Maximizing Insight from Modern Economic Analysis

by

Dolan Antenucci

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2018

Doctoral Committee:

Professor Michael Cafarella, Chair
Professor H. V. Jagadish
Assistant Professor Danai Koutra
Assistant Professor Barzan Mozafari
Professor Matthew Shapiro

Dolan Antenucci

dol@umich.edu

ORCID iD: 0000-0001-9208-6961

Dedicated to my loving wife Krista and our amazing children.

ACKNOWLEDGEMENTS

The cliché inspirational sayings about journeys and destinations come to mind when I think back on my pilgrimage to completing this dissertation: it certainly has been an interesting journey—frustrating and hopeless at times, and inspiring and full of fun at other times. I would not have arrived here without the help of many people. First, I want to thank my advisor, Michael Cafarella, for his guidance and consistently positive attitude during our time working together. While I am still amazed he finds time to sleep with his busy schedule, I am grateful for him always taking the time to meet when I needed—even if those meeting started a bit late! His out-of-the-box thinking helped shape some fascinating projects and solutions we created.

I also want to thank my past contributors, who helped bring many parts of my dissertation to fruition. Margaret Levenstein, Christopher Ré, and Matthew Shapiro were critical team members on our economics work. Mike Anderson was a core contributor to our work on declarative nowcasting. Lastly, a variety of others have contributed to our demonstration and vision papers. Working with each of these contributors was a rewarding experience, which I am grateful for.

During my time at the University of Michigan, I was fortunate enough to be a part of the database research group, where I have received a wealth of advice and feedback on my work, as well as made some great friends. Additionally, many department staff and faculty—especially my committee members—were a great help during my graduate school career. The administrative support, informative classes, and feedback on my research were all a great help.

Leading up to me applying for graduate school, several people influenced and encouraged me to pursue an advanced technical degree, as well as gave me good advice during and after the application process. Art Moore was a big influence on my renewed interest in computer science through his teachings and colorful commentary on education, life, and history. Ramesh Jain furthered this interest with his teachings and brought my attention to systems research. Additionally, he was one of the motivating factors for me choosing the University of Michigan for my studies. Along with several other faculty and instructors over the years, I am grateful for their guidance,

encouragement, and advice.

Lastly, I want to thank my family and friends for being supportive over the years. My parents and grandparents both exuded a strong work ethic throughout my life, and I think that in part helped me achieve many of my successes. I am especially thankful for my wife Krista, our son JJ, and our soon-to-arrive baby—all of which were new additions to my life during graduate school. They got to endure many hours of me doing research, yet we still managed to have some awesome adventures together, and I look forward to many more!

This work was supported by National Science Foundation DGE-0903629, IIS-1054009, IIS-1054913 and SES-1131500; Office of Naval Research N000141210041 and N000141310129; a gift from Yahoo!; and support from Google. Additionally, portions of this work are part of the University of Michigan node of the NSF-Census Research Network.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES ix

LIST OF TABLES xii

ABSTRACT xv

CHAPTER

I. Introduction 1

1.1 Transitioning to a New Workflow 4
1.1.1 New and Old Opportunities for Economic Insight . 4
1.1.2 Formalizing the Economic Analysis Workflow 6
1.1.3 Design Principles 9
1.2 Contributions and Outline 13

II. Related Work 15

2.1 Data for Empirical Analysis 15
2.1.1 Traditional Workflow 15
2.1.2 Computational Economics in the Big-Data Workflow 16
2.2 Assisting the Big-Data Workflow Phases 19
2.2.1 Identify Phase 19
2.2.2 Transform Phase 20
2.2.3 Evaluate Phase 22
2.3 Domain-Specific Tools 23

III. A Use Case of Economic Insight from Big Web Data 25

3.1 Introduction 25
3.2 Twitter Data 27

3.2.1 Strategies for Converting Social Media into Data . . 28
3.2.2 Implementation 30
3.3 A Real-time Predictor from Social Media Data 38
3.3.1 Constructing the Real-time Predictor 38
3.3.2 Analyzing the Real-Time Social Media Job Loss Index 40
3.3.3 Assessing the Information in the Real-Time Social Media Job Loss Index 42
3.3.4 Providing a Real-Time Economic Indicator from Social Media 44
3.4 Additional Applications of Social Media for Measuring Labor Market Activity 45
3.4.1 Job Search and Job Posting Indexes 45
3.4.2 Beveridge Curves 49
3.4.3 Labor Market and Hurricane Sandy 50
3.4.4 Government Shutdown 51
3.4.5 Demographics 51
3.5 Conclusion 53

IV. Morals of Nowcasting 55

4.1 Introduction 55
4.2 Survey of Nowcasting Pitfalls 57
4.2.1 Social Media Sampling Issues 57
4.2.2 Adjusting for "Grey Swans" 58
4.2.3 Weakening of Social Media Predictiveness 61
4.2.4 Too Few of Observations 64
4.3 Detecting Underlying Phenomenon Shifts 66
4.4 Lessons for Nowcasting 67
4.5 Conclusion 68

V. Feature Selection for Easier Nowcasting 69

5.1 Introduction 69
5.2 Related Work 71
5.3 System Design 73
5.3.1 Feature Preparation 73
5.3.2 Feature Selection 73
5.4 Feature Selection 75
5.4.1 Dead Ends 75
5.4.2 Unsupervised Feature Selection 77
5.5 Experiments 79
5.5.1 Experimental Setup 79
5.5.2 Feature Quality 80
5.5.3 Experimental Results 81
5.6 System Implementation 82

5.6.1 Efficient PMI Calculations . . . 82
5.6.2 User Interface . . . 83
5.6.3 Example of a User Interaction . . . 84
5.7 Future Work and Conclusion . . . 85

VI. A Declarative Query Processing System for Nowcasting . . . 86

6.1 Introduction . . . 87
6.2 Problem Statement . . . 90
6.3 System Architecture . . . 94
6.3.1 Scoring and Ranking . . . 94
6.3.2 Aggregation . . . 96
6.4 Query Optimizations . . . 97
6.4.1 Candidate Pruning . . . 97
6.4.2 Dynamic Query Optimizations . . . 99
6.4.3 Low-Level Optimizations . . . 100
6.4.4 Time Complexity . . . 101
6.5 Detecting Poor User Queries . . . 101
6.6 Experiments . . . 103
6.6.1 Experimental Setting . . . 103
6.6.2 Performance Evaluation . . . 104
6.6.3 Quality of Standard Query Model . . . 108
6.6.4 Quality of Distant Supervision Model . . . 111
6.6.5 Effectiveness of Quality Alarm . . . 113
6.7 System Implementation . . . 114
6.7.1 User Interface . . . 115
6.8 Related Work . . . 116
6.9 Conclusion and Future Work . . . 118

VII. Constraint-based Explanation and Repair of Data Transformations . . . 119

7.1 Introduction . . . 120
7.2 User Model . . . 123
7.2.1 Overview . . . 123
7.2.2 Data Transformation Primitives . . . 124
7.2.3 System Input . . . 125
7.2.4 System Output and Evaluating Results . . . 127
7.3 Reverse Data Management . . . 127
7.4 Finding Data Transformations . . . 129
7.4.1 Overview . . . 129
7.4.2 Modeling as an Optimization Problem . . . 129
7.4.3 Adding Dependence Information . . . 131
7.4.4 Prioritizing Dependence Information . . . 132

7.4.5 Adding Code-Based Constraints 133
7.4.6 Optimization Solver 134
7.4.7 Result Generation 134
7.4.8 Greedy-Hybrid Approach 135
7.4.9 Full Algorithm for Solution Finding 135
7.4.10 Time Complexity 136
7.5 Prototype System 137
7.6 Experiments 138
7.6.1 Experimental Setting 138
7.6.2 Evaluating on Synthetic Data 139
7.6.3 Evaluating with Real-World Data 144
7.6.4 Evaluating with Code-Based Constraints 146
7.6.5 Evaluating Algorithm Components 146
7.7 Related Work 149
7.8 Conclusion and Future Work 151

VIII. Conclusion and Future Work 152

8.1 Conclusion and Future Work 152
8.1.1 User Interaction for Tedious Work 152
8.1.2 Simplifying Nowcasting 153
8.1.3 Economic Analysis 153

BIBLIOGRAPHY 155

LIST OF FIGURES

Figure

3.1 Twitter Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Principal component factors calculated based on the correlation matrix of signals shown in Table 3.4. 35

3.2 Initial Claims for Unemployment Insurance and Job Loss and Unemployment Factor 1. Note: Figure shows the Department of Labor's Initial Claims for Unemployment Insurance (left scale, revised data, seasonally adjusted) and the Social Media Factor 1 (right scale). The factor is estimated as described in the text and is no way fit to the initial claims data. 36

3.3 Initial Claims for Unemployment Insurance and the Social Media Job Loss Index. Note: Figure shows the Department of Labor's Initial Claims for Unemployment Insurance and the Social Medial Job Loss Index. The Social Media Job Loss Index is estimated in sample in the shaded area and recursively thereafter. 41

3.4 Surprises Predicted by Social Media Job Loss Index. Note: Surprise is Department of Labor Initial Claims for Unemployment Insurance (preliminary or revised) minus the consensus forecast. Predicted with Social Media Job Loss Index constructed based on factor 1, as described in the text. The index is generated recursively except in the shaded area, where it is generated over the entire shaded sample. . . 44

3.5 Social Media Indexes for Job Search and Job Posting. Note: Indexes are based on factor loadings in second two columns of Table 3.12. The social media indexes are estimated in sample in the shaded area and recursively thereafter. 49

3.6 Beveridge Curves for job search and job posting trends. Note: Figures show the four-week moving averages of the Social Media Job Loss Index versus the Search and Posting indexes. 49

3.7 Social Media Signal Related to Hurricane Sandy. 50

4.1 Our original nowcasting model (Old Model) for unemployment (using only *job loss* signals), where increasing estimation errors begin in 2014. 56

4.2 Our original nowcasting model for unemployment with "everyday phrases" used to normalize the social media signals. 58

4.3 Our original nowcasting model with the different grey swan filtering methods applied. Sample period is July, 16, 2011 through April 25, 2015 (weeks ending Saturday). 59
4.4 Our new nowcasting model for unemployment (using *job loss*, *job search*, *vacancies*, *hired*, and *quits* related signals), trained with the last two years of data at each time period. 64
4.5 The different factors of our New Model, built using data from July 2011 to January 2016. The dashed red line is at $y = 0$. 65
4.6 Our original nowcasting model (OldModel) for unemployment (using only *job loss* signals), but trained only on the most recent two years of data. When the underlying usage of *job loss* in social media changed in early 2014, the model took half a year to adjust. 66
5.1 The pipeline RINGTAIL uses to convert a corpus of tweets into a set of *(gram, signal)* pairs. 74
5.2 RINGTAIL's overall architecture. 75
5.3 The RINGTAIL search result page shows **Suggested Signals** results in the left-hand column. 84
6.1 A comparison of the traditional approach to nowcasting and the RACCOONDB architecture. 94
6.2 Effects of different combinations of RACCOONDB's pruning optimizations. Results shown are for $\beta = 1$ on 1 core; runtime in logarithmic scale. 106
6.3 Runtimes for the non-optimized (NONOP) and optimized (FULLOP) systems with different values of β on 1 core. 108
6.4 RACCOONDB (RACDB) results with different β weighting preferences (0, 0.125, 0.25, 1, 128, ∞), compared with our user study baseline (MANUAL). SIGNALRANK ($\beta = 0$) and SEMANTICRANK ($\beta = \infty$) are the end points on the RACDB line. 110
6.5 A distant supervision query and nowcasting result for the target phenomenon of *Insurance* (using ITERRACDB). 113
6.6 User interface for searching in our prototype online community. . . . 114
7.1 An illustration of Example 7.1, showing the time-consuming and tedious process of repairing data transformations. 120
7.2 The user model for an explain-and-repair data transformation system, showing Janet's usage in Example 7.2. 125
7.3 Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when number of predicates is varied. Big bars are better on top chart, smaller on lower chart. 141
7.4 Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when number of data items is varied. Big bars are better on top chart, smaller on lower chart. 142
7.5 Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when the level of correlation is varied. Big bars are better on top chart, smaller on lower chart. 143

7.6 Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods evaluated using real-world datasets. Big bars are better on top chart, smaller on lower chart. 145
7.7 Percent of solutions found by EMERIL when matching multiple constraints versus just one signal-based constraint. 147
7.8 Percent of time a solution found when varying the number of predicates and using either EMERIL or NChooseK with varied max runtimes of 1, 15, 30, and 60 minutes. 147

LIST OF TABLES

Table

3.1 Summary Statistics for Signals: Job Loss and Unemployment (Weekly Rate per Million tweets). Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Sample is 19.3 billion total tweets of which 2.4 million are job loss and unemployment related. See Table 3.2 for detailed descriptions of phrases for signals. 31

3.2 Social Media Signals: Job Loss and Unemployment. Note: The signals are counts of tweets that contain 4-grams with the indicated phrase where "|" denotes a space and "*" is a wildcard. The last column indicates the number of distinct phrases found in the database of tweets matching the target phrases with wildcards. 32

3.3 Correlation of Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013. The "Unemployment" signal is purged of the Employment Situation effect, as described in the text. 33

3.4 Factor Loadings on Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Principal component factors calculated based on the correlation matrix of signals shown in Table 3.3. 34

3.5 Predicting Initial Claims: Consensus, Social Media Factor, and Lagged Dependent Variable. Note: Sample period is July 16, 2011 through November 2, 2013. Standard errors in parentheses. Dependent variable: Initial Claims for Unemployment Insurance (preliminary data in Table 3.5a, revised data in Table 3.5b). Regressors: Lagged dependent variable, consensus forecast, and social media factor 1 scaled to have same units as initial claims. 37

3.6 Constructing the Social Media Job Loss Index. Note: The dependent variable is the Department of Labor Initial Claims for Unemployment Insurance (thousands, seasonally adjusted). The independent variables are the job loss and unemployment factors. The Social Media Job Loss Index is based on regressions re-estimated each week using real-time data available as of the prediction period, as described in text. This table presents the estimates for the final week in the sample. Sample period is July 16, 2011 through November 2, 2013. Standard errors in parentheses. 39
3.7 Prediction Errors of Social Media Job Loss Index. Note: Table gives the root mean squared error (RMSE) of the Social Media Job Loss Index for initial claims for unemployment insurance (preliminary data and revised data). The models and RMSEs are estimated recursively, using data from July 16, 2011 forward, for weeks ending July 16, 2011 through November 2, 2013. 40
3.8 Incremental Information in Social Media Job Loss Index. Note: Sample period is July 16, 2011 through November 2, 2013 (recursive sample). Standard errors in parentheses. Dependent variables: Columns (1)-(3), Preliminary initial claims minus consensus; Columns (4)-(6), Revised initial claims minus consensus. 42
3.9 Summary Statistics for Signals: Job Search and Job Posting (Weekly Rate per Million tweets). Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Sample is 19.3 billion total tweets of which 2.4 million are job loss and unemployment related. See Table 3.11 for detailed descriptions of phrases for signals. 45
3.10 Correlation of Job Search and Job Posting Signals. Note: Sample period is July 16, 2011 through November 2, 2013. 45
3.11 Social Media Signals: Job Search and Job Posting. Note: The signals are counts of tweets that contain 4-grams with the indicated phrase where "|" denotes a space and "*" is a wildcard. The last column indicates the number of distinct phrases found in the database of tweets matching the target phrases with wildcards. 46
3.12 Job Search and Job Posting Factors: Loadings of First Factor, Alternative Sets of Signals. Note: Table shows the factor loading for the first factor for the selected signals. The bottom two rows report the variance of the first factor and the fraction of the overall variance accounted for by the first factor 48
3.13 Signals by Age and Sex. Table shows fraction of job-related signals by age and sex of sender. The demographics are estimated probabilistically and are coded for only a subset of signals. Because of changes in the API, this sample ends June 15, 2013. 52
4.1 Regression results for our original model and that with the new normalizer. Standard errors in parentheses, and sample period is July, 16, 2011 through April 25, 2015. 58

4.2 Regression results for our original model and that with the different grey swan adjustments applied. Standard errors in parentheses, and sample period is July, 16, 2011 through April 25, 2015. 60
4.3 A sample of phrases used in the Old and New models. The Old model uses only the Job Loss phrases, whereas the New model uses all of the phrases. Note: the asterisk works as a wildcard, including any phrases with any word in its place (e.g., "found * job" would include "found a job"). 63
5.1 Target phenomena used for testing. US Unemployment refers to the weekly number of initial unemployment insurance claims. Amazon.com, Ebay.com, Walmart.com, and 30 other top-traffic websites were selected to represent E-commerce Traffic. 80
5.2 Evaluation of different feature selection mechanisms, averaged over the tasks in Table 5.1. For average correlation and R^2, larger values are better; for average MAE, smaller is better. Best ones, or close to best, are in bold. 80
6.1 Frequently used notations. 92
6.2 Target phenomena used in experiments. All targets are measured weekly, except guns and temp, which we respectively convert from monthly and daily to weekly. 104
6.3 Query processing times for PostgreSQL, Apache Spark, and two versions of RaccoonDB: one without our pruning optimizations (NonOp) and another with them enabled (FullOp). 105
6.4 Breakdown of 1-core runtime for NonOp and FullOp, averaged over all targets. Overhead includes pruning in FullOp. 107
6.5 Nowcasting quality results for RaccoonDB's two query models and our baselines. **SS-**F_1 is the harmonic mean of the signal correlation and semantic relevance scores of each method. All values range from 0–1, with 1 being best. 109
6.6 RaccoonDB result quality improvements after revising queries with poor semantics (IterRacDB). Values are the harmonic mean of semantic relevance and signal correlation. †These statistics include RacDB results if no revisions were made. 112
6.7 Average result quality improvement from excluding queries that triggered the alarm (*Alarm*) and also excluding results with low semantic relevance (*Alarm + User*). 116
7.1 Notation used to describe our user model. 124
7.2 Real-world datasets used in our experiments. Number of predicates is using the *naïve* generation method, and correlation is average Pearson correlation between the dataset's columns. 144
7.3 Percentage of time a solution found for our synthetic datasets when using Emeril with and without prioritized dependence information (greedy-hybrid search disabled in both cases). Results are averaged across all datasets in each experiment (e.g., for "Rows," that is 30 x 10 datasets). 150

ABSTRACT

The last decade has seen a growing trend of economists exploring how to extract different economic insight from "big data" sources such as the Web. As economists move towards this model of analysis, their traditional workflow starts to become infeasible. The amount of noisy data from which to draw insights presents data management challenges for economists and limits their ability to discover meaningful information. This leads to economists needing to invest a great deal of energy in training to be *data scientists* (a catch-all role that has grown to describe the usage of statistics, data mining, and data management in the big data age), with little time being spent on applying their domain knowledge to the problem at hand. We envision an *ideal workflow* that generates accurate and reliable results, where results are generated in near-interactive time, and systems handle the "heavy lifting" required for working with big data.

This dissertation presents several systems and methodologies that bring economists closer to this ideal workflow, helping them address many of the challenges faced in transitioning to working with big data sources like the Web. To help users generate accurate and reliable results, we present approaches to identifying relevant predictors in nowcasting applications, as well as methods for identifying potentially invalid nowcasting models and their inputs. We show how a streamlined workflow, combined with pruning and shared computation, can help handle the heavy lifting of big data analysis, allowing users to generate results in near-interactive time. We also present a novel user model and architecture for helping users avoid undesirable bias when doing data preparation: users interactively define constraints for transformation code and the data that the code produces, and an *explain-and-repair* system satisfies these constraints as best it can, also providing an explanation for any problems along the way. These systems combined represent a unified effort to streamline the transition for economists to this new *big data workflow*.

CHAPTER I

Introduction

The last decade has seen a growing trend with economists exploring how to extract different economic insights from "big data" sources such as the Web. In traditional empirical analysis, economists often analyze survey-based datasets that government agencies collect and maintain. Consider the following example which illustrates the *traditional workflow* for economists:

Example 1.1. *Janet is an economist in 2012, and Hurricane Sandy just heavily damaged parts of New York and New Jersey. She wants to investigate the effect that the hurricane will have on unemployment in the area. She starts by* ***identifying*** *a monthly unemployment survey from the US Bureau of Labor Statistics, which polls 60,000 eligible households each month to determine their household members' employment status. She would prefer to have data that directly surveys businesses about the hurricane's effect on their hiring, but this data is not available and she lacks the resources to collect it. Additionally, since the unemployment survey data is only collected once a month with live interviews [21], she has to wait several weeks before data covering her desired time period is available.*

Once the data is collected, she imports it into a statistical package like STATA, where she first removes all survey results not in the affected area nor in the timeframe during which the hurricane hit. Next she applies a simple ***transformation****, where she normalizes the monthly survey data by the total eligible households nationwide for each time period.*

To ***evaluate*** *her data, she estimates linear regression models for each area, including an indicator variable for the time period after the hurricane hit. She finds that there indeed was a spike in unemployment, but her findings are not available until weeks after the hurricane. Finally, her colleague, who is excited about the discovery, asks if she can focus on Staten Island—an area hit especially hard. Janet discovers*

that the survey data has only a few observations from Staten Island, so she concludes that the results would be untrustworthy.

As Example 1.1 illustrates, this traditional workflow has some limitations. First, the standard form of data collection is slow and time-consuming. Collecting data from hundreds or thousands of individuals or businesses—over the phone, in person, or even electronically—is labor intensive and slow, delaying needed information by weeks or longer. Second, these survey-based datasets can suffer from missing data and small sample sizes, thus limiting their applicability to certain experiments. Third, the availability of these conventional datasets is limited or not available for many phenomena and trends. Because of the high cost of data collection and curation, governments and economic agencies can only collect and maintain a relatively small amount of these datasets, so there are many trends and phenomena that are not represented. Likewise, some of the datasets (such as several surveys from the US Census Bureau) require exorbitant effort to access (e.g., conducting all research onsite, getting all exported data approved).

In contrast, a new *big-data workflow*, which is centered around data sources like the Web, offers a solution to many of these shortcomings. Since the 1990s, computer storage costs have dropped drastically, so the amount of data being collected by businesses and government agencies has grown exponentially. Combine this with the advent of social media, there are many new data sources from which economic insights can be drawn. These data are constantly updated and are accessible with a quick download (and partnership if referring to private business data), which means these data are much *faster* and *cheaper* than traditional surveying methods. Additionally, these datasets often represent a much broader population and include more variables, allowing for much finer-grained analysis than traditional survey-based data.

Einav and Levin [77] summarize some of the distinguishing characteristics of this new workflow:

- Data are often available in real time, but without the guarantees of accuracy or detail some survey-based datasets have.
- Data often requires some clever or advanced processing, unlike the traditional workflow where data comes in the standard "rectangular" format (with N observations and $K \ll N$ variables).
- New variables and unmeasured activities (e.g., personal communications, geolocation data) can be captured.

As economists move towards this model of analysis, their traditional workflow starts to become infeasible. Consider the following example:

Example 1.2. *Janet is an economist in 2012, and Hurricane Sandy just heavily damaged parts of New York and New Jersey. She wants to investigate the effect that the hurricane will have on unemployment in the area. She starts by* ***identifying*** *and collecting recent tweets that mention "unemployment." She collects the tweets by searching Twitter.com and pasting the results into an Excel spreadsheet. She then has to manually visit the user profile pages for each tweet to collect the user's listed location, which she then notes in her spreadsheet. This process is quite labor intensive, taking her several days to complete.*

Next, she ***transforms*** *the tweets into two time series variables, with the first recording the daily occurrence of "unemployment" in the tweets from users in her target area (New York and New Jersey), and the second recording the daily occurrence of "unemployment" in the remaining tweets. To do this, she manually collects the counts for each day in Excel and saves these to a new spreadsheet—another laborious process. Ideally, she would hire a programmer to help with this, but she is unable to find anyone at short notice, and she wants the data as soon as possible.*

To ***evaluate*** *her data, she estimates linear regression models for each area using her social media signals, each with an indicator variable during the time period after the hurricane hit. She finds that the social media signals have little predictive power, and she realizes that the time series variable primarily captures news about monthly unemployment announcements (i.e., by the government) rather than that of social media users indicating their employment status, so she repeats the above steps several times for different phrases ("lost my job", "got fired", etc.) until she finally finds a model with good prediction quality. When her colleague inquires about Staten Island, she discovers that her Twitter-based signals have enough observations to draw reliable inference.*

Janet benefits from this new workflow by receiving her results much faster than the traditional workflow (even after the delays from her laborious collecting, labeling, and repeated iterations). Additionally, she is able to study phenomena that are under-sampled in survey-based data (e.g., the Staten Island investigation). Janet also encounters several obstacles in the process: (1) her lack of programming experience requires her to perform a lot of manual data cleaning of her social media data, and (2) due to the noisy nature of the social media data, she needed to do several iterations of her collecting and transforming before she found a model with acceptable quality. These challenges have slowed the adoption of big-data workflows.

1.1 Transitioning to a New Workflow

In this section, we further distinguish the two workflows by first presenting several example projects. We then formally define and compare these workflows. Finally, we present the characteristics of an *ideal workflow*, one which maximizes economists' ability to discover economic insight.

1.1.1 New and Old Opportunities for Economic Insight

As we saw in Examples 1.1 and 1.2, the new workflow can supplant the traditional one, but there are other workloads that do not have an obvious counterpart in the new workflow, and it is for this reason that we emphasize that the traditional workflow still has its place in an economist's repertoire of tools. Likewise, the new big-data workflow offers new opportunities for insight that were impossible under the traditional workflow.

1.1.1.1 Examples in the Traditional Workflow

Consider the following example, which illustrates the type of workload that will continue to exist as a traditional workflow use case:

Example 1.3. *Janet is back at her economic modeling. She wants to estimate how a minimum wage increase will affect the availability of jobs. Following a related project [119], she starts by* ***identifying*** *and collecting government administrative data on employment trends (e.g., the Quarterly Census of Employment and Wages) and historic state minimum wage data from the US Department of Labor. After importing these datasets into a statistical package like STATA, she* ***transforms*** *the data into a format that is conducive to analysis. To do this, she extracts the data from two states of interest—one that received a minimum wage increase and another that did not. She then normalizes these data by population (using a different dataset from the US Census Bureau). To* ***evaluate*** *the impact of the wage increase, she estimates a differences-in-differences model. She finds that a minimum wage increase does indeed appear to have a negative effect on the availability of jobs.*

The traditional workflow works perfectly well for this analysis: retrospective analysis is acceptable, so timeliness of the data is less of a concern; she is focusing on state level analysis, so her data has sufficient sample size. In contrast, if she was analyzing a smaller geographic region or she needed timely analysis (say, for investigating the

recent affect of a minimum wage increase), the new big-data workflow should be considered. (One can imagine using LinkedIn user profiles that indicate recent fast food experience, and then mining through any social media messages from these users for indication of their employment status.)

The next example illustrates a use case that is most likely not yet represented by data sources in the big-data workflow:

Example 1.4. *Our economist Janet has been commissioned by the European Union to investigate optimal shipment sizes in freight transport. She starts by* ***identifying*** *some survey data that was collected in 2004 by the European Community Humanitarian Office and is composed of approximately 10,000 shipments sent by some 3,000 shippers. For her model, she plans to follow the work of Combes [68], so she* ***transforms*** *her data with some simple cleaning, such as filtering out shipments that are missing needed data and creating new variables out of existing ones available. Finally, she* ***evaluates*** *her theory by estimating the optimal shipment size using a linear regression.*

Public data on shipment sizes and quantities is probably not available on the Web, so even though the data Janet uses is over ten years old, she can still evaluate her model and generate an estimate for optimal shipment sizes. These types of projects will continue to be popular in the traditional workflow.

1.1.1.2 Examples in the Big-Data Workflow

In this next example, we see a case where the big-data workflow allows for economic insight not available in the traditional workflow:

Example 1.5. *Janet wants to measure inflation in Argentina, a country that has been known to misrepresent this number in the past [59], so she decides to follow related work that estimates a consumer price index from online product prices [58]. She starts by* ***identifying*** *and collecting price data from different online retailers. Without a programing background, she is forced to manually collect this data, which takes several days. She then imports this data into STATA, where she* ***transforms*** *it by aggregating the prices into product categories and separating the prices by country. Finally, she* ***evaluates*** *her data by building two consumer price indexes, one for the US and another for Argentina. She compares her US index against official data released by the US government, and when it matches well, she is confident that her Argentina index gives an accurate estimate of Argentina's inflation.*

If Janet had used the traditional workflow, thus basing her inflation estimate from data collected by the Argentinian government, she would risk using fabricated data. Unfortunately, Janet struggled again with collecting and transforming her data, requiring a great deal of manual work.

In this next example, we see another case where the big-data workflow supplants the traditional workflow:

Example 1.6. *Janet wants to measure the response of spending to income, so she decides to mimic the work of others by* ***identifying*** *a financial aggregation company as a source for her analysis [87]. She contacts the company and they are willing to partner with Janet for her analysis. The data provided to her includes 60 million transactions from 75,000 users. The data is provided to her in a relational database, and since she has some experience with SQL, she performs all of her analysis in this system. She starts by* ***transforming*** *the data into weekly income and spending groups for each user, where the spending groups represent total spending, nonrecurring spending, and spending on fast food and coffee shops. Janet struggles with this grouping, where she first has to manually label all transactions, which she does by writing hundreds of SQL* UPDATE *statements, and even with all this manual work, many transactions are left unlabeled. Additionally, the volume of data is too large to update at once, so she focuses on a much smaller sample (e.g., 2% of total users) at a time. For* ***evaluation****, she exports the grouped data to analyze with STATA, where she models spending as a function of income arrival. She concludes that there is a relation between spending and income arrival, but much of this is due to recurring expenses like rent and utilities.*

We can see in this example that Janet struggles to work with larger datasets, even when using a relational database. She spends a lot of time on data cleaning and labeling.

1.1.2 Formalizing the Economic Analysis Workflow

We will now formally define the economic analysis workflow. As we showed in the examples above, these workflows follow a similar process:

1. **Identify** and collect the right data.
2. **Transform** the data into a new dataset for analysis; repeat phase 1 as needed.
3. **Evaluate** the data through modeling and evaluation metrics; repeat phases 1 and 2 as needed.

We compare and contrast these phases for both the traditional and big-data workflows below.

1.1.2.1 Identify Phase

Traditional Workflow — In the identify phase, the traditional workflow relies on high quality, expensive-to-collect datasets. Often these come from government agencies who invest a great deal of resources in creating and maintaining them, but they can also come from researchers directly, assuming they are willing to invest the resources.

Big-Data Workflow — In contrast, the big-data workflow has a massive number of potential sources of data for drawing economic insights, and domain knowledge plays a large role in determining how and from whom to collect data. Poorly chosen data can lead to overfitting and having to repeat work (as we showed in Example 1.2).

Big-Data Workflow Challenges — The volume of data available makes management and analysis quite challenging, especially for economists whose tools and training traditionally do not use big data. Collecting massive amounts of data, whether through an API or a crawl of the Web, is challenging, especially for those with little programming background. Once collected, storing and filtering is especially challenging. The traditional tools used by economists, like STATA and SAS, traditionally process data with a single machine, so a decently sized dataset can take hours or days for simple operations.[1]

In recent years, a large number of systems and tools have been developed to help address some of the challenges of working with big data (MapReduce/Hadoop ecosystems [71,163], NoSQL data stores [57,96], and NewSQL database systems [106, 149]), but these tools still have expensive setup costs and steep learning curves. If an economist is trained as a data scientist, then they can be successful with using them, but since most economists lack this training, many of the tasks in this phase (collecting, storing, filtering) require a great amount of manual work or waiting.

[1]Recent efforts have been made to better support working with big data in these tools (such as in Matlab); however, these solutions still have limitations in usability and not all functionality is supported.

1.1.2.2 Transform Phase

Traditional Workflow — In the transform phase, the traditional workflow has a pretty limited space for transformations due to the relatively small number of survey-based datasets available. When these surveys were first created, the creators had a schema in mind for how the world works. This schema limited the data collected, and thus limits the view of the world that these data can potentially represent. For example, a researcher cannot retroactively add a question to a survey and get answers for previous time periods. With many survey-based datasets established many years—or decades—in the past, researchers are limited to the economic questions that the creator of the survey foresaw at the time. Combine this with the already small nature of the data, the space for transformations is reduced even further. Filtering, normalizing, maybe some aggregations and joins with other surveys are options, but these only get you so far. Domain knowledge helps with knowing what datasets to work with and what the cultural norms are for the transformations on them.

Big-Data Workflow — In contrast, the big-data workflow has a massive number of datasets, and there are no intellectual filters on whether their schemas are good. This leads to a massive space for transformations, but it is hard to know what transformations to use; there are no cultural norms for preferring one transformation over another, as there is in the traditional workflow. Instead, researchers have to argue for the ones chosen. This leads to domain knowledge playing a much greater role. With a combination of domain knowledge and some creativity, the transform phase is often the driving force behind economic insights in the big-data workflow.

Big-Data Workflow Challenges — This new data is quite noisy—unlike the generally accurate and detailed survey-based datasets in the traditional workflow—and they often require some clever or advanced processing. Additionally, even the most trivial of transformations can introduce undesirable bias and corrupt conclusions drawn from downstream analyses. Even if one identifies that bias has been introduced, fixing it can be time-consuming and tedious while still remaining close to one's desired dataset.

As discussed above, a large number of systems and tools have been developed to help with the challenges of working with big data (MapReduce/Hadoop ecosystems, etc.) These tools can be used for different transformations like sampling, aggregating, or joining different datasets, but they still are limited by their expensive setup costs and steep learning curves. Similarly, other tools that can help with data clean-

ing [66, 107] and integration [115]—two important tasks that often happen in a loop of identifying and transforming—do little to help identify and fix undesirable bias in the transformed data.

1.1.2.3 Evaluate Phase

Traditional Workflow — In the evaluate phase, economists decide how to estimate a theoretical model they are trying to test or a trend they are trying to monitor. The previous examples often used a simple linear regression, but there are a variety of methods available. Additionally, a savvy analyst would also evaluate her data pipeline, confirming that no undesirable bias was introduced by transformations applied to the raw input data. Domain knowledge plays an important role in this phase.

Big-Data Workflow — Evaluation in the big-data workflow is very similar to the traditional workflow, except often on a larger scale: Multiple rounds of evaluation may be needed since the raw input data is often noisier, so finding one's desired indicators is not as straight-forward. Domain knowledge continues to play an important role in this phase.

Big-Data Workflow Challenges — There are challenges in the evaluation phase with identifying if the economic insights extracted are valid and if the model or approach remains valid over time. Overfitting becomes a serious problem when the number of observations is far less than the available variables from which economic insight can be drawn. For example, in some of our nowcasting work (Chapter VI), we show how spurious relationships are a serious challenge in nowcasting applications.

Additionally, as Example 1.2 shows, the big-data workflow is often an iterative process, where several rounds of identifying, transforming, and evaluating are needed before the desired economic insight is achieved. With work involving multiple systems, this process becomes even more burdensome. Likewise, transitioning between phases often requires substantial overhead. These transitions often occur between disjoint systems, and managing such a collection of systems becomes a chore in itself, requiring the expertise of a modern data scientist. This often means more work and delays in processing.

1.1.3 Design Principles

We envision an *ideal workflow*, which addresses many of the challenges listed above. If Janet was in this new world, what are the qualities of this new workflow?

1. **Accurate and Reliable Results** — Overfitting should be avoided, and users should be made aware of when models stop working.

2. **Results in Near-Interactive Time** — From data collection to generating results, systems should run in near-interactive time.

3. **Undesirable Bias Avoided** — Systems should help users identify and address any undesirable bias introduced during the transform phase.

4. **Heavy Lifting Automatically Handled** — Systems should have seamless integration. From data collection to generating results, workflows should not require excessive data management.

Accurate and Reliable Results — In order to get accurate and reliable results, economists need models that can generate them. The first step to having accurate and reliable models is to build them with *high-quality features*. These features not only contain information about one's target phenomenon, but are also relevant from a human perspective. For example, the use of the phrase "pumpkin muffins" on social media will likely yield a time series that is correlated with phenomenon of *flu activity*, because both spike as the weather cools. But future swings are unlikely to be closely linked, and no practitioner would accept results from a model using this feature. With the massive amount of potential features that can be generated from big data sources, users need help building and identifying high-quality features. Since many users are not great at describing what they want to a system initially, users often repeat a loop of identifying and testing different data or features. Systems can help with this by offering interfaces that aid with this exploration process by exploiting any domain knowledge users can provide and notifying them when they may have provided uninformative or low-quality data.

Another challenge with generating accurate and reliable results comes from identifying when an existing model starts to fail. This type of failure can happen for a variety of reasons, such as a model's features losing their relevance. For instance, a model estimating flu activity with social media phrases may have good success with the phrase, "I have a fever," but if a new, popular song is titled, "I have a Fever," this feature may get flooded with this new usage of the phrase, and the model will indicate a spike in flu activity. Identifying such model failures should be simple and not require a great deal of manual investigation.

Results in Near-Interactive Time — In order to generate results in near-interactive

time, systems should minimize the amount of time users are waiting and reduce the amount of workflow iterations they perform. To reduce user wait times, systems can exploit offline computation and indexing to help speed up the searching and identifying data. Likewise, systems should attempt to parallelize different online operations like data transformations and model training. Pruning operations can also help avoid unneeded computation. For instance, if building potential features for a model from a mass of data, many of those features will not be relevant for the user's desired model; identifying the potentially irrelevant features *before* feature building can save a lot of time.

To help reduce the number of workflow iterations that a user performs, systems should aid users in the **identify** phase to find relevant data early on. Exploiting any domain knowledge that the user has should help with this. Likewise, helping users identify which transformations to apply so that quality features are generated will help reduce additional rounds of the **transform** and **evaluate** phases.

Undesirable Bias Avoided — In order to help users avoid undesirable bias in the transform phase, systems should assist users with identifying potentially undesirable bias (such as changes in the distribution of columns or attributes between the raw input data and the transformed data). Users should then be able to provide feedback, and the system can respond with suggested revisions to the transformation code to avoid this bias. Additionally, systems can make creating transformations easier through declarative interfaces that allow users to define desired characteristics on the transformation code and on the data it produces, and the system can then use these *user constraints* to find the transformation code that best satisfies them.

Heavy Lifting Automatically Handled — In the traditional workflow for economic analysis, economists do a fair amount of data management, where they are importing and exporting data between different files and systems. The big-data workflow exacerbates this problem since the size of data being imported and exported grows significantly. Combine this with how multiple iterations of work are common, the problem is even worse. Ideally, systems would have seamless integration from start to finish of the analysis workflow. This can be done by building systems around the actual data being used; simplifying access to the data; and pre-transforming the data into potential features, helping users avoid the burden of feature engineering [33, 35].

1.1.3.1 The Ideal Workflow in Action

Now that we have an idea of our ideal workflow, let us revisit Janet in Example 1.2 to see how she may fare in this new workflow:

Example 1.7. *Our economist Janet wants to estimate Hurricane Sandy's effect on unemployment behavior. She starts by visiting a system where she indicates her target phenomenon. The system asks her to provide any domain knowledge she has in the form of keywords associated with her target (e.g., "job loss, unemployment, searching") and any trends the target may have (e.g., she may indicate that unemployment generally spikes after the holidays when temporary jobs end, and there may be another spike after the hurricane hit). The system quickly* ***identifies*** *a set of messages on social media that have a high similarity with her keywords and provided trends.*

She then uses this system to declaratively describe the ***transformations*** *she wants to apply to the messages: (1) features should represent the weekly frequency of different phrases in social media; (2) ignore English messages; (3) keep phrases "simple" (i.e., clean of punctuation and non-standard characters). The system then explores the space for applying different transformations, while considering the goodness-of-fit and complexity of potential models. After several seconds, the system suggests a set of features that* ***evaluate*** *well, keeping data and model complexity relatively low while still maintaining good result quality. Happy with the result, she exports the result and data lineage to share with colleagues.*

Additionally, consider Janet in the following example addressing bias in her transformation code:

Example 1.8. *Janet has been given access to a financial tracking service's dataset of consumer spending data from the last three years. This dataset includes demographic information on the consumers, such as age, income level, marital status, and number of children. Janet wants to validate her theory that families with several children spend more on transportation-related expenses (e.g., fuel, car insurance) than single-child families. She starts by visiting a system where she indicates her desired* ***transformations****: (1) filter for consumers with three or more children, (2) filter for married consumers; (3) and include transaction types* $\{fuel, car\ insurance, auto\ repairs\}$.

The system returns a transformed dataset, but also warns Janet that the distribution of age has skewed to older individuals and that the amount of monthly contributions to retirement accounts drastically dropped from the input dataset. Janet realizes older individuals may bias her conclusions since they often have multiple vehicles

(RVs, etc.), so she indicates that the change in age distribution is undesired. The system responds with a new dataset, updated transformation code—which includes a relaxed children filter: consumers with two *or more children—and a new list of column distribution changes. Janet sees no undesirable bias in the new list of distribution changes and happily accepts the system's result.*

1.2 Contributions and Outline

This dissertation presents several methodologies and data systems that address many of the challenges faced by economists transitioning to the new big-data workflow.

We start with a discussion of related work in Chapter II, which also includes several real-world examples of projects and data in the traditional and big-data workflows.

In Chapter III, we present a detailed example of using the new big-data workflow to contemporaneously forecast unemployment behavior using social media—a process we refer to as *nowcasting* [38]. We explore using domain experts to help in the **identify** and **evaluate** phases to efficiently choose features and recognize different trends that are well-represented by social media—while we handle the *big data heavy lifting*. We show that social media can be used to track official unemployment insurance claims data; that it carries incremental information relative to other leading indicators for forecasting unemployment behavior; and that it allows for retrospective analysis of labor market shocks (e.g., government shutdowns and natural disasters).

We then present in Chapter IV several lessons we learned through our nowcasting research to enhance the **identify**, **transform**, and **evaluate** phases of our big-data workflow. We start with a survey of some common points of failure in nowcasting models, along with ways to identify and address these issues. We then propose a method for detecting and observing underlying changes in a nowcaster's target phenomenon. Finally, we outline several methodologies for obtaining accurate and reliable results in future nowcasting projects.

In Chapter V, we propose RINGTAIL [39,40], a system that exploits domain knowledge from users to help them **identify** relevant social media signals for nowcasting. Through several experiments, we show that our wholly-automated feature selection methods are better or as good as those from a human and substantially better if RINGTAIL receives some human assistance.

In Chapter VI we further extend our nowcasting work, where we propose RACCOONDB [34, 36, 37], a declarative query processing system that yields high-quality

nowcasting results with little user effort. We show how user domain knowledge can be further exploited to simplify the **identify** and **evaluate** phases of our new big-data workflow to help users achieve accurate and reliable results (up to 57% improvement over standard nowcasting approaches). We also show how several optimizations allow us to provide results in interactive time (with a 424x speedup over our non-optimized system).

In Chapter VII, we propose a novel user model and architecture for *explain-and-repair* data transformation systems, which aids users with the **transform** phase. Users interactively define constraints for transformation code and the data that the code produces. The system satisfies these constraints as best it can, and also provides an explanation for any problems along the way. We present an algorithm that yields transformation code based on user constraints on the output data and transformation code. We implemented and evaluated a prototype of this architecture, EMERIL, using both synthetic and real-world datasets. Our approach finds solutions 34% more often and 77% more quickly than the previous state-of-the-art solution.

We conclude with our general observations and a discussion of future work that further helps economists with their transition to the new big-data workflow (Chapter VIII).

CHAPTER II

Related Work

In this chapter, we present several areas of related work. We start with some examples of data within the traditional and big-data workflows described in Chapter I. We then present existing efforts to help in the different phrases of these workflows, and lastly present some existing work in domain-specific tools.

2.1 Data for Empirical Analysis

2.1.1 Traditional Workflow

One popular example of work in the traditional workflow is the monthly unemployment rate in the US, which is maintained by the US Bureau of Labor Statistics (BLS). To collect the data for this statistic, the BLS conducts phone surveys with over 60,000 eligible households each month to determine their household members' employment status. These surveys require asking nearly a dozen questions in a live interview, with questions ranging from "Does anyone in this household have a business or a farm?" to "What are all of the things you have done to find work during the last 4 weeks?" This process delays the release of the monthly employment rate to about 3 weeks after the surveying period (generally in the middle of the following month) [21].

In another example of work in the traditional workflow, economists try to estimate the effect of minimum wage increases on different aspects of the economy. In one study [29], the authors look at the response of spending and debt to a minimum wage increase, and show that both debt and spending increase more than income increases. The data they use is from several economic surveys (e.g., Consumer Expenditure Survey (CEX), Survey of Income and Program Participation (SIPP)), where they can track households over time that have a minimum wage income, along with their

spending, income and debt levels before and after a minimum wage increase. For the CEX, the US Census Bureau electronically surveys approximately 15,000 households four times per year to collect information on buying habits, with reports issued annually [13]. Similarly, the SIPP is conducted by the US Census Bureau, where several thousand participants are surveyed in person or on the phone over a 2-4 year period to track income, labor force participation, and social program participation [16].

Combes and François [68] use survey data on freight transportation to model the choice that producers and manufacturers have for how much quantity to ship and in what size of increments. This survey data was collected in 2004 by the European Community Humanitarian Office, and is composed of approximately 10,000 shipments sent by some 3,000 shippers. This data is not maintained, so further analysis is quite limited.

These examples share some common aspects, which together demonstrate several pros and cons of such survey-driven research. In all cases, useful economic insights are being found but at the expense of having large teams dedicated to collecting and maintaining each survey. There are delays with collecting the survey data, some spanning multiple years before new reports are released, so insights can lose relevance by the time they are found. While these surveys are adjusted occasionally to add new questions, retrospective inclusion of data is impossible. Additionally, with only a small percentage of the population being surveyed, there is always a risk with sampling errors.

2.1.2 Computational Economics in the Big-Data Workflow

With the shortcomings of traditional survey-driven data, combined with the growth of cheap storage and online social activity, several alternative data sources have gained interest.

2.1.2.1 Social Media Nowcasting

Social media nowcasting is the use of online data to quantify social phenomena over time. The motivation is clear: having timely and inexpensive data can mean quicker decision making, and even a tiny improvement in policy decision making can mean the addition of billions of dollars to the economy. Google's Flu Trends project is likely the best known nowcasting project, but there have been many others, from housing prices [169] and mortgage delinquencies [42], to travel destination planning [64] and consumer sentiment [146].

The topics differ, but the methodology is roughly consistent across all of them: First, choose a number of human-uttered phrases that indicate a social phenomenon (such as "I lost my job" for unemployment). Second, count the occurrence of these phrases over time using a database of timestamped socially-generated messages, such as search engine queries or Twitter posts. Third, use the resulting time-varying signals as inputs to a trained statistical regressor that emits an estimate about the current level of the target phenomenon. Finally, repeat this regression task for many weeks in a row, and compare the resulting set of estimates to those from a more traditional dataset.

It is important to note the differences between nowcasting and forecasting. Nowcasting systems are designed to quantify a real-world social phenomenon *at the current time*, using social media data. Nowcasting systems only "predict" a hidden value in a statistical sense; they do not attempt to predict the future. The output from a nowcasting system might be fed as inputs to a forecasting model, but this kind of future prediction has generally not been the focus of nowcasting researchers.

One big challenge with nowcasting is with choosing the social media signals from which an estimate will be generated. With the diversity of content found on social media, the number of potential signals to choose from can be in the billions. For instance, in our past nowcasting work (further described in Chapters III, V, and VI), we enumerated all the potential phrases of up to four words in length, giving us over one billion phrases and accompanying signals, even after removing rarer phrases (i.e., having less than 20 total occurrences).

2.1.2.2 General Web Data

With the growth of the Web, new opportunities exist for extracting both micro- and macro-economic insights from many unconventional data sources on the Web. We will discuss two such projects.

In the first, researchers collected the prices for various products sold by online retailers to analyze the behavior of various products' pricing over time (the distribution of pricing changes, how they change relative to competitors, etc.) [58], as well as estimate inflation and the consumer price index (CPI). One of the authors' motivations was the lack of easy accessibility for the micro-price data that drives the government-created CPI: in order to get access, one must be a citizen of the US, submit a detailed proposal that takes several months for review, and once approved, do all analysis on-site at a Bureau of Labor Statistics office. The researchers were so successful in their estimation, they were able to spin the idea off into a company (PriceStats) that

makes this data available for a price. The authors verify the accuracy of their data by comparing it against offline (in-store) surveyed data.

In the second project, Talaika et al. extract unique product identifiers (GTINs) for millions of products from the web, track their occurrences, and use this information to estimate global trade flows [152]. Identifying products on the web and their associated unique identifiers required processing a general crawl of the web—covering hundreds of millions of web pages—and using named entity recognition with several other novel techniques. From the GTINs, the country of origin for the product can be identified by the first few characters. In order to estimate global trade flows, the authors identify the country of origin for websites selling the product. The researchers verify the accuracy of their data by comparing it against true trade flows as estimated by the World Trade Organization.

2.1.2.3 Private Business Data

During our work with several economists at the University of Michigan, we learned about two projects which looked at extracting insight from the customer data of different businesses. In one example [31], economists used the client account details from a large investment firm to analyze the correlation between investor confidence and their willingness to invest in their high-confident investments. This project started with a partnership between the researchers and the investment firm, with the firm requiring all analysis be done on the its servers. The firm coordinated asking clients for their willingness to participate in the study, as well as coordinated a survey with those participating to determine their investment preferences and confidence levels with their investments. Access to participating clients' data ($n = 10000$) was provided to the researchers in a tabular text format, after being anonymized, with each row being a client account, some clients having several rows. The researchers spent a good amount of time cleaning and integrating the data was ready for use in their modeling.

In a second project [87], economists looked at the credit card transactions of different users of a personal finance management program—where users import their credit card and bank transactions—to investigate trends such as, how patient consumers are with their spending (i.e., by looking at spending patterns around when they received paychecks or tax refunds). The data for this analysis came from a partnership between the operator of the personal finance management program and a separate university. Similar to the previous project, researchers spent a good amount of time with data cleaning and integration.

In other work, Einav et al. [75] collaborated with eBay to study the effectiveness of sales taxes on online shopping, with insights being drawn from eBay's marketplace data. They also used this data to study online pricing and sales strategies [74, 76].

2.2 Assisting the Big-Data Workflow Phases

As discussed in Chapter I, there is some existing work that can help with some of the challenges in the new big-data workflow, but other challenges remain. We discuss these in further detail below.

2.2.1 Identify Phase

The **identify** phase focuses on collecting, storing, and searching data. There are several systems and areas of research that are applicable to these tasks:

Web Crawling — Much work has been done on web crawling, which applies to the collecting component of the **identify** phase. Olston and Najork provide a thorough survey on web crawling approaches [134]. While these approaches are useful to programmers and computer scientists, most economists lack the programming background to exploit them. Instead, data should be easily accessible in the systems that economists use. We take this approach with our nowcasting systems described in Chapters V and VI.

Large Scale Storage and Processing — Parallel database systems like Teradata [154] and Gamma [72] were some of the first systems to help with the storage and searching of large scale data. More recently, MapReduce [71] and the associated Hadoop ecosystem (HDFS, Hive, Pig, etc.) have been the popular choice over the last several years, with Apache Spark [174] and its ecosystem supplanting it over the last few years. These tools generally run on a large amount of commodity hardware and offer technically-savy users simple and reliable storage, flexible processing, and a variety of other features.

Database researchers have more recently developed NoSQL data stores [57, 96] and NewSQL database systems [106, 149]. Both focus on scalability, with the latter also offering some form of ACID guarantees.

Unfortunately, many of these systems fall short of making it easy for non-technical people to adopt the tools; instead, often requiring users to have a background in programming to use them. Our work instead builds on top of these systems from which economists can easily interact.

Query Formulation — Query formulation is a well studied problem in the database community, where the goal is to either suggest or revise queries for users. The systems making these suggestions generally rely on either query logs [108, 120] or the underlying data being queried to form suggestions [45, 171]. Some of our nowcasting work shares the spirit of these systems by helping users find results that they want but their queries do not exactly describe. However, we also differ in that we can evaluate results in their final usage—in a nowcasting model—so we can further exploit this fact, in a manner similar to feature selection.

Feature Selection — Feature selection is a well-studied problem in the statistics and machine learning domains. A survey by Guyon and Elisseeff [94] gives a good overview of the subject, where they discuss different classes of selection methods such as using domain knowledge, using a ranking or scoring mechanism, and using dimensionality reduction techniques. In our nowcasting work, we borrow ideas and methods from all three of these categories. However, our focus is also on helping users refine their query to better describe what they are looking for, which is not necessarily in the scope of feature selection.

Information Retrieval — Information retrieval systems aim to make accessing and searching data easy for users, with search engines serving as a classic example of this. Multimedia retrieval systems offer a variety of querying methods that differ from those found in traditional text information retrieval. One such method allows users to provide examples of the multimedia objects they desire [86, 102, 138], which our nowcasting work is similar to since our users also interact with our nowcasting system, RaccoonDB (Chapter VI), by providing examples of their target phenomena (i.e., with their semantic and signal query components).

Other systems allow for users to provide multiple data types in their queries—a method known as multi-modality queries—to best match the multimedia objects [81]. Similarly, our RaccoonDB system uses multiple heterogeneous retrieval methods together (i.e., our semantic and signal scoring methods), also providing a synergistic effect when combined together.

2.2.2 Transform Phase

The **transform** phase is where the raw data collected in the **identify** phase is transformed into a new dataset from which economic insights are found. This includes tasks such as joins, aggregations, data cleaning, data integration, and one-to-many or many-to-many transformations. The following work relates to these transformations:

Large Scale Storage and Processing — Many of the systems that we list as being helpful to the **identify** phase are equally helpful in the **transform** phase. Systems like Mapreduce/Hadoop and Apache Spark make parallelizing of transformation operations simple for someone with a programming background. For instance, filtering, aggregations, and one-to-many operations are fairly straight forward programs to develop. NoSQL and NewSQL database systems have mixed support for similar transformations along with joins and support for more advanced operations via custom UDFs. As with the **identify** phase, these solutions do not offer the best usability for non-technical people. Our work builds on top of these tools to offer better usability.

Feature Engineering — Feature engineering is field of research that has been growing in popularity recently. Features, sometimes called signals, encode information from raw data that allows machine learning algorithms to classify an unknown object or estimate an unknown value. With machine learning being recommended for the new big-data workflow [77], feature engineering will be an important part of this workflow. Anderson et al. show how tedious feature engineering can be and provide research on speeding this process up [33, 35]. Our nowcasting systems in Chapters V and VI also attempt to alleviate some of the featuring engineering burden on users by pre-building the social media signals (i.e., features) that are used for building nowcasting models.

Data Preparation — Data preparation is an important task in the **transform** phase, and economists often spend a great deal of time in this step. It is the process of transforming a raw dataset into a new data object, which is then used in some downstream analysis (e.g., in a statistical or machine learning model). When using some raw data source (e.g., financial reports from different businesses), much effort has to go into converting the data to a standard format from which analysis can be done. The economists we worked with often used a mix of manual editing (e.g., in Microsoft Excel), some scripting (e.g., in SAS or STATA), and some SQL. Much work has gone into making this process easier for users. This includes helping users with data extraction, such as done by TextRunner [172], WebTables [56], and Senbazuru [63], which provide systems for extracting different types of data from large corpora like the Web. This also includes helping users with data wrangling, such as programming-by-example work from Wu et al. [165–167], BlinkFill [93], and Foofah [105]; or with interactive systems like Wrangler [107] and Trifacta [80]. We follow this trend in Chapter VII with our constraint-based explanation and repair of data transformations. Our work is a subtype of automatic program generation like

Trifacta or Foofah, but the user guides program generation by providing constraints around desired outputs, instead of programming-by-demonstration or concrete output tuples.

Data Integration — Data integration focuses on integrating heterogenous data sources into a new unified view. For example, economists we spoke with described an integration task at the US Federal Reserve, which involved integrating US tax records across families to study income mobility. This process involved a mix of manual editing and scripting in SAS and STATA. In data research, much work has been done on data integration. Halevy et al. [95] provide a good survey on different approaches. While our work does not focus on data integration specifically, approaches we use in our constraint-based data transformation system can be used to make this process easier.

Why-Not Provenance and Reverse Data Management — In *why-so* and *why-not* querying, users want to understand why query results are as they are, such as why particular tuples are included or missing from a result (e.g., a user searching for flights and wondering why a particular direct flight does not exist). Methods include finding modifications to a database that cause the original query to yield an expected result [103], and by tracing through the query execution graph, finding the manipulation that caused a tuple to not be included [62]. In `ConQueR` [157], the system suggests query modifications that will include user-provided why-not tuples (e.g., the flight searcher may see a suggestion to increase her maximum price). In reverse data management (RDM), an action is taken on an input dataset to achieve a certain effect on the output dataset [121]. Sub-domains of RDM include *how-to* querying [122], updating through views [47], and constraint-based repairs [41]. Our work with data transformations in Chapter VII can be considered a subdomain of reverse data management, and we share a similar goal with both domains in that we want to aid users with finding the data they want.

2.2.3 Evaluate Phase

In the **evaluate** phase, tasks like modeling and evaluation take place. There are several existing systems and areas of research that can help:

Machine Learning at Scale — Traditional economic modeling and evaluation is done in tools like STATA, SAS, R, and MatLab. These tools offer many popular modeling approaches like regressions, classification, and simulation-based modeling. Likewise, evaluation metrics and model comparison methods are built in, making eval-

uation fairly straightforward. Unfortunately, data created in the **transform** phase is not always in a size that these tools can support. Some of these tools are adding support for working with larger datasets (e.g., with parallel processing and integration with parallel systems like Hadoop); however, these solutions still have limitations in usability and not all functionality is supported. Systems like Hadoop's Mahout [135], Spark's MLlib [123], and Hogwild [144] offer several scalable machine learning algorithms, but these systems still require a programming background to use.

Workflow Management — There are several systems that help with implementing a multi-step workflow, going from raw data to generating a result or insight. Streaming systems like Storm [156] and Spark Streaming [175] are popular for processing data created at a fast velocity, but they also offer ways to piece together several functions or programs in a workflow. AMP Lab's MLBase [111], KeystoneML [27], and ML Pipelines [28], along with Google Tensorflow [30], Amazon Machine Learning [25], and Microsoft's Azure Machine Learning [26] ofter tools for creating machine learning pipelines, while also scaling well. Unfortunately, these systems fall short of making it easy for non-technical people to adopt the tools, instead requiring users to be data scientists or at least have some programming background.

Our work similarly aids with workflow management, but instead of offering users a variety of modeling and evaluation methods, we use simple methods, focusing more on the **identify** and **transform** phases, as well as the overall workflow integration. Users can also export data after the **transform** phase, opting to instead use more sophisticated modeling and evaluation approaches.

2.3 Domain-Specific Tools

The use of domain-specific tools and databases has had great results in domains like astronomy, biology, and oceanography. Jim Gray made several contributions to astronomy including to the data management and query system, Skyserver [92, 150, 151]. More recently, the SciDB data management system was developed to handle very large (i.e., petabyte scale) array data [51], a popular need in fields like astronomy, climate modeling, and bioinformatics. The AMP Lab has developed several tools tailored to DNA and genomic research [153, 173]. Additionally, Howe and Halperin tailored tools for smaller oceanography, molecular biology, and ecology labs [101] (i.e., those teams that have limited IT budgets) to make importing, cleaning, and retrieving of data easy.

Other database research has looked at the growing need for "pay as you go" data

management solutions in the *makeshift database world*, where data exists in a variety of schemas (structured, semi-structured, and unstructured) across different data types, and traditional database benefits are in the background. This work includes dataspaces [85], data lakes [12], and homebrew databases [161]. Economists also benefit from this domain of research, but there is room for further development of economic-specific features.

In our work, we extend the work of the eScience and makeshift database world to include economic-specific data systems and methodologies.

CHAPTER III

A Use Case of Economic Insight from Big Web Data

In this chapter, we present a detailed example of using the new big-data workflow to contemporaneously forecast unemployment behavior using social media—a process we refer to as *nowcasting*. We worked closely with domain experts, using their expertise to efficiently choose features and recognize different trends that are well-represented by social media. We show that social media can be used to track official unemployment insurance claims data; that it carries incremental information relative to other leading indicators for forecasting unemployment behavior; and that it allows for retrospective analysis of labor market shocks (e.g., government shutdowns and natural disasters).

This chapter is based on collaborative work with Michael Cafarella, Margaret C. Levenstein, Christopher Ré, and Matthew D. Shapiro [38].

3.1 Introduction

Social media provide an enormous amount of information that can be tapped to create measures that potentially serve as both substitutes and complements to traditional sources of data from surveys and administrative records. The use of social media to construct economic indicators has a number of potential benefits. First, social media data are available in real time and at very high frequency. Such timely and high-frequency data may be useful to policymakers and market participants who often need to make decisions prior to the availability of official indicators. The fine time-series resolution may be particularly helpful in identifying turning points in economic activity. Second, social media data are potentially a low-cost source of valuable information, in contrast to traditional surveys that are costly for both the respondent and the organization collecting the data. Third, social media offer a distinctive

window into economic activity. They represent naturally-occurring personal communication among individuals about events in their everyday lives without reference to any particular economic concept. Like administrative data, but unlike surveys, social media challenge economists to map the observed information into the economic concept being measured. Fourth, social media can be used to answer questions we would have liked to ask in surveys had we known about events in advance. In ordinary survey design, we frame the questions and then collect the data. Social media allows us to reverse this order and generate ex post "surveys." For example, we use the indexes to examine the impact of two shocks to the labor market, Hurricane Sandy in October 2012 and the October 2013 government shutdown.

This chapter develops new measures of flows in the labor market using social media data. Specifically, we use Twitter data to produce and analyze new weekly estimates of job flows from July 2011 to early November 2013. Why do we focus on job flows? Substantively, job flows are of central interest to economists, market participants, and policymakers. Practically, the weekly frequency of the official UI claims data makes them a good benchmark for testing the performance of our social media measures. We have Twitter data for only 28 months, so there is insufficient time-series variation against which to compare national aggregates such as GDP or employment. Given that the UI series is available at high frequency and without sampling error, one might ask what the Twitter signal has to add. We chose the unemployment flows concept as the case study for this chapter precisely because of the availability of a high quality, frequent series against which to compare it. This comparison should give researchers confidence to use the techniques developed in this research to study domains that are not as well-covered by official statistics.

Official UI data and our job loss index track related but not identical phenomena. Our aim is therefore to track the official index with our social media index, but not perfectly so. Since they are designed to measure the same general economic concept, they should certainly have strong co-movements. Yet they should not be perfectly correlated because of differences in population, timing, and the underlying data generation process. Indeed, one of the promises of social media for measuring economic concepts is that it will provide incremental information relative to official statistics. We find that the social media index not only does a very good job of tracking the official data, it also has important independent movements that we show—both statistically and anecdotally—carry incremental information.

The contributions of this chapter are as follows:

- We present methods for validating such novel economic measures and articulate

principles for assessing the usefulness of time series derived from social media (Section 3.2).

- By comparing our estimates with official data, we show that our Twitter derived job loss index tracks initial claims for unemployment insurance (UI) and carries incremental information relative to both lagged UI data and the consensus forecast (Section 3.3).
- We propose social media indexes to measure concepts with weaker analogues in official statistics—job search and job posting—and then use these measures to study shifts in the relationship between posting and job loss (Section 3.4).
- We show that social media allows for retrospective analysis of labor market shocks, such as government shutdowns and natural disasters (Sections 3.4.3 and 3.4.4).

3.2 Twitter Data

Twitter is a social media service through which individuals and enterprises can post short, 140-character messages of any subject of their choosing. These messages are known as tweets.[1] Unless restricted by the user, they are available publicly. These messages can be read on the web through internet browsers and by a variety of other software. Individuals can subscribe to the tweets of particular users, or subscribe to them by topic (denoted by a hash tag, i.e., a keyword with a "#" prefix). A common use of Twitter is to communicate news about life events to a community of friends. These can be mundane ("I am standing at 3rd and Elm waiting for a bus"), communicate plans or whereabouts ("Let's meet at Showcase Cinema at 7:15 to see the new Bond film"), or momentous ("George and I are pleased to announce the birth of Polly, 7lb, 8oz"). The following tweet contains a job loss phrase of the type we analyze:

2011 was interesting. I ended an engagement, got laid off, started a small biz, and it looks like I'll be moving this year too. Whew!

Our analysis is based on a roughly 10 percent sample of all tweets between July 2011 and early November 2013. The dataset contains 19.3 billion tweets and is 43.8 terabytes (TB) in size.

[1]The length of a tweet derives from the 160 character limit on an SMS text message. Twitter reserves 20 characters for the identifier.

Web search queries, an alternative source of naturally-occurring web data, have also been examined for their economic content. Web search queries are framed very differently from social media data and contain different types of information. Our approach based on social media thus is both similar and complementary to approaches based on web searches [64,65,146]. There are several differences in technique between our system and approaches based on web searches. First, web search queries and social media data likely capture different kinds of information. Social media data capture communications among individuals about their lives, while web search queries reflect individuals trying to find information on the web. These datasets capture phenomena that likely overlap, but are not identical. For example, an individual may be likelier to announce a new job to friends via social media, but may be more willing to reveal personal health information via a web search query. Second, Twitter messages are tied to a user, who exists in a public social network. User metadata can be used, for example, to classify messages by demographic groups or geography. Potentially, user information could be used to relate Twitter messages across users. As of this writing, Google Trends does not give information about the person who generated the search query, so controlling for individual characteristics or following a user over time is not possible. (The current version of Google Trends does give country-level geographic distributions of the users who generated the search queries.) Last, but not least, raw Google search queries are not public and so cannot be analyzed directly; in contrast, tweets are public. Google's web search query data are made public only via the Google Trends tool. It currently does not reveal actual frequencies for search terms, but instead places frequencies on a 0-100 scale, making some uses of the data difficult or impossible. The techniques used to collect and prepare the data are not public. In contrast, the methods we propose here are transparent, so researchers can more easily inspect and reproduce analyses performed on the resulting signal data.

3.2.1 Strategies for Converting Social Media into Data

A core challenge in this work is to develop a rigorous methodology to convert the corpus of social media texts into time-varying signals that have both predictive and explanatory power for labor market flows. We can convert a given set of relevant tweets into a signal by first counting their frequency in each 24-hour period in the sample period and then compiling these daily counts into a single time-varying signal. Obtaining m signals amounts to choosing m relevant sets of tweets. Clearly, the power of our social media index depends on how well we choose these sets.

Given the very large number of tweets, automated statistical techniques for choos-

ing predictive features—in our case, sets of tweets—are appealing [94]. These techniques, however, pose serious challenges for our task. First and most basic, the technique for enumerating all the possible signals in the tweet collection—i.e., modeling the feature space—is not obvious. One approach is to create a tweet set for each unique Twitter author; another is to compose a tweet set for every k-gram, in which a k-gram is a sequence of k or fewer consecutive ordered words found in the tweet corpus. Considering a restrictive set of features may make feature selection easier, while a larger set of features may enable creation of an index with better predictive power. Second, even a relatively restrictive set of potential features, such as k-grams with $k \leq 4$, yields vastly more features than we have time-series macroeconomic data.[2] (There are roughly one billion 4-grams that occur at least 20 times in our data.) Some features with high correlations will in fact be entirely spurious and thus carry no predictive power. Other features may be predictive but not causal (e.g., Lysol as a feature for flu). Such features can be useful for the predictive model but may be logically opaque to human observers (that is, the tweets in a set will not have an obvious common thread or will have a common thread that appears to be nonsensical).

In Chapter V, we discuss more recent efforts to explore the problem of feature selection when applied to social media and any macroeconomic or similar topic. Macroeconomic tasks tend to offer very small in-sample datasets in comparison to other data-intensive trained system tasks in computer science. For example, web search engines can exploit billions of human judgments about web page relevance, derived by observing users' clicks on search engine result pages. In the absence of large datasets that can automatically validate feature selections, the techniques under development would have the researcher describe feature preferences (that is, provide "domain knowledge") and then observe a set of features suggested by the system. The researcher would then reject features that violate real but implicit researcher preferences. (We give several examples of this procedure from our own experience in the section below.) One early version of this automated system offered suggestions based on a combination of user-suggested terms, thesauri, and statistics derived from web text [39, 40]. Ideally, this system would give results in interactive timeframes, but the massive number of possible feature combinations (roughly 2.7×10^{103} when choosing 10 features from among 4-grams in our corpus) makes known suggestion

[2] In their development of a flu index using Google web search queries, Ginsberg, et al. [89] followed a variable ranking strategy that chose signals that were highly correlated with a target signal. In preliminary work, we considered the correlation of phrases found in tweets with weekly unemployment. None of the top 100 most correlated phrases had any plausible connection to unemployment [39].

techniques infeasible. Solving these problems is the subject of ongoing research, and could substantially lower the researcher burdens associated with applying social media techniques to any novel topic.

In this chapter, we solve this problem by first limiting our analysis to k-grams, which are essentially repeated cross sections of tweets, aggregated first to days and then weeks. For the economics task at hand, we chose signals from a feature space in which each feature corresponds to a k-gram where $k \leq 4$. Second, we narrow the feature space further by using domain knowledge to select signals that we strongly believe are causally connected to job loss. Specifically, the research team identified terms that it believes are indicative of the phenomenon being measured, based on knowledge and expertise in the area. We describe this procedure in more detail in Section 3.3.

This approach has drawbacks. First, we may unwittingly add bias during our selection of phrases. Second, some tweet sets cannot be described at all (e.g., because we restrict ourselves to 4-grams, we cannot characterize the set of all tweets that contain the five-word phrase, "my mom no longer works"). Finally, the feature space does not automatically group phrases that are textually distinct but semantically similar (e.g., "I got fired" and "my boss canned me" express the same idea but are not identical k-grams). Our choice of feature modeling has the benefits of being easy to describe and enabling many tweet sets that are understandable to the user (e.g., all tweets that contain the phrase, "I lost my job"). Moreover, despite its restrictiveness, our design is sufficient to demonstrate that social media data contain genuinely useful information about labor flows.

3.2.2 Implementation

To implement the domain knowledge strategy, we developed a list of phrases related to job loss and unemployment that we expected to be found in tweets that carried information about job loss in general and initial claims for unemployment insurance in particular. The phrases we use to aggregate signals of job loss and unemployment are listed in Table 3.1. A space is denoted as "|" and a wildcard as "*" as in the detailed descriptions in Table 3.2.

The process for generating the list of phrases includes the following steps:

1. A priori specification of terms such as "lost job," "laid off," and "unemployment" that we expect to be contained in tweets of interest.

Signal	Mean	Standard Deviation	Coefficient of Variation
Axed	3.25	1.51	0.46
Canned	8.86	3.42	0.39
Downsized	0.49	0.25	0.51
Outsourced	2.11	1.35	0.64
Pink slip	1.34	1.31	0.98
Lost job	3.21	0.86	0.27
Fired job	27.45	6.67	0.24
Been fired	15.19	6.76	0.45
Laid off	15.70	3.59	0.23
Unemployment	53.33	20.07	0.38

Table 3.1: Summary Statistics for Signals: Job Loss and Unemployment (Weekly Rate per Million tweets). Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Sample is 19.3 billion total tweets of which 2.4 million are job loss and unemployment related. See Table 3.2 for detailed descriptions of phrases for signals.

2. Expansion of the specification of the target phrases to include plausible misspellings and wildcards to capture variants such as "lost my job" or "lost his job."

3. Deletion of phrases where—upon inspection—it becomes clear that the a priori specified phrases have little to do with the labor market.

The first column of Table 3.1 gives the ten job loss and unemployment signals that we will analyze. We allow for variants in spelling and spacing. The variants we consider include the 27 search phrases listed in Table 3.2.

There are some terms one might expect to include a priori, but which we exclude or include only in combination with other words. For example, we do not include a search for the words "fired," "benefits," and "insurance" alone because each was used much more frequently in unrelated contexts (e.g., fired up). Note that "unemployment benefits" or "unemployment insurance" are captured because we do include any k-gram including the word "unemployment." In general, singleton terms can be problematic. We originally included the term "sacked" but eliminated the signal from further analysis because its frequency in the data—several orders of magnitude greater than other employment-related terms—suggested that its use referred to other linguistic meanings. Similarly, we eliminated "let go" because it appeared much more frequently than other employment-related phrases and seemed to have other plausible meanings.

Category	Signal	Phrase	Number of distinct matched phrases
Job loss	Axed	axed	1
	Canned	canned	1
	Downsized	downsized	1
		down|sized	1
	Outsourced	outsourced	1
	Pink slip	pinkslip	1
		pink|slip	1
	Lost job	lost|*|job	45
	Fired	fired|*|job	28
		fired|*|work	16
		fired|from	1
		fired|lol	1
		get|fired	1
		got|fired	1
		just|fired	1
	Been fired	been|fired	1
		being|fired	1
		be|fired	1
		was|fired	1
	Laid off	laidoff	1
		laid|off	1
		layed|off	1
		layoff	1
		lay|off	1
Unemployment	Unemployment	unemploy	1
		unemployed	1
		unemployment	1

Table 3.2: Social Media Signals: Job Loss and Unemployment. Note: The signals are counts of tweets that contain 4-grams with the indicated phrase where "|" denotes a space and "*" is a wildcard. The last column indicates the number of distinct phrases found in the database of tweets matching the target phrases with wildcards.

In the case of the phrase "lost * work," inspection of the matched k-grams clearly indicated nearly universal non-employment related concepts. Many phrases referred to computer problems such as "lost all my work" and "lost my #$% work," as well as happier references such as "lost in my work" and "lost Beethoven work." As a consequence, we excluded all candidates related to this signal in the creation of the job loss measure. Having the wildcard in this search was critical for revealing that the "lost work" phrases were not about employment.

One concern about the use of social media to measure economic activity is that it will capture comments on releases of official statistics rather than provide independent measures of activity.[3] We did not see evidence that there is a lot of tweeting about the Department of Labor's release of initial claims data, but the monthly Employment Situation release does get a lot of attention and might account for a significant num-

[3]There is evidence that a substantial amount of communication over social media consists of links to internet sites of content creators (CNN, Justin Bieber's tweets, etc.). See Goel, Watts, and Goldstein [91].

	Axed	Canned	Down-sized	Out-sourced	Pink slip	Lost job	Fired job	Been fired	Laid off	Un-emp.
Axed	1									
Canned	0.37	1								
Downsized	0.34	0.29	1							
Outsourced	0.18	0.31	0.34	1						
Pink slip	-0.05	0.00	-0.10	-0.12	1					
Lost job	0.45	0.49	0.46	0.40	0.02	1				
Fired job	0.48	0.46	0.28	0.18	-0.08	0.52	1			
Been fired	0.45	0.36	0.04	0.01	-0.03	0.30	0.65	1		
Laid off	0.43	0.52	0.46	0.43	-0.05	0.59	0.63	0.24	1	
Unemployment	0.35	0.40	0.50	0.44	-0.14	0.54	0.47	0.21	0.66	1

Table 3.3: Correlation of Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013. The "Unemployment" signal is purged of the Employment Situation effect, as described in the text.

ber of mentions of "unemployment." Indeed, the Bureau of Labor Statistics (BLS) plans to use Twitter as an official release channel, so re-tweets of the unemployment report may be a significant confound in the future. To check for the importance of the unemployment report per se on tweets about unemployment, we estimate a linear regression with the unemployment signal as the dependent variable and a dummy for weeks containing the unemployment report as the regressor. The estimated relationship is

$$r\_unemp_t = \underset{(1.9)}{49.4} + \underset{(4.1)}{17.4} * emp\_sit_t + u_t \tag{3.1}$$

where r_unemp is the unemployment signal, emp_sit is the Employment Situation dummy, and u is the residual.[4] tweets about unemployment are about a third higher in an Employment Situation week than average, so we purge tweet-derived signals containing "unemployment" of the Employment Situation effect using a regression as shown above.

Table 3.1 presents summary statistics of the signals. The signals are expressed as weekly rates per million tweets. While the signals derived from the selected phrases are fairly rare—between 0.5 and 54 per million tweets—there are so many tweets that the signals still provide a rich dataset. Of the 19.3 billion tweets reflected in Table 3.1, there are 2.4 million associated with job loss and unemployment. The signals have roughly comparable coefficients of variation, so there is potentially information in each of them. The correlation matrix in Table 3.3 shows that the signals from the selected phrases are positively correlated (with the exception of "pink slip"), so they do appear to be picking up related phenomena in the tweets.

[4]Standard errors are in parentheses.

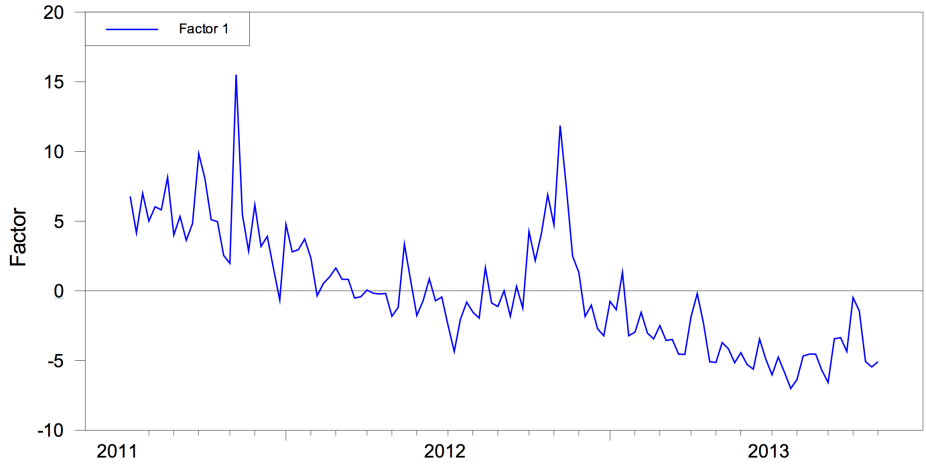
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Axed	0.65	0.29	-0.06	0.39	0.45	0.13	-0.33	0.07	0.03	-0.04
Canned	0.68	0.11	0.14	-0.36	0.21	-0.56	-0.02	0.11	0.04	-0.05
Downsized	0.60	-0.42	0.04	0.51	0.01	-0.18	0.36	0.14	-0.12	0.01
Outsourced	0.52	-0.54	0.02	-0.40	0.33	0.37	0.13	0.09	-0.09	-0.03
Pink slip	-0.11	0.22	0.95	0.04	-0.03	0.13	0.03	0.11	0.01	-0.02
Lost job	0.78	-0.07	0.19	0.03	0.04	0.00	0.08	-0.56	0.12	0.02
Fired job	0.77	0.41	-0.11	-0.05	-0.27	0.11	0.05	-0.02	-0.24	-0.28
Been fired	0.51	0.72	-0.17	-0.08	0.04	0.14	0.32	0.11	0.11	0.20
Laid off	0.83	-0.13	0.08	-0.07	-0.26	0.00	-0.29	0.03	-0.26	0.25
Unemployment	0.76	-0.29	-0.04	0.01	-0.32	0.09	-0.10	0.18	0.42	-0.06
Variance of factor	4.27	1.40	1.02	0.72	0.60	0.55	0.46	0.42	0.36	0.19
Cumulative % of variance	0.43	0.57	0.67	0.74	0.80	0.86	0.90	0.95	0.98	1.00

Table 3.4: Factor Loadings on Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Principal component factors calculated based on the correlation matrix of signals shown in Table 3.3.

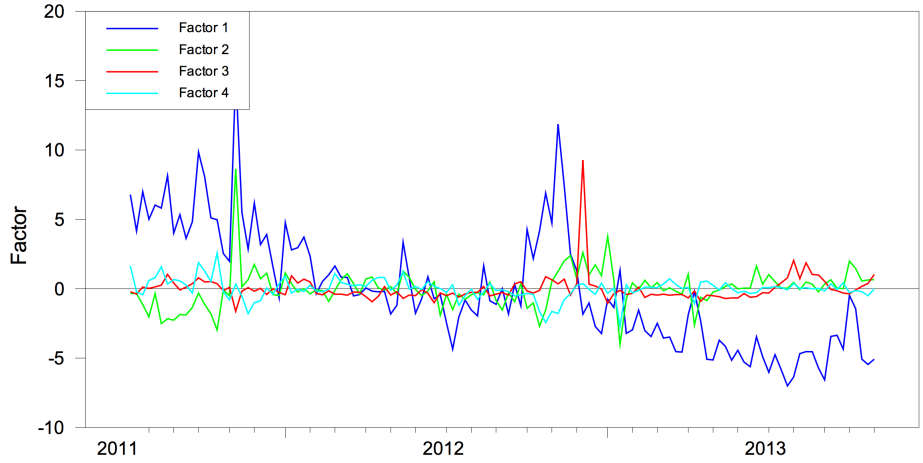
In order to preserve degrees of freedom while extracting as much information as possible from the Twitter signals, we perform a principal components analysis on the ten signals. Table 3.4 reports the factor loadings and variances. Not surprisingly, given the positive and fairly uniform correlation structure reported in Table 3.3, the first factor has fairly uniform coefficients across the signals and accounts for 43 percent of the variance. The next four factors each account for about 10 percent of the variance. Figure 3.1 plots the factors estimated over the entire sample from July 2011 through early November 2013. Figure 3.1a shows only the first factor, and Figure 3.1b shows the first four. The first factor is fairly volatile in the second half of 2011 and has a noticeable downward trend into 2012, when it flattens. This pattern is interrupted in late 2012. In 2013, the downward trend evident throughout the period resumes. Figure 3.1b adds the factors 2, 3, and 4. By construction, they have signal less and have spikes that might be suspect. Factor 2 has a spike in late 2011 that matches a spike in Factor 1, so it might be genuinely related to job flows.

On the other hand, note that Factor 3 is dominated by the "pink slip" signal (see Table 3.4). There is evidently a spike in that signal in December 2012. Without it, Factor 3 would not have emerged from the principal components analysis as having significant variance, so unless we are prepared to believe this spike is job related, it should be discounted.[5]

[5]We suspect that the spike was driven by tweets about the November 19, 2012 launch of a marketing campaign titled "Pink Slip" featuring football player Tom Brady [140].



(a) Factor 1



(b) Factor 1 - 4

Figure 3.1: Twitter Job Loss and Unemployment Signals. Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Principal component factors calculated based on the correlation matrix of signals shown in Table 3.4.

3.2.2.1 Relating Social Media Data to the Economy and Economic Data

These signals from social media and the factors that summarize them are new measures of economic activity. They are not based in any way on standard measures using conventional sources of data. It is natural to ask how they relate to a standard measure of economic activity: initial claims for unemployment insurance (UI). The initial claims data are well-suited for evaluating the social media signals. First, they are available at weekly frequency. Given that we have just over two years of Twitter data, a high-frequency economic indicator for comparison is very important. Second, initial claims for UI are a direct measure of transitions in the labor market. Hence, they are likely to have much more high-frequency variation than variables that mea-

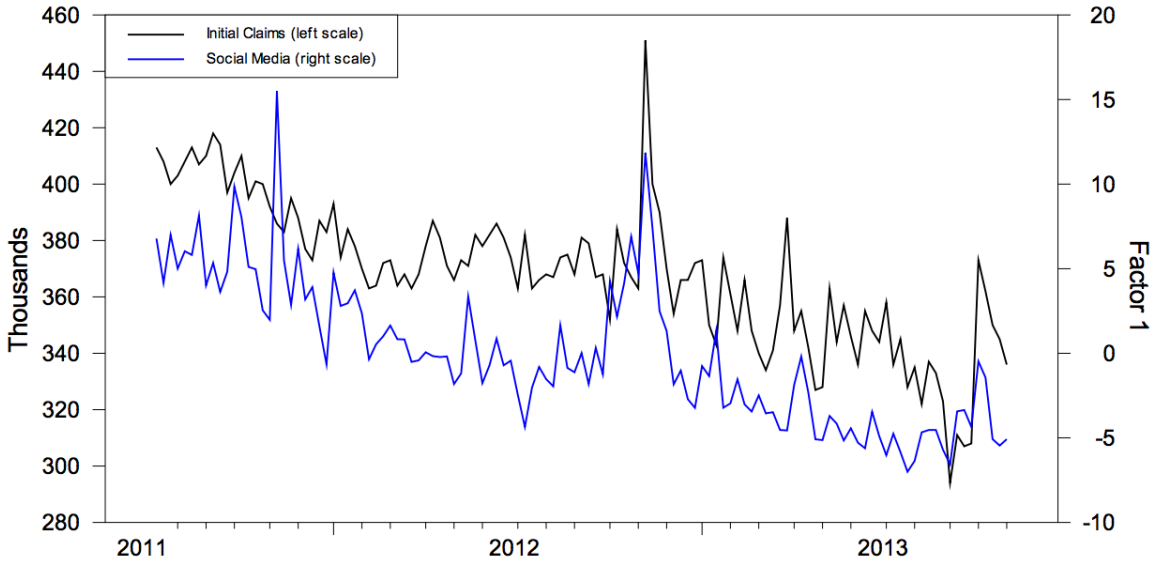


Figure 3.2: Initial Claims for Unemployment Insurance and Job Loss and Unemployment Factor 1. Note: Figure shows the Department of Labor's Initial Claims for Unemployment Insurance (left scale, revised data, seasonally adjusted) and the Social Media Factor 1 (right scale). The factor is estimated as described in the text and is no way fit to the initial claims data.

sure stocks (e.g., the unemployment rate). We expect that social media data will be useful precisely for measuring such high-frequency changes in activity.

Figure 3.2 shows initial claims for UI (left scale) and the first factor from the Twitter job loss and unemployment signals (right scale). The social media series is estimated completely independently from the new claims data. The relationship between these two indicators of job loss is quite strong—both in the general trend and in some notable spikes. Over the sample period from July 2011 to early November 2013, initial claims have a general downward trend in new claims. They flatten in the first half of 2012 and then resume the downward trend in 2013. The social media series has a very similar pattern.

There are also some high-frequency changes in new claims—notably the spike in late fall 2012. Our indicator also captures that spike in job loss. We will investigate this spike, associated with Hurricane Sandy, in some detail below (Section 3.4.3).

Note also that the fit of the social media series to the new claims series is not perfect. Aside from period-by-period variation, the social media series has a spike in 2011 that is not in the initial claims data. More interestingly, it does not indicate the slowdown in job loss seen in the new claims data in September 2013. The social media information contains independent information about the job market. Indeed,

	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Constant	90.24	30.16	98.46	31.57	43.00	19.87	23.60
	(21.98)	(23.09)	(22.93)	(23.56)	(21.51)	(22.48)	(22.72)
Lagged initial claims	0.75			0.05	0.48		0.17
	(0.06)			(0.16)	(0.07)		(0.15)
Consensus forecast		0.92		0.86		0.67	0.48
		(0.06)		(0.18)		(0.10)	(0.21)
Social media: factor 1 (scaled)			0.73		0.40	0.27	0.29
			(0.06)		(0.07)	(0.08)	(0.09)
Adjusted R^2	0.57	0.64	0.53	0.63	0.65	0.66	0.66

(a) Preliminary Data

	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Constant	74.73	46.33	103.43	46.56	43.36	34.17	34.22
	(20.27)	(20.75)	(20.02)	(20.55)	(18.81)	(19.56)	(19.28)
Lagged initial claims	0.80			0.27	0.50		0.29
	(0.05)			(0.15)	(0.07)		(0.14)
Consensus forecast		0.88	0.61		0.59	0.30	
		(0.06)		(0.16)		(0.08)	(0.16)
Social media: factor 1 (scaled)			0.73		0.38	0.32	0.33
			(0.05)		(0.07)	(0.07)	(0.07)
Adjusted R^2	0.64	0.67	0.59	0.67	0.71	0.71	0.72

(b) Revised Data

Table 3.5: Predicting Initial Claims: Consensus, Social Media Factor, and Lagged Dependent Variable. Note: Sample period is July 16, 2011 through November 2, 2013. Standard errors in parentheses. Dependent variable: Initial Claims for Unemployment Insurance (preliminary data in Table 3.5a, revised data in Table 3.5b). Regressors: Lagged dependent variable, consensus forecast, and social media factor 1 scaled to have same units as initial claims.

as we will discuss below, the drop in initial claims in September relates to a processing problem in California.

Not all job loss is associated with applications for UI, so we are not seeking simply to predict UI. Nonetheless, the high- and low-frequency association of the series with the official data is reassuring.

We can test the association of the social media signal with the UI initial claims data statistically. Table 3.5 presents regressions of initial claims on the social media series and for comparison, lagged initial claims and the consensus forecast.[6,7] The social media series is not as good a predictor of new claims as are the lagged dependent variable or the consensus, though of course there is no reason to expect or hope it to be. Nonetheless, it is strongly predictive of new claims and remains significant in the

[6]The consensus forecast is produced by Bloomberg Surveys [46]. It is the median forecast of a panel of approximately 50 economists.

[7]For this set of regressions, the social media index is normalized to make the regression coefficients comparable, normalizing so it has the same mean and standard deviation as the dependent variable. This normalization, of course, has no effect on the t-statistics or fit of the regression.

regressions that include the lagged dependent variable and the consensus.

3.3 A Real-time Predictor from Social Media Data

The social media series for job loss successfully tracks official data at both high and low frequency. This section constructs a real-time index for predicting initial claims for unemployment insurance, and evaluates its ability to provide a real-time indicator of economic activity. In contrast with the previous section, which sought to estimate time series from social media and show that they are related to economic activity, this section aims to construct a predictor that is feasible in real-time.

3.3.1 Constructing the Real-time Predictor

To construct the University of Michigan Social Media Job Loss Index, we estimate a model relating initial claims for unemployment insurance to social media signals recursively, using only data that are available at the point of the prediction. The Twitter data are available almost immediately, so we can construct a prediction of the current week's new claims with virtually no lag. The procedure is as follows:

1. Estimate the factors on the social media signals from the beginning of the sample through the current week.

2. Estimate the University of Michigan Social Media Job Loss Index by regressing real-time initial claims data on the factors. The regression coefficients are updated each week.

3. Construct the prediction as the fitted value for the current week from that regression.

4. Update the data weekly and repeat this procedure.

We carry out this procedure recursively over periods ending July 7, 2012 through November 2, 2013. The starting period of the estimation is always July 16, 2011.[8] We consider various specifications for the regression in step 2. Table 3.6 reports the estimates of these specifications for the final period. Table 3.7 reports the root mean squared error of these different specifications using the predictions estimated

[8]We experimented with various alternatives to having a fixed starting period. These included estimating over a rolling, one-year window and using the whole period, but with exponentially declining weights on older observations. The results were quite similar, so we report the simpler specification using OLS estimated over all the data available in real time.

	(1)	(2)	(3)	(4)	(5)	(6)
Constant	368.26	368.25	368.23	368.23	365.77	368.81
	(1.52)	(1.50)	(1.50)	(1.48)	(9.11)	(1.73)
Factor 1	4.70	4.70	4.71	4.71	4.69	4.69
	(0.36)	(0.35)	(0.35)	(0.35)	(0.36)	(0.36)
Factor 2		-2.35	-2.35	-2.35		
		(1.07)	(1.07)	(1.06)		
Factor 3			-1.52	-1.52		
			(1.48)	(1.47)		
Factor 4				3.76		
				(2.05)		
Seasonal factor for initial claims					2.54	
					(9.16)	
Employment Situation week						-2.43
						(3.66)
Adjusted R^2	0.59	0.60	0.60	0.61	0.59	0.59

Table 3.6: Constructing the Social Media Job Loss Index. Note: The dependent variable is the Department of Labor Initial Claims for Unemployment Insurance (thousands, seasonally adjusted). The independent variables are the job loss and unemployment factors. The Social Media Job Loss Index is based on regressions re-estimated each week using real-time data available as of the prediction period, as described in text. This table presents the estimates for the final week in the sample. Sample period is July 16, 2011 through November 2, 2013. Standard errors in parentheses.

recursively. The specification with a constant and the first factor yields a strongly significant coefficient and an adjusted R^2 of 59 percent.[9] Adding factors 2 through 4 adds little to the fit of the regression. Table 3.7 shows that the RMSE of the specification with one factor is the lowest, so that is our preferred specification based both on goodness of fit and parsimony.

Table 3.6 also includes specifications with two additional explanatory variables. Specification 5 includes the seasonal factor for initial claims as an additional explanatory variable to evaluate whether there is discernible seasonality in the relationship between the social media index and initial claims. Flows into unemployment are highly seasonal with peaks in December/January and the summer. The Twitter data may also exhibit seasonality, but with less than three years of data, we cannot seasonally adjust it. Using the new claims seasonal factor implicitly seasonally adjusts, assuming the same seasonality in both series. The seasonal factor is small and insignificant, so we do not include it in our preferred specification. Given that job loss is indeed seasonal, it is interesting to note that the social media mentions of job loss do not have the same spike as the official data. An interpretation of this finding is that a predictable job transition relating, for example, to the end of a seasonal spell

[9]Note that the estimate in the first column of Table 3.6 is the same, apart from normalization, of that in the third column of Table 3.5b.

Specification	Root Mean Squared Error	
	Preliminary Data	Revised Data
(1) Factor 1	21.9	19.2
(2) Factor 1,2	22.7	20.0
(3) Factor 1,2,3	23.7	21.4
(4) Factor 1,2,3,4	22.6	20.6
(5) Factor 1, Seasonal Factor	22.1	19.4
(6) Factor 1, Employment Situation Week	22.1	19.3

Table 3.7: Prediction Errors of Social Media Job Loss Index. Note: Table gives the root mean squared error (RMSE) of the Social Media Job Loss Index for initial claims for unemployment insurance (preliminary data and revised data). The models and RMSEs are estimated recursively, using data from July 16, 2011 forward, for weeks ending July 16, 2011 through November 2, 2013.

of employment, is not something that one would mention in a tweet using the phrases we use to construct the signals. The absence of such predictable transitions is not necessarily a problem for the social media index—indeed for some purposes it might be an advantage—but it needs to be kept in mind for the use and interpretation of the indicator.

The last column of Table 3.6 considers whether the announcement of the Bureau of Labor Statistics unemployment data affects the index. As we describe in Section I, the signals mentioning "unemployment" are already purged of this announcement effect. The estimate in column (7) checks whether this processing is sufficient for removing the announcement effect from the index. The dummy for weeks that the unemployment rate is released is insignificant, so the procedure discussed in Section 3.2 does appear to suffice.

3.3.2 Analyzing the Real-Time Social Media Job Loss Index

Figure 3.3 shows our preferred specification for the social media index with a constant and Factor 1. It is plotted against the initial claims data. The shaded area is the first year of data. Because it is not feasible to estimate the factors and perform the regression recursively, as described in steps 1 and 2 above, they are estimated over the whole period. The balance of the data shown in Figure 3.3 is estimated recursively, as described in the previous section. The social media index tracks the official data closely, both in overall trend and in some of the movements. On the other hand, it carries independent information about job loss, for example, indicating a spike in 2011 not present in the official data and failing to show the decline in job

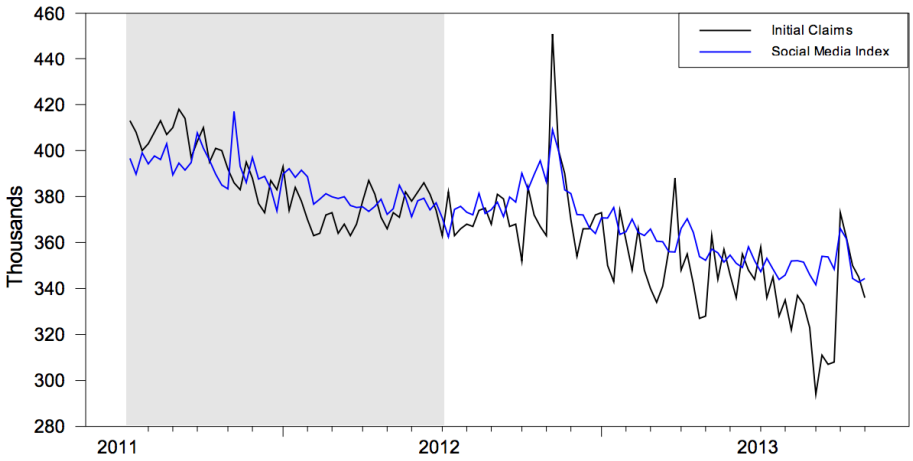


Figure 3.3: Initial Claims for Unemployment Insurance and the Social Media Job Loss Index. Note: Figure shows the Department of Labor's Initial Claims for Unemployment Insurance and the Social Medial Job Loss Index. The Social Media Job Loss Index is estimated in sample in the shaded area and recursively thereafter.

loss in September 2013 at the end of our sample. This drop in reported initial claims in the official data in September 2013 relates to a data processing issue in California;[10] this is an example of where the social media index does not suffer from measurement error encountered by the official data, and thus may more closely track the true state of the economy.

Additionally, the social media index tracks increases in job loss evidently associated with the government shutdown during the first two weeks of October 2013. The index rises noticeably in the first half of October and declines by about the same amount in the second half of the month. Initial claims have a similar pattern (after accounting for the rebound from the resolution of the processing issues in California).[11]

While the UI series and social media index generally move together, they are certainly not perfectly correlated. This is to be expected, since they measure different things. While part of the proof of concept is to show that the social media index moves with the official data, the aim is not to replicate the official data perfectly. For myriad reasons relating to the concept being measured, the coverage and take-up of

[10]Employment and Training Administration's Unemployment Insurance Weekly Claims Report, issued September 19, 2013, reports a decrease of 25,412 UI claims in California "due to Labor Day holiday and computer system updates" [78]. The following week, they reported that a comparable increase in UI claims in California "reflects return to 5 day workweek and a full week of processing after computer system updates" [79].

[11]Federal workers apply to a different unemployment insurance system. They are not included in the preliminary initial claims data used to construct the real-time job loss index.

Dependent Variable	Prelim. IC minus Consensus			Revised IC minus Consensus		
	(1)	(2)	(3)	(4)	(5)	(6)
Constant	-9.41	-55.49	-10.57	-6.78	-75.54	-7.29
	(3.16)	(53.78)	(3.43)	(2.92)	(49.19)	(3.18)
Job Loss Index minus consensus	0.72		0.63	0.75		0.71
	(0.20)		(0.22)	(0.18)		(0.21)
Job Loss Index		0.85			0.94	
		(0.25)			(0.23)	
Consensus		-0.72			-0.75	
		(0.20)			(0.18)	
Lag of Job Loss Index minus consensus			0.19			0.09
			(0.22)			(0.20)
Adjusted R^2	0.15	0.15	0.15	0.19	0.20	0.18

Table 3.8: Incremental Information in Social Media Job Loss Index. Note: Sample period is July 16, 2011 through November 2, 2013 (recursive sample). Standard errors in parentheses. Dependent variables: Columns (1)-(3), Preliminary initial claims minus consensus; Columns (4)-(6), Revised initial claims minus consensus.

unemployment insurance benefits, and the makeup of the samples, the social media index measures something different from the official series. Nonetheless, our findings that they are related do provide evidence that the social media index is a meaningful measure of economic activity.

3.3.3 Assessing the Information in the Real-Time Social Media Job Loss Index

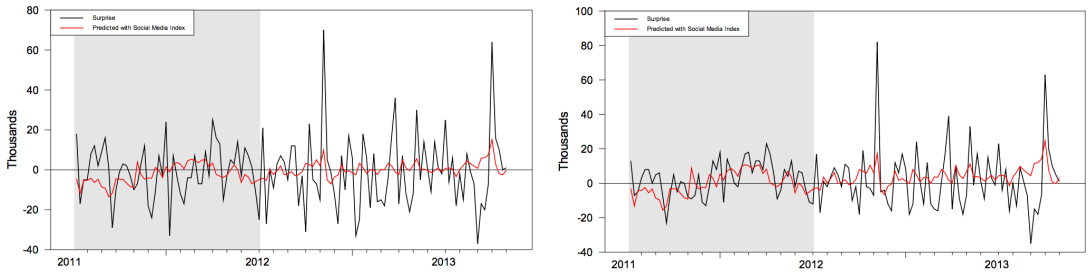
Next we ask whether, from the perspective of predicting the state of the economy in real-time, there is incremental information in our social media index. The results in the previous section suggest this might be so. We know from column 6 of Table 3.5 that the consensus forecast is a very good predictor of the initial claims data, but that the social media factor has incremental explanatory power. In order to address the question of incremental information, we compare our Social Media Job Loss Index to the consensus forecast on the eve of the initial claims announcement. This consensus forecast is based on a survey of market experts several days prior to the release of initial claims for UI. Table 3.8 reports the results of this analysis. First, we examine the preliminary report of new UI claims. We subtract the consensus estimate from the preliminary UI claims report to calculate the error in the consensus view. We then compare these errors to the Social Media Job Loss Index, which we construct based on information available in real time as described above.

To assess the incremental information in the Social Media Job Loss Index, we examine the regression of the error (preliminary initial claims minus consensus) on the Social Media Job Loss Index minus the consensus (Table 3.8, Column 1). The

social media index carries incremental information. It is statistically significant and explains about 15 percent of the variation in the surprise relative to the consensus. In Table 3.8, Column 2 we report an estimate that separates the impact of the University of Michigan Social Media Job Loss Index and the consensus. The University of Michigan Social Media Job Loss Index remains a significant predictor of the error in the consensus, while the coefficient on consensus itself is roughly equal and in opposite sign to the coefficient on the University of Michigan Social Media Job Loss Index minus consensus in the first estimate (the p-value of the test of equal and opposite coefficients is 0.16). These results are included to show that the correlation of the consensus with the surprise is not driving the result. Finally, in Column 3 we include a lagged index. It has a very small coefficient that is not significantly different from zero, suggesting that, after one week the information content in the tweets had been incorporated into the consensus view. Indeed, there is little evidence of any lags in the relationship between the social media signals and the UI data.

In the second part of Table 3.8, we compare our social media index to the UI claims, revised one week after the initial numbers. The results are similar to those for the preliminary UI claims, except that the explanatory power of the social media index increases to 19 percent of the variance, suggesting that the social media index is better at predicting the true, revised UI number than it is at predicting the original estimate. This finding suggests that the social media index is capturing information about the true state of the job market that is not captured in either the consensus or the preliminary UI claims estimate. The incremental information in the social media index is relevant relative to both the preliminary and revised data. Policymakers and forecasters will be more interested in information about the revised data. Market participants may be more interested in the incremental information for the preliminary announcement.

Figure 3.4 shows the incremental information in the social media index on a week-by-week basis. The figures show the surprise (initial claims minus consensus) and the part predicted by the University of Michigan Social Media Job Loss Index (that is, fitted value of the regression of the surprise on the University of Michigan Social Media Job Loss Index minus consensus) for the preliminary and revised initial claims data. Again, we do not aim to track all the surprise and indeed account for 15 to 20 percent of it. Much of the surprise is serially uncorrelated noise with no intrinsic interest. The social media index does capture some of the unexpected increase in initial claims in early 2012 and some of the swing from positive to negative surprises



(a) Preliminary Data (b) Revised Data

Figure 3.4: Surprises Predicted by Social Media Job Loss Index. Note: Surprise is Department of Labor Initial Claims for Unemployment Insurance (preliminary or revised) minus the consensus forecast. Predicted with Social Media Job Loss Index constructed based on factor 1, as described in the text. The index is generated recursively except in the shaded area, where it is generated over the entire shaded sample.

in the last quarter of 2012 to the first quarter of 2013.[12]

3.3.4 Providing a Real-Time Economic Indicator from Social Media

This research project has implemented the creation of the University of Michigan Social Media Job Loss Index in real time. At the end of each week ending Saturday, our automated computer program processes the latest tweets, recalculates the job loss index based on the one-factor model described in the previous section, and updates the prediction. The prediction is posted on the web each week at `http://econprediction.eecs.umich.edu/`. In this way, we are able to provide policymakers, forecasters, and other interested parties with a useful high-frequency economic indicator with virtually no lag between availability of the source data and availability of the indicator. Such virtually contemporaneous information should be useful to policymakers and market participants who need to make decisions in real time with incomplete information.

Category	Signal	Mean	Standard Deviation	Coefficient of Variation
Search	Find	25.23	4.96	0.20
	Look	25.31	8.28	0.33
	Need	90.74	19.79	0.22
	Search	2.13	0.93	0.44
Posting	Hiring	220.81	212.79	0.96
	Job	161.14	59.97	0.37
	Work	384.37	67.66	0.18
Search and Posting	Seek	0.64	0.29	0.45

Table 3.9: Summary Statistics for Signals: Job Search and Job Posting (Weekly Rate per Million tweets). Note: Sample period is July 16, 2011 through November 2, 2013 (weeks ending Saturday). Sample is 19.3 billion total tweets of which 2.4 million are job loss and unemployment related. See Table 3.11 for detailed descriptions of phrases for signals.

	Find	Look	Need	Search	Seek	Hiring	Job	Work
Find	1							
Look	0.20	1						
Need	0.34	0.85	1					
Search	0.14	0.45	0.45	1				
Seek	0.25	0.33	0.29	0.45	1			
Hiring	-0.01	0.05	0.21	0.13	0.21	1		
Job	0.26	0.24	0.13	0.30	0.46	0.24	1	
Work	0.39	0.50	0.52	0.31	0.50	0.19	0.56	1

Table 3.10: Correlation of Job Search and Job Posting Signals. Note: Sample period is July 16, 2011 through November 2, 2013.

3.4 Additional Applications of Social Media for Measuring Labor Market Activity

3.4.1 Job Search and Job Posting Indexes

We create and describe two additional series related to search, matching, and labor market equilibrium. Specifically, we examine tweets containing phrases indicating that the *tweeter* (i.e., the creator of the tweet) is searching for a job (e.g., "find," "look," "need," "search," or "seek," each followed by "job" or "work") and others that suggest that the tweeter is searching for an employee ("hiring," and "job" or "work" followed by "opportunity" or a phrase indicating location or job type). The signals for job search and job posting are listed in Table 3.9 and the detailed phrases are

[12]Choi and Varian [65] use Google search queries to predict initial claims. For "Welfare and Unemployment" (though less so for "Jobs") Google Trends captures the increase in unemployment at the onset of the Great Recession.

Category	Signal	Phrase	Number of distinct matched phrases
Search	Find	find|*|job	242
		find|*|work	178
	Look	look*|*|job	237
		look*|*|work	497
	Need	need|*|job	398
		need|*|work	515
	Search	search*|*|job	93
		search*|*|work	38
Search and Posting	Seek	seek*|*|job	30
		seek*|*|position	11
		seek*|*|work	29
Posting	Hiring	hiring|*	17278
	Job	job|opportunities	1
		job|opportunity	1
		jobs|in|*	3040
		jobs|near|*	36
		job|in|*	4397
		job|near|*	18
	Work	work|in|*	10163
		work|near|*	32
		work|opportunities	1
		work|opportunity	1

Table 3.11: Social Media Signals: Job Search and Job Posting. Note: The signals are counts of tweets that contain 4-grams with the indicated phrase where "|" denotes a space and "*" is a wildcard. The last column indicates the number of distinct phrases found in the database of tweets matching the target phrases with wildcards.

given in Table 3.11.

Signals reflecting a job posting are much more frequent than those reflecting job search. Search signals are comparable in their frequency to those reflecting job loss (compare to Table 3.1). Table 3.10 presents the correlation matrix of all the job search and job posting terms. While "find" is not closely correlated with any other terms, the other search terms are positively and similarly correlated. As expected, the posting terms are more closely correlated to one another than to the search terms. The "seek" term is correlated across search and posting terms, and is syntactically related to both, so it is included in both sets of terms.

There are analogues to the Twitter signals for search and posting in conventional data sources:

- The unemployment rate is a measure of search activity, especially since the BLS requires a modicum of job search activity as part of the CPS definition of being unemployed.
- Help wanted advertising has been a traditional source of data on vacancies.[13]
- The BLS JOLTS data provide a survey measure of job openings.[14]

For the job loss index developed in the previous section, UI claims were a particularly good analogue in official data. New claims for UI are high frequency. Moreover, both the "lost job" tweets and the new claims data are direct measures of flows. In contrast, the match of the search and posting terms with the unemployment rate and vacancies is not as clear-cut. First, it is not obvious which side of the market is generating the search and posting terms. Second, unlike in the job loss analysis, there may be a mismatch between stocks and flows when comparing the social media signals to unemployment or vacancies. There has been imaginative recent work on addressing the measurement issues relating to stocks versus flows in the labor market [44, 69]. Given the less than perfect analogy between our search and matching measures and potential official benchmarks, we do not pursue an econometric analysis along the lines of the previous section. We do, however, discuss our series in the context of recent findings from the JOLTS.

To construct the search and posting indexes, we do a factor analysis as discussed in the previous section. Table 3.12 presents the factor loadings for the first factor for the search and posting signals. As expected from the correlation matrix, the "find" signal has a smaller loading. Aside from the "find" signal in the first set of loadings and the "hiring" signal in the second, the loadings are fairly even across the signals. Figure 3.5 shows the indexes based on the first factors reported in Table 3.12. Since we are not benchmarking against an official index, there is no renormalization of this index. Hence, the index is not measured in meaningful units: it is the change in the index that has meaning. We use the same recursive procedure as before: after the starting period shaded in gray, the indexes are estimated recursively.

[13]The Conference Board formerly produced a monthly Help Wanted Advertising Index based on print advertisements, but discontinued it in 2008. It currently produces a comparable series, drawn from internet postings of job advertisements, the Conference Board Help Wanted OnLine data series [155].

[14]In the JOLTS, an opening needs to meet three criteria: a specific position exists; work could start within 30 days; and the firm is actively seeking workers from outside its location to fill the position [54]. The JOLTS data also have data on separations that can be compared to our job loss index. The social media index is more frequent and more timely than the JOLTS data. The JOLTS data are produced monthly and are available about two months after the reference week of the survey.

	Search	**Posting**
Find	0.45	
Look	0.86	
Need	0.87	
Search	0.70	
Seek	0.61	0.77
Hiring		0.46
Job		0.81
Work		0.82
Variance	2.57	2.14
Fraction of variance	0.51	0.53

Table 3.12: Job Search and Job Posting Factors: Loadings of First Factor, Alternative Sets of Signals. Note: Table shows the factor loading for the first factor for the selected signals. The bottom two rows report the variance of the first factor and the fraction of the overall variance accounted for by the first factor

Overall, there is a downward trend similar to that found in the job loss index. There are also spikes at various points, for example, December 2012, but not the previous December. The dip in October 2013 associated with the government shutdown is discussed in Section 3.4.4.

Note that the downward trend in our posting and search indexes is not seen in the JOLTS job openings rate over the same period. The JOLTS job opening rate was 2.4 percent in July 2011 (the start of our sample). In 2012 and 2013, it was higher than in 2011, but with no discernible trend. It reached 2.8 percent in early 2012 and—with small ups and downs—was still at 2.8 in late 2013 (see Chart 1 of November 2013 JOLTS report [52]). In contrast, our search and posting indexes have a substantial movement down in the second half of 2011 and a slight downward trend in 2012 and 2013. These differences suggest that the social media indexes are measuring something different from the JOLTS openings, perhaps because of stock/flow considerations.[15]

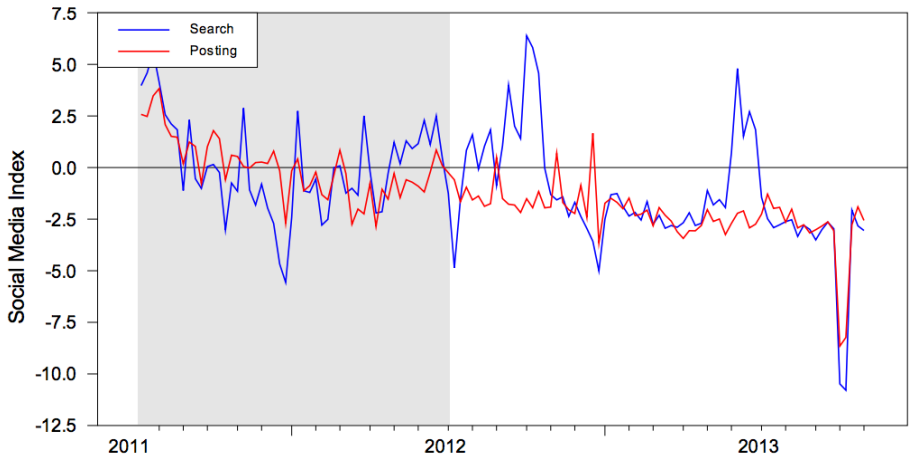
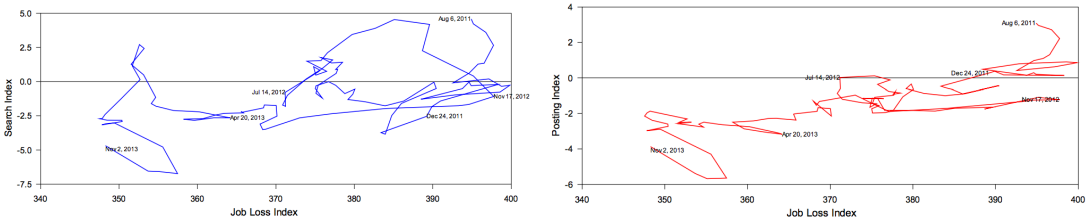


Figure 3.5: Social Media Indexes for Job Search and Job Posting. Note: Indexes are based on factor loadings in second two columns of Table 3.12. The social media indexes are estimated in sample in the shaded area and recursively thereafter.



(a) Search (b) Posting

Figure 3.6: Beveridge Curves for job search and job posting trends. Note: Figures show the four-week moving averages of the Social Media Job Loss Index versus the Search and Posting indexes.

3.4.2 Beveridge Curves

We use our labor market indexes to study the relationship between job loss, search, and posting akin to the Beveridge Curve. Figure 3.6a shows the Search/Job Loss Beveridge Curve and Figure 3.6b show the Posting/Job Loss Beveridge Curve. Of course, these indicators are not identical to vacancies and unemployment in the standard Beveridge Curve, but they have potential to shed light on labor market equilibrium. Note that the relationship between the variables is mainly positive, especially in the

[15]The Conference Board HWOL series has a similar flattening in 2013 after a recovery from the 2009 trough. The JOLTS separation rate (see Chart 2 of November 2013 JOLTS report [52]) have a similar pattern to the job openings rate: moving slightly up from 3.2 percent to 3.4 percent from mid-2011 to the beginning of 2012, then bouncing between 3.1 and 3.4 in 2012 and 2013 with no discernible trend. Note that we are comparing our series to JOLTS for the period where we have data, beginning in 2011. The JOLTS openings rate has a strong upward movement from its trough in 2009 at the depth of the Great Recession.

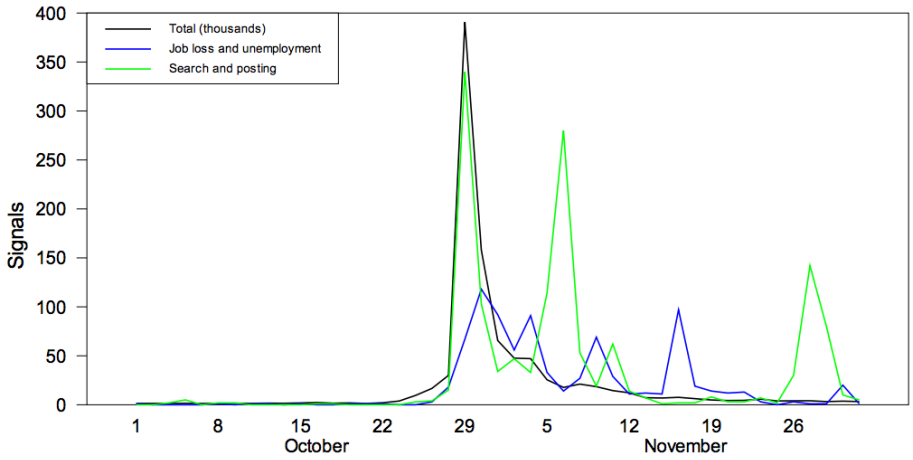


Figure 3.7: Social Media Signal Related to Hurricane Sandy.

posting/job loss figure that is most analogous to the traditional curve. Taken at face value, this finding suggests that over this period (July 2011 to early November 2013), inward shifts of the Beveridge Curve dominated movement along the Beveridge Curve. This finding is consistent with recent work by Barnichon, et al. [44] and Hobijn and Şahin [99] that shows that there were significant outward shifts of the Beveridge Curve, as measured by JOLTS data, with the onset of the Great Recession.[16] There are various explanations of the outward shift in the Beveridge Curve that started at the onset of the Great Recession relating to deterioration in matching jobs to the unemployed, especially those unemployed for long durations. Our social media indexes suggest a reversal of this deterioration of labor market conditions at least since mid-2011.

3.4.3 Labor Market and Hurricane Sandy

One of the potential benefits of analyses using social media data is that researchers may examine the impact of unexpected events as they happen without relying on recall or chance surveying during such events.[17] We examine signals related to Hurricane Sandy to demonstrate this type of analysis. Figure 3.7 shows all tweets (measured in thousands) that include the words "Sandy" or "hurricane." Unlike our previous

[16]In Section 3.4.1, we noted that the JOLTS openings and separation rates both shift up from 2011 to 2012 and then exhibit no trend. In contrast, our job loss and search and posting indexes both have opposite shifts over the same period. The work cited concerning the JOLTS focuses on the bigger shifts in the Beveridge curve surrounding the 2009 trough. Unfortunately, our Twitter data does not encompass the Great Recession.

[17]For example, Kimball et al. [109] had a survey in the field when Hurricane Katrina hit; they used it to study the hurricane's effect on psychometric measures of happiness.

analysis, carried out at weekly frequency, Figure 3.7 shows daily data. Additionally, we simply present raw counts rather than rates or a statistical index, because we have no historical baseline.[18] Not surprisingly, the number of such tweets increases sharply as Hurricane Sandy headed toward the northeast coast of the United States in late October 2012. The series peaks on October 29 when Hurricane Sandy hit New York and New Jersey. Figure 3.7 also shows the subsets of signals that include either our job loss or search and posting terms. (Note that the scale differs by a factor of 1,000 from that of the total.) We can see that search and posting terms spike simultaneously with Sandy's arrival, while job loss references increase just after the hurricane arrived. The job market related signals continued at elevated rates well after mentions of the storm per se peaked.[19]

3.4.4 Government Shutdown

The effects of the government shutdown in October 2013 are clearly evident in the job loss index (Figure 3.3) and in the search and posting indexes (Figure 3.5). All have pronounced falls in the first two weeks during the shutdown and equal bounce backs in the weeks following the shutdown. These results imply that the shutdown had a significant effect on labor market activity, but that the effect was short lived.

Interestingly, beginning in September 2013, there are changes in search and posting relative to job loss that are consistent with movements along the Beveridge Curve (Figure 6). These observations are dominated by the effects of the government shutdown and reopening that are evident in the time series. Hence the labor market disruption associated with the government shutdown appears to be a classic demand shock instead of a disruption of the matching function. This episode illustrates the usefulness of social media for measuring and analyzing the impact of unexpected events. Our social media indexes provide high-frequency and contemporaneous information that is not available in conventional sources.

3.4.5 Demographics

A concern about the use of social media data is that those who participate in social media are not representative of the population. We can assess this concern by

[18] We do see increased mentions of hurricanes in the signals during the storm seasons in our data.

[19] There is a sharp peak on December 7, 2012 in tweets that mention "hurricane" or "Sandy" and "unemployment," presumably reflecting the release of the first Bureau of Labor Statistics Employment Situation report to reflect unemployment data after Sandy. This example illustrates the importance of controlling for data releases when analyzing the social media signals.

		Fraction of Signals (percent)			
		All	**Job loss**	**Search**	**Posting**
Age	14-18	20.9	12.3	37.1	9.6
	19-21	8.2	5.6	13.0	5.4
	22-24	10.1	9.0	12.4	6.8
	25-34	14.5	15.2	14.0	11.0
	35-44	13.1	15.9	8.3	14.5
	45-64	33.2	42.1	15.1	52.7
	Total	100.0	100.0	100.0	100.0
Sex	Male	60.6	66.7	50.5	59.8
	Female	39.4	33.3	49.5	40.2
	Total	100.0	100.0	100.0	100.0

Table 3.13: Signals by Age and Sex. Table shows fraction of job-related signals by age and sex of sender. The demographics are estimated probabilistically and are coded for only a subset of signals. Because of changes in the API, this sample ends June 15, 2013.

estimating demographic characteristics of tweeters. For a subset of signals, we can probabilistically estimate the age and sex of the sender based on attributes of the tweet. By examining the distribution of word choices in a set of tweets written by people with known age and gender, we can train statistical models to predict age and gender for the author of a novel tweet. The training set for the age predictor includes up to 3,200 tweets for each of 24,000 users, while the training set for gender includes 12,500 users' tweets. We identify users with known age and gender by searching for tweets that contain self-admission of demographic details, for example, "I'm 30 years old now, but still live with my mom" or "I'm a strong woman." The statistical technique we employ is a randomized decision tree classifier [50].

We use six age brackets: 14-18, 19-21, 22-24, 25-34, 35-44, and 45-64. Classifier accuracy on held-out data is 47.3 percent for age, and 82.4 percent for gender. Table 3.13 shows the fraction of job-related signals by age and sex for the subset of signals for which we can estimate demographic characteristics.[20] Though the distribution of age and sex does not match the population, the use of social media to communicate about job-related issues is much more evenly spread across demographic groups than one might have expected. In particular, middle-aged and older individ-

[20]The estimates in Table 3.13 are based on data only through June 2013. After that time, Twitter changed its public API, thereby reducing our ability to gather large numbers of tweets for specific individuals. Further work is required to quantify how this change in data availability will affect the accuracy of demographic classification, and if accuracy is reduced, what novel methods can be used to improve quality.

uals are over-represented in the job-related signals, in comparison to how frequently they tweet overall. Note that senders of a signal need not be tweeting about themselves: for example, messages by a teenager could be commenting on a job transition for a parent. Even so, it is reassuring that a substantial majority of our job-related signals are from the working-age population.

3.5 Conclusion

This chapter addresses the challenge of turning the vast output of social media into data that can be used to create meaningful measurements. Doing so requires processing a very large dataset, coding social media signals for analysis, and using statistical methods to transform them into economic data. In this chapter, we accomplish these tasks. We create a social media signal of job loss that closely tracks initial claims for unemployment insurance. Despite obvious differences in the underlying processes generating unemployment insurance claims and tweets about job loss, the social media index tracks the official data at both high and medium frequencies. We construct a real-time index and show that this index has information about initial claims not reflected in either the consensus forecast or the lagged data. The indexes shed light on specific events such as Hurricane Sandy and the government shutdown.

We began our analysis with a concept—job loss—that has a relatively well-measured analogue in high-frequency official data. Having shown that a social media index can track a concept that is relatively well-measured, we turn to concepts that are less well measured. In particular, we construct indexes of job search and job posting, concepts of keen interest to analysts of the labor market, but less well measured in official statistics. We apply these series to show that the Beveridge Curve appears to be shifting inward since mid-2011—reversing outward shifts that other researchers identified during the Great Recession.

Longer time series and further analysis are needed to confirm the usefulness of social media in constructing indicators of economic activity. Nonetheless, this chapter has demonstrated that it is both feasible and useful to infer information about the state of the labor market from postings on social media that are generated by individuals in the normal course of their social interactions. Variables derived from social media can be both substitutes and complements to data generated from surveys and administrative records by statistical agencies and the private sector. They have the promise of providing measurements at relatively low cost, with high frequency, and virtually in real time, so they have potential advantages over traditional data

sources. That is not to suggest, however, that social media could supplant official statistics. Official statistics provide necessary benchmarks for understanding even the best measured variables from social media. In practice, the rapid evolution of the use of social media could make the relationship between the measurement and the underlying fundamental being measured unstable. Our recursive procedure in constructing the University of Michigan Social Media Job Loss Index is one approach to addressing this potential instability. As we accumulate longer time series, research using methods described in this chapter is necessary to evaluate the extent to which social media data do track activity. Nonetheless, as with the search and posting series constructed in this chapter, social media data provide an opportunity to track hard-to-measure components of economic activity by capturing information that has been previously neglected or is difficult to measure in traditional sources.

CHAPTER IV

Morals of Nowcasting

In the previous chapter, we described how we developed a social media nowcasting model for estimating initial claims of unemployment insurance in the US. A year or so after publishing our first results [38], our model estimates began to diverge from the official data released by the US government. We began investigating why our estimates were diverging, and during the process, we learned several interesting lessons. In this chapter, we present these lessons, along with the methodologies we developed from them, to help guide future nowcasting efforts.

This chapter is based on collaborative work with Michael Cafarella, Margaret C. Levenstein, and Matthew D. Shapiro.

4.1 Introduction

Using Twitter data that stretched back to mid-2011, we built a nowcasting model that generated a novel estimate of initial unemployment insurance (IUI) claims each week in the US. This system's output has been updated and publicly available since January of 2014. Also in January of 2014, we wrote an NBER working paper that described our system and its remarkable accuracy up to that point [38].

Then, shortly afterwards, our nowcasting model's accuracy began to fall precipitously. In the first year of estimates (July 2012 to July 2013), our model's weekly IUI estimates had a correlation of 0.677 with the official data; in following year (July 2013 to July 2014), our model obtained a correlation of just 0.475. In the most recent period (July 2014 to April 2015), the model did even worse, obtaining a negative correlation of -0.416. Figure 4.1 illustrates these changes in our estimates.

This chapter presents a diagnosis of why our nowcasting system failed and the mechanism by which our estimates rapidly became nearly valueless. Interestingly, the system did not fail because of technical reasons or any deep change in the use

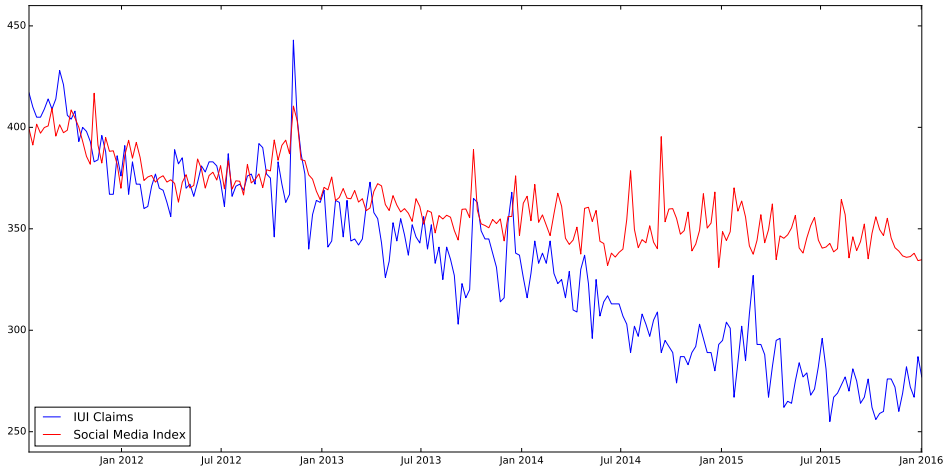


Figure 4.1: Our original nowcasting model (Old Model) for unemployment (using only *job loss* signals), where increasing estimation errors begin in 2014.

of social media. Rather, it failed because of economic modeling reasons: the real economy changed character in a way our statistical model could not capture. While the nowcasting system's *predictions* became valueless, the system's failure actually embodied an interesting finding about the real economy.

We believe our experience holds useful lessons about the best way to exploit nowcasting systems in the future. These lessons include how to make concrete estimates more accurately, but more importantly, include how to use nowcasting as a tool for generating meaningful economic insights.

The contributions of this chapter are as follows:

- We present a survey of some common pitfalls and points of failure in nowcasting models, along with ways to identify and address these issues (Section 4.2).
- We propose a method for detecting and observing underlying changes in a nowcaster's target phenomenon—a novel approach to generating new economic insights (Section 4.3).
- We present several useful lessons we learned from our experiences about the best way to exploit nowcasting systems in the future (Section 4.4).

4.2 Survey of Nowcasting Pitfalls

For more than 18 months, our nowcasting model—which worked so well in 2013—has consistently overestimated the official IUI number. We examined several possible reasons the nowcaster's results could have diverged so badly from official data, starting in early 2014.

In this section, we describe four reasons why a nowcasting model can fail, and describe how they relate to our own nowcasting model failure.

4.2.1 Social Media Sampling Issues

For many researchers working with social media, they have limited access to social media data. For example, Twitter provides an API for retrieving tweets, but when retrieving a full sample, this is limited to 1% of the total tweets, unless elevated access is established.[1]

Depending on how such sampling is done, the retrieved data may not be a perfectly random sample. For example, imagine that Twitter had become far more aggressive about detecting and deleting advertising-centric "spam" messages. The number of people in the overall population who utter "I lost my job" might decrease, but the percentage of all non-deleted tweets that contain "I lost my job" might stay steady or actually increase.

Ideally we would normalize message counts by the number of individuals represented in the sample, rather than the raw message count (which is more subject to filtering and sampling decisions). We therefore introduced a panel of "everyday phrases" that should not be impacted by any real-world shifting phenomenon: simple greetings, descriptions of lunchtime, etc. We retrieved the signal for each of these phrases, then performed PCA and obtained the first factor. We used this resulting "everyday signal" to normalize our subject-specific social media signals, rather than the raw count we used previously.

In our case, this adjustment did not help; however, biased sampling can be a point of failure in other models, so it should be considered when building or debugging a nowcasting model. Our results can be seen in Figure 4.2, with the regression results in Table 4.1.

[1] In our work, we have access to approximately 10% of the total tweets each day.

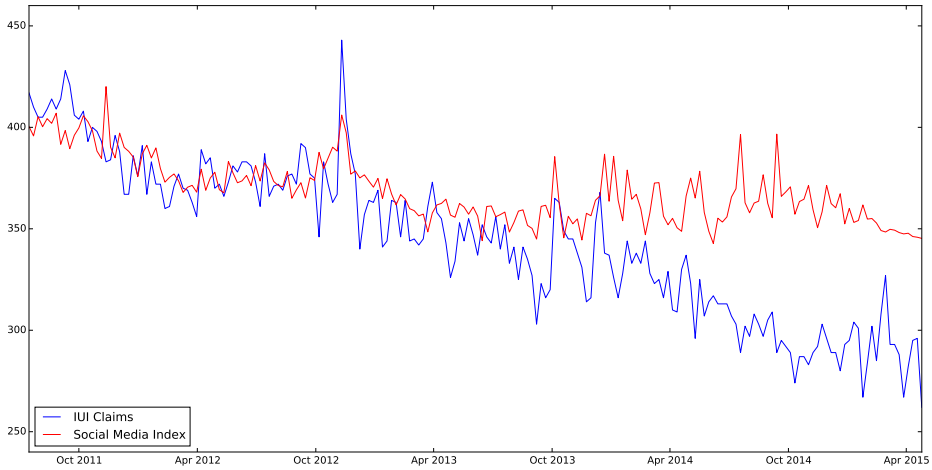


Figure 4.2: Our original nowcasting model for unemployment with "everyday phrases" used to normalize the social media signals.

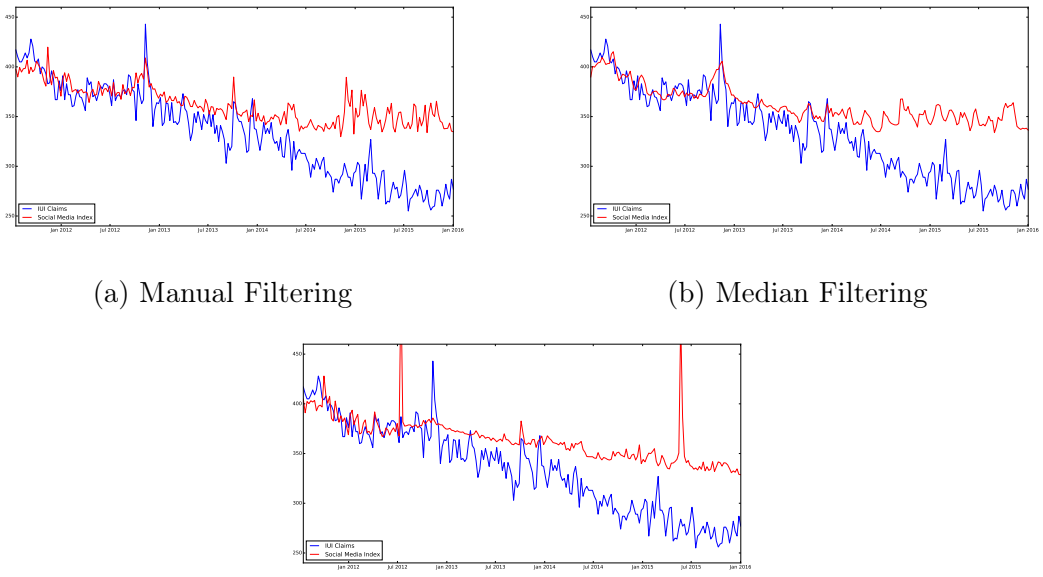
	Original	**New Normalizer**
Constant	345.4	345.4
	(2.4)	(2.7)
Job Loss (Factor 1)	6.8	0.0
	(0.9)	(0.8)
Adjusted R^2	0.24	0.0
RMSE	38.5	43.5
Correlation	0.62	0.43

Table 4.1: Regression results for our original model and that with the new normalizer. Standard errors in parentheses, and sample period is July, 16, 2011 through April 25, 2015.

4.2.2 Adjusting for "Grey Swans"

When we derived our social media signals, we extracted the different phrases of up to four words in length from each social media message, and then enumerated all the occurrences for each phrase as a daily signal. This approach to creating social media signals can be affected by "grey swan" events, where a particular phrase that generally captures one meaning, and some event causes a second meaning to swamp the original usage for some period of time. For example, the phrase "fired" can relate to being fired from a job, but might also be used to describe missiles that were "fired" on another country.

For our target phenomenon of unemployment behavior, true economically-motivated



(a) Manual Filtering

(b) Median Filtering

(c) Heuristic Filtering

Figure 4.3: Our original nowcasting model with the different grey swan filtering methods applied. Sample period is July, 16, 2011 through April 25, 2015 (weeks ending Saturday).

messages are few in number compared to many news-driven usages, so it is possible that over time our topic-specific signals have accumulated large and non-economic spikes and other movements. These non-economic movements would eventually ruin the fidelity of our social media-derived signals.

We explored three methods that help adjust for grey swans. In the first, we manually identify potential spikes and adjust them with a previous period's data. In the second, we use median filtering to smooth away short term grey swans. And in the last method we explore, we use a smoothing heuristic, which works similar to median filtering but offers a bit more flexibility.

4.2.2.1 Method 1: Manual Identification and Adjustment

In this approach, each input signal is manually investigated for unusual spikes. For each of these spikes, a search is done for news stories involving the particular signal's phrase and dates around the spike. If any matches are found, the spike is replaced with the previous year's trend, or in the case where the previous year is unavailable, the most recent period without the spike is used. Similarly, if using this data in model, an indicator variable can be used instead so that the model can weight the original data accordingly. The manual identify and adjust approach can be ad hoc

	Original	Manual	Median	Heuristic
Constant	345.4	345.4	345.4	345.4
	(2.4)	(2.3)	(2.2)	(2.3)
Job Loss (Factor 1)	6.8	5.7	5.9	6.4
	(0.9)	(0.7)	(0.6)	(0.7)
Adjusted R^2	0.24	0.27	0.32	0.27
RMSE	38.5	46.8	36.6	37.8
Correlation	0.62	0.67	0.68	0.69

Table 4.2: Regression results for our original model and that with the different grey swan adjustments applied. Standard errors in parentheses, and sample period is July, 16, 2011 through April 25, 2015.

and require a great deal of manual work, but the approach can achieve good results.

For our model, we tried applying this adjustment to signals such as "fired from" and "pink slip," where they were respectively affected by news stories about a rocket "fired from" the Ukraine at an airliner [148] and the US Army issuing "pink slips" to active officers serving overseas [97]. Results from applying these adjustments to over 20 social media signals can be seen in Figure 4.3a, with regression results in Table 4.2. These show a slight improvement, but results are not as good as the pre-2014 period. Moreover, the overall direction of the system's IUI estimates is still contrary to the government's official data after early 2014.

4.2.2.2 Method 2: Median Filtering

This method of adjusting for grey swans requires far less work than the manual approach. To apply this, each input signal has median filtering applied to it, which replaces each data point with the median value found in a window of size k centered around the to-be-replaced point [164]. The effect should be to eliminate nearby spike-style outlier values. If one expects that most grey swan occurrences happen due to news-style spike events, this filter could remove many problems.

When we applied this filtering method to our unemployment model, the resulting accuracy is markedly better, but still not as good as the pre-2014 period. Again, the overall direction of the system's IUI estimates is still contrary to the government's official data after early 2014. We also explored a variety of different values for k, but they all suffer from the same deviation from the official data. Figure 4.3b and Table 4.2 show the results for $k = 4$.

4.2.2.3 Method 3: Smoothing Heuristic

In our last method, we mimic the concept of median filtering, but in a slightly different manner. First, each input signal is examined for "spikes" (i.e., magnitude changes) that are $x\%$ greater than the previous k weeks. One advantage this approach has over median filtering is that it is flexible in which spikes are targeted. If one wants to leave smaller magnitude spikes, simply increasing x allows for this.

Similar to the previous grey swan filtering methods, the resulting accuracy improves, but the overall direction of the system's IUI estimates is still contrary to the government's official data after early 2014. Figure 4.3c and Table 4.2 show the results for $x = 400\%$ and $k = 4$.

Overall, applying these smoothing and filtering methods helped to remove our model's non-targeted signal components, but they do not explain most of the nowcaster's failure. However, the positive effects of the grey swan filtering should encourage its use with future nowcasting models that potentially face features with grey swans.

4.2.3 Weakening of Social Media Predictiveness

A third point of failure in nowcasting models is that the relationship between peoples' utterances in social media and the real trend in the target phenomenon have changed over the nowcasting period. For example, in our unemployment model, perhaps our chosen phrases and accompanying signals—"I lost my job", "I need a job", and so on—once reflected our target phenomenon, but this connection became weaker over time. This weakening can result from behavioral changes among people using social media, or shifts in the underlying phenomenon. We discuss each of these below.

4.2.3.1 Behavioral Changes of Social Media Users

The usage of online social media has changed drastically from when it first began a decade ago. As social media has gained widespread adoption, the population demographics have also changed. These users have different interests and roles in their lives, so the content they create differs from that of early adopters. User groups who were under-represented initially have grown significantly over the years. This includes seniors, users in rural populations, and those in lower-income households [137]. These changing populations can affect the behavior of social media signals and their

predictiveness.

New features introduced to social media also play a role in changing the behavior of social media signals. For instance, Twitter hashtags are a feature introduced in 2007, but grew in popularity when Twitter began publishing trending hashtags on its homepage in 2009 [142]. Additionally, the way features are used can change. Shapp describes how hashtags have grown to have two popular uses: indicating the topic for a message ("tagging") and as part of the message's content ("commentary") [147].

Another way that behavioral changes can occur in social media signals is through new usage of a term or phrase swamping the previous usage. This is similar to the grey swans discussed above, except that in this case, the usage continues in the long term. For example, one can imagine a phrase like "looking for a job" being used by people seeking employment, but later, job hunters use the phrase as a marketing tool, swamping the usage of people indicating their need for a job.

Identifying a behavioral change in social media usage can be tricky. One way to go about it is to manually look at the social message messages containing the term or phrase in question. If a behavioral change like the "looking for a job" case is occurring, it will be easy to spot. Traditional demographic analysis can help identify changes in social media populations. Finally, having an understanding for *how* social media data is being used (e.g., knowing when new features are released) can help identify these behavioral changes.

4.2.3.2 Trend Shifts in the Target Phenomenon

A different reason for peoples' utterances in social media no longer matching well with a target phenomenon can be due to shifts in the underlying phenomenon. For example, a nowcaster may be estimating traffic in a city by looking at social media messages; if that city installs a massive transit system, that can have an effect on the trend of traffic in the city.

In our particular model, we were concerned that our model reflected only job loss, and did not include job search, posting, or vacancies information. When formulating our model originally, we explicitly included social media signals that reflect job search, but our training procedure—tested and performed in late 2013, based on data from mid-2011 to late 2013—gave this component a negligible weight; it appeared to be entirely irrelevant to the target IUI signal. As we eventually discovered, the underlying trend of the economy changed, thus affecting the relevant features in our model.

To test this hypothesis on our model, we reformulated it to create a "New Model"

Job Loss	Job Search	Vacancies	Hired	Quits
axed	find * job	hiring *	found * job	quit * job
canned	look * job	jobs in *	got * job	quit * boss
downsized	need * job	work in *	hired	quiting * job
outsourced	search * job	job * open	new * job	
lost * job	seek * job	job * vacancy	start * job	
fired * job			took * job	
been fired				
laid off				
unemployment				

Table 4.3: A sample of phrases used in the Old and New models. The Old model uses only the Job Loss phrases, whereas the New model uses all of the phrases. Note: the asterisk works as a wildcard, including any phrases with any word in its place (e.g., "found * job" would include "found a job").

that includes additional social media signals, reflecting different job-related behaviors. The signal phrases for the New and Old Models are listed in Table 4.3. The New Model expands upon the Old Model by including phrases relevant to four new explanatory factors for IUI: *job search* (find * job, etc.), *vacancies* (hiring *, etc.), *hired* (found * job, etc.), and *quits* (quit * job, etc.). We also shortened the training period for each model to only include the last two years of data—to help prevent past trends from muddling any new trend changes. This is in contrast to our Old Model, which trained on all available data and only included phrases relevant to the *job loss* explanatory factor. After building the signals for each phrase, we performed PCA on each of the resulting distinct sets of signals.

Our New Model builds a regression model using these five resulting signals, again, only using the most recent two years of data. Figure 4.4 shows that the New Model is substantially closer to the target IUI signal (ICSA Realtime) than the Old Model. In late 2013, training a similar model gave negligible weights to non-job loss signals, but this New Model has non-trivial weights for all inputs.

The difference between our late-2013 model and the New Model suggests there might have been a meaningful shift in the economy around the time our Old Model started to fail in early 2014. There is some real economic evidence for such a shift; for example, the Federal Reserve withdrew quantitative easing in February 2014. Further, JOLTS data suggests that job loss (i.e., "separations") stayed steady in 2014, but that job gains (i.e., "openings") rapidly increased [53]. But there is also evidence in the Twitter data. First, we see our *vacancies* and *job loss* factors both have major shifts in early-2014 (Figure 4.5).

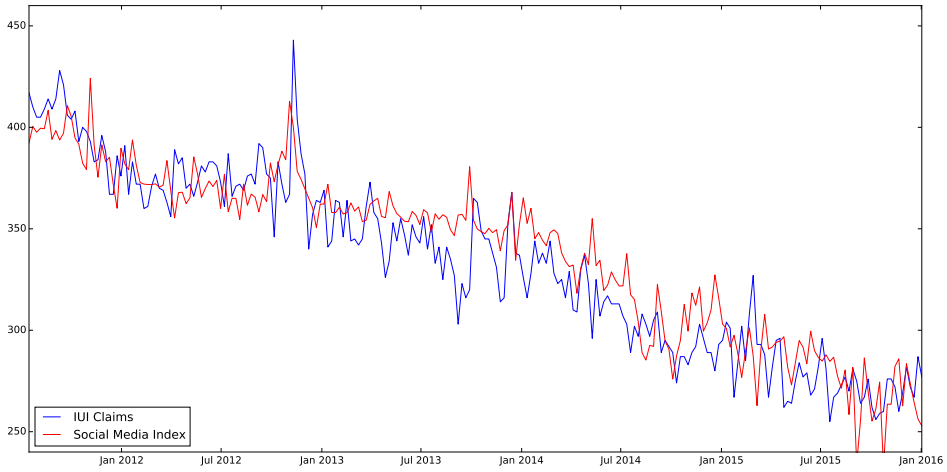


Figure 4.4: Our new nowcasting model for unemployment (using *job loss*, *job search*, *vacancies*, *hired*, and *quits* related signals), trained with the last two years of data at each time period.

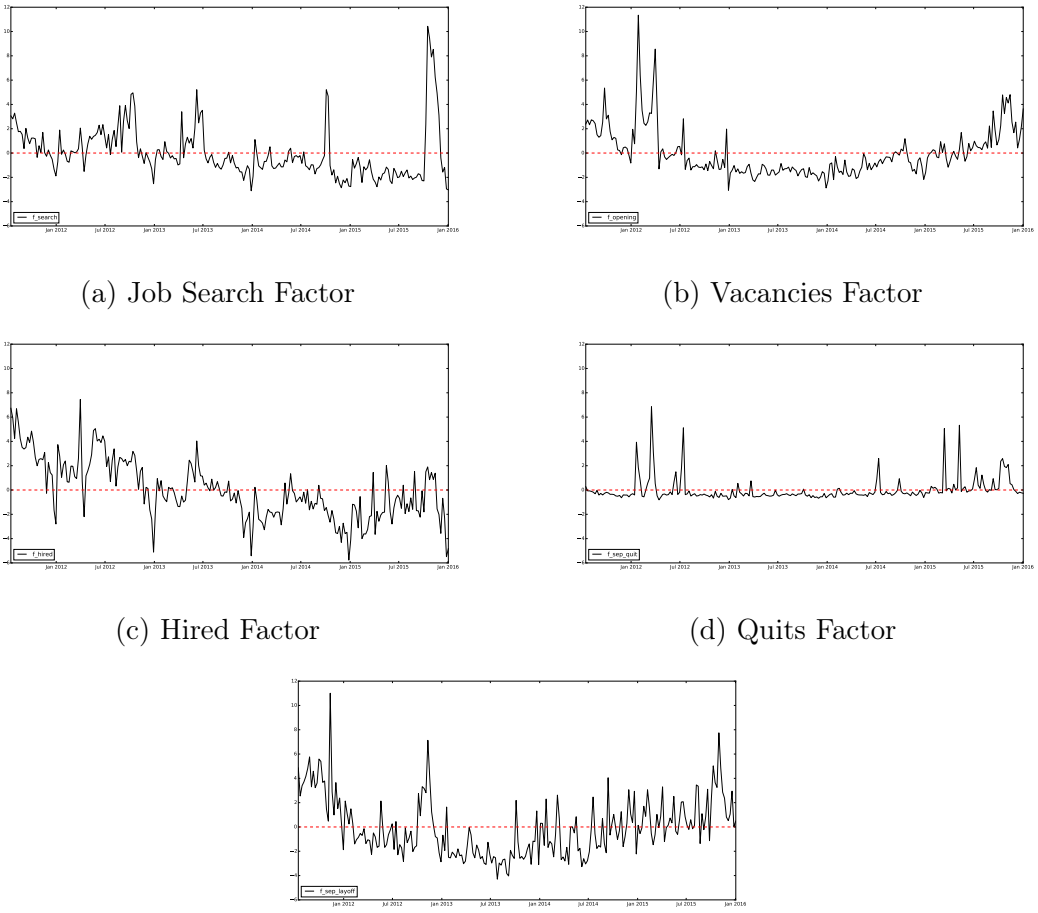
Second, if there was a meaningful shift in the real economy in early 2014, then we would expect to see that a model trained on the preceding k months of data would show good accuracy leading up to 2014, then would be highly inaccurate until the trailing k window included a good amount of post-shift data. That is exactly what we see in Figure 4.6; each data point shows (on the y-axis) the accuracy of a model trained on the date on the x-axis for the preceding 24 months of data. By late 2014, the model has swung back to its 2013-level of accuracy.

However, this return-to-accuracy only takes place when using the New Model's training period (most recent two years of data). The same experiment using the Old Model shows that models trained in early 2014 go to a high level of inaccuracy *and do not improve* (Figure 4.1). Our Old Model that included only job loss information cannot account for the early-2014 rise in job gains.

Following a similar process, one can detect changes in the underlying phenomenon of any nowcaster. We describe this in more detail in Section 4.3.

4.2.4 Too Few of Observations

For our last potential point of failure, we will discuss a common problem with nowcasting applications: lacking enough observations. Without a proper number of observations, a nowcasting model becomes highly vulnerable to overfitting. By only having a small number of observations, there is a high likelihood that there



(a) Job Search Factor

(b) Vacancies Factor

(c) Hired Factor

(d) Quits Factor

(e) Job Loss Factor

Figure 4.5: The different factors of our New Model, built using data from July 2011 to January 2016. The dashed red line is at $y = 0$.

are scenarios that are not represented by these observations, and when one of these scenarios surfaces, the model will do a poor job at estimating the scenario. For example, consider the naïve scenario of unemployment seasonality: Unemployment generally spikes after the winter holiday when season jobs, such as at a shopping mall, end. If a nowcasting model is only trained on summer months, holiday-related job loss indicators will not be recognized as relevant by the model, and estimates will suffer.

When we first established our unemployment nowcasting model, we used one year of weekly observations to train our initial model, with a new model being trained each week as new data became available. While we only used a few dozen variables, which were reduced by principal component analysis (PCA) to just one predictor, we still risked having scenarios that were not represented by the observations we had.

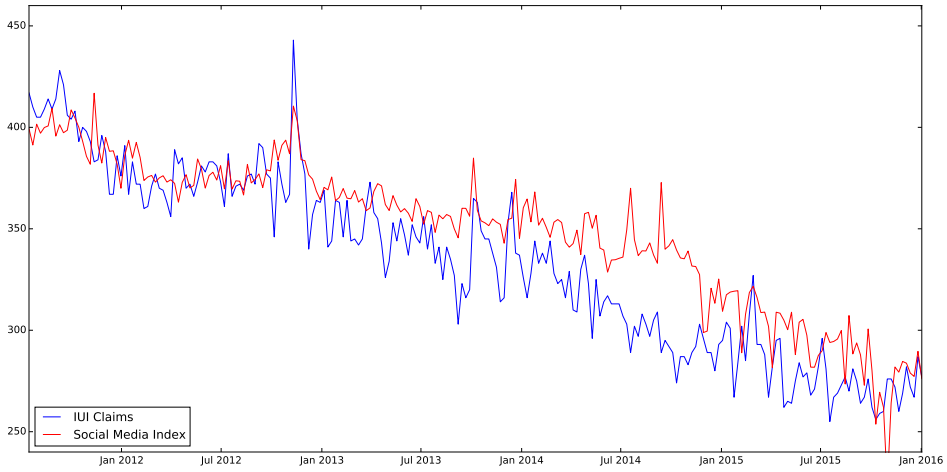


Figure 4.6: Our original nowcasting model (OldModel) for unemployment (using only *job loss* signals), but trained only on the most recent two years of data. When the underlying usage of *job loss* in social media changed in early 2014, the model took half a year to adjust.

To test if too few of observations were impacting the quality of our nowcasting estimates, we increased the initial observation size to two years of weekly observations in our New Model (Figure 4.4). The results are fairly negligible, which is likely because of our principal component analysis (PCA) feature reduction, which helps avoid overfitting. PCA can be a useful tool to use when access to more observations is limited.

4.3 Detecting Underlying Phenomenon Shifts

Our experiment in Section 4.2.3.2 suggests to us a new way to use nowcasting systems: not as a simple predictor for missing data values, but as a method to test hypotheses about the target phenomenon's inner workings. By testing to see which signals best improve the nowcaster's accuracy, we can see which real-world strings best explain the target phenomenon.

We propose the following heuristic for detecting underlying phenomenon changes:

1. Formulate a set of model test data. In our case, this set consists of points in time in which we would like to predict the IUI level.

2. Formulate a set of explanation sets. Each explanation set is a set of string-

groups that can be found in social media. For example, our Old Model had an explanation set of just one string group: *job loss*. Our New Model had an explanation set of four additional string groups.

3. Build a model for each explanation set. The model with the best fit yields the best explanation.

From a statistical perspective, this hypothesis testing procedure can be seen as a simple form of feature selection and is hardly new. What is new is the economist's ability to formulate novel and complicated hypotheses through the generation of string sets. The possible hypotheses are limited mainly by the user's ability to identify coherent string groups from the social media corpus. (And, as discussed in Section 4.2.4, the user is also limited by the risk of overfitting. She must either use a very large set of test data—which is likely difficult if these data points are drawn from the time domain—or use statistical regularization when building each model.)

4.4 Lessons for Nowcasting

Our experience with nowcasting does not appear to be unique. The high-profile Google Flu Trends project also encountered several collapses in accuracy after the original work [89], which the authors addressed substantially by using different statistical techniques [20]. The use of tree ring data to "predict" missing values in the temperature record also encountered well-documented accuracy problems [32, 117, 118].

All of these systems have a problem that is likely impossible to avoid with the overall nowcasting project: these systems are called upon to make repeated predictions throughout time, while using what is likely a very limited sample from the time domain. Even if the size of data appears to be large, building predictive models based on a limited number of time domain samples means that nowcasting systems will be vulnerable to relatively rapid changes in the target phenomenon. It is not clear if this failure mode characterizes the Google Flu Trends and tree ring projects, but our experiments clearly indicate such a regime change in the US economy in late 2013 and early 2014.

This failure indicates a serious problem with nowcasting over long time spans; in the future, we would likely distrust models trained long ago using a small sample from time. However, we believe nowcasting remains valuable, as long as the practitioner follows a few rules of thumb:

First, assume that any nowcasting model will "rot" fairly quickly. Since extremely large samples from the time domain are generally infeasible, one must assume that the test regime for a nowcaster closely resembles the regime in which it was trained. This assumption is likely only valid for a short time period after the model is trained, though the exact length of time depends on the problem area.

Second, incorporate strong domain knowledge into the nowcasting model when possible. We knew that job search is an important part of unemployment phenomena, but we removed this element because of lack of support in the 2011-2013 data. If we had included it, our system could have responded to the economic shifts of early 2014 by increasing the coefficient on job search, in a manner similar to our "regime break" experiments. A caveat: the practitioner can only include a small number of factors supported by "domain, not data" before encountering statistical overfitting issues.

Third—and to us, most importantly—reimagine nowcasting systems as a tool for generating economic insights as well as accurate data. If we had followed the precepts above, our nowcaster in early 2014 would have remained accurate. Accurate data is important, but it would have been very exciting to observe a rapid shift in the relative importance of job loss and job search in 2014. Identifying such a shift is a deeper insight into the working economy than nowcasting systems have generally attempted. This ability to generate these insights is a power we aim to exploit in future work.

4.5 Conclusion

Nowcasting has grown in popularity over the years, but little has been done to help guide nowcasters on the best practices. In this chapter, we present a survey of some common pitfalls that can cause a nowcasting model to fail, along with ways to identify and address these issues. We also propose a new use case for nowcasting: a tool for detecting when a nowcaster's target phenomenon experiences an underlying shift. This can serve as a new mechanism for discovering economic insight from social media. Finally, we conclude with several lessons we learned from our own experiences about how best to exploit nowcasting in the future.

CHAPTER V

Feature Selection for Easier Nowcasting

One observation we made with our nowcasting research in Chapter III was that *feature selection* when nowcasting a particular phenomenon is a challenging problem. Typical nowcasting systems require the user to choose a set of relevant social media objects, which is difficult, time-consuming, and can imply a statistical background that users may not have. We believe this may serve as a major obstacle to widespread adoption of nowcasting.

In this chapter, we propose RINGTAIL [39, 40], a nowcasting system which helps the user choose relevant social media signals. It takes a single user input string (e.g., "unemployment") and yields a number of relevant signals the user can use to build a nowcasting model. We evaluate RINGTAIL on six different topics using a corpus of almost 6 billion tweets, showing that features chosen by RINGTAIL in a wholly-automated way are better or as good as those from a human and substantially better if RINGTAIL receives some human assistance. In all cases, RINGTAIL reduces the burden on the user.

This chapter is based on collaborative work with Michael Cafarella, Margaret C. Levenstein, Christopher Ré, and Matthew D. Shapiro [39]; along with demonstration system help from Erdong Li, and Shaobo Liu, and Bochun Zhang [40].

5.1 Introduction

Let us first formalize the *feature selection problem*. Consider the steps the user follows in most nowcasting projects:

1. Aggregate the relevant objects over time to yield a set of time-varying *signals*—such as the daily frequency of various phrases.

2. Determine whether each signal is *relevant* to the target phenomenon (e.g., perhaps the signal "I feel sick" is relevant for flu levels).

3. Use the selected signals, plus conventional data that describes the phenomenon (such as health system flu statistics), to train a predictive model; later, feed novel social media signals to the model to obtain nowcasting results.

The middle step, in which the user chooses a set of relevant social media objects, is a feature selection problem [33]. The post-aggregation database contains a vast number of candidate signals and the user must choose just a small number to yield a high-quality model in the final step. Most systems have used a human to identify a signal by listing strings that the social media object must contain. For example, a user interested in unemployment might choose a signal corresponding to all tweets that contain *laid off* and others for *got let go*, *looking for a job*, and *was canned*.

We believe this user-directed feature selection is much more burdensome than it first appears. The difficulty arises because users are only weakly able to choose good signals. For example, we obtained Twitter-derived signals for each of the above four phrases for the time period of mid-2011 to mid-2012 and measured their correlation to official US initial unemployment insurance claims data. To the human eye, each of the four phrases above seems reasonable, but their Pearson correlations with initial claims ranged from a terrific 0.74 (*laid off*) to a terrible 0.14 (*looking for a job*). In the experiments we describe in Section 5.5, the best three user-chosen signals for each of six different phenomena average a correlation of 0.58, while the worst three average just 0.15. Clearly, humans are unreliable signal-choosers.

This fact has terrible consequences for the usability of nowcasting systems. Users must preemptively choose a very large number of signals, or must engage in a repetitive choose-and-test loop until the nowcaster's performance is "good enough." Further, because nowcasting is useful in exactly those scenarios where conventional test data is rare, users must also be concerned with statistical issues such as overfitting. As a result, creating a nowcasting system is currently a time-consuming process that requires a statistical background. It is not surprising that non-computational domain experts have largely failed to embrace them.

Technical Challenge — This work describes RINGTAIL, a nowcasting system, which helps the user choose features. In response to a user's single topic query (e.g., "unemployment"), it yields a number of signals the user can immediately use to build a statistical model. Rather than relying on user expertise or scarce conventional data (e.g., as part of a *variable ranking* approach), RINGTAIL uses statistics from a Web

corpus to obtain semantic similarity information between the user's query and each candidate signal's label. The resulting system is domain independent and yields results that in our experiments range from slightly better to roughly break-even with the human-suggested labels (depending on how many suggestions the humans give).

We do not claim that we have found the strongest possible architecture for a nowcasting system. (Indeed, we expand upon this work in Chapter VI with an improved nowcasting system.) Instead, RINGTAIL is designed to roughly emulate the designs embodied in most previous nowcasting systems, while using the feature selection techniques that are this work's primary contribution.

Contributions — This work is organized as follows. We propose the RINGTAIL architecture for building and selecting nowcasting features (Section 5.3). We then present several feature selection techniques that do not consume precious conventional data (Section 5.4). Finally, we evaluate RINGTAIL on six different topics using a corpus of almost 6 billion tweets. Using multiple evaluation criteria, we show that features automatically chosen by RINGTAIL are on average better or as good as those from a human (Section 5.5).

We discuss related work in Section 5.2. We think this work addresses just the first of a wide range of interesting nowcasting questions, which we discuss in Section 5.7.

5.2 Related Work

There are two main areas of work relevant to our research.

Nowcasting There have been many recent projects that have attempted to use social media to characterize real-world phenomena. Most use hand-chosen social media signals and a small amount of conventionally collected data to train a statistical model, which predicts a real-world value. Target phenomena have included mortgage delinquencies [42], unemployment rates [43], auto sales [141], and home sales [168]. Goel, et al. [90] used search logs to predict sales of media products. Choi and Varian [64] hand-select categorical search data for some phenomena—such as unemployment in the US—as well as use a statistical approach (spike and slab) for other phenomena, such as consumer confidence. Their work relies on a pre-classified set of features, which we do not have. Additionally, our selection method brings a novel information source to bear on the problem, and combining these approaches is a possible area of future work.

Ginsberg, et al., which predicted flu levels from users' search queries [89], is a

notable exception to the standard method of choosing input signals. Their system composes a time-varying signal for each of the 50 million most popular searches, then chooses the 100 that have the best correlation with the target flu signal. This approach was possible because of the long history of both search queries and the conventional flu data; it would not work with vastly more candidate signals, or many fewer conventional data points. Doornik proposed to make flu trend prediction more reliable through a combination of purely statistical techniques and use of a broader set of search queries [73].

Of course, in this work we are primarily interested in other nowcasting projects for their method of choosing inputs. We do not compare their accuracy against our system's, because accuracy numbers depend not only on feature selection methods, but also on data availability, preparation techniques, and other factors beyond the scope of this work. That said, the larger RINGTAIL project aims to obtain high-accuracy results to support economists who are interested in accurate nowcasting.

Social media data has been used a number of related projects such as measuring happiness [128] or spatially tracking disease epidemics [112]. In the current work, we focus only on phenomena we can also measure through a traditional mechanism, and do not incorporate geographic information. However, our feature selection techniques may be useful to these projects.

Feature Selection Feature selection is a well-known problem in the statistics literature. Guyon and Elisseeff [94] provided a good survey of the field. They place feature selection techniques into a few broad categories. With *domain knowledge* techniques, a human uses first-hand knowledge of a topic to choose features manually. As we discussed, humans' domain knowledge does not appear to be sufficient when choosing nowcasting features. *Variable ranking* techniques use a scoring mechanism—such as correlation criteria, single variable classifiers, or information criteria—to determine the relative worth of each potential feature (also called a signal or variable). These techniques are popular, but will overfit when the set of candidate signals is large and the conventional data used for computing the score is small; unfortunately, that is the common case for nowcasting applications. (Leinweber [114] provided a vivid example of the pitfalls linked to overfitting and spurious correlations.) Finally, there are *dimensionality reduction* techniques such as clustering, principal component analysis, and matrix factorization. These approaches consume no conventional data points and are applicable to nowcasting applications.

5.3 System Design

In this section we present RINGTAIL's basic architecture, including the *feature preparation* pipeline that most nowcasting systems have in common, plus the *feature selection* process that is unique to RINGTAIL.

5.3.1 Feature Preparation

Figure 5.1 illustrates the data preparation pipeline that most nowcasting systems have to some rough degree. The process starts with a large corpus of social media messages. In our experiments, we used almost 6 billion tweets collected between July 2011 and July 2012. Other implementations could use individual web search queries or similar messages. Each message might have various metadata (e.g., username, source IP address, etc.) but the only strict requirement is that it has a timestamp.

In the next step, we aggregate these messages to obtain a large number of time-varying signals. This process is straightforward for very short texts like web searches: for each unique search, count the number of appearances in each 24-hour period over the collection's timespan.

For longer messages like tweets, the process is slightly more involved. For each tweet, RINGTAIL enumerates each sequence of t or fewer words into *grams*. For example, the tweet "lost my job" generates the grams {*lost, my, job, lost my, my job, lost my job*}. RINGTAIL then computes a signal for each of these unique grams observed in the entire corpus of tweets. Because a single tweet almost always contains multiple grams, it can contribute to multiple signals.

After processing the social media messages, we obtain the raw input to the nowcasting feature selection step: a huge number of *(gram, signal)* pairs—3.2 billion in our experiments when grams are 4 or fewer words. The user will eventually use a small number of these signals, plus a relatively small amount of conventional data, to train a statistical model, which predicts the target phenomenon. RINGTAIL's task is to choose the k signals from this massive set that will yield as accurate a model as possible.

5.3.2 Feature Selection

RINGTAIL's basic architecture is described in Figure 5.2. Note that in order to sidestep some computational issues that are outside the scope of the current work, we currently precompute some of the work in Step **D**, though we intend this work to

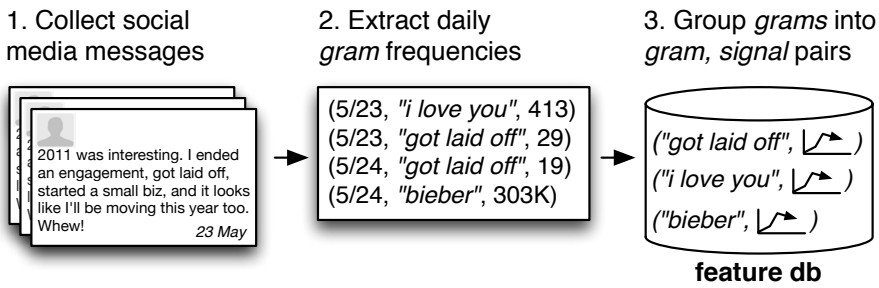


Figure 5.1: The pipeline RINGTAIL uses to convert a corpus of tweets into a set of *(gram, signal)* pairs.

be computed entirely at query-time—this is an active area of our research, which we describe in Section 5.7.

Using the pipeline described in Figure 5.1, Step **A** precomputes the candidate signals. The rest of the steps begin when the user enters a single *query* in step **B**. This includes a *topic query* describing the target—such as "unemployment"—and a conventional dataset—such as the US government's weekly data on unemployment insurance claims. The next step, *feature selection*, is where our contributions lie.

RINGTAIL feature selection comprises three steps. To start, we expand the user's *topic query* into a large number of gram candidates as pictured in step **C**. This uses an off-the-shelf thesaurus to expand each of the query's words and can yield up to several dozen *topic synonyms*. In **D**, we look up each topic synonym in the web-derived pointwise mutual information (PMI) database; PMI is a method for measuring the semantic relatedness of two strings [67]. Ranking candidate grams by PMI score can yield thousands of *synonym-PMI grams* that are related to the topic query. This set likely contains a large number of strings that should be semantically linked to the user's topic. We have not used any conventional data so far.

In **E**, we use principal component analysis to distill this still-large candidate set into a small number of synthetic signals we can return to the user. This process may cause us to lose signals that are "eccentric but correct" and so we must implicitly trust that any real-world phenomenon whose signal we want to observe will be captured by multiple social media grams. We have still not used any of the user's conventional data. Finally, in **F**, we use the conventional data to compute a statistical model and return it to the user. In the next section we examine RINGTAIL's feature selection steps—**C**, **D**, and **E**—in detail.

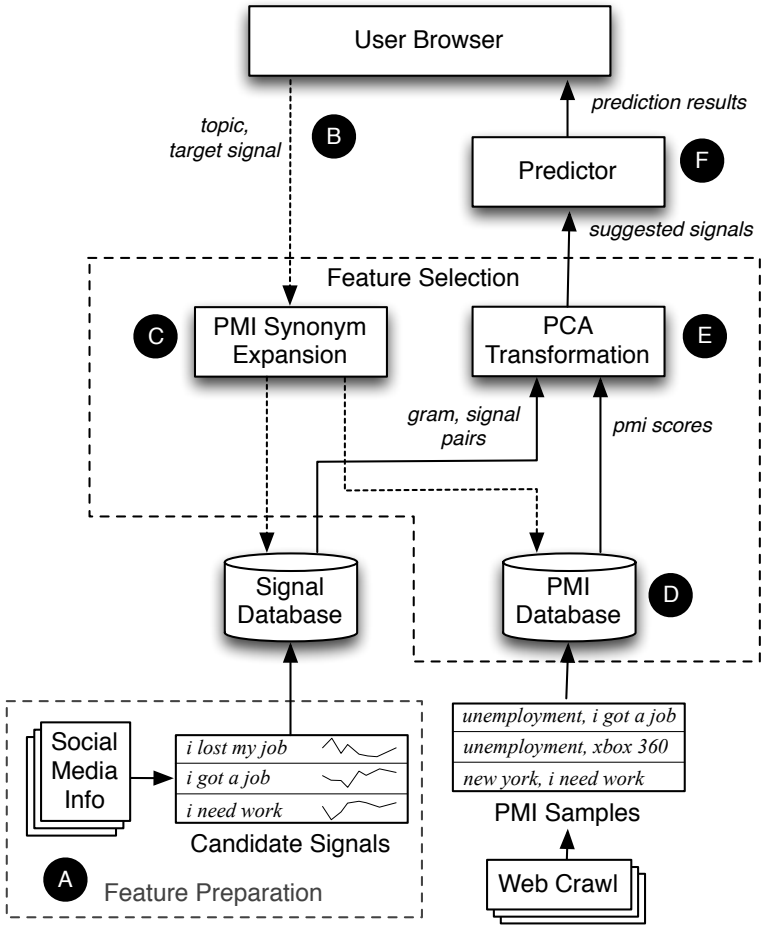


Figure 5.2: RINGTAIL's overall architecture.

5.4 Feature Selection

RINGTAIL's central goal is to choose a small number of relevant features from the massive set of candidates we can compute from a social media corpus. Feature selection has a vast literature; we will explain that most standard techniques do not apply in the nowcasting setting, with its relative paucity of conventional data. We then describe our proposed solution, which uses as little conventional data as possible.

5.4.1 Dead Ends

We now describe a few common feature selection techniques that appear reasonable at first, but will not work in our target nowcasting setting. In the course of preparing this work we tried each of them and met with no success (with the partial exception of subset selection, a limited form of which we incorporate into our final

technique).

Domain knowledge relies on a knowledgeable human to suggest good features. This approach is not terrible in terms of result quality, and is widely used by other nowcasting systems. But in practical terms, a domain knowledge approach yields a system that is difficult to use for all the reasons we described in Section 5.1: users have only weakly-accurate ideas about what signals are effective, and so engage in a repetitive choose-and-test loop that is both tedious and prone to overfitting.

Feature selection with signal data scores each feature according to a data-driven similarity metric based on the target phenomenon—such as Pearson correlation or R^2 from a single-variable regression. The underlying problem with this approach is the steep imbalance in size of our two datasets (e.g., 3.2 billion *(gram, signal)* pairs vs. 52 conventional samples). Given so many candidate signals, it is easy to find variables that score highly through sheer chance. For example, the gram whose signal has the highest in-sample Pearson correlation with our unemployment dataset is *out this facebook* (0.991). It is unlikely that this gram has any predictive power for the unemployment level. Of the 100 most highly correlated grams, *none* of them are plausibly connected to unemployment.

Variable ranking with a human filter uses data-driven scoring to obtain an initial ranking, then asks a human to manually remove spurious correlations. This is a plausible technique when the number of spurious correlations is small; however, the number of spurious correlations in nowcasting settings is so large that simply examining this list amounts to a substantial burden on the user. For example, in our correlation-ranked list of variables, the first one that is plausibly connected to *unemployment* is *ski job*, which does not appear until position 1,376—and this may still not carry much information. RINGTAIL will not be very usable if each nowcasting query requires the user to manually examine thousands of candidates.

Subset selection with signal data attempts to find not just a ranking of variables, but the best possible combination of them. Forward selection starts from an empty set of variables and grows a high-quality set, while backward selection gradually removes variables from a full set. Unfortunately, subset selection methods suffer from the same data imbalance as the above methods. Spurious subsets of variables will appear to yield high performance only because of the limited size of our test data.

All of the above techniques are either labor-intensive or suffer because our conventional dataset is so small in comparison to the potential number of grams. In

principle, variable ranking against the conventional signal data should work, if only we had sufficient data to avoid spurious correlations. Unfortunately, nowcasting systems are most useful exactly in those settings where conventional data is hard to obtain; we will never have as much as we would like.

It is easy to overlook that finding a source of this conventional data is only half the problem. We also need conventional data that *overlaps in time* with the social media data. Since no social media data source is more than a few years old, the best way to evaluate the relative scarcity of these data sources is how quickly we expect them to arrive in the future. By that measure, the quantity of social media data will always swamp conventional data. A small conventional dataset is not simply an artifact of our experiments, but rather has to be considered a basic challenge for usable nowcasting. Thus, the feature selection techniques we apply below are designed to use no conventional data at all.

5.4.2 Unsupervised Feature Selection

The central observation behind our approach is that while the signal part of each *(gram, signal)* pair is constrained by the availability of conventional data, the gram portion is not. When choosing signals by hand, humans are able to examine the gram alone and still yield good, though imperfect and burdensome, results. We attempt to build a system that similarly exploits common sense about the gram.

We use two techniques (SYN and PMI) to expand the user's topic query into a large family of potential grams.

5.4.2.1 Synonym Expansion (SYN)

In the first step we expand the user's single *topic query* into several roughly synonymous queries. A good topic query (*unemployment*) may be different from a good gram (*I need a job*), and the goal of this step is not to find good grams. Rather, our goal is to make the system robust to different word choices on the part of the user. Finding good grams is the next step.

Synonyms for a given topic query are generated from three sources: WordNet [10], Thesaurus.com [162], and Big Huge Thesaurus [3]. The user's topic query is split on whitespace into tokens, and each individual token is run through these three services. Out of the resulting words and phrases returned, all permutations from the different sets are used. For example, if the input label "gas prices" returns the sets *gas = {gas, fuel}; prices = {prices, costs}*, then the final list of *synonym topic queries* would

be *{gas prices, gas costs, fuel prices, fuel costs}*. This is a fairly naïve expansion algorithm, but as we will see in Section 5.5.3, its performance is roughly comparable to human-generated synonyms.

5.4.2.2 PMI Scoring (PMI)

For each topic query synonym, we now want to find all the related phrases that could embody useful signal data. With each query, we want to sort all grams in descending order of *relatedness* with it, then pick the top k. Fortunately, the information retrieval and Web search research communities have developed a straightforward technique for computing the relatedness of two strings.

Pointwise Mutual Information, or PMI, is often used to check for an association between words or phrases in a particular corpus [67]. For example, Turney used PMI to build a system that performed well enough to pass the synonym portion of a English language test [159].

Given two phrases x and y where $P(x)$ and $P(y)$ are the respective probabilities of each phrase occurring in the corpus, and $P(x, y)$ is the probability of the phrases occurring together, PMI is defined as

$$\text{PMI}(x, y) \equiv \log \frac{P(x, y)}{P(x)P(y)} \tag{5.1}$$

If there is an association between the two, then $P(x, y)$ will be much larger than $P(x)P(y)$, thus yielding a high PMI score. If there is no association between the two, the PMI computation will yield something close to zero.

Computing these probabilities is a critical ingredient to PMI. We can calculate them using any large text corpus. We used the ClueWeb09 English web crawl dataset [9]. This contains 500 million English web pages crawled during 2009. We processed this corpus using Hadoop MapReduce. We define two grams as "related" if they occur within 100 tokens of each other. With a large number of candidate grams (e.g., we use over 3.2 billion in our experiments), naïvely calculating all of these PMI scores is intractable, even with a large cluster of servers. We describe in Section 5.6.1 how we estimate these scores in a more efficient manner.

To combine these n synonyms' PMI rankings into one list, we take the top 1,000 grams with the highest PMI for each topic query synonym. In principle, this gives us $n \times 1,000$ potential signals; however, many of these signals appear in multiple lists, so the number of unique signal phrases is less. We rank these signal phrases by the number of lists on which they appear. Where there are ties, we rank signals by their average rank across synonym lists.

5.4.2.3 Data Reduction (PCA)

The first two steps expand the user's query, operating strictly at the text level. The final step performs data reduction on the actual signal data.

Once the features have been ordered, we then need to select features for use in nowcasting. Since there is a limited number of data points—just 52 when we have a year of social media data and the conventional data is a weekly signal—we can only select a handful of features to avoid overfitting when training the nowcasting predictor (step **F** in Figure 5.2).

We first explore *k-thresholding*, which involves selecting the top k features from each ranking method—our experiments in Section 5.5 set $k = 100$. While k-thresholding is a common practice, there may be a feature at *k+1* that carries important information, so we explored transforming a larger number of features with principal component analysis (PCA). PCA is used in a range of tasks that require unsupervised feature transformation, such as computer vision [70] and asset pricing [60]. The top j signals from each ranking mechanism are passed into the PCA algorithm—our experiments have $j = 500$. This yields a set of transformed signals, ordered by the amount of variance in the data explained by each. As before, we apply k-thresholding to the resulting list of signals.

5.5 Experiments

The central experimental claim of this work is that we can use automated methods to choose signals that are at least as good as those chosen by hand. In other words, we are concerned with the quality of the signals from step **E**. Note we do *not* claim this method always obtains a high-accuracy nowcasting system (the results of step **F**). Nowcasting accuracy is dependent on many factors besides feature selection, such as the amount and type of social media data, the exact phenomenon being predicted, and so on. Poor prediction accuracy might be due to any of these causes.

5.5.1 Experimental Setup

Our initial design of RINGTAIL uses roughly 6 billion tweets collected between July 2011 and July 2012. We transformed these into roughly 3.2 billion candidate signals (Figure 5.1 and Step **A** of Figure 5.2) using a series of MapReduce jobs. We evaluated RINGTAIL on six phenomena (listed in Table 5.1) that are past or plausible future nowcasting applications. Each target has a label and a conventional dataset

Target Phenomenon	Source	User Label
Box Office Sales	B.O. Mojo [14]	movie tickets
Flu Activity	CDC [15]	flu rates
Gas Prices	U.S. EIA [23]	gas prices
Mortgage Refinancings	MBA [5]	mortgage refinance
E-commerce Traffic	Alexa [1]	online shopping
US Unemployment	US DOL [22]	unemployment

Table 5.1: Target phenomena used for testing. US Unemployment refers to the weekly number of initial unemployment insurance claims. Amazon.com, Ebay.com, Walmart.com, and 30 other top-traffic websites were selected to represent E-commerce Traffic.

Metric	Random	Human	PMI	PMI+Syn	PMI+PCA		PMI+Syn+PCA	
					$k = 100$	$k = 3$	$k = 100$	$k = 3$
Average Correlation	0.2448	0.3630	0.2567	0.2543	0.1669	**0.4496**	0.1637	0.3993
Average R^2	0.0912	0.1809	0.0986	0.0970	0.0474	**0.2406**	0.0449	0.2176
Average MAE	0.1661	**0.1277**	0.1282	0.1308	0.1293	0.1313	**0.1278**	0.1349

Table 5.2: Evaluation of different feature selection mechanisms, averaged over the tasks in Table 5.1. For average correlation and R^2, larger values are better; for average MAE, smaller is better. Best ones, or close to best, are in bold.

associated with it, which the user provides in Step **B**.

We lightly preprocess the tweets, removing punctuation and discarding non-English messages. We also translate "one-off" text strings such as URLs and reply indicators (e.g., *@carlos*) into generic tokens (e.g., *<URL>* and *<REPLY>*).

Each gram consists of a sequence of four or fewer tokens from a tweet, which are generated as described in Section 5.3.1. These are then aggregated together into *(gram, signal)* pairs, where the signal represents the daily frequency for each gram between July 2011 to July 2012. Finally, we normalize the resulting signals to account for growth in total tweets. Each conventional dataset is weekly. For correlation and R^2 metrics, we convert the target signal and *(gram, signal)* pairs into four-week moving averages.

5.5.2 Feature Quality

We now show that RINGTAIL can obtain social media features that are at least as good as those given by a human expert. We first describe our evaluation metrics, then the benchmarked techniques.

Evaluation Metrics – We evaluate the quality of the emitted set of signals using

three metrics.

- **Average Correlation** – For each signal in the emitted set, we measure the Pearson correlation with the conventional data signal. Each emitted signal and the conventional data signal consist of the entire conventional data timespan—in our case, 52 data points. We average over all the emitted signals. Values are the absolute value of correlation, ranging 0 to 1, higher being better.

- **Average** R^2 – For each signal in the emitted set, we perform a linear regression computation with the signal as a predictor variable and the conventional data signal as the response variable. We compute the R^2 error quantity that arises from the regression. This is a standard error metric in economics and other fields. Again, we use the entire 52-week dataset and average over all emitted signals. Values range 0 to 1, with higher being better.

- **Average Mean Absolute Error (MAE)** – Finally, we compute the accuracy of the predictor in step **F**. This is the only metric that evaluates our performance using held-out conventional data. For each signal in the emitted set, we build a series of linear regression predictive models using subset selection (max three features per model). Each model uses 30 data points from both the emitted signal and the conventional data series. We then ask the model to predict the value for the *next* data point in the conventional series. The difference between the prediction and the held-out data is the error. We average this error over $52 - 30 = 22$ rolling predictors. Finally, we average the MAE from each of the models. MAE is described in percentages, and smaller values are best.

Benchmarked Techniques – We tested several feature selection methods, all different combinations of the three techniques described in Section 5.4.2: SYN, PMI, and PCA. We compared these approaches against two baseline methods. The first, RANDOM, is simply a set of 100 signals drawn randomly from the overall signal database. The second, HUMAN, is the result of asking seven graduate students to each suggest 10 relevant grams for each target phenomenon (e.g., *I need a job* for US unemployment). RINGTAIL's goal is to match or beat HUMAN's quality with an entirely automated method.

5.5.3 Experimental Results

Our experimental results are summarized in Table 5.2. Not surprisingly, HUMAN outperforms RANDOM on all three metrics. In addition, at least one of RING-

TAIL's techniques can beat or essentially tie HUMAN. For *Average MAE*, the best RINGTAIL technique is the one that uses all three approaches from Section 5.4.2 (PMI+SYN+PCA).

For *Average Correlation* and *Average* R^2, the result is somewhat different. When $k = 100$—as is the case with the other benchmarks—PMI+PCA and PMI+SYN+PCA perform worse than RANDOM on average, yet the top signals ($k = 3$) easily outperform HUMAN on average. We suspect this is due to the way PCA ranks the signals by the amount of variance of PCA inputs that they explain. The bottom ones carry little of the information that the top ones do, so they perform quite poorly, thus lowering the overall average. While the seemingly low correlation and R^2 values in Table 5.2 may cause one to question the quality of RINGTAIL, we emphasize that we make no claim the signals are perfect, only that we can essentially match a human on average (0.1277 for HUMAN, vs 0.1278 for RINGTAIL). Although correlation and R^2 are useful metrics, MAE is likely the metric the typical nowcaster will want to maximize.

To test if there is room for improvement with our synonym combining heuristic, we asked a human choose the synonyms manually instead of using an automated thesaurus-driven process. Results show this algorithm performs 5% better than HUMAN using average MAE, compared to the tie that arises from our automatically generated synonyms.

5.6 System Implementation

As part of this work, we implemented RINGTAIL as a live system that allows users to query for arbitrary topics and manage the resulting list of signals. In this section, we first discuss how we efficiently calculate PMI scores to allow for fast query processing. We then describe the user interface for RINGTAIL and provide a walk-through of how a typical user interaction would occur.

5.6.1 Efficient PMI Calculations

Much of the systems-building challenge of RINGTAIL lies in efficiently calculating PMI scores for each topic query. A single query requires a huge number of PMI scores (one for each candidate), a single computation could integrate frequency counts from many Web pages, and PMI is difficult to precompute (since in principle there are as many possible PMI scores as the square of the number of grams, which lies in the billions). RINGTAIL has three techniques for obtaining PMI scores.

The first method is accurate but induces huge latency: for each query it runs

a MapReduce job across a cluster of servers. When considering a large corpus of candidate grams (i.e., in the billions), calculating all the required PMI scores is intractable—even with a large cluster of servers—so this method of operation is reserved for use cases where the number of grams can be reduced drastically (e.g., by pre-filtering or limiting to unigrams).

The second method approximates the PMI scores by using a random sampling of the Web corpus. Some accuracy is lost, but speedup is near linear with corpus reduction. Unfortunately, even with a reasonable sized cluster, we have to reduce the corpus size quite a bit, which results in a significant drop in accuracy.

Finally, we have experimented with using matrix completion methods [144] to very efficiently approximate PMI scores even when we have relatively few samples of PMI scores. This last approach again trades accuracy of the scores for improved runtime performance, but we have found in our experiments that accuracy losses are fairly small, so this is our preferred PMI computation method for when the number of candidate grams is greater than a few million.

5.6.2 User Interface

RINGTAIL's external presentation is that of a web application with three primary pages. The first page is a simple search box, which prompts the user to provide a topic query ("unemployment") and a conventional dataset (e.g., the US weekly initial unemployment claims data for the past year). This is processed by RINGTAIL as described in Section 5.3.

The second page is shown in Figure 5.3 and is where the user spends most of his or her time. The **Suggested Signals** are the main output of the system, given in response to the user's **Original Topic Query**. The suggested signals can be highlighted by clicking on them. When a gram is clicked, the **Single-Signal Closeup** shows the corresponding signal data, as a percentage of social media messages over time. When the user clicks on a specific date within that signal, RINGTAIL displays some **Sample Messages** drawn from that date. Finally, the user can retain or reject RINGTAIL's suggestions to build the set of **User-Chosen Signals** that are sent to the last phase.

The final page allows the user to process the signals chosen in the previous step. It offers a few basic model training methods (e.g., a simple linear regression, autoregressive model); these methods are offered as a convenience rather than a deep contribution of the system. The user can also download the signal data for use with an external tool.

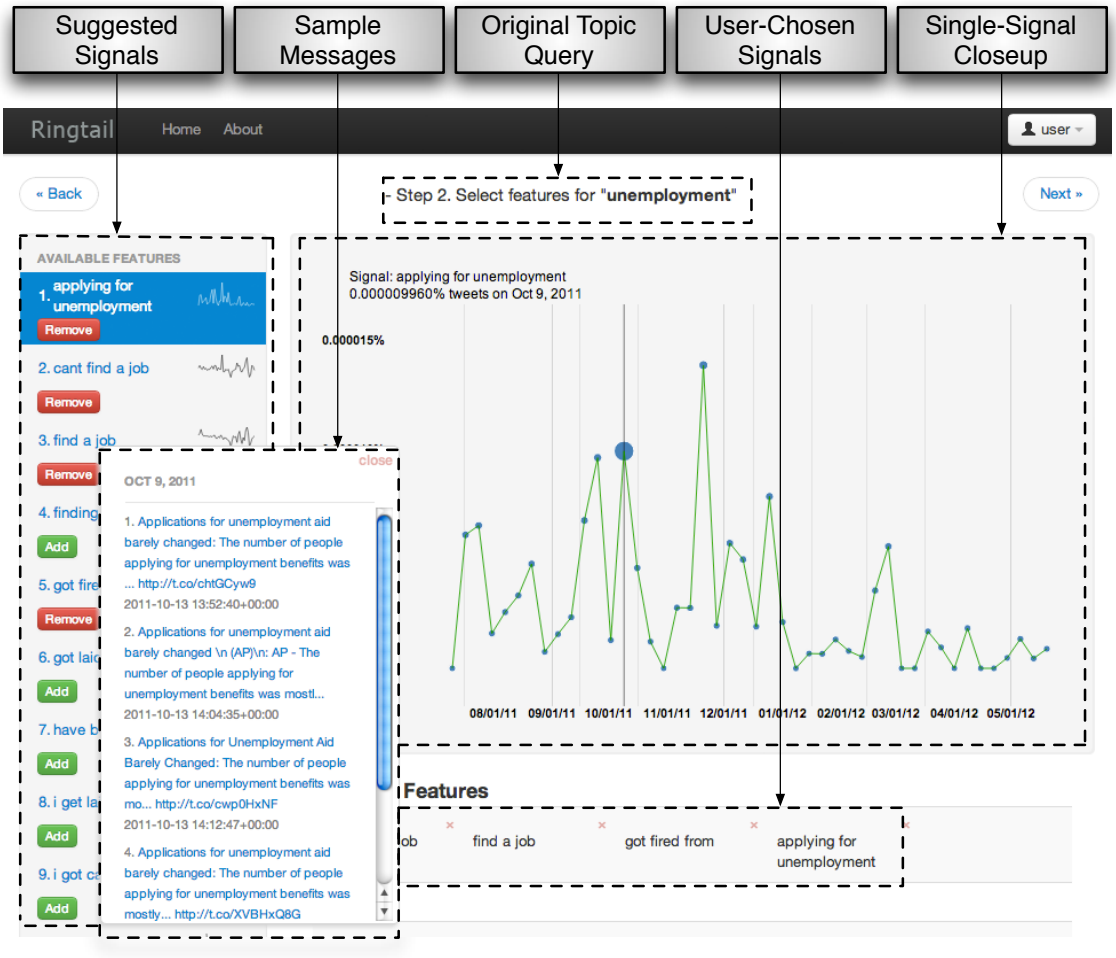


Figure 5.3: The RINGTAIL search result page shows **Suggested Signals** results in the left-hand column.

5.6.3 Example of a User Interaction

Consider "Ben," an economic policymaker:

1. Ben is concerned about the unemployment rate. He visits the first page of RINGTAIL and enters "unemployment" into the search box. He also provides a recent weekly dataset: a year's worth of unemployment insurance claims. He then clicks on the **Go** button.

2. RINGTAIL quickly shows a result similar to that seen in Figure 5.3. Ben can see a number of signal grams proposed by the system: *I need a job*, *I got laid off*, and so on. When Ben clicks on a gram, the relevant time series is shown in the central window. He can also click on an individual data point to list tweets that contain the relevant gram on the day in question; this feature is often useful

when trying to explain sudden and surprising moves in the data.

3. Ben adjusts the list of grams until he is satisfied. He then clicks **Next** to proceed to the final page of the application. On this page, he can train a simple regression model based on the chosen signals and the uploaded conventional data. The central window shows the conventional data series overlaid with a predicted signal generated by the trained model.

5.7 Future Work and Conclusion

We have demonstrated that RINGTAIL can suggest features as well as a human on a range of nowcasting tasks. We believe this approach is a critical step in making nowcasting a practical tool rather than a research curiosity.

Our immediate future focuses on efficiently computing PMI values (step **D**), which we believe would also have an impact on other Web research projects; we are currently exploring approximation techniques for PMI. RINGTAIL avoids conventional data for most of the feature selection process, but still uses it to train a statistical model (bottom row of Table 5.2). Nowcasting would be most useful in cases where conventional data does not exist at all. Indeed, it would be terrific if the predictive model could be built using social media data exclusively. Finally, improving nowcasting accuracy will likely continue to be a lively research area.

CHAPTER VI

A Declarative Query Processing System for Nowcasting

In previous chapters, we show how social media nowcasting can be used to estimate various real-world phenomena. However, the typical nowcasting workflow requires either slow and tedious manual searching of relevant social media messages or automated statistical approaches that are prone to spurious and low-quality results.

In this chapter, we propose a method for declaratively specifying a nowcasting model; this method involves processing a user query over a very large social media database, which can take hours. Due to the human-in-the-loop nature of constructing nowcasting models, slow runtimes place an extreme burden on the user. Thus we also propose a novel set of query optimization techniques, which allow users to quickly construct nowcasting models over very large datasets. Further, we propose a novel query quality alarm that helps users estimate phenomena even when historical ground truth data is not available. These contributions allow us to build a *declarative nowcasting data management system*, RaccoonDB, which yields high-quality results in interactive time.

We evaluate RaccoonDB using 40 billion tweets collected over five years. We show that our automated system saves work over traditional manual approaches while improving result quality—57% more accurate in our user study—and that its query optimizations yield a 424x speedup, allowing it to process queries 123x faster than a 300-core Spark cluster, using only 10% of the computational resources.

This chapter is based on collaborative work with Michael R. Anderson and Michael Cafarella [34, 36, 37].

6.1 Introduction

The past several years have seen a growing interest in social media *nowcasting*, which is the process of using trends extracted from social media to generate a time-varying signal that accurately describes and quantifies a real-world phenomenon. For example, the frequency of tweets mentioning unemployment-related topics can be used to generate a weekly signal that closely mirrors the US government's unemployment insurance claims data [38]. Researchers have applied nowcasting to flu activity [89], mortgage refinancings [42], and more [48, 64, 146, 169].

Although nowcasting has had more research attention than wide adoption, it has the potential for massive impact. Building datasets with nowcasting is likely to be faster and less expensive than traditional surveying methods. This can be a great benefit to many fields—economics, public health, and others—that to a computer scientist appear starved for data. For example, US government economists use a relatively small number of expensive conventional economic datasets to make decisions that impact *trillions of dollars* of annual economic activity. Even a tiny improvement in policymaking—enabled by nowcasting datasets—could mean the addition of billions of dollars to the economy.

Nowcasting research to date has been driven by research into particular *target phenomena*; the software has been ad hoc rather than general-purpose. Yet, the literature [42, 48, 64, 146, 169] and our first-hand experience [38, 39] suggests that building a nowcasting model entails a standard database interaction model, the human-in-the-loop workflow:

1. Select: The user selects one or more *topics* about the phenomenon derived from social media messages. A topic consists of a descriptive label (e.g., "job loss") and a time-varying signal (e.g., daily frequency of tweets about job loss).

2. Aggregate: Using an aggregate function (e.g., averaging), the user combines the selected topics into a single *nowcasting result*. The result consists of a set of topic labels (e.g., *job loss* and *searching for jobs*) and a single time-varying signal.

3. Evaluate: The user evaluates the nowcasting result. A good result contains relevant topic labels (e.g., they are about unemployment) and a high-quality signal (e.g., it closely tracks real-world unemployment).

The nowcasting user repeats the above loop until reaching success in Step 3 or giving up. Unfortunately, the user has the difficult task in Step 1 of trying to select topics that will optimize two potentially competing criteria. As a result, the traditional approach to nowcasting topic selection—hand-choosing topics with seemingly

relevant descriptive labels—is tedious and error-prone. For example, in one of our user studies, participants employing this manual approach achieved 35% lower quality results than our automated solution and exerted much more effort in doing so (see MANUAL in Section 6.6.3). We illustrate this with the following example:

Example 6.1. *Imagine a government economist, Janet, who wants to use social media to estimate the target phenomenon of unemployment behavior. She transforms a database of social media messages into a database of social media topics. She then* ***selects*** *all topics that contain the word "fired." She uses historical ground truth data to train a statistical model that* ***aggregates*** *the topics' signals into a nowcasting result. When* ***evaluating****, Janet finds the output signal poorly tracks unemployment. She also notes that many "fired" topics actually reflect news about war-torn countries in which missiles were "fired." She chooses different topics about "benefits," but the new result also tracks unemployment poorly; she sees that the messages describe "unemployment benefits" as well as the film, "Friends With Benefits." This loop continues dozens of times until she obtains a satisfactory result.*

A nowcasting software system might follow the naïve strategy of ranking all possible topic candidates by both signal and label relevance to Janet's target phenomenon. Such a system would still be burdensome to use. Consider:

Example 6.2. *Janet visits the hypothetical nowcasting system, enters a query for "unemployment" and provides historical data for the US monthly unemployment rate. The system scores and ranks all topics by correlation with the ground truth signal and their labels' semantic relevance to "unemployment." The system then* ***selects*** *the top-scoring topics and* ***aggregates*** *them together into a nowcasting result. Janet has successfully avoided much of the manual trial-and-error process in Example 1, but due to the massive number of candidates, the new system takes a considerable amount of time—anywhere from fifteen minutes to several hours*[1]*—to compute results. After* ***evaluating*** *the result, she finds that many of the topics' social media messages do not reflect on-the-ground observations, but rather discuss the government's monthly data announcement. She modifies her query to use a weekly unemployment dataset instead of the monthly one and waits again for an extended time to find she has been only partially successful in avoiding the monthly unemployment announcement. She changes her text query from "unemployment" to "job loss, hired, benefits," waits yet again for an extended period, and finally obtains a good result.*

System Requirements — While using the feature selection system in Example 2 has allowed Janet to avoid many of the tedious iterations of Example 1's ad hoc nowcaster,

[1] As seen with our baseline systems in Section 6.6.2.

some amount of iterating is unavoidable. Even with high-quality automatic feature selection, Janet will inevitably want to tweak the nowcaster to use different historical data or avoid certain social media topics, but these tweaks are now incredibly painful, as she waits for standard feature selection techniques to complete. In contrast, an ideal nowcasting system would combine high-quality feature selection with interactive-speed runtimes, so Janet can quickly make a small number of effective revisions. (Our target usage model is that of web search, where users can iteratively improve queries very rapidly, so even fifteen minutes can seem like an eternity when having to repeatedly refine ineffective queries.)

In this chapter, we propose a framework for a *declarative nowcasting data management system*. The user expresses a desired target phenomenon, and in interactive time, the system automatically selects social media topics to satisfy the competing statistical and semantic relevance criteria. The user can now enjoy the benefits of automatic feature selection (fewer iterations) with the advantages of fast execution (quick iterations when strictly necessary). We are unaware of any existing data systems that can do this. Our initial prototype of this framework is called RACCOONDB.

Technical Challenge — Our automated method for predicting phenomena can be viewed as a form of multi-criteria feature selection from the set of all distinct social media topics—a truly massive set (e.g., we have 150 million topics in our experiments). Previous work in feature selection [176] has integrated selection with model construction, but existing approaches require several seconds to process a few hundred features—too slow for our intended application. Another area of related work is multi-criteria ranking [82, 104], but existing systems assume candidate scores are already provided or inexpensive to compute; in contrast, obtaining our scores is costly. Processing nowcasting queries requires a different approach that can choose the best topics from hundreds of millions of candidates in interactive time.

Our Approach — We expand upon the work of feature selection and multi-criteria ranking to handle the scale and speed required by nowcasting. We apply *candidate pruning* methods that exploit both signal and semantic information, along with several *low-level optimizations*, to achieve interactive-speed query processing. We also employ *query expansion* principles to find relevant results that user queries do not explicitly declare. Additionally, *user query log statistics* help identify when a query cannot be answered effectively.

Contributions — Our contributions are as follows:

- We define a novel query model for declaratively specifying a nowcasting model,

which allows users to estimate real-world phenomena with little effort (Section 6.2).

- We propose a novel framework for generating nowcasting results that uses both semantic and signal information from social media topics to generate high-quality nowcasting results (Section 6.3).
- We describe a novel set of query optimizations that enable query processing in interactive time (Section 6.4).
- We propose a novel method for detecting low-quality nowcasting queries—even for targets lacking historical ground truth data—alerting users when results may be unreliable (Section 6.5).
- We built a prototype system, RaccoonDB, and evaluated it using 40 billion tweets collected over five years. We show that it is 57% more accurate than manual approaches in our user study and that its optimizations yield a 424x speedup. Compared to a 300-core Spark cluster, it processes queries on average 123x faster using only 10% of the computational resources (Section 6.6).

We note that this chapter builds upon our past nowcasting work. Our early work in economics [38] was successful at nowcasting US unemployment behavior, but building that nowcasting model entailed the tedious process of manual searching. In later computer science work [39], we showed the potential of using topic label semantics to select relevant topics for nowcasting; however, that system had two crucial flaws: it did not use any signal information to find relevant topics—excluding an important dimension for determining topic relevancy—and it took hours to process a single query. In contrast, this chapter's contributions enable finding more relevant topics and generating higher-quality results, all within interactive runtimes. In a short demonstration paper [37], we described the user interaction of RaccoonDB, which includes a web-based interface for users to easily create nowcasting queries and explore results in interactive time.

6.2 Problem Statement

We will now formally define the nowcasting data management problem. Table 6.1 summarizes our notation.

Nowcasting — The goal of a nowcasting system is to use social media data to produce a time-varying signal that describes a real-life, time-varying phenomenon. As the name implies, nowcasting systems estimate a *current* quantity based on *current* social media data. They do not attempt to predict future events or one-off events such

as riots. Nowcasting is possible and reasonable in cases where a target phenomenon—such as unemployment, or going to the movies—also yields discussion in social media. For phenomena with little or no discussion, nowcasting will not be applicable.

Social Media Data — A nowcasting system operates on a *social media database* comprised of social media messages.

Definition 6.3 (Social media database)**.** A social media database is a set $\mathcal{M}$, where each $m \in \mathcal{M}$ is a tuple $m = (msg, tstamp)$ containing user-created textual content (msg) and a timestamp of the content creation date ($tstamp$).

From $\mathcal{M}$, a set of topics can be extracted into a *topic database* to represent different trends on social media.

Definition 6.4 (Topic database)**.** A topic database is a set $\mathcal{T}$ where each topic $t \in \mathcal{T}$ is a tuple $t = (l, s)$ such that:

1. Topic t is associated with a set of messages $M_t \subset \mathcal{M}$.
2. Label l is a textual description of the messages in M_t.
3. Time-varying signal $s = \{s_1, \ldots, s_c\}$ is defined by operator $S : M_t \to s$, where $s_i = (time_i, y_i)$. Value y_i is the number of messages in M_t occurring in time period $time_i$, which is in a chosen domain (weekly, etc.).

As an example, a topic with label "Christmas" may have a set of messages discussing the holiday and a signal s that spikes in December of each year. To extract topics from $\mathcal{M}$, several nowcasting projects have used simple string containment in the messages [38, 39, 42, 64, 89] (e.g., a single topic might be all the messages that contain, "I lost my job"). This approach has the advantage of supplying an easy-to-understand label for each topic. Others have used sentiment analysis [48] (e.g., a single topic might be all the messages that connote happiness). Similarly, methods like topic modeling [100] have shown success in topic extraction.

User Queries — In order to find relevant topics for a target phenomenon, users describe their target with a *user query*.

Definition 6.5 (User query)**.** A user query is a tuple $\mathcal{Q} = (q, r)$ such that:

1. Query string q describes the target phenomenon.
2. Time-varying signal $r = \{r_1, \ldots, r_c\}$ describes the target's historic trend, where $r_i = (time_i, y_i)$. Value $time_i$ is in the same domain as $\mathcal{T}$, and y_i is a real number.

Query string q is distinct from the topics above; a single q = "unemployment" could describe topics related to *losing a job*, *needing a job*, and so on. Users have

Notation	Description
$\mathcal{M}$	Social media database
$\mathcal{T}$	Topic database
$\mathcal{Q} = (q, r)$	User query with query string q and signal r
$O = (A, o)$	Nowcasting result with label set A and signal o
$\mathcal{R}$	Topics in $\mathcal{T}$ used to create output O
k	Target size of $\mathcal{R}$
β	User preference for signal vs. semantic weighting

Table 6.1: Frequently used notations.

two options for declaring query signal r. In the *standard query model*, r would be historical data of the target phenomenon, such as government survey or administrative data. For some phenomena, this ground truth data is not available; however, a popular practice in machine learning is to use synthetic or incomplete data instead of a traditional dataset; these *distantly supervised learners* are often able to achieve high quality results even when no good traditional data is available [127]. This motivated our *distant supervision query model*, where r would be a partial signal that describes the user's domain knowledge. For example, our economist knows that unemployment always spikes when Christmas seasonal jobs end, so she creates a signal r that shows spikes at the end of each year. Under this model, RaccoonDB processes queries as it does in the standard model, but it also provides a query quality alarm to help users identify low-quality queries (further discussed in Section 6.5).

Nowcasting Results — For a given user query Q, RaccoonDB creates $\mathcal{R}$, a set of k topics selected from $\mathcal{T}$, which are then used to generate a *nowcasting result* that estimates the target phenomenon using social media.

Definition 6.6 (Nowcasting result)**.** A nowcasting result is a tuple $\mathcal{O} = (A, o)$ such that:

1. Topic label set A is the set of topic labels in $\mathcal{R}$.
2. Output signal $o = \{o_1, \ldots, o_c\}$ is a time-varying signal defined by a given aggregate function $f : \mathcal{R} \to o$, where $o_i = (time_i, y_i)$. Value y_i is an estimate of the target in period $time_i$, which is in the same time domain as $\mathcal{T}$.

Several choices for the aggregation function f are found in past nowcasting projects, such as simply combining the signals [42, 89] or choosing the most important factor(s) from a principal components calculation [38, 39]. The size of k will vary for different aggregation functions, but it is generally on the order of tens or hundreds.

Evaluating Success — The quality of nowcasting result $\mathcal{O}$ is measured using two criteria. First, *semantic relevance* measures the relatedness of output topic label set

A to query string q. It is unacceptable to build a nowcasting result using a topic that is semantically unrelated to the researcher's goal, even if such a topic has been highly correlated with the target phenomenon in the past. For example, in certain time periods, the topic *pumpkin muffins* will likely yield a time series that is correlated with *flu activity*, because both spike as the weather cools. However, it is highly unlikely the two are causally linked: pumpkin muffin activity may fluctuate because of harvest success or changes in consumer tastes, whereas flu activity may fluctuate because of vaccine efficacy or prevalence of certain viruses.[2] We will evaluate output label set A using human annotators.

The second criterion is *signal correlation*, which measures the similarity between output signal o and query signal r. For both our query models, r represents (either directly or indirectly) the target phenomenon's behavior over time, and o should track this closely. We use Pearson correlation to measure this, a popular measure of signal similarity.

In nowcasting practice, the relative weight of these two criteria—call them *semScore* and *sigScore*—is quite unclear, owing to the informal evaluation process for output label set A. RaccoonDB will allow the user to control the relative importance of these criteria with a parameter β ($\beta > 1.0$ favors semantic scoring, while $\beta < 1.0$ favors signal scoring), which we use to compute *harmonic mean scores* using a formula we borrow from an evaluation metric used in information retrieval, *generalized F-score* [145]. It is defined as:

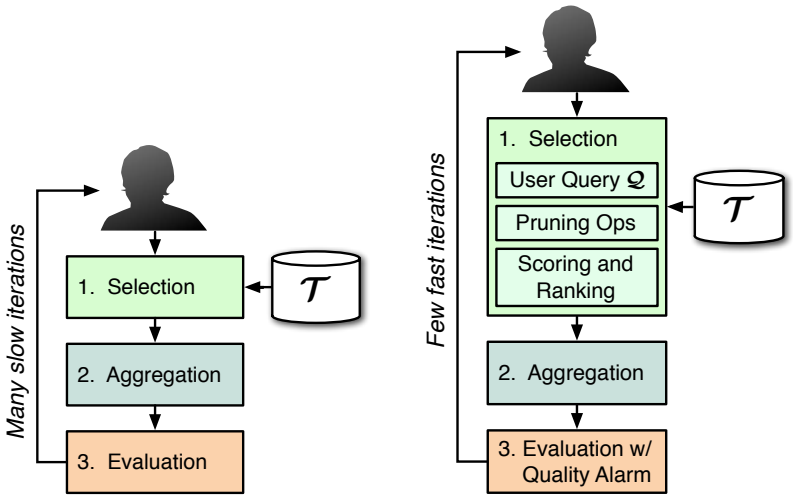
$$F_{\boldsymbol{\beta}} = (1 + \beta^2) \cdot \frac{sigScore \cdot semScore}{(\beta^2 \cdot sigScore) + semScore} \tag{6.1}$$

We can now formally state the nowcasting data management problem we are solving with RaccoonDB:

Problem 6.7. *Given user query $\mathcal{Q}$ and weighting preference β, find a nowcasting result $\mathcal{O}$ that maximizes $F_{\boldsymbol{\beta}}$ over all choices of $\mathcal{R} \subset \mathcal{T}$.*

Further, because of the human-in-the-loop nature of building nowcasting models (Section 6.1), we want to find a result quickly enough to allow the user to interactively refine her query over many iterations (i.e., within several seconds).

[2]Some seemingly unrelated topics may actually be quite indicative (e.g., maybe pumpkin muffins suppress the immune system), but this sort of *hypothesis generation* is different from our chapter's focus, which is on *model generation*.



(a) Traditional (b) RaccoonDB

Figure 6.1: A comparison of the traditional approach to nowcasting and the RaccoonDB architecture.

6.3 System Architecture

In the traditional approach to nowcasting (Figure 6.1a), users spend many slow iterations manually searching for topics and evaluating their effectiveness (or manually filtering out spurious topics from statistical ranking approaches). In contrast, RaccoonDB introduces several novel components that allow for fewer, yet faster iterations and more expressive querying, as shown in Figure 6.1b.

First, RaccoonDB uses the two-part query $\mathcal{Q}$ to find topics in $\mathcal{T}$ that match well in both the semantic and signal domains (*Scoring and Ranking*). Additionally, *Pruning Optimizations* are used to avoid scoring and ranking irrelevant topics (further described in Section 6.4). RaccoonDB performs the same aggregation as the traditional workflow, and then, before showing the results to the user for evaluation, RaccoonDB uses a *Quality Alarm* to warn users of potentially low-quality queries (further discussed in Section 6.5).

6.3.1 Scoring and Ranking

The set of social-media-derived topics $\mathcal{T}$ contains a large number of topics (roughly 150 million in our experiments), which in principle need to be scored for signal and semantic relevance, then sorted, for *each novel nowcasting query*. Performing this step is the primary bottleneck in obtaining interactive query times in the naïve implementation of RaccoonDB (Algorithm 1), where similarity scores are generated in two parts for all $t \in \mathcal{T}$ and the user's query $\mathcal{Q}$.

Algorithm 1 Naïve Query Processing

Input: $\mathcal{T}$, $\mathcal{Q}$, k, β

1: *semScores* = computeSemScores($\mathcal{T}$, $\mathcal{Q}$)
2: *sigScores* = computeSigScores($\mathcal{T}$, $\mathcal{Q}$)
3: $\mathcal{R}$ = getTopFScores(*semScores*, *sigScores*, β, k)
4: *factors* = PCA($\mathcal{R}$)
5: (A, o) = factorCombiner(*factors*, $\mathcal{Q}$)
6: *isLowQuality* = queryQualityAlarm(A, o)
7: return (A, o, *isLowQuality*)

Algorithm 2 Thesaurus Semantic Scoring

Input: q, l

1: *toks1* = tokenizeAndStem(q)
2: *toks2* = tokenizeAndStem(l)
3: *bests1* = [], *bests2* = []
4: **for all** t1 in toks1 **do**
5: *scores1* = []
6: **for all** t2 in toks2 **do**
7: *scores1*.append(jaccardDistance(*t1*, *t2*))
8: **end for**
9: *bests1*.append(min(*scores1*))
10: **end for**
11: **for all** t2 in toks2 **do**
12: *scores2* = []
13: **for all** t1 in toks1 **do**
14: *scores2*.append(jaccardDistance(*t1*, *t2*))
15: **end for**
16: *bests2*.append(min(*scores2*))
17: **end for**
18: return 1 - (average(*bests1*) + average(*bests2*))/2

Semantic Scoring — On line 1 of Algorithm 1, RaccoonDB calculates a semantic similarity score between each topic's label and the user query string q. Several existing methods can be used for this, including pointwise mutual information [67] and thesaurus- or lexical-based metrics [125, 158]. We chose a variation of thesaurus-based similarity for its simplicity and performance, as well as its shared spirit with query expansion principles, allowing topics that share no immediate tokens with a user query to still be relevant. Algorithm 2 describes how this is computed. On lines 1–2, the topic label l and the query string q are tokenized and stemmed with Porter stemming. Then on lines 4–18, for each pair of topic and query tokens, we compute the Jaccard distance between the tokens' thesaurus sets; all such distances are then combined using a method from Tsatsaronis, et al. [158], where an average is produced

across them.

Signal Scoring — On line 2 of Algorithm 1, a signal similarity score is calculated between each topic's signal and the user's query signal r. Several existing methods can be used for this, including correlation-based metrics (Pearson, Spearman, etc.) and mean squared error. We use Pearson correlation due to its simplicity and popularity, as well as its statistical properties that we can exploit in our query processing optimizations (Section 6.4.1.1). If the user provides an incomplete query signal r (i.e., our distant supervision query model), we only compute Pearson correlation for the timespans explicitly provided.

Harmonic Mean Scoring and Ranking — On line 3 of Algorithm 1, RaccoonDB calculates a harmonic mean score (F_β) for each topic in $\mathcal{T}$ using Equation 6.1 and the user-provided weighting preference β. Finally, the topics are sorted by their harmonic mean scores, and the top-k topics are selected as $\mathcal{R}$.

Time Complexity — For the naïve implementation of RaccoonDB (Algorithm 1), if topic signals have c observations and topic labels have on average u tokens, then for the n topics in $\mathcal{T}$, RaccoonDB scoring has $O((c+u)n)$ worst-case time. Since c and u are relatively small and slow-growing, this complexity can be simplified to $O(n)$. Ranking the topics requires $O(n \log(n))$ sorting, so the non-optimized system's complexity becomes $O(n + n \log(n))$, which can be simplified to $O(n \log(n))$ worst-case time. In Section 6.4.4, we discuss the time complexity of our optimized system, which drastically reduces the number of items scored and ranked.

6.3.2 Aggregation

A core part of nowcasting projects is aggregating the topics in $\mathcal{R}$ into the output signal o. Past systems have used a variety of techniques, including summing signals together [42, 89], dimensionality reduction [38, 39], or simply setting $k = 1$ (thus, no aggregation). This module is not a core contribution of RaccoonDB but is a pluggable component chosen by the domain experts using the system. By default, we use a form of principal component analysis (PCA) aggregation with $k = 100$, based on our past economic work [38].

We first perform dimensionality reduction on the topic signals in $\mathcal{R}$ using PCA (line 4 of Algorithm 1). We use this output to generate k factors that represent weighted combinations of the signals in $\mathcal{R}$. We then choose a subset of them using a heuristic based on a *scree plot* approach to choosing principal components. In traditional scree plot usage, a researcher plots each component's captured variance

and eigenvalue and then looks for an "elbow" in this plot, defining a break between *steep* and *not steep* portions of the plot. We mimic this process by looking for the first set of neighboring points where the delta between them drops below a fixed threshold that we tuned across a workload of queries. Finally, we use this subset as input to a linear regression task, in which the user's signal r is the training data. To prevent overfitting, the system always chooses a subset size so that it is smaller than the number of points in r. The output of this regression is the query output o. This aggregation has a negligible effect on the time complexity of RaccoonDB since the input of our aggregation has only k items and $k \ll |\mathcal{T}|$, the input size of our scoring and ranking procedures, which dominates the overall time complexity.

6.4 Query Optimizations

In order to process queries in interactive time, RaccoonDB automatically balances several optimizations according to the characteristics of the user query $\mathcal{Q} = (q, r)$ and weighting preference β. RaccoonDB uses two types of candidate pruning to avoid fully scoring all items in $\mathcal{T}$, the first using signal information and the second using semantics. The parameters of these pruning methods are adjusted for each individual query. Further, shared computation methods like SIMD instructions and parallelization, which are applicable to all user queries, allow RaccoonDB to further speed up candidate scoring. We discuss each of these below.

6.4.1 Candidate Pruning

We will now discuss how RaccoonDB performs confidence interval signal pruning and semantic pruning.

6.4.1.1 Confidence Interval Signal Pruning

To address the cost of calculating hundreds of millions of signal similarity scores for a given query, we use a form of top-k confidence interval pruning to eliminate candidates without calculating their exact score, an approach used when answering top-k queries over probabilistic data [143] and for recommending data visualizations over large datasets [160].

The core idea is to first quickly compute an approximate score using a *lower resolution* version of the data. A set of approximate scores, even with large error bounds, is often sufficient to disqualify many low-scoring candidates from further

Algorithm 3 Confidence Interval Signal Pruning

Input: $\mathcal{T}$, $\mathcal{Q}$, β, *semScores*, k, δ

1: $aggLB = []$, $aggUB = []$, $resolution = 0$, $origSize = |\mathcal{T}|$
2: **repeat**
3: $resolution = \text{getNextResolution}(resolution, k, |\mathcal{T}|)$
4: $r_{\text{lowres}} = \text{sample}(\mathcal{Q}.r, resolution)$
5: **for all** $i, (l, s) \in \mathcal{T}$ **do**
6: $s_{\text{lowres}} = \text{sample}(\mathcal{Q}.r, resolution)$
7: $scoreLB, scoreUB = \text{signalSimCI}(r_{\text{lowres}}, s_{\text{lowres}})$
8: $aggLB[i] = \text{getTopFScores}(scoreLB, semScores[i], \beta, k)$
9: $aggUB[i] = \text{getTopFScores}(scoreUB, semScores[i], \beta, k)$
10: **end for**
11: $thresLB = \text{getLowerBoundThres}(aggLB, k)$
12: $\mathcal{T}' = []$
13: **for all** $i, sim \in aggUB$ **do**
14: **if** $sim \geq thresLB$ **then**
15: $\mathcal{T}'.\text{append}(\mathcal{T}[i])$
16: **end if**
17: **end for**
18: $\mathcal{T} = \mathcal{T}'$
19: **until** $|\mathcal{T}'| \ / \ origSize < \delta$
20: return $\mathcal{T}'$

consideration. For the few potentially high-scoring candidates that remain, the system uses additional data to reduce the error bounds until it obtains the top-k topics.

Algorithm 3 details our pruning method. The input includes the topic database $\mathcal{T}$; user query $\mathcal{Q}$ and weighting preference β; the *semScores* for each topic in $\mathcal{T}$; the target size of $\mathcal{R}$ (k); and the desired pruning ratio (δ). On lines 3–6, the system creates low-resolution versions of the query signal r and each topic in $\mathcal{T}$ using a *resolution* value chosen by examining the size of set $\mathcal{T}$, k, and the previous *resolution* value (further discussed in Section 6.4.2). Then, the system calculates approximate signal similarity scores with 95% confidence intervals for each $(r_{\text{lowres}}, s_{\text{lowres}})$ pair. Using these intervals, the system finds lower and upper bounds on the harmonic mean scores (lines 7–9). The k-th highest lower bound in the harmonic mean scores becomes the pruning threshold (line 11); RACCOONDB prunes all topics with upper bounds below the threshold to produce $\mathcal{T}'$ (lines 12–17). The pruning repeats until the pruning ratio is less than δ.

6.4.1.2 Semantic Scoring Pruning

We can also use a pruning approach to avoid work during semantic scoring. We note that the size of a typical thesaurus is relatively small, and it is easy to precompute

Algorithm 4 Semantic Pruning

Input: $\mathcal{T}$, $\mathcal{Q}$, *thres*, k

1: $tokens = \text{getTokensAboveThres}(\mathcal{Q}.q, thres)$
2: $candidates = \text{getTopics}(\mathcal{T}, tokens)$
3: $\mathcal{T}' = []$
4: **for all** $(t, s) \in candidates$ **do**
5: **if** $\text{jaccardScore}(t, \mathcal{Q}.q) \geq thres$ **then**
6: $\mathcal{T}'.\text{append}((t, s))$
7: **end if**
8: **end for**
9: return $\mathcal{T}'$

Jaccard similarities for all token pairs with non-zero similarity.

We exploit this fact in our semantic pruning (Algorithm 4). Like signal pruning, semantic pruning has $\mathcal{T}$, $\mathcal{Q}$, and k as input. Additionally, it requires a threshold (*thres*) that controls the aggressiveness of the pruning. On lines 1–2, the algorithm uses a precomputed data structure to retrieve tokens that have a Jaccard similarity with any token in the query higher than *thres*; topics containing any of these tokens are then retrieved. Next, the full Jaccard score is calculated for the retrieved topics, and any topics with a Jaccard score below *thres* are pruned (lines 3–8). Controlling the value of *thres* over repeated calls is a crucial part of the algorithm and is described below in Section 6.4.2.

One guarantee we get from our thesaurus scoring is that our semantic pruning will always return *all* of the topics with a score greater than or equal to our threshold. As Algorithm 2 describes, the final similarity score is essentially the average across all token similarities. Therefore, if a topic has a score greater than or equal to *thres*, it must have a token with a Jaccard score greater than or equal to *thres*. Otherwise, the average of the Jaccard scores would be below *thres*.

6.4.2 Dynamic Query Optimizations

Algorithm 5 shows our full algorithm for optimized nowcasting query processing, which combines signal and semantic pruning, and dynamically adjusts pruning parameters based on $\mathcal{Q}$ and β. The core idea is that each pass through the main loop of the algorithm (lines 3–12) uses incremental amounts of semantic and signal information to eliminate candidates for $\mathcal{R}$, until there are only k candidates remaining.

For many queries, RaccoonDB could simply perform the pruning in Algorithms 3 and 4 with static parameters (e.g., $thres = 0.5$). Other queries, though, benefit from different parameter settings. For example, queries that skew topic semantic scores to

high values should use a higher threshold to avoid excess work in semantic pruning.

When first processing a query, RaccoonDB decides an initial value for *thres* (lines 1–2): given the user's query string q, RaccoonDB creates a histogram of Jaccard scores to find a value of *thres* such that the number of candidate items is greater than or equal to N, which is chosen based on past query processing statistics. If N is too large, many items will be unnecessarily processed due to inadequate pruning; if it is too small, too many items will be pruned and extra rounds of query processing will be needed. As hinted at above, the histogram can vary greatly for each query, and one that has a distribution skewed towards higher or lower values will respectively benefit from lower and higher *thres* values.

After the first round of semantic pruning and scoring (lines 4–5), RaccoonDB applies signal pruning and scoring (lines 6–7). In this phase, since each iteration has relatively expensive sampling and data copying costs, RaccoonDB attempts to minimize the number of iterations needed to meet the stopping criteria on line 19 of Algorithm 3 by dynamically adjusting the *resolution* value. A query signal that correlates well with many topics benefits from a higher *resolution*.

Once scoring is complete (lines 4–7), RaccoonDB uses the k-th best item's score to find the minimum semantic score that satisfies Equation 6.1 with a perfect signal score of 1.0. This becomes our minimum Jaccard score (*minJacScore*), which determines a new Jaccard threshold (lines 9–11). This Jaccard threshold is guaranteed to be monotonically increasing since at each round of pruning, only higher scoring items are included (as described in Section 6.4.1.2). Finally, after one or more rounds of pruning occur, the rest of the pipeline continues as in the unoptimized system (lines 13–16).

6.4.3 Low-Level Optimizations

There are several low-level database system optimizations available to RaccoonDB for all user queries. First, we can represent signal, semantic, and harmonic mean scoring (lines 4–8 of Algorithm 5) as vector operations, enabling us to use SIMD instructions. Second, by using a compact integer encoding for token labels, we can speed up lookups on lines 1–2 of Algorithm 4. Third, by partitioning $\mathcal{T}$, we can run lines 1–8 of Algorithm 5 in parallel on multiple CPUs.

Algorithm 5 Query Processing with Optimizations

Input: $\mathcal{T}$, $\mathcal{Q}$, β, k, δ

1: $minJacScore = +\infty$
2: $thres = \text{getNewThres}(\mathcal{Q}, \beta, minJacScore)$
3: **while** $thres < minJacScore$ **do**
4: $\quad \mathcal{T}' = \text{semanticPruning}(\mathcal{T}, \mathcal{Q}, thres, k)$
5: $\quad semScores = \text{computeSemScores}(\mathcal{T}', \mathcal{Q})$
6: $\quad \mathcal{T}'' = \text{signalPruning}(\mathcal{T}', \mathcal{Q}, semScores, \beta, k, \delta)$
7: $\quad sigScores = \text{computeSigScores}(\mathcal{T}'', \mathcal{Q})$
8: $\quad \mathcal{R} = \text{getTopFScores}(semScores, sigScores, \beta, k)$
9: $\quad kthBest = \mathcal{R}[k].score$
10: $\quad minJacScore = (kthBest * \beta^2)/(1 + \beta^2 - kthBest)$
11: $\quad thres = \text{getNewThres}(\mathcal{Q}, \beta, minJacScore)$
12: **end while**
13: $factors = \text{PCA}(\mathcal{R})$
14: $(A, o) = \text{factorCombiner}(factors, \mathcal{Q})$
15: $isLowQuality = \text{queryQualityAlarm}(\mathcal{Q}, A, o)$
16: return $(A, o, isLowQuality)$

6.4.4 Time Complexity

As discussed in Section 6.3.1, our naïve query processing runs in $O(n \log(n))$ worst-case time. However, our pruning methods reduce the amount of candidate topics that are sorted from n to p. With the $O(n)$ scoring pass over the data, our optimized system runs in $O(n + p \log(p))$ time. In most reasonable cases, $p \ll n$, so RaccoonDB processes queries in $O(n)$ time (i.e., the scoring time). In rare cases where pruning is ineffective (that is, p is not significantly smaller than n), the worst case running time is $O(p \log(p))$.

6.5 Detecting Poor User Queries

As discussed in Section 6.2, RaccoonDB supports two query models. In the standard model, the user's query signal r is historical ground truth data about her target phenomenon, while in the distant supervision query model, r is whatever domain knowledge she has about her target's trend. In the latter query model, even if the user is a domain expert, it may be easy for her to unintentionally provide a low-quality version of r. Knowing that unemployment claims, for example, peak after the winter holidays allows the user to define a signal peak in January, but the amplitude and exact location on the timeline may be far from the truth.

The distant supervision query model can provide a user with valuable nowcasting results that have been heretofore impossible, but when the user's query signal r is unknowingly wrong, the nowcasting result may also be wrong without the user realizing it. In this section, we propose a method of detecting when r is likely to be far from the ground truth when *no ground truth is available for comparison*, so the user can reevaluate her query before accepting a result.

Query Consistency — When a user's query contains no conventional ground truth data, our system cannot quantitatively evaluate how well its estimate of the target phenomenon matches reality. We can, however, evaluate properties of the user's query to judge the likelihood that the query will produce a high-quality estimate. More specifically, the query's two components (q, r) must be mutually consistent; that is, topics satisfying the semantic portion of the query should be related to the topics that satisfy the signal portion, and vice versa. If the two query components are unrelated, the nowcasting output of the system will be unrelated to at least one of the query inputs, which would qualify as a low-quality nowcasting result.

The quality of the user's query string q is fairly easy for a user to judge for herself (i.e., "Does 'I need a job' indicate unemployment?" should be an easy question for most users to answer). The query signal r, conversely, is difficult to evaluate without access to ground truth data. To assist the user, RACCOONDB uses a *query quality alarm* to examine her query to see if it is likely to produce a poor-quality result.

Building a Query Quality Alarm — The query quality alarm is a domain-independent classifier that predicts whether or not a query will produce a poor nowcasting result. The classifier uses the following groups of features to characterize important properties of the data:

- *Query Consistency* — These two features, f_{sem} and f_{sig}, measure the consistency of the two query components, as described above. To do this, the system finds the top-v topics as measured by signal similarity and computes the mean semantic similarity for them, and vice versa. For top signal topics S and top semantic topics T:

$$f_{\text{sem}} = \frac{1}{v} \sum_{s \in S} semScore(s) \qquad f_{\text{sig}} = \frac{1}{v} \sum_{t \in T} sigScore(t)$$

- *Variance Explained* — By applying PCA to a set of signals, one can measure the variety among the signals by looking at the resulting eigenvalues. RACCOONDB uses this observation to generate two features: the variance explained (i.e., eigenvalue) by the first PCA factor for the top-k topics as measured by signal similarity,

and vice versa using semantic similarity.

- *Consistency Across β Settings* — To compute another series of features that indicate query consistency, the system finds the top topics as measured by one of our similarity metrics (e.g., signal similarity) and uses Equation 6.1 to find the average harmonic mean score (F_β) for the topics. We repeat this for both of our similarity metrics and β settings of 0.25, 0.5, 1, 2, and 4.

We use a random forest classifier built using the scikit-learn toolkit [136] and trained with past queries submitted to the system, allowing the system to improve as usage continues over time.[3] We label training queries as high quality if RaccoonDB's output scores greater than a user-defined threshold (we use 0.5 in our experiments) in both semantic relevance and signal correlation and as poor quality, if not.

6.6 Experiments

In this section, we present our evaluation of our four main claims about RaccoonDB's performance and quality:

1. The optimizations in Section 6.4 allow RaccoonDB to process queries in interactive time (Section 6.6.2).
2. The standard query model helps users with historical ground truth data generate *relevant* and *accurate* estimates of various real-world phenomena (Section 6.6.3).
3. The distant supervision query model helps users *without any ground truth data* generate *relevant* and *accurate* estimates of various real-world phenomena (Section 6.6.4).
4. The query quality alarm in Section 6.5 can accurately detect low-quality queries (Section 6.6.5).

6.6.1 Experimental Setting

Target Phenomena — We evaluated RaccoonDB using six publicly available datasets (Table 6.2). We chose these datasets because: (1) their availability of ground truth data; (2) their use in past nowcasting research [38, 39, 64, 89]; and (3) they describe a good mixture of both seasonal (e.g., flu) and non-seasonal phenomena (e.g., guns). Additionally, we evaluated RaccoonDB using a dozen other target

[3]When being brought online initially when no query history exists, one can "prime" the alarm with conventional datasets.

Phenomenon	Description
boxoffice	US movie box office returns [14]
flu	US reported flu cases [15]
gas	US average gasoline price per gallon [23]
guns	US gun sales [17]
temp	New York City temperature [24]
unemployment	US unemployment insurance claims [22]

Table 6.2: Target phenomena used in experiments. All targets are measured weekly, except guns and temp, which we respectively convert from monthly and daily to weekly.

phenomena: those targeted by Google Finance's Domestic Trends [18]. These latter targets are discussed in Section 6.6.4.

Social Media Data — Our experiments were run against a corpus of 40 billion tweets, collected from mid-2011 to the beginning of 2015. This represents roughly 10% of the Twitter stream. We filtered out non-English messages using a trained language classifier that shows 98% precision.

Topic Extraction — We extracted topics using a simple string containment approach as described in Section 6.2. We first applied "commodity" natural language preprocessing: removing obscure punctuation, replacing low-frequency tokens with generic type-based tokens (e.g., URLs become $<URL>$), and normalizing capitalization. We then populated $\mathcal{T}$ by extracting n-grams from each message, enumerating every consecutive sequence of four or fewer words (e.g., the message "Merry Christmas" would be associated with three topics: *merry*, *christmas*, and *merry christmas*). We then removed all topics with fewer than 150 associated messages. Instead of raw counts for the signal data, we use the counts normalized by the size of $\mathcal{M}$.

System Configuration — We ran our RaccoonDB and PostgreSQL experiments on a 32-core (2.8GHz) Opteron 6100 server with 512GB RAM, and we ran our Apache Spark experiments using Amazon c3.8xlarge EC2 instances.

6.6.2 Performance Evaluation

Summary: RaccoonDB answers nowcasting queries orders of magnitude faster than nowcasting with popular data management tools (PostgreSQL and Apache Spark). Our optimizations allow for interactive speed processing, with a **424x** speedup over a non-optimized version of RaccoonDB.

Overview — In this experiment, we measured the runtime of several nowcasting

	1 core			30 cores		
Target	PostgreSQL	Non Op	Full Op	Spark	Non Op	Full Op
boxoffice	> 6 h	909 s	6.2 s	1141 s	62.0 s	1.5 s
flu	> 6 h	935 s	1.4 s	1155 s	62.6 s	0.6 s
gas	> 6 h	1003 s	1.6 s	1157 s	65.9 s	0.5 s
guns	> 6 h	959 s	1.4 s	1173 s	64.7 s	0.8 s
temp	> 6 h	1003 s	3.5 s	1197 s	66.1 s	1.1 s
unemp	> 6 h	959 s	7.6 s	1215 s	68.2 s	2.5 s
Average	> 6 h	962 s	3.6 s	1173 s	64.9 s	1.2 s
FullOp Speedup	$> 10^4$x	424x	-	1356x	75x	-

Table 6.3: Query processing times for PostgreSQL, Apache Spark, and two versions of RaccoonDB: one without our pruning optimizations (NonOp) and another with them enabled (FullOp).

query processing systems. An ideal system would return results in interactive time so that users can quickly iterate through revisions toward their final result.

Evaluation Metrics — We evaluated the performance of RaccoonDB by measuring the query processing time for each of our target phenomena. Additionally, since our pruning optimizations eliminate candidates in part based on a 95% confidence bound—potentially removing some high-quality candidates—we also measured our optimized system's recall of $\mathcal{R}$ compared to a non-optimized version of RaccoonDB.

Baselines — We first measured how well two popular data systems, PostgreSQL and Apache Spark, can be used for declarative nowcasting. For PostgreSQL, we stored our topics in a table and iterate over them using the built-in `CORR()` function for signal scoring and our own UDF for semantic scoring. For Apache Spark, we tested two clusters of Amazon EC2 c3.8xlarge instances (each using 30 cores), the first with 1 node, the other with 10 nodes. The Spark-based system was written in Python, and it preloaded topics into memory and cached intermediate data between job stages.

Since PostgreSQL and Spark are more generalized systems, we also compared RaccoonDB against a "non-optimized" version of itself (NonOp), which lacks the optimizations discussed in Sections 6.4.1 and 6.4.2, but does include the low-level optimizations in Section 6.4.3 (vectorized operations, parallelization, etc.), allowing us to measure the impact of our core optimizations. NonOp, along with the optimized system, were written in C++, with topics and optimization-based data structures preloaded into memory.

Overall Performance Results — As Table 6.3 shows, our optimized system (Ful-

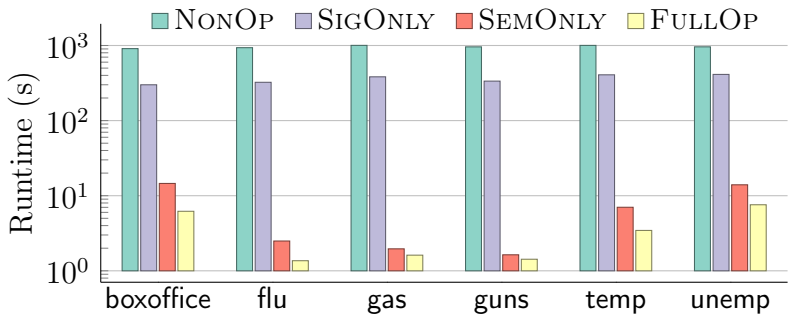


Figure 6.2: Effects of different combinations of RaccoonDB's pruning optimizations. Results shown are for $\beta = 1$ on 1 core; runtime in logarithmic scale.

lOp) averaged just 3.6 seconds per query on a single processor core, achieving interactive query processing times with very few resources. In contrast, PostgreSQL was far from efficient, not finishing even after 6 hours (at which point we ended the evaluation). RaccoonDB itself is far less efficient without its optimizations: using a single core, FullOp yielded a **424x** average speedup over NonOp, which averaged 962 seconds (about 16 minutes) per query—far from interactive time.

If we allow for parallelization, systems like Apache Spark may appear to be reasonable choices; however, when using 30 cores (on one node), Spark averaged 1173 seconds (about 20 minutes) per query. In contrast, FullOp averaged just 1.2 seconds per query using 30 cores, with a **1356x** average speedup over Spark. Increasing the Spark cluster size to 300 cores (over 10 nodes) reduced the average runtime to 106.8 seconds, but our 30-core FullOp system was still on average **123x** faster *using only 10% of the computational resources.*

To measure the direct benefit of our optimizations on RaccoonDB with parallelization, we enabled parallelization on our non-optimized system and compared it against our optimized system. Using 30 cores, the non-optimized system averaged 65 seconds per query, still too long to allow for interactive, human-in-the-loop style querying. In contrast, our optimized system's 1.2 second average runtime resulted in a **75.2x** speedup over our 30-core non-optimized system.

Impact of Pruning on Recall — Unlike RaccoonDB's semantic pruning, the confidence interval-based signal pruning is lossy; the 95% confidence interval used for pruning may cause the pruning away of high-quality topics. We can measure the impact of lossy pruning by measuring the recall of the optimized system's $\mathcal{R}$ against the non-optimized $\mathcal{R}$. In the experiments for Table 6.3, RaccoonDB achieved 100% recall with signal pruning (i.e., *never* losing a single topic).

	NonOp		FullOp	
Semantic Scoring	214.6 s	(22%)	0.97 s	(27%)
Signal Scoring	737.8 s	(77%)	1.16 s	(32%)
Rank Aggregation	9.2 s	(1%)	0.11 s	(3%)
Optimization Overhead	*n/a*	(0%)	1.37 s	(38%)
Total	961.6 s		3.61 s	

Table 6.4: Breakdown of 1-core runtime for NonOp and FullOp, averaged over all targets. Overhead includes pruning in FullOp.

Impact of Pruning on Runtime — Figure 6.2 shows the impact of our different pruning optimization methods on runtime using a single core. With no optimizations (NonOp), query processing took 962 seconds on average for our target phenomena. Using signal pruning alone (SigOnly) reduced runtime by 62.5% to 360 seconds (6 minutes) on average. Using semantic pruning alone (SemOnly) reduced NonOp's runtime by 99.3% to 7 seconds. Using both pruning techniques together further reduced the runtime to 3.6 seconds—a 48% reduction from SemOnly and a 99.6% reduction overall.

Semantic pruning was so effective because it is able to cheaply identify items that can be eliminated, removing over 99% of topics on average for our target queries. Signal pruning eliminated a similar percentage of topics, but because it requires semantic scores be computed, using signal pruning alone resulted in a much smaller computational savings.

Table 6.4 shows how the non-optimized (NonOp) and optimized (FullOp) versions of RaccoonDB spent their time (in this case, using 1 core). NonOp spent the majority of its time scoring, plus a small amount on combining the scores. For FullOp, a significant portion of its total runtime was due to pruning overhead. Of course, the absolute runtimes for the optimized system are drastically smaller.

Impact of β on Runtime — As discussed in Section 6.4, there is an interaction between β and our optimizations. In this experiment, we measured how this interaction affects runtime. Figure 6.3 shows the results of varying β for RaccoonDB (with 1 core), averaged over all nowcasting queries. When β weights signal scores more heavily, our semantic pruning had a smaller effect. In the extreme case of SignalRank, in which the *semScores* do not impact the final output at all, nothing can be pruned semantically. Fortunately, this is an unusual use case of RaccoonDB, where the goal is generally to use both query components to select topics. However, these runtimes can be improved to interactive time by increasing the number of cores.

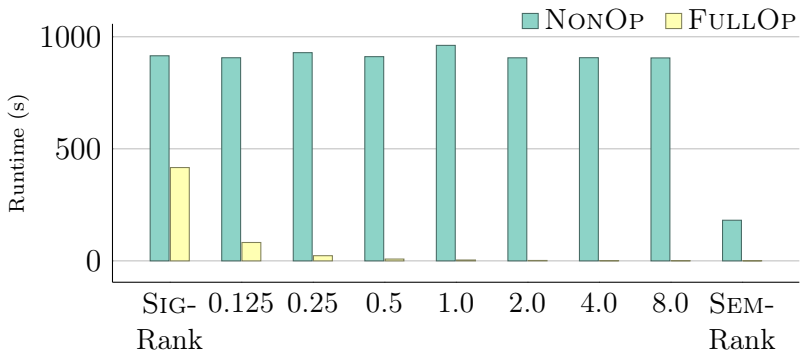


Figure 6.3: Runtimes for the non-optimized (NonOp) and optimized (FullOp) systems with different values of β on 1 core.

6.6.3 Quality of Standard Query Model

Summary: Compared to our baseline methods of users manually searching for topics or ranking them by similarity scores, results generated with RaccoonDB's standard query model require far less work from users and are of higher quality.

Overview — In this experiment, we measured the result quality from RaccoonDB and our baselines performing a traditional nowcasting task in which high-quality historical data is available to the user. An ideal result should closely estimate the target phenomenon and should be generated from relevant topics. An effective nowcasting query system should be able to beat our baselines (see below) for at least some setting of β, and ideally should do it for many settings.

Evaluation Metrics — We evaluated signal quality using a standard nowcasting evaluation model [38]: using the available social media signals at time t, we generated a statistical model to estimate a value at time $t + 1$; then, the updated social media signals at time $t + 1$ are used to build a new model and estimate for $t + 2$. This process repeats for the data's entire time period, starting one year into the dataset. Pearson correlation is then used to evaluate these forecasts against the ground truth data for the same time period.

In order to measure semantic relevance, we asked a set of human evaluators (via Amazon Mechanical Turk) to rate topic labels as *relevant* or not when applied to a given nowcasting query. Evaluators rated 10 topic labels for a single query—chosen randomly from the top-100 $\mathcal{R}$ results returned by RaccoonDB—with the majority vote from five evaluators deciding if the topic label is relevant. The semantic relevance for a nowcasting result is the normalized number of topic labels that were marked as relevant.

	Standard Query Model			Dist. Supervision Query Model		
Method	**Sig. Corr.**	**Sem. Rel.**	**SS-F_1**	**Sig. Corr.**	**Sem. Rel.**	**SS-F_1**
RaccoonDB	0.82	0.94	**0.88**	0.67	0.97	**0.79**
Manual	0.53	0.80	0.64	0.53	0.80	0.64
SemanticRank	0.35	0.94	0.51	0.35	0.94	0.51
SignalRank	0.92	0.19	0.31	0.79	0.06	0.11

Table 6.5: Nowcasting quality results for RaccoonDB's two query models and our baselines. **SS-F_1** is the harmonic mean of the signal correlation and semantic relevance scores of each method. All values range from 0–1, with 1 being best.

Baselines — We compared RaccoonDB to several baseline approaches that we created to simulate the workflow found in existing nowcasting literature. For the first baseline (Manual), we performed a user study designed to mimic the interactive string-picking procedure followed by most past nowcasting projects. We asked eleven technically sophisticated users to build a nowcasting model for our six target phenomena. We gave users a search tool that would take a topic label as input and then display the corresponding topic signal (assuming it was found in $\mathcal{T}$). After each search, we also displayed a composite signal, which our system generated by following the aggregation procedure described in Section 6.3. We allowed the users to add or remove up to fifteen topics without any time limit. We chose each user's best result for each nowcasting target and we averaged the signal relevance and semantic correlation scores of all users.

For the next two baselines, we mimicked the nowcasting workflow of ranking candidate topics by a similarity metric. The first of these (SemanticRank) is an "all semantics" method that uses only semantic-based ranking of topics from a nowcasting query string (e.g., [42]). The second (SignalRank) is an "all signal" technique that ranks topics based on their signal correlation with a nowcasting query signal (e.g., [89]). RaccoonDB can emulate SemanticRank and SignalRank by setting β to extreme values ($\beta = 0, \infty$).

Overall Results — Figure 6.4 and Table 6.5 summarize these results. On the x-axis of each plot is the semantic relevance (measured by human judges), and on the y-axis is the signal correlation (measured by Pearson correlation on held-out ground truth signal). RaccoonDB results are shown as a line to indicate the possible answers RaccoonDB provides based on different β parameter values. On these figures, we display RaccoonDB results for $\beta = \{0, 0.125, 0.25, 1, 128, \infty\}$, with the extreme settings being our baselines SemanticRank ($\beta = 0$) and SignalRank ($\beta = \infty$).

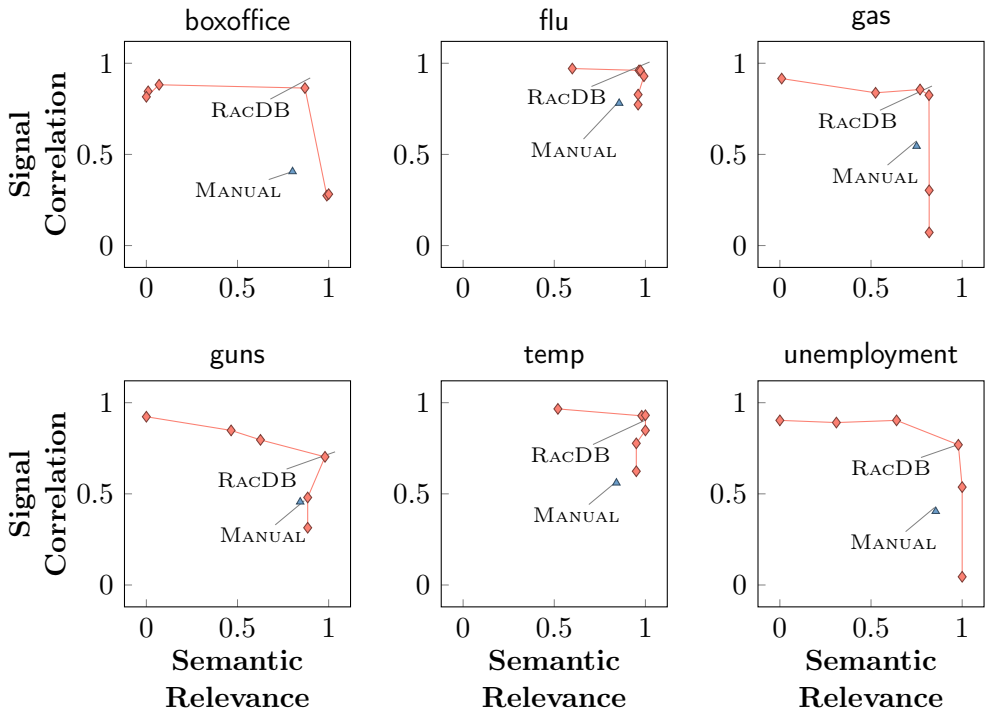


Figure 6.4: RaccoonDB (RacDB) results with different β weighting preferences (0, 0.125, 0.25, 1, 128, ∞), compared with our user study baseline (Manual). SignalRank ($\beta = 0$) and SemanticRank ($\beta = \infty$) are the end points on the RacDB line.

Our user study baseline (Manual) is shown as an individual point.

Not surprisingly, Manual and SemanticRank had great semantic relevance (on average, 0.83 and 0.94), but their signal correlation was mediocre for many targets (flu was an exception), averaging 0.53 and 0.35. In contrast, SignalRank had high signal correlation (0.92), but the semantic relevance was very low (0.19), indicating that the high signal correlation was likely due to spurious statistical correlations with the user query signal r. Conversely, RaccoonDB performed well on both signal correlation and semantic relevance (0.82 and 0.94), with results that dominate Manual and often SemanticRank and SignalRank as well. More balanced β settings (i.e., closer to 1) generally performed best.

The strongly seasonal targets like flu and temp were easier to estimate with RaccoonDB and the baselines, while other less-seasonal targets like guns were harder to estimate accurately. On average, if we set $\beta = 1$, RaccoonDB used relevant topics and correlated quite well with the target signal, correlating **57%** better than Manual and **135%** better than SemanticRank. While SignalRank correlated well with the ground truth signal, its topics had virtually no semantic relevance, so a

user would not accept its results.

6.6.4 Quality of Distant Supervision Model

Summary: RaccoonDB can reasonably estimate real-world phenomena using no ground truth data, requiring far less work and producing higher-quality results than our baseline methods. Further, with a few query revisions, RaccoonDB results can improve significantly.

Overview — As mentioned in Section 6.2, it is often possible for researchers to encode domain knowledge (*unemployment follows Christmas* or *movies are popular in summer*) in a hand-made distantly supervised signal. The obvious risk here is that providing RaccoonDB with low quality signal data will yield low-grade or misleading nowcasting results.

To test this scenario, we evaluated RaccoonDB with the same target phenomena described in Section 6.6.1, but we replaced the query signal r with a degraded replacement meant to emulate crude user domain knowledge. We transformed each r into a low-quality r' by replacing each human-marked signal peak with a synthetic parameterized peak characterized by just one of four possible amplitude values, one of four widths, and a hand-chosen maximum.

Evaluation Metrics and Baselines — We used the same evaluation metrics and baselines as in Section 6.6.3.

Overall Results — As Table 6.5 shows, RaccoonDB's signal correlation dropped from 0.82 to 0.67 when going from the standard query model to the distant supervision model, but RaccoonDB still achieved a higher signal correlation than Manual (0.52) and SemanticRank (0.35)—all methods with more than acceptable semantic relevance scores. As with the standard query model, SignalRank achieved a high signal correlation (0.79), but the terrible semantic relevance (0.06) leads us to not trust this result.

Iterative Querying Quality — In this experiment, we evaluated how RaccoonDB result quality improves when users iteratively revise their queries. We assumed that users revise queries when the result has a semantic relevance below 0.5. For example, in an effort to improve the second query for *Insurance*, we added several insurance company names (e.g., *Aetna* and *Geico*). Since users can easily identify irrelevant topics, this is more analogous to RaccoonDB's real-world usage. We tested this using 12 new target phenomena that lack ground truth data (chosen from Google Domestic Trends [18]). For each target, we generated queries with synthetic signals

Target	Manual	Sem-Rank	Sig-Rank	RacDB	IterRacDB
Air Travel	**0.62**	0.00	0.06	0.37	-
Auto Finance	0.28	0.00	0.29	0.48	**0.76**
Bankruptcy	0.45	0.00	0.47	0.25	**0.50**
Construction	0.41	0.02	0.61	0.79	-
Credit Cards	0.62	0.00	0.49	**0.73**	
Dur. Goods	0.03	0.00	0.12	**0.81**	-
Education	0.83	0.72	0.65	**0.86**	-
Furniture	0.36	0.00	0.42	**0.45**	-
Insurance	0.12	0.00	0.25	0.41	**0.69**
Mobile	**0.92**	0.35	0.86	0.88	-
Real Estate	0.04	0.08	0.19	0.51	**0.70**
Shopping	0.27	0.04	0.07	**0.92**	-
Average	0.41	0.10	0.37	0.62	**0.71**†
Std. Dev.	0.28	0.21	0.24	0.22	**0.17**†

Table 6.6: RaccoonDB result quality improvements after revising queries with poor semantics (IterRacDB). Values are the harmonic mean of semantic relevance and signal correlation. †These statistics include RacDB results if no revisions were made.

(as described above), and we measured signal correlation with the original Google Trends estimate. We used the same baselines as above, except for Manual, where we use the keywords listed by Google. Figure 6.5 shows the query and result for *Insurance* after one round of revising.

As Table 6.6 shows, RaccoonDB with only one round of querying (RacDB) generated higher-quality results than our baselines for 9 out of 12 targets, with an average harmonic mean between semantic relevance and signal correlation of 0.62. When allowing for query revisions (IterRacDB), result quality improved to 0.71, beating our baselines on 10 out of 12 targets and achieving a 73% increase on average over using the Google-chosen keywords (Manual). For consistency, these results used $\beta = 1.0$; however, some targets had higher-quality results with a different weighting preference (e.g., *Air Travel* with $\beta = 2.0$ had a harmonic mean score of 0.63). RaccoonDB allows users to explore these different results.

Additional Results — To explore RaccoonDB's capabilities in light of low-quality data, we introduced varied levels of error into our original distant supervision queries (boxoffice, etc.). RaccoonDB still did surprisingly well. Due to limited space, these results are omitted.

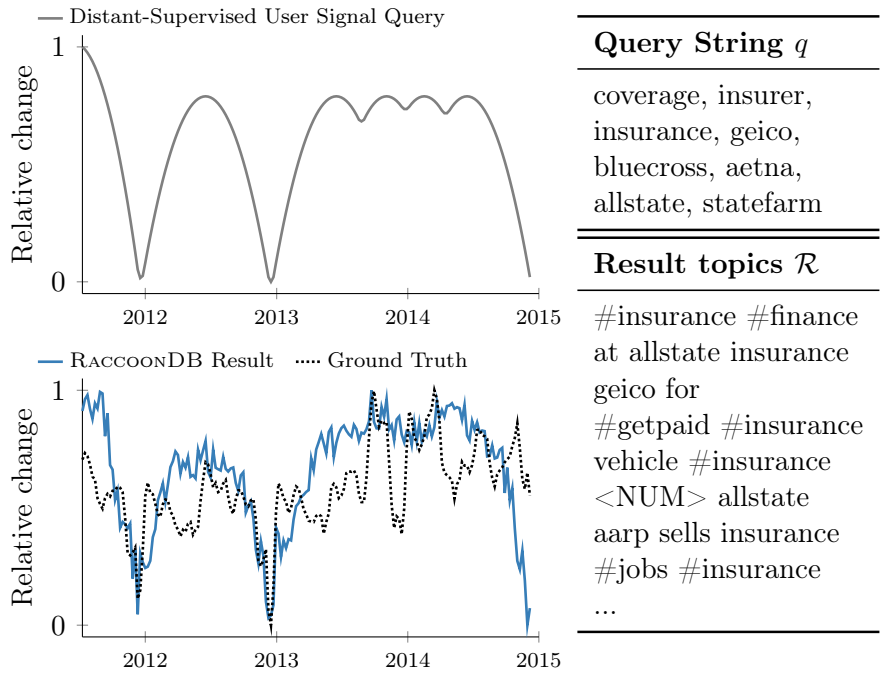
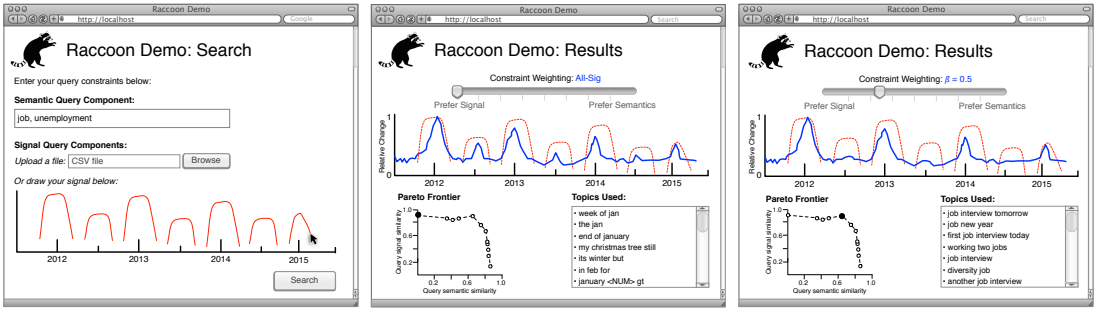


Figure 6.5: A distant supervision query and nowcasting result for the target phenomenon of *Insurance* (using IterRacDB).

6.6.5 Effectiveness of Quality Alarm

Summary: Through a user study, we show that our query quality alarm is able to identify low-quality queries, improving average result quality by 31.6%.

Overview — In this experiment, we evaluated how well our query quality alarm can identify queries that produce low-quality results. The failure mode our alarm guards against is in our distant supervision query model when the user provides a very poor query signal r without intending to, and as a result, gets a very surprising query result. To test our alarm, we collected a labeled training set from a user study, then we used leave-one-out cross validation, training on all but one target phenomenon for each round of evaluation. We presented each of our 18 study participants with each of our target phenomena and asked them to provide the query tuple (q, r); they used an online drawing tool to provide a distantly supervised version of r. To allow us to train and test our classifier on a substantial number of user queries, we synthesized new queries by taking the Cartesian product of the query strings and signals provided by our 18 users. This method allowed us to create an additional 306 synthetic queries for each of the 6 nowcasting targets.



(a) Query Page (b) SignalRank Result (c) User-Chosen Result

Figure 6.6: User interface for searching in our prototype online community.

Evaluation Metrics — We measured the average improvement of semantic relevance and signal correlation before and after filtering out alarmed results. We also measured precision and recall of our alarm's ability to identify low-quality queries. To simplify our analysis, we used $\beta = 1$ for all queries; however, it is worth noting that for some queries, a different β value achieved higher-quality results.

Overall Results — Table 6.7 shows the result of applying the query quality alarm in two different settings, with each result averaged over our six target phenomena. *None* shows the average quality across all queries. *Alarm* shows the results for those queries that did not trigger the alarm, increasing average signal correlation 30.4% (from 0.35 to 0.45).

In the second setting, *Alarm + User*, we assumed that users perform an additional step of pre-filtering results with a semantic relevance below 0.5. Since users can easily identify irrelevant topics, this is more analogous to RaccoonDB's real-world usage. The signal correlation increased to 0.43 (a 24.2% increase over not using the alarm). While this is not quite as large an increase as using the alarm alone, this method did show a significant increase in semantic relevance. This simple extended alarm had a precision of 0.74 and a recall of 0.89, with a false positive rate of 16% over all of the targets. For all targets, both the average semantic relevance and signal correlation scores showed improvement.

6.7 System Implementation

In addition to our experimental work surrounding RaccoonDB, we also implemented a prototype online community using the same underlying technologies [37]. Our prototype system lets users search for signals within a large Twitter corpus using a dynamic web-based interface. Also, users can share results with the general public,

review and comment on others' shared results, and clone these results as starting points for further exploration and querying.

6.7.1 User Interface

The user interface for our prototype system consists of two main parts. First, the user enters her query using an intuitive two-part form (Figure 6.6a). The semantic query component is entered as a comma-separated series of words or phrases in a simple text field. The signal query component can either be uploaded via a CSV file or drawn directly on a time series-type graph using an interactive JavaScript-based widget. Partial or fragmented signals can be entered here, allowing the user to encode as much (or as little) domain knowledge she possesses about the actual historical behavior of her target phenomenon. Second, after submitting the form, our system processes the query and returns the results to the user on a page similar to Figures 6.6b and 6.6c. Here the user can use interactive controls to investigate the contributions of individual topic-signal pairs to the nowcasting result.

Further, the user can explore *in real time* a tradeoff between the influence of her query's semantic and signal components via a slider widget. This tradeoff can be visualized in a "Pareto Frontier" plot (Figures 6.6b and 6.6c), where the x-axis and y-axis indicate each result's similarity with the user's semantic and signal query components, respectively. Each score generally decreases as the other's influence increases, where the "best" tradeoff can vary from query to query—and from user to user—because for any given query, a user may have more confidence in her signal query component than in her semantic query component, or vice versa. By letting the user adjust this balance, our system offers more flexibility across a range of phenomena.

For example, imagine our hypothetical economist Janet is using the system: While Janet is interactively exploring these results, she sees a nowcasting result (Figure 6.6b) that heavily favors her signal query component and that has a very high similarity score with her query signal; however, the topics used to create this signal have no relevance to her task and instead deal with the seasonal time period (e.g., "end of january"). Moving the constraint weighting slider to more heavily favor her semantic query component results in a signal generated from topics that are now very relevant to the unemployment task (e.g., "job interview tomorrow") and only cause a slight reduction of the similarity score with her signal query component (Figure 6.6c). Satisfied, she exports this result for use in her economic modeling, and then shares the result for other users to explore.

Alarm Model	Semantic Relevance	Signal Correlation	Harmonic Mean Score
None	0.57	0.35	0.43
Alarm	0.61 (5.4%)	0.45 (30.4%)	0.52 (19.6%)
Alarm + User	0.84 (46.0%)	0.43 (24.2%)	0.57 (31.6%)

Table 6.7: Average result quality improvement from excluding queries that triggered the alarm (*Alarm*) and also excluding results with low semantic relevance (*Alarm + User*).

Sharing Results — One of the goals of our system is to create an online community where users can share interesting results with the public, as well as get feedback on potentially questionable results. If a user makes an outlandish claim supported by data generated by RaccoonDB, others can review the components that went into creating the data, respond with commentary, or "clone" the result as a starting point for their own exploration. This type of analysis and reproducibility is especially important for economists, where researchers at the US Federal Reserve recently showed they were unable to reproduce over 50% of selected published economic results—and that was even with the original authors' help [61].

In addition to allowing users to review, comment, and clone shared results, RaccoonDB can keep shared results updated regularly with new social media data, thus making it easy for anyone to monitor for changing trends. We envision economists and other researchers using this community as a means for better nowcasting collaboration, discovery, and debugging.

6.8 Related Work

There are several areas of research that are related to our work, which we discuss below.

Nowcasting — Several research projects [42,64,89,146], including our own work [38], have used ad hoc social media nowcasting to estimate real-world phenomena, relying on either slow and tedious manual searching or automated statistical approaches that are prone to spurious and low-quality results. We extend this work by proposing a general-purpose declarative nowcasting query system that quickly generates high-quality results with little user effort. Our work is also similar to Google Trends [64]—where users can search for social media topics by keyword—and Google Correlate [129]—where users can rank topics by signal correlation. However, in contrast to our sys-

tem, neither take advantage of both criteria at the same time. Our work is also distinct from Socioscope [170], which assumes relevant topics have already been identified. While nowcasting has faced some criticism due to model failures [113] and the cherry-picking of results [124], we view nowcasting as a tool that can be used either effectively or maliciously—like statistics in general—and with proper domain knowledge and training, these issues can be avoided.

At the end of Section 6.1, we discuss how our current work differs from our past nowcasting work [37–39].

Feature Selection — Feature selection is a well-studied problem in the statistics and machine learning domains. Guyon and Elisseeff have a survey on the subject [94], where they discuss different classes of selection methods such as using domain knowledge, ranking methods, and dimensionality reduction techniques. In our work, we borrow ideas and methods from all three of these categories. More recently, Zhang et al. [176] introduce a system for fast interactive feature selection. We share their motivation of improving human-in-the-loop feature selection, but our semantic-based approach to scoring candidates takes this a step further by automating much of the human-driven relevance evaluation. Additionally, much of their work focuses on combining selection with model construction, but the scale of our topic database requires RaccoonDB to use a ranking-based selection process.

Query Optimization — Query optimization is a well-studied problem in database literature and has been applied to other domains, though standard RDBMS optimization techniques do not apply in our setting. Optimizations based on confidence interval pruning have been implemented in a number of works, including top-k query processing over probabilistic data [143] and in recommending visualizations for large datasets in the SeeDB project [160]. While similar to these other projects, our confidence interval signal pruning works in tandem with our novel semantic scoring pruning method, requiring dynamic parameter configuration on a query-by-query basis. Herodotou et al. [98] developed a system to identify good system parameters for MapReduce jobs, but they computed these separately from the actual job execution. RaccoonDB instead dynamically adjusts its parameters as the query is being executed.

Multi-criteria Ranking — The goal of multi-criteria ranking, or rank aggregation, is to combine multiple ranked orderings to produce a top-k list without evaluating all items. Rank-Join [104] and Fagin's algorithm [82] are two popular approaches. Our work also needs to achieve a top-k list with multiple scoring metrics; however, in our

case, the scoring of candidates is most expensive, so we have to rely on a different set of optimizations (discussed in Section 6.4). Additionally, our work hints at a relationship to skyline queries [110]; however, our massive number of topics requires that we approximate the skyline by selecting one point from it (the top topic in a result) and aggregate it with $k-1$ of its neighbors.

Query Formulation — This area of work focuses on using external information—such as query logs [108] or the data being queried [171]—to expand query results to include relevant items that user queries do not exactly describe. Our work shares a similar goal, but we can further improve results by evaluating them in their final usage in a nowcasting model.

Complex Event Processing (CEP) — CEP focuses on extracting trends from a stream of events, and could be used to identify topic-based signals; however, doing so would be extremely slow since CEP systems identify patterns across events directly (e.g., taking 30 seconds to process a simple "W" pattern match on 500K tuples [130]). Further, since they lack semantic-based topic selection, they would require the tedious manual topic selection that RaccoonDB avoids.

Sequential Pattern Mining — Note that our work is quite distinct from sequential pattern mining [126], in which the goal is to identify latent interesting events from time-stamped data. Rather, our task is to choose the most useful few sequences from a vast set of candidates.

6.9 Conclusion and Future Work

In this work, we proposed a novel query model and framework for declaratively specifying a nowcasting model and yielding high-quality results with little user effort. Our query optimizations and quality alarm aid the human-in-the-loop nature of nowcasting by allowing for fast query iteration on many phenomena—even those lacking historical ground truth data. In the future, we would like to deploy RaccoonDB. We also see an opportunity for applying its principles to building other socially driven dataset construction tools.

CHAPTER VII

Constraint-based Explanation and Repair of Data Transformations

A common task in data analysis pipelines is transforming a dataset into a new one for performing analysis on, and much effort has gone into improving this process for users (Wrangler, WebTables, etc.). Unfortunately, even the most trivial mistake in this phase can introduce bias and lead to invalid conclusions about the data. For example, consider a researcher who is identifying subjects to test a new statin drug: in an effort to find a population likely to benefit from the drug, she might select patients with a high dietary cholesterol intake; however, it could bias the test population to those with a generally unhealthy lifestyle, thereby spoiling the analysis. Even after identifying such bias, it can be time-consuming and tedious to remove while still remaining close to one's desired dataset.

In this chapter, we propose a novel user model and architecture for *explain-and-repair* data transformation systems, where users interactively define constraints for transformation code and the data that the code produces. The system satisfies these constraints as best it can, and also provides an explanation for any problems along the way. We present an algorithm that yields transformation code based on user constraints on the output data and transformation code. We implemented and evaluated a prototype of this architecture, Emeril, using both synthetic and real-world datasets. Our approach finds solutions 34% more often and 77% more quickly than the previous state-of-the-art solution.

This chapter is based on collaborative work with Michael Cafarella.

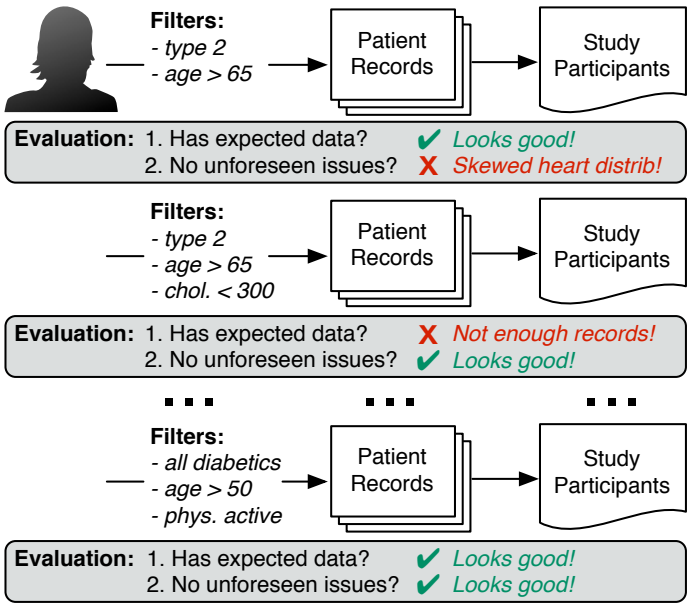


Figure 7.1: An illustration of Example 7.1, showing the time-consuming and tedious process of repairing data transformations.

7.1 Introduction

A common task in data analysis pipelines is transforming data from a raw input dataset into a new dataset for performing analysis on. For example, a medical researcher interested in testing a new statin drug might need to filter a patient database so that it contains people likely to benefit from the test drug. Such transformations may include filtering, aggregations, and data wrangling. Much effort has gone into improving this process for users via interactive interfaces [80, 107], transformations by example [93, 105, 165–167], and various other methods that aid with syntactic and semantic transformations [56, 63, 172].

Unfortunately, even the most trivial of transformations can introduce undesirable bias and corrupt conclusions drawn from the analyses. Filtering for subjects with high dietary cholesterol intake is reasonable on its face, however, it could bias the test population to those with a generally unhealthy lifestyle, thereby spoiling the analysis. While an analyst with some training in statistics would likely identify this bias, similar mistakes—whether through statistical ignorance or cognitive bias—still appear in the literature [19, 83, 131].

Even if one identifies that bias has been introduced, fixing it can be time-consuming and tedious while still remaining close to one's desired dataset. The best transformation code is one that does not introduce any undesirable bias, and is otherwise

very similar to the analyst's original plan. Consider the following example, which we illustrate in Figure 7.1:

Example 7.1. *Janet is a researcher who is identifying subjects for a study investigating a new diabetes medication. She believes the medication will be most effective for type 2 diabetics, so she* ***adds a filter for*** `(type2 = True)`*. In the past, she had problems with middle-aged participants following up, so she* ***filters for anyone with*** `(age > 65)`*. After discussing the plan with her colleague, she realizes that older people have worse cardiovascular health than is typical, thereby giving a misleading picture of her drug's effectiveness, so she* ***filters for patients with*** `(cholesterol < 300)`*. The resulting set of patients is too small for her study, so she* ***removes*** `(age > 65)`*. The new result still has issues with the distribution of cardiovascular health, so she* ***removes*** `(cholesterol < 300)` *and* **adds** `(exercises = True)`*. The result of these changes yield too few patients again, so she* ***removes*** `(type2 = True)` *and* ***adds*** `(diabetic = True)`*. The results are closer to what she wants, so she* ***re-adds*** `(age > 65)`*. Now the patient counts are again too low, so she* ***relaxes the age filter to*** `(age > 50)`*.*

Janet has now had to perform six rounds of filtering and time-consuming data analysis to obtain her desired dataset.

In an ideal environment, Janet would describe her target dataset, and an external system would find a dataset that most closely matches her goal—while helping her avoid any undesirable bias. Consider Janet using an *explain-and-repair* data transformation system:

Example 7.2. *Once again, Janet is identifying subjects for a study. She describes her ideal dataset to the system* `(type2 = True, age > 65, COUNT(subjects) > 500)`*, and the system responds with a transformed dataset and a list of columns whose distributions have changed (e.g., prevalence of cardiovascular disease is skewed). Janet clicks "Do Not Allow" for the cardiovascular disease bias warning, and the system responds with a new dataset, updated transformation code, and a new list of column distribution changes. Janet sees no undesirable bias in the new list of distribution changes and happily accepts the system's result.*

System Goals — In this chapter, we propose an explain-and-repair data transformation system. The system takes the following input:

- A dataset that needs transforming;
- User-provided *signal-based* constraints that indicate desired characteristics of the transformed dataset;

- User-provided *code-based* constraints that indicate desired characteristics of the transformation code;
- User-provided *aggregate-based* constraints that indicate desired aggregate conditions on the transformed dataset.

The system has the following desiderata:

- Finds a set of transformations and the data that the code produces, which best match the user's constraints;
- Explains potentially undesirable bias to the user, requests feedback, and uses this to create a new result;
- Responds with a result within a reasonable timeframe (i.e., several minutes).

For the current work, we assume all transformations are in the form of filters (i.e., relational selections); thus, some transformations are not possible (e.g., folding or extraction), but for many applications our filter-only assumption is fine.

Technical Challenge — Using a large pool of candidate transformations, the explain-and-repair system must find the set of transformations that best match the user's constraints. This amounts to a combinatorial search problem, but one with a costly objective function: the code must be evaluated with any code-based constraints, the transformations need applying to the input dataset, and the resulting dataset must be evaluated with any signal-based constraints.

A somewhat similar challenge is faced in *how-to* querying [122], where the goal is to find a dataset that best satisfies a set of constraints, but this goal does not include determining the transformation code to use. Extending how-to querying work in a straightforward manner so that both the resulting dataset and the transformation code are considered causes the problem to be intractable with only a small pool of candidate transformations (as detailed in Section 7.3).

Our Approach — One popular approach to combinatorial optimization is with integer programming, where much effort has gone into creating efficient methods for finding solutions [133]. We represent the problem as a constrained-optimization problem, which we solve using nonlinear programming. We preprocess the input dataset and pool of candidate transformations to determine how the different transformations would impact the data distribution of the input dataset, and then use this information in our problem. Binary variables exist for each transformation, and the objective

function is to find the set of variable values that maximize the similarity with the user's constraints.

Contributions — Our contributions are as follows:

- We define a novel user model for constraint-based explanation and repair of data transformations, helping users find data that best matches their constraints without any surprise issues (Section 7.2).
- We present an algorithm for finding transformations that best match a set of constraints. The algorithm models this as a constrained-optimization problem, which it solves using nonlinear programming (Section 7.4).
- We built a prototype system, EMERIL, and evaluated it with synthetic and real-world data, showing that our approach finds solutions 34% more often and 77% quicker than the previous state-of-the-art solution (Section 7.6).

7.2 User Model

In this section, we present a user model for explain-and-repair data transformation systems. The terminology introduced in this section is summarized in Table 7.1.

7.2.1 Overview

When formulating a ***transformation program***, an analyst has a ***raw input dataset*** that she wants to transform into a ***desired output dataset***. She has an initial set of goals for her desired output dataset, which may include:

- Which items it should contain (e.g., *type 2 diabetics*);
- The distribution of particular items (e.g., *a Gaussian-distributed prevalence of cardiovascular disease*);
- The number of data items (e.g., *at least 500 subjects*);
- Which transformations to use (e.g., *no gender filtering*).

This information, or set of ***user constraints***, drives the creation of the transformation program that yields a ***generated output dataset***. Unfortunately, finding a transformation program that satisfies all of the constraints *can be a challenge for an analyst*, so she will likely revise her goals at least once. Even after several revisions,

	Description
$\mathcal{D}$	Raw input dataset
$\mathcal{T}$	Desired output dataset
$\mathcal{C}$	Set of candidate transformations
$\mathcal{U}_i$	User constraints at query cycle i
$\mathcal{W}_i$	User preference weights for $\mathcal{U}_i$
$\mathcal{R}_i$	Generated output dataset that best matches $\mathcal{U}_i$
C_i	Transformation program that yields $\mathcal{R}_i$
$\mathcal{P}_i$	Set of system-identified problems for $\mathcal{R}_i$

Table 7.1: Notation used to describe our user model.

without careful inspection or foresight, the generated output dataset can still contain potential problems, or rather, unknowingly differ from the analyst's desired result. This problem is an example of a *data engineering loop* [33], a class of problems that includes feature engineering [35, 111, 177].

Figure 7.2 summarizes how a user can build a transformation program in conjunction with an explain-and-repair system. Janet (from Example 7.2) starts by providing her raw dataset and initial constraints to the explain-and-repair system (**Step 1**). The system responds with a generated output dataset, transformation program, and a warning that the distribution for cardiovascular disease prevalence has changed (**Step 2**). Since the system treats the constraints as *soft constraints*, they may be violated or imperfectly true in the output data. Janet responds with a constraint that indicates the cardiovascular distribution should be Gaussian (**Step 3**), and the system responds with a new transformation program and generated dataset (**Step 4**). These ***query cycles***, where the user specifies constraints and receives a result, continue until the user is happy with her result.

7.2.2 Data Transformation Primitives

Raw Input Dataset — This is the dataset that the user wants to transform into a desired output dataset, both of which we assume are relational datasets:

Definition 7.3 (Raw input dataset)**.** An input dataset is a set $\mathcal{D}$, where each $d \in \mathcal{D}$ is a tuple of values $(v_1, v_2, ..., v_k)$ with an associated tuple of column names $(c_1, c_2, ..., c_k)$.

Definition 7.4 (Desired output dataset)**.** Dataset $\mathcal{T}$ is the output dataset a user desires from applying a transformation program C to raw input dataset $\mathcal{D}$.

Transformation Program — This is the means of producing a generated output

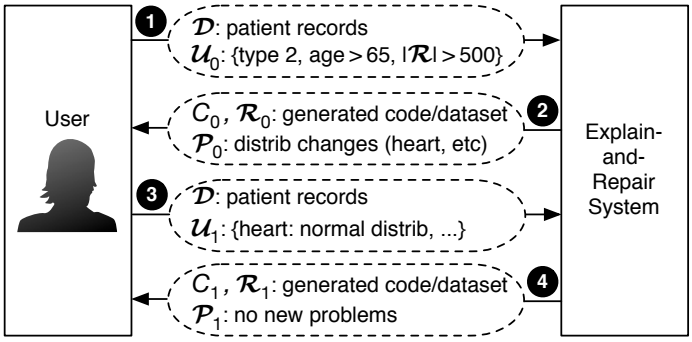


Figure 7.2: The user model for an explain-and-repair data transformation system, showing Janet's usage in Example 7.2.

dataset. We limit this chapter's focus to only filter-based transformations, or *predicates*:

Definition 7.5 (Predicate)**.** Predicate $p = (c, o, v)$ is a tuple representing a filter-based transformation such that operator o and value v form a binary test on column c in $\mathcal{D}$, indicating the included rows in generated output $\mathcal{R}$.

Definition 7.6 (Candidate Transformations)**.** $\mathcal{C}$ is the set of all potential predicates applicable to $\mathcal{D}$.

Definition 7.7 (Transformation program)**.** The set $C \subseteq \mathcal{C}$ is a set of transformations. The conjunction of C applied to $\mathcal{D}$ produces generated output dataset $\mathcal{R}$.

Generated Output Dataset — This is the result of applying a transformation program to a raw input dataset:

Definition 7.8 (Generated output dataset)**.** Dataset $\mathcal{R}$ is the generated output dataset from applying transformation program C to $\mathcal{D}$.

We can now formalize the ***data transformation problem***:

Problem 7.9. *Given raw input dataset $\mathcal{D}$ and set of candidate transformations $\mathcal{C}$, find a transformation program $C \subseteq \mathcal{C}$ that maximizes similarity$(\mathcal{R}, \mathcal{T})$.*

7.2.3 System Input

When using an explain-and-repair system, users provide an input dataset $\mathcal{D}$ and a set of user constraints.

User Constraints — This is a collection of constraints that define the user's desired output dataset at query cycle i:

Definition 7.10 (User Constraints)**.** $\mathcal{U}_i$ is a set of user constraints $\mathcal{U}_i = \{U_a, U_s, U_x, U_t\}$, such that:

1. U_a is a set of TargetDistrib constraints.
2. U_s is a set of DesiredPred constraints.
3. U_x is a set of NoPred constraints.
4. U_t is a set of TupleCount constraints.

TargetDistrib Constraints — These constraints are used to indicate a desired distribution for a column in the output dataset (e.g., Janet would specify that the *prevalence of cardiovascular disease* should have a Gaussian distribution).

Definition 7.11 (TargetDistrib constraint)**.** A TargetDistrib constraint is a tuple $u = (c, s)$ such that:

1. Column name c indicates the name of a column in $\mathcal{D}$.
2. Density estimation signal s indicates the desired distribution of column c.
3. For probability density function f, $f(\mathcal{R}_i[c]) \simeq s$.

When specifying density estimation signal s, users have two options: for continuous values, users provide a signal. In past work [36], we provided a web-based tool for "drawing" signals, so something similar could be used to assist users. For multinomial values, users provide a set of desired counts or percentages for each value (e.g., $gender = \{male : 60\%, female : 40\%\}$).

DesiredPred Constraints — These constraints are used to indicate when a predicate should be used to generate a result. For example, Janet may require that her study include only females, thus *(gender = 'female')* would be used.

Definition 7.12 (DesiredPred constraint)**.** Predicate u is a DesiredPred constraint indicating $\exists\ p \in C_i$ s.t. $p \simeq u$.

NoPred Constraints — These constraints are used to indicate when a particular column should *not* be used to generate a result. For instance, Janet may require that all levels of patient income be included in her study, so she would specify *income_level* as a NoPred constraint.

Definition 7.13 (NoPred constraint)**.** Column name u is a NoPred constraint indicating a column in $\mathcal{D}$, where $\forall p = (c, o, v) \in C_i, c \neq u$.

TupleCount Constraints — These constraints indicate a required minimum or maximum number of data items in the generated output dataset. For instance, Janet would specify that she wants a minimum of 500 study subjects.

Definition 7.14 (TupleCount constraints)**.** Tuple $u = (m, n)$ is a TupleCount constraint, where $m = \min \mid \max$ and $|\mathcal{R}_i| \leq n$ if $m = \min$, otherwise, $|\mathcal{R}_i| \geq n$.

Constraint Preference Weights — Satisfying all of the user constraints simultaneously is often not possible, and so the explain-and-repair system treats them as soft constraints. To help the system prioritize which to relax first, users can specify a set of *preference weights* for their constraints (or if not provided, a uniform weighting is assumed):

Definition 7.15 (Preference Weights)**.** Set $\mathcal{W}_i$ is a set of sets, where $W_{ij} \in \mathcal{W}_i$ defines the preference weights for one of the user constraint types in $\mathcal{U}_i$, and $\sum \mathcal{W}_i = 1$.

7.2.4 System Output and Evaluating Results

The explain-and-repair system output at a given query cycle i includes generated output dataset $\mathcal{R}_i$, transformation program C_i, and a set $\mathcal{P}_i$ of *system-identified problems*.

System-Identified Problems — For a given result, the system identifies any distribution changes amongst columns in $\mathcal{R}_i$, notifying the user that these may be potential problems. For instance, Janet is notified that the distribution of *cardiovascular disease prevalence* has changed.

Definition 7.16 (System-identified problem)**.** Tuple $p = (c, s_1, s_2)$ is a system-identified problem, where c is the name of a column in $\mathcal{D}$, and for a density estimation function f, $s_1 = f(\mathcal{D}[c])$, $s_2 = f(\mathcal{R}_i[c])$, and $s_1 \neq s_2$.

Evaluating Results — When evaluating $\mathcal{P}_i$, a user will indicate if any distribution changes are undesired by providing a TargetDistrib signal-based constraint for each problem. After iterating until all problems have been resolved between the user and the system, the user can also evaluate the transformation program C_i, and if any problems are found there, the user will provide a DesiredPred or NoPred code-based constraint for each problem. Similarly, if output dataset $\mathcal{R}_i$ has too many or too few records, the user can provide a TupleCount aggregate-based constraint.

7.3 Reverse Data Management

The problem of finding a transformation program that best matches a set of user constraints is an example of a *reverse data management* problem [121]. The most similar area of research from this domain is *how-to* querying, where the goal is to

Algorithm 6 Tiresias adaption to our problem

Input: $\mathcal{D}, \mathcal{C}, \mathcal{U}$

Result: constrained-optimization problem formulation

1: Create bin/tuple/predicate mappings:
2: $binTuples = \{b_1 = (...), ..., b_k = (...)\}$
3: $tuplePreds = \{t_1 = (...), ..., t_k = (...)\}$
4: Create binary variables to indicate tuple/predicate usage:
5: $tuples = \{0, 0, ...\}$
6: $preds = \{0, 0, ...\}$
7: Create constraints to define if tuple included:
8: $t_i = (!p_1 \vee tp[i, 1]) \wedge (!p_2 \vee tp[i, 2]) \wedge ... \wedge (!p_k \vee tp[i, k])$
9: where $t_i = tuples[i], p_k = preds[k], tp = tuplePreds$
10: Create constraints to define estimated counts:
11: $binCounts[b] = \sum_{t \in binTuples[b]} tuples[t]$
12: Define objective:
13: minimize $\sum_{b \in binIds} \text{abs}(binCounts[b] - targetCounts[b])$

modify an existing dataset to satisfy some constraints. Meliou and Suciu developed Tiresias [122], which solves general how-to queries using linear programming: Users describe with a declarative query language their data, constraints, and desired actions (modify a row value, add tuples, etc.), and Tiresias finds the best set of actions that creates the desired dataset.

Tiresias works great for how-to querying, but it does not work well for our particular problem: Both Tiresias and an explain-and-repair system aim to produce a dataset that matches a set of constraints, but an explain-and-repair system also needs to find a transformation program that produces the dataset (and matches well with any relevant constraints). In contrast, Tiresias does not need to build a transformation program. However, with a few alterations, we can use Tiresias for our problem. Consider Algorithm 6 whose inputs are a raw input dataset $\mathcal{D}$, a set of candidate transformations $\mathcal{C}$, and user constraints $\mathcal{U}$. We first create mappings between tuples, bins (bins are from a histogram on the user constraint signal), and predicates (lines 2 – 3). Then we create binary variables to indicate if a tuple or predicate is used in a solution (lines 5 – 6). Next we create constraints that define whether each tuple is used in a solution (line 8) and what the estimated counts are for each bin (line 11). Finally, we define the objective function for the constrained-optimization solver (line 13).

Unfortunately, representing just one of the logical statements in line 8 as a constraint in the optimization problem actually requires *several* constraints and variables. For n tuples and p predicates, the number of constraints and variables are each $O(np)$. While these only grow linearly with the input size, constrained-optimization solvers

can only handle a finite number of constraints and variables. For this particular approach, we found that with any more than 500 tuples and 100 predicates, the solver Tiresias uses (GLPK [116]) was unable to find a solution after three hours, at which point we terminated the program.

7.4 Finding Data Transformations

In this section, we propose a novel algorithm for finding a data transformation program that best matches a user's constraints. Unlike a Tiresias-based solution, our approach scales well with the number of tuples in a dataset.

7.4.1 Overview

The core idea of our approach is to represent our problem as a constrained-optimization problem, where we model the impact each candidate transformation has on matching the user's constraints and use a constrained-optimization solver to find the set of transformations that maximize this match.

Algorithm 7 shows the steps that are needed. The user provides raw input dataset $\mathcal{D}$, user constraints $\mathcal{U}_i$, and an optional preference weights $\mathcal{W}_i$. The first step is to analyze the input dataset to generate candidate predicates and precompute some information used in the problem formulation (line 1). This work can be done offline, as we discuss in Section 7.5. We then formulate the problem as a constrained-optimization problem (line 2), find a solution using nonlinear programming (lines 3 – 4), and analyze the result for potential problems (line 5). This is the core of our chapter's contribution, and we discuss it in detail below.

7.4.1.1 Background on Constrained Optimization

Constrained-optimization solvers take as input a set of constants, variables, and constraints, along with an objective function. The constants dictate the initial conditions, and the solver finds values for the variables that maximize the objective function, subject to the provided constraints.

7.4.2 Modeling as an Optimization Problem

To start, we will assume that predicates are independent of each other and that we are matching a single TargetDistrib constraint (e.g., Janet specifying that the distribution of *cardiovascular disease prevalence* should be Gaussian and centered

Algorithm 7 Overview of our algorithm's solution finding

Input: $\mathcal{D}$,$\mathcal{U}_i, \mathcal{W}_i$

1: $\mathcal{C}, pp$ = analyzeDataset($\mathcal{D}$)
2: *model* = formulateProblem($\mathcal{D}$,$\mathcal{U}_i, \mathcal{C}, pp$)
3: *solverResult* = runOpSolver(*model*)
4: $C_i, \mathcal{R}_i$ = processSolverResult(*solverResult*)
5: $\mathcal{P}_i$ = identifyProblems($\mathcal{R}_i$,$\mathcal{U}_i, \mathcal{W}_i$)
6: return $C_i, \mathcal{R}_i, \mathcal{P}_i$

on an average level). If two predicates are independent, then the probability of a row being included when both predicates are applied to a dataset is the same as the product of their marginal probabilities (i.e., $P(A, B) = P(A)P(B)$). Later we will relax these two assumptions.

Target Bounds — For a TargetDistrib constraint with column c and signal s, the goal is to maximize the similarity between s and the distribution of c in the generated output dataset $\mathcal{R}_i$. In order to avoid having the solver calculate probability densities for every candidate solution (or having to perform time-consuming signal comparisons), our algorithm discretizes s into *bins* to create a *target histogram*. This allows for faster distribution matching with only a small loss to accuracy.

Our algorithm then discretizes the original distribution of c in $\mathcal{D}$ as a set of percentages and finds the maximum number of data items in each bin that satisfies the target histogram. This defines the *target counts*. For example, if *cardiovascular disease prevalence* originally has a uniform distribution of {200, 200, ..., 200}, then a target histogram of {1%, 4%, 15%, 30%, 30%, 15%, 4%, 1%} would yield target counts of {7, 27, 100, 200, 200, 100, 27, 7} (maximizing the largest bins, relatively adjusting the others). Lastly, our algorithm converts the target counts into *target bounds* by creating an interval around the counts with a *slack* parameter ϵ, allowing for an ϵ margin of error when finding solutions.

Model Variables — When formulating the optimization problem, our algorithm creates a set of binary variables, *preds*, with one for each predicate $p \in \mathcal{C}$. These are used to indicate if a predicate is included in the transformation program C_i. An additional set of variables, *bc*, is created to store the estimated bin counts from applying a candidate solution to $\mathcal{D}$. The constants defined by our algorithm include the precomputed bin probabilities for each predicate (see Section 7.5), the target counts, the target bounds, and the probabilities for each bin in the target histogram.

Model Constraints — Our algorithm creates a constraint to define the estimated bin counts from applying a candidate solution on $\mathcal{D}$. The intuition of this is that the probability of a row being included by a set of predicates is the probability of

the bin multiplied by the product of the predicates conditioned on the bin. Thus, the estimated count for a particular bin is the product of this joint probability and the size of the dataset. This probabilistic approach allows the solver to estimate the result of applying a set of predicates without having to decide if particular tuples are included in a resulting dataset (as a Tiresias-based approach would).

Equation 7.1 displays how the count is calculated for bin b, where $bc[b]$ is the estimated bin count, $bp[b]$ is the bin probability (or percentage of $\mathcal{D}$ in bin b), $bpp[i, b]$ is the bin probability for predicate variable $preds[i]$, and c is a small constant (e.g., 1×10^{-6}) used to avoid a log of zero:

$$bc[b] = |\mathcal{D}| * bp[b] * \exp(\sum_{i=1}^{|preds|} preds[i] * \log(bpp[i, b] + c)) \tag{7.1}$$

Our algorithm also adds constraints that limit the estimated counts to within the target bounds (Equation 7.2) and ensure that at least one predicate is chosen (Equation 7.3):

$$lower\_bounds[b] \leq bc[b] \leq upper\_bounds[b] \tag{7.2}$$

$$\sum_{i=1}^{|preds|} preds[i] \geq 1.0 \tag{7.3}$$

Objective Function — Using the variables and constraints defined above, the constrained-optimization solver finds the set of predicates that minimize the sum of differences between the target and estimated bin counts:

$$\text{minimize} \sum_{b=1}^{|bins|} \text{abs}(tc[b] - bc[b]) \tag{7.4}$$

This approach fixes the core problem with a Tiresias-based approach, but its assumption of independent predicates is a key weakness that would make it unsuitable for many datasets. In the next section, we will remove this assumption.

7.4.3 Adding Dependence Information

In the previous section, we describe how our algorithm uses a probabilistic approach to estimating the impact of including a predicate in a candidate solution. However, if two predicates are included in a candidate solution that have some dependency between them (i.e., their matching rows overlap more often than would be expected by chance), then the estimated bin counts (Equation 7.1) will be incorrect.

To remedy this problem, our algorithm has a policy that emits dependency information of the form $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$, where $\mathbf{d}_i = \{p_1, p_2, \ldots, p_q \mid 0 < p_j \leq |\mathcal{C}|\}$. It then uses D to create additional constants, variables, and constraints in its optimization problem, which allows the solver to correctly estimate the bin counts for a candidate transformation program that has dependent predicates.

Additional Variables — For each $\mathbf{d}_i \in D$, our algorithm creates a binary variable in the optimization model, which if true then the bin counts are estimated as if $\forall p \in \mathbf{d}_i$ are included in a candidate solution. These binary variables are combined with the previously created *preds* variables, creating *preds*$'$, which has $|\mathcal{C}|+|D|$ binary variables that the solver is finding values for.

Our algorithm creates additional constants for the bin probabilities of each dependent predicates set, similar to the ones added for the individual predicates. Additionally, a set of constants, *pred_sets*, is added to the optimization model, where each *pred_set* $\in$ *pred_sets* links a synthetic predicate variable with its associated predicates.

Additional Constraints — In order to avoid the solver misestimating the bin counts for a candidate solution with dependent predicates, our algorithm imposes the following restriction: If the synthetic variable for a dependent predicates set $\mathbf{d}_i$ is true, then $\forall p \in \mathbf{d}_i : preds'[p] = \text{false}$. Likewise, if $\exists p \in \mathbf{d}_i : preds'[p] = \text{true}$, then the synthetic variable for dependent predicates set $\mathbf{d}_i$ must be false. Equation 7.5 summarizes this constraint:

$$\forall pred\_set \in pred\_sets : \sum_{i \in pred\_set} preds'[i] \leq 1 \tag{7.5}$$

7.4.4 Prioritizing Dependence Information

Our approach to including dependency information fixes the problem of inaccurate estimates from dependent predicates, but another problem remains: adding dependency information to the optimization model leads to more constraints, and finding solutions becomes more difficult.

Our algorithm addresses this problem by using a *filtering with exploration* policy for limiting dependence information in a model (Algorithm 8). Using precomputed dependence scores for each dependent predicates set (described in Section 7.5), each set is tested to see if its estimated counts is within a margin of error (ϵ) of being independent, and if so, the dependency set is excluded (lines 4 – 9). Next, the top *pdp* percent of the remaining dependency sets are chosen as the final output, with a

randExplore percent of those replaced by a random selection of dependency sets (lines 10 – 14). This *random exploration* of predicate dependency helps find solutions that are more common but have relatively low dependency amongst its predicates.

7.4.5 Adding Code-Based Constraints

In addition to signal-based constraints, users can provide code-based DesiredPred or NoPred constraints, where the objective is to include (or exclude) predicates that are similar (or dissimilar) to the constraint. For instance, Janet in Example 7.2 provides DesiredPred constraints of `(type2 = True)` and `(age > 65)`, indicating that her ideal transformation program would have predicates similar to these. Thus, a transformation program with `(age > 50)` is preferred over one with no age filter, or one that has `(age < 65)`.

To model this, our algorithm creates additional constants, variables, and constraints, along with an updated objective function, so that the solver prefers solutions that match well on both signal- and code-based constraints.

Additional Variables — A set is added to hold the code-distance calculations, which is used in the objective function. Additionally, a constant is added that defines a similarity matrix between the code-based constraints and each of the predicates. This is generated using a simple code-similarity function that rewards matching columns, operators, and similar ranges of values.

Additional Constraints — One additional constraint is added, which uses the code-similarity matrix to define the code distances for any predicate variables that are true. Equation 7.6 defines this constraint, where U_c is the set of DesiredPred constraints, D_c is array of code distances being estimated, and S is the code-similarity matrix:

$$\forall u \in U_c : D_c[u] = \sum_{p \in preds} (p * (1 - S[i, u])) \tag{7.6}$$

Updated Objective Function — The updated objective function, which supports both signal- and code-based constraints, is defined in Equation 7.7, where D_s is an array of signal distances (i.e., the objective function in Equation 7.4), D_c is the array of code distances from Equation 7.6, and $\mathcal{W}$ is the user's constraint preference weights:

$$\text{minimize} \sum_{i=1}^{|U_s|} \mathcal{W}[U_s, i] * D_s[i] + \sum_{i=1}^{|U_c|} \mathcal{W}[U_c, i] * D_c[i] \tag{7.7}$$

Algorithm 8 Prioritizing of dependence information

Input: $\mathcal{D}$,$\mathcal{U}_i$, pp, pdp, ϵ, $randExplore$

```
depScores = getDepScores(pp)
binCounts = getBinCounts(D, U_i)
T = getTargetBounds(D, U_i)
for all pids, s ∈ depScores do
    estCounts = (∏_{p∈pids} pp[p]) * binCounts
    if withinBounds(T, estCounts) then
        delete depScores[(p_1, p_2)]
    end if
end for
numRand = randExplore * pdp * |depScores|
numDep = (1 − randExplore) * pdp * |depScores|
finalDepScores = depScores[0 : numDep]
delete depScores[0 : numDep]
finalDepScores += randomSample(depScores, numRand)
return finalDepScores
```

7.4.6 Optimization Solver

Our algorithm assumes that an off-the-shelf constrained-optimization solver is used to process its formulated optimization problem; however, one common problem with these solvers is they have several internal parameters that control their success at finding solutions. For many problems, the default settings work fine, but for others, the solver can reach a local minimum prematurely and decide that a problem is infeasible or that a solution cannot be found. To work around this, our algorithm offers dynamic configuration changing, where multiple configurations can be used.

7.4.7 Result Generation

After the optimization solver produces a result, our algorithm processes it into a transformation program and generated output dataset, and then checks for potential problems.

Processing the Solver Output — When a solution is found by the solver, it returns a list of predicates in its chosen solution. This list can contain synthetic predicates (i.e., each $\mathbf{d}_i \in D$), so our algorithm replaces these with the associated set of predicates and adds them to the transformation program C_i. The remaining non-synthetic predicates returned from the solver are also added to C_i.

Our algorithm then verifies the solution is valid by applying C_i to input dataset $\mathcal{D}$ to generate output dataset $\mathcal{R}_i$, and then compares the discretized distribution of any TargetDistrib constraints' columns against its associated target bounds. If an

answer is invalid, or the solver did not return a solution, then our algorithm has three options: First, if the solver deemed the problem as infeasible—which may be due to it finding a local minimum—our algorithm will adjust the configuration for the solver (as described in Section 7.4.6). Second, our algorithm will adjust the slack parameter ϵ to provide more leniency with solution validity. Third, the amount of dependence information included in the optimization problem can be increased.

Identifying Potential Problems — For a given result, our algorithm iterates through the different columns to compare their distributions before and after applying a transformation program, where Pearson correlation is used to identify columns with at least a δ percentage change. These are then added to the system-identified problem set $\mathcal{P}_i$.

7.4.8 Greedy-Hybrid Approach

For certain problems, a greedy heuristic can replace the role of the constrained-optimization solver in our algorithm. For this reason, our algorithm has an optional *greedy-hybrid* approach, where it uses a greedy heuristic to search for a valid solution, and if none is found, our algorithm proceeds with using the constrained-optimization solver.

The greedy heuristic works as follows: First, a threshold k is defined to control the maximum number of predicates in a solution. It then finds the single predicate that best matches the target counts. Next it pairs this predicate with all remaining predicates, finding the set of predicates that best match the target counts. This process repeats until k candidate solutions are determined, and then the best of these is chosen as the final answer.

The advantage of the greedy heuristic is it can search for solutions relatively quickly (with $O(k|\mathcal{C}|)$ time complexity), but it requires the best single predicate (found in its first step) to be part of a valid solution. For this reason, the heuristic on its own is not as generalizable as a greedy-hybrid approach (as we show in our experiments in Section 7.6).

7.4.9 Full Algorithm for Solution Finding

Algorithm 9 summarizes our full algorithm for find solutions. In line 1, the raw input dataset is analyzed. Next in lines 2 – 3, the dependent predicate sets are filtered using the method in Algorithm 8 and the target bounds are determined. In lines 4 – 5, the greedy-hybrid approach is used, and if a valid solution is found, the optimization

Algorithm 9 Full algorithm for finding solutions

Input: $\mathcal{D}$,$\mathcal{U}_i, \mathcal{W}_i$, *pdp*, ϵ, *randExplore*

1: $\mathcal{C}, pp$ = analyzeDataset($\mathcal{D}$)
2: S = getFilteredDepSets($\mathcal{D}$,$\mathcal{U}_i$, *pp*, *pdp*, *randExplore*, ϵ)
3: T = getTargetBounds($\mathcal{U}_i, \epsilon$)
4: $C_i, \mathcal{R}_i$ = greedySearch($\mathcal{D}$,$\mathcal{U}_i, \mathcal{W}, \mathcal{C}, \epsilon$)
5: **if not** isValid($C_i, \mathcal{R}_i$,$\mathcal{U}_i$) **then**
6: *preds*$'$ = getModelPreds($\mathcal{D}$, *pp*, $\mathcal{C}$)
7: *config*, ϵ' = getSolverConfig($\mathcal{D}$,$\mathcal{U}_i$)
8: *model* = formulateProblem($\mathcal{D}$,$\mathcal{U}_i$, *pp*, T, S, *preds*$'$, ϵ')
9: *solverResult* = runOpSolver(*model*, *config*)
10: $C_i, \mathcal{R}_i$ = processSolverResult(*solverResult*)
11: **end if**
12: $\mathcal{P}_i$ = identifyProblems($\mathcal{R}_i$,$\mathcal{U}_i, \mathcal{W}_i$)
13: return $C_i, \mathcal{R}_i, \mathcal{P}_i$

problem is skipped. Otherwise, the optimization problem is formulated (lines 6 – 8), and the optimization solver is used to find a solution (line 9). In line 10, the solver result is processed into the transformation program and generated output dataset. Lastly, the transformed dataset is evaluated for potential problems (line 12), and the result is returned to the user.

7.4.10 Time Complexity

Our algorithm has two main components that affect its runtime: problem formulation and solving the constrained optimization problem.

Problem Formulation — The main bottleneck in problem formulation is scoring and sorting the dependent predicate sets. Since sets can be up to k in size, for p predicates, roughly p^k sets can exist, so if scoring a single set has $O(s)$ time complexity, the total scoring has $O(s*p^k)$ time complexity. When we include sorting these scores, time complexity increases to $O(s*p^k+p^k\log(p^k))$. However, we have found that $k=2$ is good enough for most datasets, so this simplifies to $O(s*p^2+p^2\log(p^2))$.

Optimization Solver — The solvers we use employ active set methods for solving constrained-optimization problems, and these approaches can have exponential time complexity for the worse case input instance, but in practice, are generally recognized as much better: $O(n^3)$ for dense problems, but the cost of nonlinear functions and gradients enters into this complexity. Our problem grows mostly with the amount of dependent predicate sets, which creates a new model constraint for each set added. Generally, any more than a few million sets results in a runtime of over an hour, and that still has no guarantees of converging on a potential solution.

7.5 Prototype System

We embody our algorithm in a prototype software system, EMERIL, which was written in over 6,800 lines of mostly Python source code. It also makes a few decisions regarding data-specific policies, which we describe below.

Emeril Interaction Model — EMERIL uses a web-based interface for interacting with users. Users can either explicitly declare their code-based constraints in an SQL-like syntax, or submit actual SQL queries as if working with a traditional database, which EMERIL uses to infer the user constraints. Users provide signal-based constraints by "drawing" them or defining them as a comma-separated list, as we allowed in past work [36].

Optimization Solver — EMERIL uses the constrained-optimization solvers MINOS [132] and SNOPT [88], where SNOPT is preferred for larger datasets. We use AMPL [84], an algebraic modeling library, to interface with these solvers. As discussed in Section 7.4.6, our algorithm allows for dynamic configuration changing, so EMERIL uses a few configurations for each solver, which we determined through a parameter grid search on several test datasets.

Candidate Predicates — Generating candidate predicates is a pluggable component of EMERIL, but by default it uses the following approach: equality predicates are generated for each value in any categorical columns; then for numerical columns, a histogram is generated with the column's values, with three predicates generated for each bin edge: a range predicate between it and the next edge, and greater-than and less-than inequality predicates. This yields the set of candidate transformations $\mathcal{C}$.

Identifying Predicate Dependence — Since our algorithm uses predicate dependency information to improve its accuracy, EMERIL precomputes this before processing queries: Using the formula for independence ($P(A,B) = P(A)P(B)$), we measure dependence between two predicates using $d = abs(P(A,B) - P(A)P(B))$. EMERIL parallelizes this step to speedup their creation. This yields the *depScores* variable in Algorithm 8 from Section 7.4.4.

Predicate Probability Distributions — As part of the dataset analysis, EMERIL precomputes probability distributions for all predicates and sets of dependent predicates: EMERIL first determines the data items that each predicate or set of predicates matches. It then calculates a density estimation signal for each column in input dataset $\mathcal{D}$ to produce the probability distributions. This yields the *pp* variable in

Algorithm 9 from Section 7.4.9.

Time Complexity — The complexity of the dataset analysis is primarily dependent on the number of candidate predicates for a dataset, where the biggest bottleneck is from calculating the dependent predicate sets. If there are p predicates, calculating dependency scores for sets with up to k items requires roughly a $O(p^k)$ time complexity; however, we have found that $k = 2$ is good enough for most datasets, so this reduces to $O(p^2)$.

7.6 Experiments

In this section, we evaluate our three claims about EMERIL:

1. EMERIL performs better than competing methods at finding high-quality data transformation programs. We evaluated this on synthetic datasets with varying levels complexity (Section 7.6.2), as well as on several popular real-world datasets (Section 7.6.3).
2. EMERIL performs well at finding solutions when code-based user constraints are specified, all with no meaningful impact on runtime (Section 7.6.4).
3. Each individual component of EMERIL contributes meaningfully to EMERIL's ability to find transformation programs that best satisfy user constraints (Section 7.6.5).

7.6.1 Experimental Setting

In this section, we describe our baseline methods, evaluation metrics, and our experimental configuration.

Baseline Methods — We evaluated EMERIL against three baseline methods:

- **Tiresias** — For this approach, we use the adaption of Tiresias we describe in Section 7.3. We expect this approach to scale poorly with dataset size.
- **NChooseK** — For this approach, we test all combinations of up to k predicates ($k = 2$), using the set of predicates with the best score as the final answer. We expect this approach to work well for smaller problems but to quickly get bogged down for larger problems.
- **Greedy** — For this approach, we use the greedy search algorithm described in Section 7.4.8, with $k = 2$. Greedy incrementally builds candidate solutions by choosing a single predicate that best matches a target signal, so we expect Greedy

to only perform well on problems where the first selected predicate is part of a valid solution.

Evaluation Metrics — Our synthetic datasets and answers have some randomness associated with their generation, so we repeated each experiment for all variations of the datasets. From this, we measured the ***percentage of time a solution is found*** and the ***average runtime***. Runtimes for all systems do not include offline analysis or pre-loading of data from disk. In all cases, a production system can pre-load this to improve query times.

Experimental Configuration — We ran all of our experiments on a 32-core (2.8GHz) Opteron 6110 server with 512 GB RAM. We imposed a one hour time limit on EMERIL and our baseline methods for finding a solution. While users will have different preferences on how long they will wait, we feel this is a reasonable timeframe to expect an answer. For NChooseK, if the time limit expired, the best answer found up to that point was used as the final answer.

7.6.2 Evaluating on Synthetic Data

In this set of experiments, we tested EMERIL and our baseline methods using synthetic datasets with varying levels of size, correlation, and input data complexity. EMERIL used the top 10% of dependency sets (with a 50% *randExplore* setting) for these experiments.

7.6.2.1 Synthetic Datasets

We generated our synthetic datasets to control for the following properties, which impact discovery of high-quality transformation programs:

- **Input data complexity** — We varied the number of candidate transformations (a rough proxy for input data complexity since a more varied dataset will yield more candidates) from 1,000 to 5,000, in increments of 1,000.
- **Number of rows** — We varied the number of rows from 100,000 to 1 million, in increments of 100,000.
- **Level of data correlation** — We varied the level between columns from 0.0 to 1.0, in increments of 0.1.

Dataset Generation — We repeated the following process with different random seeds to generate 30 synthetic datasets for each property configuration described above:

1. We chose a target number of rows, columns, and data correlation. When varying a property, we used one of the variations listed above; otherwise, we used 50,000 rows, 100 columns, and a 0.5 average correlation.
2. We generated a relational dataset with the target number of rows and columns by sampling values from a Gaussian distribution. To control the level of correlation between columns, we used a covariance matrix filled with the target level to create this distribution.
3. For candidate predicates in our varied-row and correlation datasets, we created one inequality predicate per column that captured all of the column's values. For the varied-complexity datasets, we generated inequality predicates for each unique column value, and then randomly sampled our target number.

Synthetic Answer Generation — The primary challenge of EMERIL is with matching signal-based constraints, so to represent this in our experiments without entangling our results with user behavior, we mimicked a user providing a TargetDistrib constraint that specifies a desired uniform distribution on a column that was Gaussian-distributed. This constraint was chosen to support operations like those we described in Section 7.1. For example, Janet may see that *age* in her raw dataset follows a Gaussian distribution, but needs it to be uniform for drug testing.

To account for the Greedy baseline's success being heavily dependent upon the synthetic answer, we added a "noise" column and predicate that we randomly generate, where around half of the time, it has the best individual score amongst predicates. Thus, Greedy will choose it first and not find a valid answer.

7.6.2.2 Varied Input Data Complexity

Summary: EMERIL found a valid solution *34% more often* than our best baseline (NChooseK), using *77% less time* to do so, all while Tiresias never found a valid solution.

Overview — In this experiment, we vary the number of candidate predicates (a rough proxy for input data complexity) from which a solution is selected. This number is the most important factor in problem difficulty.

Results — Figure 7.3 summarizes the results, which shows that EMERIL was able to find answers much more often than our baseline methods and with a reasonable runtime—finding a solution 91% of the time, averaging 14 minutes per run. In contrast, our baseline methods have mixed success. Tiresias was unable to find any

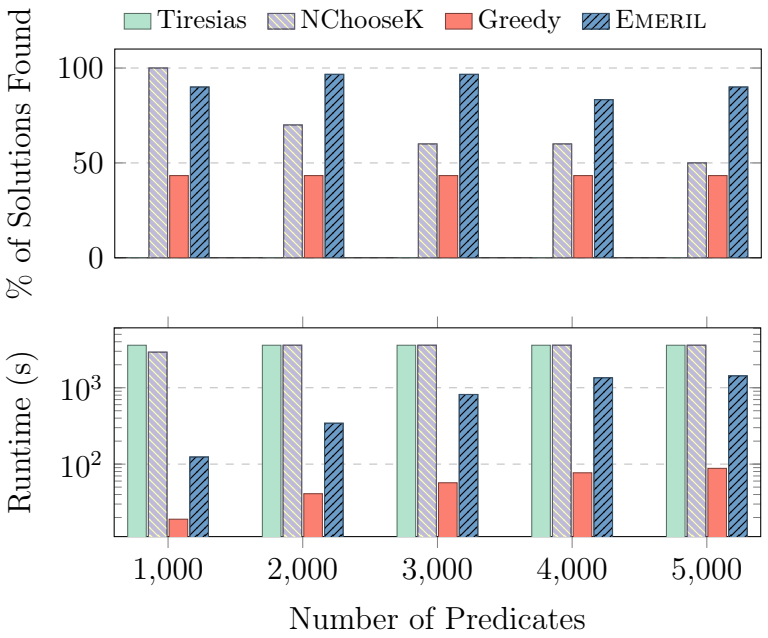


Figure 7.3: Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when number of predicates is varied. Big bars are better on top chart, smaller on lower chart.

answers in the allotted time, but this failure was expected for the reasons described in Section 7.3. Greedy and NChooseK were able to find answers 43% and 68% of the time on average. Greedy had the best runtime of all methods (averaging 1 minute per run), and NChooseK was generally using the full hour allowed. While the combinatorial methods have some promising results, they only work in limited cases, and thus are not a comprehensive solution for our problem.

7.6.2.3 Varied Number of Rows

Summary: EMERIL *used 96% less time* on average to achieve the same level of solution finding (100%) as our best baseline (NChooseK), all while Tiresias never found a valid solution.

Overview — In this experiment, we varied the number of rows in each dataset. For most methods, this number impacts how quickly a candidate answer can be evaluated.

Results — As shown in Figure 7.4, EMERIL found valid solutions 100% of the time and had great runtimes, averaging 34 seconds per run. We designed our algorithm in Section 7.4 to have a time complexity independent of the number of rows in a dataset, but when using the greedy-hybrid search (as these results are), the runtime of a greedy search is added to EMERIL's total runtime. The greedy search on its own

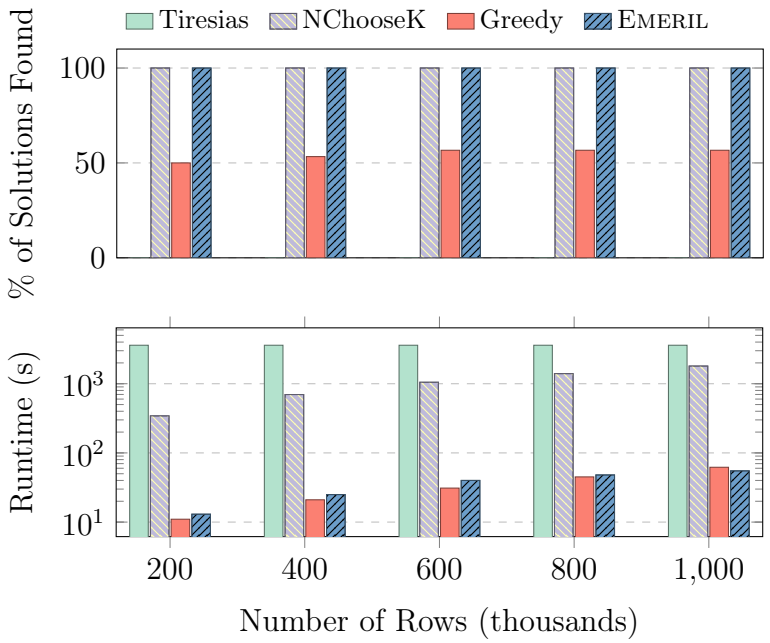


Figure 7.4: Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when number of data items is varied. Big bars are better on top chart, smaller on lower chart.

averaged 29 seconds per run, but only found valid solutions 54% of the time. Tiresias was unable to find a solution for any of the experiment runs, reaching the one hour time limit in all cases. NChooseK found valid solutions 100% of the time, but at the cost of a large runtime, averaging 16 minutes per run; however, had we chosen a larger number of candidate predicates than the 100 we used for these experiments, NChooseK would perform much more poorly (as Section 7.6.2.2 shows).

7.6.2.4 Varied Level of Data Correlation

Summary: For realistic levels of correlation, EMERIL found valid solutions often and did so very quickly, all while Tiresias never found a valid solution.

Overview — In this experiment, we varied the level of correlation between columns in the datasets. As this number increases, candidate predicates become more correlated, and since EMERIL's constrained optimization model assumes predicate independence when no other information is provided, we expect it to be more challenging to find valid solutions as the correlation level increases.

Results — Figure 7.5 summarizes the results. Across all datasets, EMERIL found 77% of valid solutions, with a runtime of around 3 seconds per run. For datasets with correlation of 0.5 or below, EMERIL found 100% of solutions, but for those with

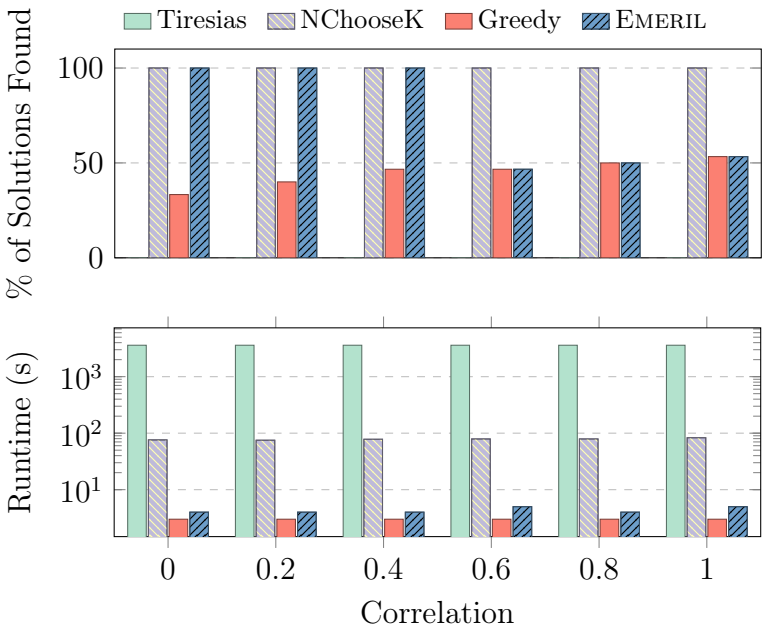


Figure 7.5: Percent of solutions found and runtime (log scale) for EMERIL and our baseline methods when the level of correlation is varied. Big bars are better on top chart, smaller on lower chart.

correlation above 0.5, EMERIL found solutions 50% of the time. This lower success rate is because EMERIL has a small budget of dependence information in its constrained optimization model (to help keep runtimes low), and when data correlation levels get too high, the synthetic answer's dependence information is generally not included, even though it is needed. While EMERIL may struggle with highly correlated datasets, we have found this to be less common with real-world datasets. For instance, only one of the six real-world datasets discussed in this chapter have an average correlation above 0.18 (see Table 7.2).

For our baselines, Tiresias reached the timeout limit before finding any solutions. Even with a relatively small dataset as we used in these experiments, the required model becomes too complex for an optimization solver to find a solution in the allotted time. Greedy was able to find valid solutions 45% of the time, averaging 3 seconds of runtime. NChooseK was able to find valid solutions 100% of the time, but at the cost of a higher runtime (averaging 77 seconds of runtime); however, as discussed in Section 7.6.2.3, had we chosen a larger number of candidate predicates for these experiments, NChooseK would perform much more poorly.

Dataset	Rows	Cols	Preds	Corr
ATUS [2]	68,077	374	3,363	0.05
Food Facts [8]	65,503	159	1,103	0.18
GTD [4]	170,350	135	1,262	0.02
NFL Play Data [7]	46,129	66	535	0.02
NHANES [6]	9,813	693	9,618	0.13
WDI [11]	409,992	62	1,712	0.78

Table 7.2: Real-world datasets used in our experiments. Number of predicates is using the *naïve* generation method, and correlation is average Pearson correlation between the dataset's columns.

7.6.3 Evaluating with Real-World Data

Summary: Emeril was able to find solutions 44% more often than the best baseline (Greedy), while only increasing average runtime from 3 minutes to 14 minutes. Tiresias on the other hand, never finds a solution.

Overview — In these experiments, we evaluated Emeril and our baseline methods using real-world datasets (summarized in Table 7.2), many of which are used extensively in research (e.g., ATUS and NHANES are US government health surveys, GTD is a database on terrorist activity, and WDI is the World Bank's Development Indicators). The candidate predicates were created using the method described in Section 7.5. Emeril used the top 1% of dependency sets (with a 0% *randExplore* setting) for these experiments. We chose slightly different values from the synthetic dataset experiments to account for the larger dataset sizes and less common target answers.

We defined a TargetDistrib constraint for a column in each dataset (e.g., *carbohydrates* was used for Food Facts), which the competing systems are trying to match. In order to ensure that there is a valid answer to evaluate against, we inserted a synthetic answer using a similar approach to our previous experiments: two columns in the data were replaced with synthetic values that match with two predicates we created so that when applied together on the dataset, the result matches the signal of the TargetDistrib constraint. While using a synthetic answer does not fully capture a real-world scenario, it allowed us to evaluate our systems on data that has lots of real-life dataset characteristics without getting bogged down in arbitrary user preferences.

Results — As shown in Figure 7.6, Emeril found solutions 81% of the time, averaging 14 minutes per run. For our baseline methods, Tiresias was unable to find any solutions during the allotted time; NChooseK found a valid solution 38% of the time,

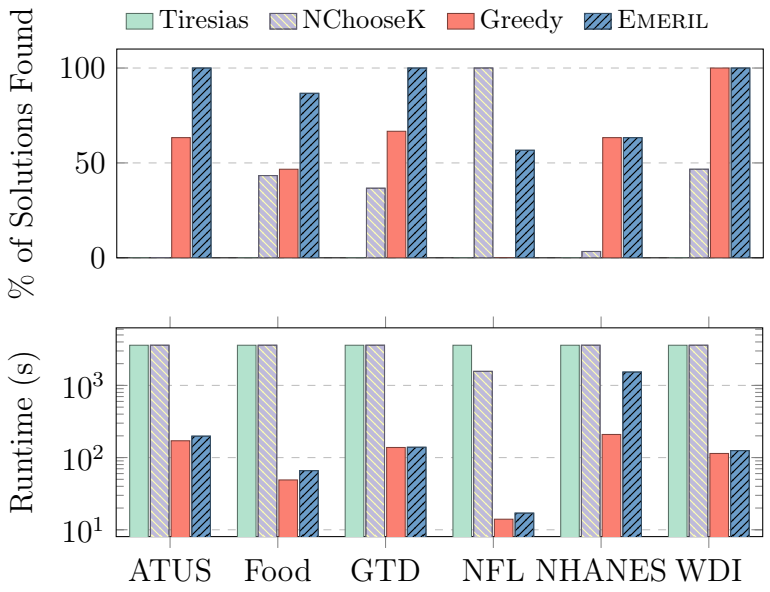


Figure 7.6: Percent of solutions found and runtime (log scale) for Emeril and our baseline methods evaluated using real-world datasets. Big bars are better on top chart, smaller on lower chart.

averaging 54 minutes per run; and Greedy had a much lower runtime, averaging 2 minutes per run, finding valid solutions 57% of the time.

Emeril has comparatively poor performance on the NFL and NHANES datasets (finding 57% and 63% of the solutions respectively), but for different reasons: NHANES has the largest number of predicates of any test set, so the 1% limit added a very large amount of dependency information to our algorithm; this caused Emeril to exhaust the 1-hour time limit before finding a solution. For NFL, through pure chance, the target answers in the datasets have predicates that are almost—but not entirely—independent. That means our naïve approach from Section 7.4.2, which assumes total independence, would not find the answer, nor does our standard algorithm, which does not rank the relevant predicates very highly; we will have to go very deep into the ranking in order to find the correct answer. Indeed, we found that when increasing the percentage of dependency sets included in the optimization model from 1% to 20%, the percentage of solutions found for NFL increases from 57% to 67%. Somewhat ironically, NChooseK does very well on the NFL dataset, as it is small enough to exhaustively search all predicate sets. The power of the naïve NChooseK algorithm on this dataset suggests that it might be useful to explore an intelligent optimizer front-end to our system, which runs NChooseK when the input data is sufficiently small.

7.6.4 Evaluating with Code-Based Constraints

Summary: When the user provides a code-based constraint, and there are multiple solutions that match identically on a signal-based constraint, Emeril prefers the solution with better code similarity 100% of the time, with no meaningful impact on runtime.

Overview — In this experiment, we tested how well Emeril performs when matching both a signal- and code-based constraint. As described in Section 7.2, a user may specify that a particular predicate should be in the answer. The solver will then prefer predicates that are most similar with it. For example ($age > 30$) would have a decent match with ($age \geq 40$), but a poor match with ($gender =$ "female").

We assumed the user provided both code- and signal-based constraints. We used the same datasets as in our varied schema complexity experiments (Section 7.6.2.2), except we duplicated the synthetic answer with new column names. Thus, two synthetic answers exist, each equally match the signal-based constraint, but one better matches the code-based constraint.

Results — As Figure 7.7 shows, including a code-based constraint makes finding solutions slightly more difficult, finding 85% of solutions instead of 91% when matching just the signal-based constraint. Of the valid solutions found when matching multiple constraints, Emeril preferred the solution with the better code similarity 100% of the time. Finally, total runtime of Emeril was unaffected by adding the additional code-based constraint, where both sets of experiments averaged 14 minutes per run.

7.6.5 Evaluating Algorithm Components

In this set of experiments, we tested whether each of the different components of Emeril are indeed needed.

7.6.5.1 Optimization Solver

Summary: Using the constrained optimization solver allowed Emeril to perform better than any competing method.

Overview — Using the optimization solver is core to Emeril, and the alternative to using it is to simply use one of our baseline methods (e.g., Greedy or NChooseK). In Section 7.6.2, we show that these baseline methods did not perform as well as Emeril; however, we can also evaluate a few variations of NChooseK, where the allowed runtime is decreased, thus the best answer within that time limit is used. We

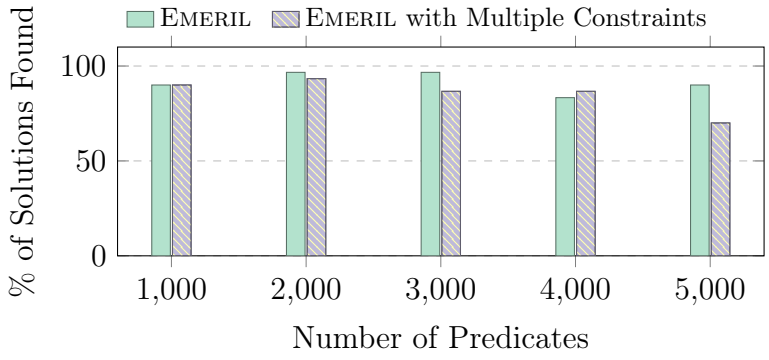


Figure 7.7: Percent of solutions found by Emeril when matching multiple constraints versus just one signal-based constraint.

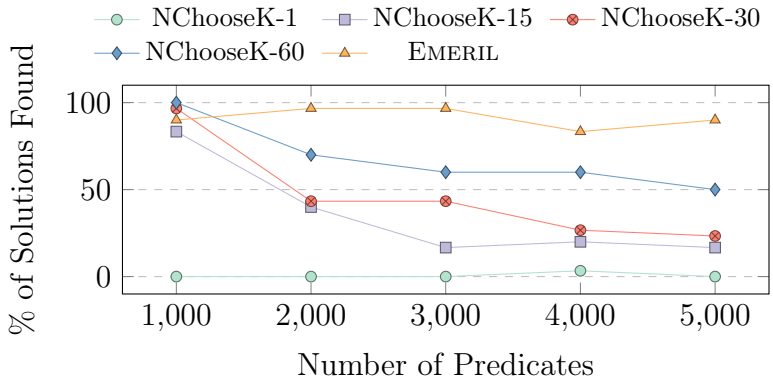


Figure 7.8: Percent of time a solution found when varying the number of predicates and using either Emeril or NChooseK with varied max runtimes of 1, 15, 30, and 60 minutes.

tried four variations of NChooseK, with the allowed time limit set to 1, 15, 30, and 60 minutes each.

Results — As Figure 7.8 shows, Emeril found 91% of solutions (averaging 14 minutes per run). The best version of NChooseK, which used 60 minutes per run, only found 68% of solutions. If given enough time, NChooseK would find solutions 100% of the time, but as the number of predicates grows (N), or the maximum number of predicates in an answer grows (K), an exhaustive combinatorial search becomes very slow.

7.6.5.2 Using Dependence Information

Summary: When Emeril used dependence information, it found valid solutions 3x more often than Emeril without it.

Overview — The next component we tested was the use of predicate dependence

information in our optimization problem. As described in Section 7.4.3, this information is needed for most real world datasets that have some dependence amongst its data; if completely independent, then no dependence information is needed.

Results — Table 7.3 shows the results from using EMERIL with dependence information (EMERIL10) and without it (EMNODEP) on all of our synthetic datasets. When including dependence information, EMERIL found valid solutions 78% of the time, whereas EMNODEP only found 19% of valid solutions—a 3x improvement when including dependence information. EMNODEP is unable to find any valid solutions for the varied-predicates datasets nor the varied-rows datasets. With the varied-correlation datasets, EMNODEP does great with lower levels of correlation, finding solutions 83% of the time for correlations from 0.0 – 0.5. This is due to the independence assumption holding, at least within our target bounds' margin of error.

7.6.5.3 Prioritizing of Dependence Information

Summary: Using EMERIL's prioritizing of dependence information found solutions 50% more often than without it.

Overview — In this experiment, we tested the benefit of EMERIL's prioritizing of predicate dependence information in our optimization model. We tested this by disabling our greedy-hybrid search and replacing EMERIL's approach for including dependency information with a process that randomly selected 5% and 10% of the unordered dependence information (EMRAND5 and EMRAND10 respectively). We compared this with EMERIL using the top 5% and 10% of dependence information (EMERIL5 and EMERIL10).

Results — Table 7.3 summarizes these results. When comparing each method using 5% of the dependence information, EMERIL5 found solutions more often than EMRAND5 on each of the varied data experiments. EMRAND5 still performed relatively well on some experiments: in the predicate-based experiment, as the number of predicates grow, there is a greater chance of a valid answer being included by random, especially if there are multiple valid solutions in a dataset. This is why EMERIL includes a percentage of random dependence information in its model. In the correlation-based experiment, datasets with lower levels of correlation do not require the dependence information, so the included information is irrelevant to solution finding. For the higher correlation levels, both methods struggle for the same reason: too many highly-dependent predicate sets were dominating the list of included dependency information. When comparing the two methods using 10% of the dependence

information, we see similar results: EMERIL10 found valid solutions more often than EMRAND10 on all of experiments datasets.

7.7 Related Work

There are several areas of research that are related to our work, which we discuss below.

Data Preparation — Data preparation is the process of transforming a raw dataset into a new data object, which is then used in some downstream analysis (e.g., in a statistical or machine learning model). Much work has gone into making this process easier for users. This includes helping users with data extraction, such as done by TextRunner [172], WebTables [56], and Senbazuru [63], which provide systems for extracting different types of data from large corpora like the Web. This also includes helping users with data wrangling, such as programming-by-example work from Wu et al. [165–167], BlinkFill [93], and Foofah [105]; or with interactive systems like Wrangler [107] and Trifacta [80]. Our work is a subtype of automatic program generation like Trifacta or Foofah, but the user guides program generation by providing constraints around desired outputs, instead of programming-by-demonstration or concrete output tuples.

Reverse Data Management — In reverse data management (RDM), an action is taken on an input dataset to achieve a certain effect on the output dataset [121]. Sub-domains of RDM include *how-to* querying [122], updating through views [47], and constraint-based repairs [41], with the most similar area to our work being how-to querying. In how-to querying, the goal is to compute hypothetical updates to a database such that the output matches one or more constraints. The most similar work to ours is Tiresias [122], which also uses constrained optimization in their solution, but as we show in Section 7.3, using their approach on our problem quickly becomes intractable as the input dataset size or space of candidate transformations grows.

Why-Not Provenance — In *why-so* and *why-not* querying, users want to understand why query results are as they are, such as why particular tuples are included or missing from a result (e.g., a user searching for flights and wondering why a particular direct flight does not exist). Methods include finding modifications to a database that cause the original query to yield an expected result [103], and by tracing through the query execution graph, finding the manipulation that caused a tuple to not be

Experiment	EmNoDep	EmRand5	Emeril5	EmRand10	Emeril10
Predicates (1k – 5k)	0.00	0.70	**0.77**	0.72	**0.82**
Rows (100k – 1m)	0.00	0.04	**1.00**	0.10	**1.00**
Correlation (0.0 – 1.0)	0.45	0.48	**0.56**	0.51	**0.56**
Average	0.19	0.35	**0.77**	0.39	**0.78**

Table 7.3: Percentage of time a solution found for our synthetic datasets when using EMERIL with and without prioritized dependence information (greedy-hybrid search disabled in both cases). Results are averaged across all datasets in each experiment (e.g., for "Rows," that is 30 x 10 datasets).

included [62]. In `ConQueR` [157], the system suggests query modifications that will include user-provided why-not tuples (e.g., the flight searcher may see a suggestion to increase her maximum price). Our work is similar to `ConQueR` in that we suggest query modifications, but our goal is to alter the distribution of a particular column, and while one might be able to specify why-not tuples from those that should be removed to produce a desired distribution, the opposite is also needed—removing tuples from the generated output dataset. Further, `ConQueR` assumes inequality predicates when determining how to include why-not tuples, which would further limit its ability to help with our problem.

Query Formulation and Refinement — Query formulation is a well-studied problem in database literature, where the goal is to help users formulate or revise queries so that generated results better match users' desired results. Past work has used query logs [108] and the data being queried [171] to aid with this formulation. Our work shares a similar goal of aiding users with query refinement, except we ask the users for explicit constraints, which we use to formulate a transformation program.

Constraint-Based Querying — Constraint-based querying systems, such as CONQUEST [49] and our work with RACCOONDB [34, 36, 37], provide custom query languages for defining constraints, which are used to find relevant results. Our work is most similar with RACCOONDB since both use code- and signal-based constraints to find a small subset of items in a large pool of candidates. However, RACCOONDB's constraints are less expressive than EMERIL and assume a PCA-based pipeline that makes transformation program generation substantially easier than in the explain-and-repair task.

7.8 Conclusion and Future Work

In this work, we present a novel user model and algorithm for explain-and-repair data transformation systems, where users provide constraints on their desired transformation program and the data it produces, and the system finds the best match for both. We show that our constrained-optimization based algorithm for matching user constraints outperforms the previous state-of-the-art solution. Future work includes expanding beyond filter-based transformations and allowing for near real-time interactive exploration of the tradeoff between different constraints.

CHAPTER VIII

Conclusion and Future Work

8.1 Conclusion and Future Work

There are three high-level contributions this dissertation contains: helping users avoid tedious work, simplifying nowcasting, and improving economic analysis in the era of big data. We discuss each of these below.

8.1.1 User Interaction for Tedious Work

The first high-level contribution of this dissertation is a pattern of showing how declarative user interaction can help users avoid tedious work. The problem—which can exist in many domains—is of tedious searching to satisfy multiple criteria. For instance in RaccoonDB, users want to search for social media topics that are relevant both semantically and statistically. As we show in Chapter VI, humans are generally bad at satisfying multiple criteria. The formula we use to address this problem is to have an interactive loop, where users define their goals through some declarative query language, and a system does the tedious searching to try satisfying them. The result is less work by users, and often higher query results than the user can attain through manual efforts. We demonstrate this formula with our RaccoonDB system and with our Emeril work.

One future direction for this declarative user interaction is to find other applications that need multiple criteria satisfaction, and simply apply this formula to the problem. For instance, consider a financial advisor creating a portfolio. He has many investments to choose from, and he may have two competing criteria for the chosen investments, such as low volatility and high returns. Similarly, a researcher creating a machine learning model may have criteria of, high accuracy and low parameter count—where the solution is a generalized version of RaccoonDB. And finally, con-

sider a researcher write rich transformations in a language like Python, where the criteria is matching on both code and data-based constraints—again, where the solution is a generalized version of EMERIL.

8.1.2 Simplifying Nowcasting

The second pattern we exemplify in our research is with simplifying nowcasting. The problem faced is that organizations, such as governments or businesses, want insights on trends, people, and other real-world phenomena. The high-level solution is to use new data sources to capture these trends, and make it easy for the non-computer scientist to do. The result is that new phenomena are explained, and often much quicker than with traditional data—such as surveys. Example of this contribution can be found in our economics nowcasting work and our RACCOONDB system.

There are several areas for future work with simplifying nowcasting. First, *micro-nowcasting*. The idea is to track various phenomena on a per-user basis. For example, users may indicate changing jobs, or moving, or being depressed. The aggregate of these can capture new phenomena, such as job changes over time. A second area for future work is to expand into new media, such as images and video. Some work has started in this, such as using satellite images of hospital parking lots to estimate disease outbreak [55] or of retailer parking lots to estimate economic activity [139], but there's potential for much more. Lastly, one additional direction for future work is with outlier detection in nowcasting applications. This is about distinguishing noise from actual trend changes. For instance, discussion of being "fired" from a job may be a good indicator of unemployment, until news of a rocket being fired at an airliner floods the social media discussion. Helping automatically distinguish noise corrupting an indicator versus a significant change in the underlying phenomenon will greatly benefit users.

8.1.3 Economic Analysis

Economists face several challenges with transitioning from their traditional workflow to the new big-data workflow for economic insight. While adopting some of the data scientist training will help, a better short-term solution is for computer scientists and database researchers to develop domain-specific systems that can aid economists in this transition.

In this dissertation, we present several systems and methodologies that bring

economists closer to our *ideal workflow*, where users generate accurate and reliable results in a fast manner, all while avoiding the "heavy lifting" of working with big data sources like the Web. To help users generate accurate and reliable results, we present approaches to identifying relevant predictors in nowcasting applications, as well as methods for identifying potentially invalid nowcasting models and their inputs. We show how a streamlined workflow, combined with pruning and shared computation can help handle the heavy lifting of working with big data to generate results in near-interactive time. Finally, we propose a novel user model and architecture for *explain-and-repair* data transformation systems, which aid users with data preparation by allowing them to define their ideal transformation code with declarative constraints, and the system does its best to satisfy them, while offering an explanation for any problems along the way. These systems combined represent a unified effort to streamline the transition for economists to this new *big data workflow*.

Future work with helping economists in the big data age of economic analysis includes further collaborations between computer scientists and economists to develop systems that handle domain-specific problems. This can include helping economists **identify** novel data sources for drawing insights from; helping them **transform** this into formats needed for analysis—without introducing any undesirable bias; and further aiding them with their **evaluation** process. Lastly, economists will continue to benefit from better training on big data analysis and through improved usability of general big data processing tools.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Alexa web information service. http://aws.amazon.com/awis/.

[2] American Time Use Survey - Bureau of Labor Statistics. http://www.bls.gov/tus. Accessed: 2016-06-01.

[3] Big Huge Thesaurus. http://words.bighugelabs.com.

[4] Global Terrorism Database. http://start.umd.edu/gtd. Accessed: 2016-06-01.

[5] Mortgage bankers association's weekly app. survey. http://www.mortgagebankers.org/ResearchandForecasts/ProductsandSurveys/WeeklyApplicationSurvey.

[6] National Health and Nutrition Examination Survey - CDC. http://www.cdc.gov/nchs/nhanes. Accessed: 2016-06-01.

[7] NFL API. http://api.nfl.com/docs. Accessed: 2016-06-01.

[8] Open Food Facts - World. http://world.openfoodfacts.org. Accessed: 2016-06-01.

[9] The ClueWeb09 Dataset. http://www.lemurproject.org/clueweb09.php.

[10] WordNet: A lexical database for english. http://wordnet.princeton.edu.

[11] World Development Indicators - World Bank. http://data.worldbank.org. Accessed: 2016-06-01.

[12] Big data requires a big, new architecture. http://www.forbes.com/sites/ciocentral/2011/07/21/big-data-requires-a-big-new-architecture/, July 2011.

[13] BLS Methods Handbook - Chapter 16 Consumer Expenditures and Income. http://www.bls.gov/opub/hom/pdf/homch16.pdf, December 2015.

[14] Box Office Mojo Weekly Box Office Index. http://boxofficemojo.com/weekly, January 2015.

[15] CDC Influenza Surveillance Data via Google. http://www.google.org/flutrends/data.txt, January 2015.

[16] Census.gov - SIPP Introduction and History. http://www.census.gov/programs-surveys/sipp/about/sipp-introduction-history.html, December 2015.

[17] FBI - Gun Checks/NICBCS. http://www.fbi.gov/about-us/cjis/nics, January 2015.

[18] Google Domestic Trends - Google Finance. https://www.google.com/finance/domestic_trends, January 2015.

[19] Let's think about cognitive bias. *Nature*, 526(163), Oct 2015.

[20] Research Blog: Google Flu Trends gets a brand new engine. http://googleresearch.blogspot.com/2014/10/google-flu-trends-gets-brand-new-engine.html, January 2015.

[21] US Dept. of Labor - How the Government Measures Unemployment. http://www.bls.gov/cps/cps_htgm.htm, January 2015.

[22] US Dept. of Labor - Unemployment Insurance Weekly Claims Data. http://research.stlouisfed.org/fred2/graph/?id=ICSA,ICNSA, January 2015.

[23] US Energy Information Administration - Weekly Retail Gasoline and Diesel Prices. http://www.eia.gov/dnav/pet/pet_pri_gnd_dcus_nus_w.htm, January 2015.

[24] Weather Underground - Weather History New York City. http://www.wunderground.com/history/airport/KNYC/, January 2015.

[25] Amazon Machine Learning - Predictive Analytics with AWS. https://aws.amazon.com/machine-learning/, January 2016.

[26] Azure Machine Learning - Microsoft. https://azure.microsoft.com/en-us/services/machine-learning/, January 2016.

[27] KeystoneML - AMPLab - UC Berkeley. https://amplab.cs.berkeley.edu/projects/keystoneml/, January 2016.

[28] ML Pipelines: A New High-Level API for MLlib - Databricks Blog. https://databricks.com/blog/2015/01/07/ml-pipelines-a-new-high-level-api-for-mllib.html, January 2016.

[29] Daniel Aaronson, Sumit Agarwal, and Eric French. The spending and debt response to minimum wage hikes. 2008.

[30] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow.org.*

[31] John Ameriks, Gábor Kézdi, Minjoon Lee, and Matthew D Shapiro. Heterogeneity in expectations, risk tolerance, and household stock shares. *Ann Arbor*, 1001:48109–1248, 2015.

[32] Kevin J Anchukaitis, Petra Breitenmoser, Keith R Briffa, Agata Buchwal, Ulf Büntgen, Edward R Cook, Rosanne D D'Arrigo, Jan Esper, Michael N Evans, David Frank, et al. Tree rings and volcanic cooling. *Nature Geoscience*, 5(12):836–837, 2012.

[33] Michael Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[34] Michael R Anderson, Dolan Antenucci, and Michael J Cafarella. Runtime support for human-in-the-loop feature engineering system. *IEEE Data Eng. Bull.*, 39(4):62–84, 2016.

[35] Michael R Anderson and Michael Cafarella. Input selection for fast feature engineering. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 577–588. IEEE, 2016.

[36] Dolan Antenucci, Michael R Anderson, and Michael Cafarella. A declarative query processing system for nowcasting. *Proceedings of the VLDB Endowment*, 10(3):145–156, 2016.

[37] Dolan Antenucci, Michael R Anderson, Penghua Zhao, and Michael Cafarella. A query system for social media signals. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1326–1329. IEEE, 2016.

[38] Dolan Antenucci, Michael Cafarella, Margaret C Levenstein, Christopher Ré, and Matthew D Shapiro. Using social media to measure labor market flows. Working Paper 20010, National Bureau of Economic Research, March 2014.

[39] Dolan Antenucci, Michael J Cafarella, Margaret Levenstein, Christopher Ré, and Matthew Shapiro. Ringtail: Feature selection for easier nowcasting. In *WebDB*, 2013.

[40] Dolan Antenucci, Erdong Li, Shaobo Liu, Bochun Zhang, Michael J Cafarella, and Christopher Ré. Ringtail: a generalized nowcasting system. *PVLDB*, 2013.

[41] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79. ACM, 1999.

[42] N. Askitas and K. F. Zimmerman. Detecting mortgage delinquencies. Technical report, Forschungsinstitut zur Zunkuft der Arbeit, 2011.

[43] N. Askitas and K. F. Zimmerman. Google econometrics and unemployment forecasting. Technical report, Forschungsinstitut zur Zunkuft der Arbeit, 2011.

[44] Regis Barnichon, Michael Elsby, Bart Hobijn, and Ayşegül Şahin. Which industries are shifting the beveridge curve. *Monthly Lab. Rev.*, 135:25, 2012.

[45] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 795–804. ACM, 2011.

[46] Bloomberg. Bloomberg economic calendar. bloomberg.com, January 2016. Online; accessed 18 January 2016.

[47] Aaron Bohannon, Benjamin C Pierce, and Jeffrey A Vaughan. Relational lenses: a language for updatable views. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 338–347. ACM, 2006.

[48] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2011.

[49] Francesco Bonchi, Fosca Giannotti, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Roberto Trasarti. A constraint-based querying system for exploratory pattern discovery. *Information Systems*, 34(1):3–27, 2009.

[50] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[51] Paul G Brown. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 963–968. ACM, 2010.

[52] Bureau of Labor Statistics. Job openings and labor turnover – november 2013. bls.gov, November 2015. Online; accessed 18 January 2016.

[53] Bureau of Labor Statistics. Job openings and labor turnover – november 2015. bls.gov, January 2016. Online; accessed 20 January 2016.

[54] Bureau of Labor Statistics. Job openings and labor turnover report. bls.gov, January 2016. Online; accessed 18 January 2016.

[55] Patrick Butler, Naren Ramakrishnan, Elaine O Nsoesie, John S Brownstein, et al. Satellite imagery analysis: What can hospital parking lots tell us about a disease outbreak? 2014.

[56] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[57] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[58] Alberto Cavallo. Scraped data and sticky prices. Working Paper 4976-12, Massachusetts Institute of Technology Sloan, 2012.

[59] Alberto Cavallo. Online and official price indexes: measuring argentina's inflation. *Journal of Monetary Economics*, 60(2):152–165, 2013.

[60] Gary Chamberlain and Michael Rothschild. Arbitrage, factor structure, and mean-variance analysis on large asset markets, 1984.

[61] Andrew C. Chang and Phillip Li. Is economics research replicable? sixty published papers from thirteen journals say "usually not". 2015.

[62] Adriane Chapman and HV Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534. ACM, 2009.

[63] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. Senbazuru: a prototype spreadsheet database management system. *Proceedings of the VLDB Endowment*, 6(12):1202–1205, 2013.

[64] H. Choi and H. Varian. Predicting the present with Google Trends. Technical report, Google, Inc., 2011.

[65] Hyunyoung Choi and Hal Varian. Predicting initial claims for unemployment benefits. *Google Inc*, pages 1–5, 2009.

[66] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469. IEEE, 2013.

[67] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

[68] François Combes. Empirical evaluation of economic order quantity model for choice of shipment size in freight transport. *Transportation Research Record: Journal of the Transportation Research Board*, (2269):92–98, 2012.

[69] Steven J Davis, R Jason Faberman, and John C Haltiwanger. The establishment-level behavior of vacancies and hiring. Technical report, National Bureau of Economic Research, 2010.

[70] Fernando De la Torre and Michael J Black. Robust principal component analysis for computer vision. In *Computer Vision, 2001. ICCV 2001. Eighth IEEE International Conference*, volume 1, pages 362–369. IEEE, 2001.

[71] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[72] David J DeWitt, Shahram Ghandeharizadeh, Donovan Schneider, Allan Bricker, Hui-I Hsiao, Rick Rasmussen, et al. The gamma database machine project. *Knowledge and Data Engineering, IEEE Transactions on*, 2(1):44–62, 1990.

[73] J. A. Doornik. Improving the timeliness of data on influenza-like illnesses using Google Trends. Technical report, Oxford University, 2010.

[74] Liran Einav, Chiara Farronato, Jonathan D Levin, and Neel Sundaresan. Sales mechanisms in online markets: What happened to internet auctions? Technical report, National Bureau of Economic Research, 2013.

[75] Liran Einav, Dan Knoepfle, Jonathan D Levin, and Neel Sundaresan. Sales taxes and internet commerce. Technical report, National Bureau of Economic Research, 2012.

[76] Liran Einav, Theresa Kuchler, Jonathan D Levin, and Neel Sundaresan. Learning from seller experiments in online markets. Technical report, National Bureau of Economic Research, 2011.

[77] Liran Einav and Jonathan Levin. Economics in the age of big data. *Science*, 346(6210):1243089, 2014.

[78] Employment and Training Administration. Unemployment insurance weekly claims report. US Department of Labor, September 2013. Online; accessed 18 January 2016.

[79] Employment and Training Administration. Unemployment insurance weekly claims report. US Department of Labor, September 2013. Online; accessed 18 January 2016.

[80] Trifacta Wrangler Enterprise. Trifacta wrangler (2015), 2016.

[81] Hugo Jair Escalante, Carlos A Hérnadez, Luis Enrique Sucar, and Manuel Montes. Late fusion of heterogeneous methods for multimedia image retrieval. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 172–179. ACM, 2008.

[82] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226. ACM, 1996.

[83] Daniele Fanelli, Rodrigo Costas, and John PA Ioannidis. Meta-assessment of bias in science. *Proceedings of the National Academy of Sciences*, page 201618569, 2017.

[84] Robert Fourer, David M Gay, and Brian Kernighan. *Ampl*, volume 117. Boyd & Fraser Danvers, MA, 1993.

[85] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[86] Tobias Gass, Tobias Weyand, Thomas Deselaers, and Hermann Ney. FIRE in ImageCLEF 2007: Support vector machines and logistic models to fuse image descriptors for photo retrieval. In *Advances in Multilingual and Multimodal Information Retrieval*, pages 492–499. Springer, 2008.

[87] Michael Gelman, Shachar Kariv, Matthew D Shapiro, Dan Silverman, and Steven Tadelis. Harnessing naturally occurring data to measure the response of spending to income. *Science*, 345(6193):212–215, 2014.

[88] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[89] J. Ginsberg, M. H. Mohebbi, R. Patel, L. Brammer, M. S. Smolinksi, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, February 2009.

[90] S. Goel, J.M. Hofman, S. Lehaie, D.M. Pennock, and D.J. Watts. Predicting consumer behavior with web search. *Proceedings of the National Academy of Sciences*, 2010.

[91] Sharad Goel, Duncan J Watts, and Daniel G Goldstein. The structure of online diffusion networks. In *Proceedings of the 13th ACM conference on electronic commerce*, pages 623–638. ACM, 2012.

[92] Jim Gray, David T Liu, Maria Nieto-Santisteban, Alex Szalay, David J DeWitt, and Gerd Heber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, 2005.

[93] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–330. ACM, 2011.

[94] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[95] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[96] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[97] Jonathan Hendershott. Sending pink slips to a war zone. New York Post, July 2014. Online; accessed 12 January 2016.

[98] Herodotos Herodotou, Fei Dong, and Shivnath Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *SoCC*, 2011.

[99] Bart Hobijn and Ayşegül Şahin. Beveridge curve shifts across countries since the great recession. *IMF Economic Review*, 61(4):566–600, 2013.

[100] Liangjie Hong and Brian D Davison. Empirical study of topic modeling in Twitter. In *Workshop on Social Media Analytics*, 2010.

[101] Bill Howe and Daniel Halperin. Advancing declarative query in the long tail of science. *IEEE Data Eng. Bull.*, 35(3):16–26, 2012.

[102] Xian-Sheng Hua and Yong Rui. Content-based multimedia retrieval. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[103] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F Naughton. On the provenance of non-answers to queries over extracted data. *Proceedings of the VLDB Endowment*, 1(1):736–747, 2008.

[104] Ihab F Ilyas, Walid G Aref, and Ahmed K Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 13(3):207–221, 2004.

[105] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 683–698. ACM, 2017.

[106] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan PC Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, et al. H-store: a high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 1(2):1496–1499, 2008.

[107] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[108] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. Snipsuggest: Context-aware autocompletion for sql. *Proceedings of the VLDB Endowment*, 4(1):22–33, 2010.

[109] Miles Kimball, Helen Levy, Fumio Ohtake, and Yoshiro Tsutsui. Unhappiness after hurricane katrina. Technical report, National Bureau of Economic Research, 2006.

[110] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002.

[111] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, volume 1, pages 2–1, 2013.

[112] Vasileios Lampos and Nello Cristianini. Nowcasting events from the social web with statistical learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(4):72, 2012.

[113] David M Lazer, Ryan Kennedy, Gary King, and Alessandro Vespignani. The parable of Google Flu: traps in big data analysis. 2014.

[114] David Leinweber. Stupid data miner tricks: Overfitting the S&P500. *The Journal of Investing*, 16(1):15–22, 2007.

[115] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[116] Andrew Makhorin. Glpk (gnu linear programming kit). *http://www.gnu.org/software/glpk/*, 2008.

[117] Michael E Mann, Jose D Fuentes, and Scott Rutherford. Reply to'tree rings and volcanic cooling'. *Nature Geoscience*, 5(12):837–838, 2012.

[118] Michael E Mann, Jose D Fuentes, and Scott Rutherford. Underestimation of volcanic cooling in tree-ring-based reconstructions of hemispheric temperatures. *Nature Geoscience*, 5(3):202–205, 2012.

[119] Jonathan Meer and Jeremy West. Effects of the minimum wage on employment dynamics. Technical report, National Bureau of Economic Research, 2013.

[120] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 469–478. ACM, 2008.

[121] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Reverse data management. *Proceedings of the VLDB Endowment*, 4(12), 2011.

[122] Alexandra Meliou and Dan Suciu. Tiresias: the database oracle for how-to queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 337–348. ACM, 2012.

[123] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015.

[124] Panagiotis T Metaxas, Eni Mustafaraj, and Daniel Gayo-Avello. How (not) to predict elections. In *IEEE SocialCom*, 2011.

[125] Donald Metzler, Susan Dumais, and Christopher Meek. *Similarity measures for short segments of text.* Springer, 2007.

[126] Morten Middelfart and Torben Bach Pedersen. Implementing sentinels in the targit bi suite. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1187–1198. IEEE, 2011.

[127] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL-IJCNLP*, 2009.

[128] Lewis Mitchell, Morgan R Frank, Kameron Decker Harris, Peter Sheridan Dodds, and Christopher M Danforth. The geography of happiness: Connecting Twitter sentiment and expression, demographics, and objective characteristics of place. *PloS one*, 8(5), 2013.

[129] M. Mohebbi, D. Vanderkam, J. Kodysh, R. Schonberger, H. Choi, and S. Kumar. Google Correlate whitepaper. Technical report, Google, Inc., 2011.

[130] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. From regular expressions to nested words: Unifying languages and query execution for relational and XML sequences. *VLDB*, 2010.

[131] Kevin Mullane, Michael Williams, et al. Bias in research: the rule rather than the exception? 2013.

[132] Bruce A Murtagh and Michael A Saunders. MINOS 5.4 User's Guide, Report SOL 83-20R, Systems Optimization Laboratory, 1983.

[133] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[134] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[135] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in action.* Manning Shelter Island, 2011.

[136] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[137] Andrew Perrin. Social media usage: 2005-2015. 2015.

[138] Kriengkrai Porkaew and Kaushik Chakrabarti. Query refinement for multimedia similarity retrieval in MARS. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 235–238. ACM, 1999.

[139] David Potere, Neal Feierabend, Alan H Strahler, and Eddie A Bright. Wal-mart from space. *Photogrammetric Engineering & Remote Sensing*, 74(7):913–919, 2008.

[140] Press Release. Ugg for men launches new "pink slip" integrated campaign for holiday 2012 featuring tom brady. Reuters, November 2012. Online; accessed 18 January 2016.

[141] Y. Qingyu, P. Geng, L. Ying, and L. Benfu. A prediction study on the car sales based on web search data. In *International Conference on E-Business and E-Government (ICEE)*, 2011.

[142] Leena Rao. Twitter makes hashtags more #useful. Techcrunch.com, July 2009. Online; accessed 12 January 2016.

[143] Christopher Re, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[144] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[145] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[146] Steven L Scott and Hal Varian. Bayesian variable selection for nowcasting economic time series. In *Economics of Digitization*. University of Chicago Press, 2014.

[147] A Shapp. Variation in the use of twitter hashtags. *Qualifying paper in sociolinguistics, New York University*, 2014.

[148] Paul Sonne and Anton Troianovski. Ukraine says plane downed by missile probably fired from russia. The Wall Street Journal, July 2014. Online; accessed 12 January 2016.

[149] Michael Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[150] Chris Stoughton, Robert H Lupton, Mariangela Bernardi, Michael R Blanton, Scott Burles, Francisco J Castander, AJ Connolly, Daniel J Eisenstein, Joshua A Frieman, GS Hennessy, et al. Sloan digital sky survey: early data release. *The Astronomical Journal*, 123(1):485, 2002.

[151] Alexander S Szalay, Jim Gray, Ani R Thakar, Peter Z Kunszt, Tanu Malik, Jordan Raddick, Christopher Stoughton, and Jan vandenBerg. The sdss skyserver: public access to the sloan digital sky server data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 570–581. ACM, 2002.

[152] Aliaksandr Talaika, Joanna Biega, Antoine Amarilli, and Fabian M Suchanek. Ibex: Harvesting entities from the web using unique identifiers. In *WebDB*. ACM, 2015.

[153] Ameet Talwalkar, Jesse Liptrap, Julie Newcomb, Christopher Hartl, Jonathan Terhorst, Kristal Curtis, Ma'ayan Bresler, Yun S Song, Michael I Jordan, and David Patterson. Smash: a benchmarking toolkit for human genome variant calling. *Bioinformatics*, 30(19):2787–2795, 2014.

[154] DBC Teradata. 1012 data base computer systems manual, release 1.3, teradata corp., document no. Technical report, C10-0001-00, February, 1985.

[155] The Conference Board. The conference board help wanted online. conferenceboard.org, January 2016. Online; accessed 18 January 2016.

[156] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[157] Quoc Trung Tran and Chee-Yong Chan. How to conquer why-not questions. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 15–26. ACM, 2010.

[158] George Tsatsaronis, Iraklis Varlamis, and Michalis Vazirgiannis. Text relatedness based on a word thesaurus. *Journal of Artificial Intelligence Research*, 37(1):1–40, 2010.

[159] Peter D. Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *ECML*, pages 491–502, 2001.

[160] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. S ee db: efficient data-driven visualization recommendations to support visual analytics. *Proceedings of the VLDB Endowment*, 8(13):2182–2193, 2015.

[161] Amy Voida, Ellie Harmon, and Ban Al-Ani. Homebrew databases: Complexities of everyday information management in nonprofit organizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 915–924. ACM, 2011.

[162] Grady Ward. The Institute for Language, Speech and Hearing - The Moby Lexicon Project. `http://icon.shef.ac.uk/Moby/`, January 2015.

[163] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[164] Douglas B Williams and Vijay Madisetti. *Digital signal processing Handbook.* CRC Press, Inc., 1997.

[165] Bo Wu and Craig A Knoblock. An iterative approach to synthesize data transformation programs. In *IJCAI*, pages 1726–1732, 2015.

[166] Bo Wu, Pedro Szekely, and Craig A Knoblock. Learning data transformation rules through examples: Preliminary results. In *Proceedings of the Ninth International Workshop on Information Integration on the Web*, page 8. ACM, 2012.

[167] Bo Wu, Pedro Szekely, and Craig A Knoblock. Minimizing user effort in transforming data by example. In *Proceedings of the 19th international conference on Intelligent User Interfaces*, pages 317–322. ACM, 2014.

[168] L. Wu and E. Brynjolfsson. The future of prediction: How Google searches foreshadow housing prices and sales. Technical report, MIT Sloan School of Management, 2009.

[169] Lynn Wu and Erik Brynjolfsson. The future of prediction: How google searches foreshadow housing prices and sales. In *Economics of Digitization.* U. of Chicago Press, 2014.

[170] Jun-Ming Xu, Aniruddha Bhargava, Robert Nowak, and Xiaojin Zhu. Socioscope: Spatio-temporal signal recovery from social media. In *ECML-PKDD*, 2012.

[171] Junjie Yao, Bin Cui, Liansheng Hua, and Yuxin Huang. Keyword query reformulation on structured data. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 953–964. IEEE, 2012.

[172] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.

[173] Matei Zaharia, William J Bolosky, Kristal Curtis, Armando Fox, David Patterson, Scott Shenker, Ion Stoica, Richard M Karp, and Taylor Sittler. Faster and more accurate sequence alignment with snap. *arXiv preprint arXiv:1111.5572*, 2011.

[174] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[175] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM, 2013.

[176] Ce Zhang, Arun Kumar, and Christopher Ré. Materialization optimizations for feature selection workloads. In *SIGMOD*, 2014.

[177] Ce Zhang, Arun Kumar, and Christopher Ré. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)*, 41(1):2, 2016.