An Empirical Comparison of Various Online Binary Classification Algorithms

Undergraduate Statistics Honors Research Thesis

Xinyan Han

Research Supervisor: Prof. Ambuj Tewari

Graduate Student Supervisor: Sougata Chaudhuri

Apr. 25th 2015

This research is about empirical comparison about the following five algorithms: online gradient descent (OGD) algorithm for learning linear classification function, OGD algorithm for learning kernel based non-linear classification function with no budget restriction, OGD algorithm for learning kernel based non-linear classification function with budget restriction, fast bounded OGD algorithm for scalable kernel based online learning – ICML 2012, online boosting – ICML 2012.

To implement these algorithms, we used four datasets from UCI machine-learning Repository with size ranging from 1000 to 19020 as below:

- German Credit dataset (24 attributes, 1000 instances, class:1,2)
- Spambase dataset (57 attributes, 4601 instances, class:0,1)
- Magic Gamma Telescope data (10 attributes, 19020 instances, class:g,h)
- EEG Eye State Data Set (14 attributes, 14980 instances, class: 0,1)

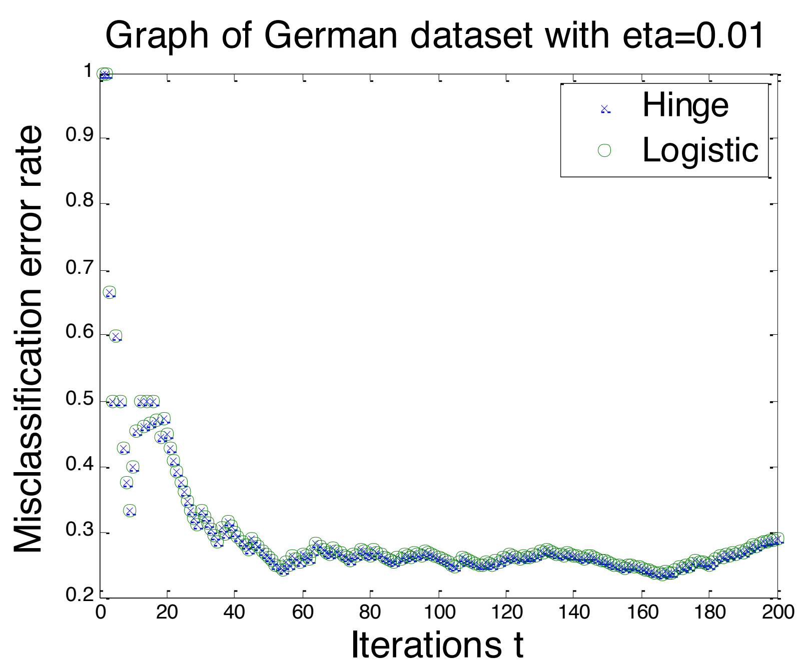
For all datasets, the feature vectors $\{x\}$ are normalized ($x = \frac{x}{\|x\|^2}$).

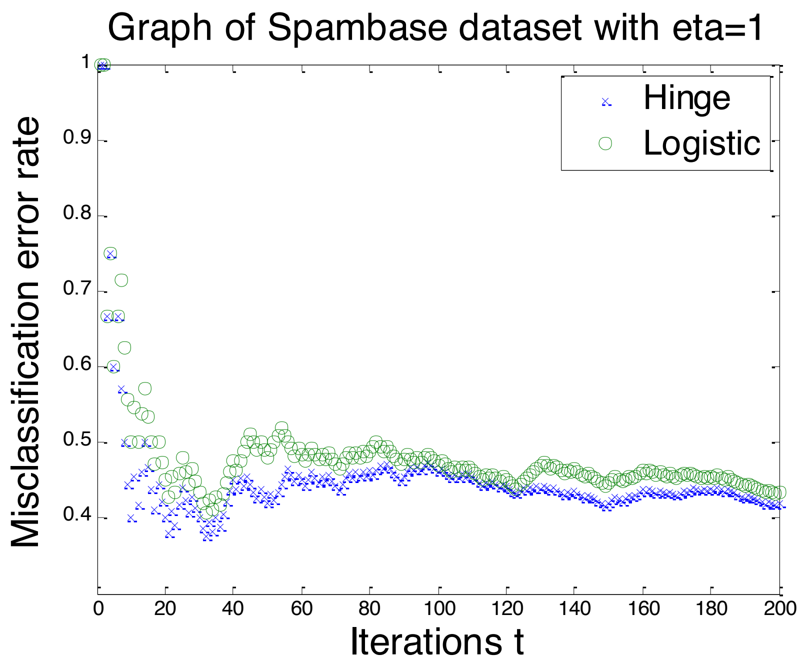
1. Online Gradient Descent for Learning Linear Classification Function

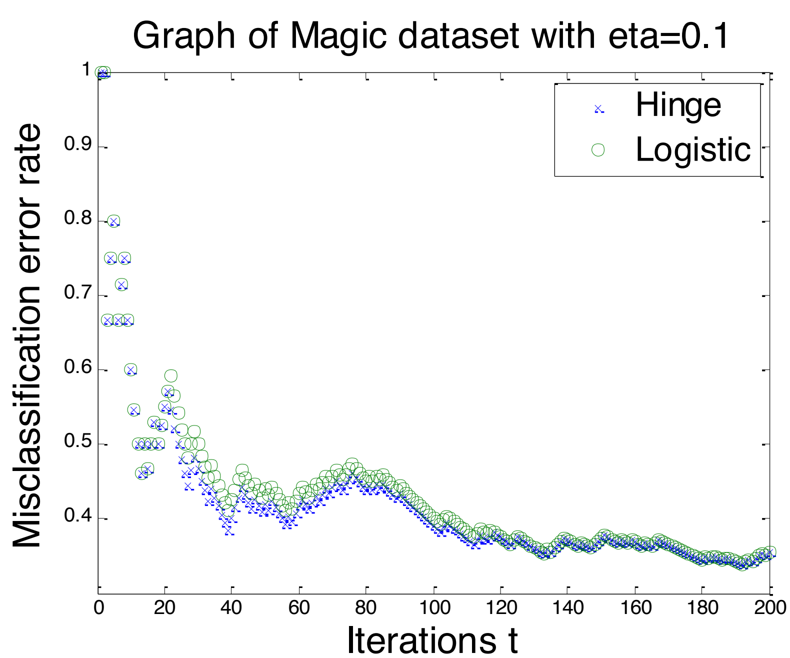
The misclassification error rate is calculated as $\frac{\sum_{i=1}^{t} I(\hat{y}_i \neq y_i)}{t}$, where t = {1, 2, …, T} and T is the dataset size varying depending on different datasets. Below is the table of misclassification error rates for the four datasets using hinge and logistic function with learning rate eta 0.01, 0.1, 1, 10, $\frac{1}{\sqrt{t}}$ and **without projection**.

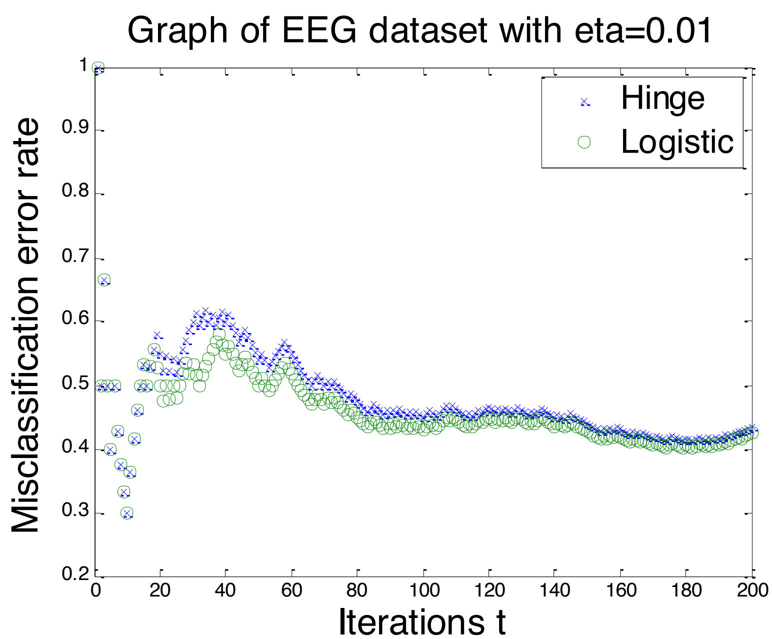
	T	Eta = 0.01	Eta = 0.1	Eta = 1	Eta = 10	Eta=1/sqrt(t)
German, Hinge	1000	0.3007	0.3007	0.3317	0.3696	0.3007
German, Logistic	1000	0.3007	0.3067	0.3237	0.3816	0.3027
Spambase, Hinge	4601	0.3942	0.3777	0.3701	0.3850	0.3942
Spambase, Logistic	4601	0.3948	0.3742	0.3618	0.3979	0.3896
Magic, Hinge	19020	0.3143	0.2835	0.3135	0.3578	0.2933
Magic, Logistic	19020	0.3147	0.2872	0.2998	0.3693	0.2989
EEG, Hinge	14980	0.4490	0.4575	0.4950	0.4747	0.4490
EEG, Logistic	14980	0.4492	0.4654	0.4928	0.4948	0.4508

Below are the graphs of misclassification error rates with eta which gives the smallest error rate for each of the four dataset:




Graph of Spambase dataset with eta=1
Misclassification error rate
Iterations t
Hinge
Logistic
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20
40
60
80
100
120
140
160
180
200

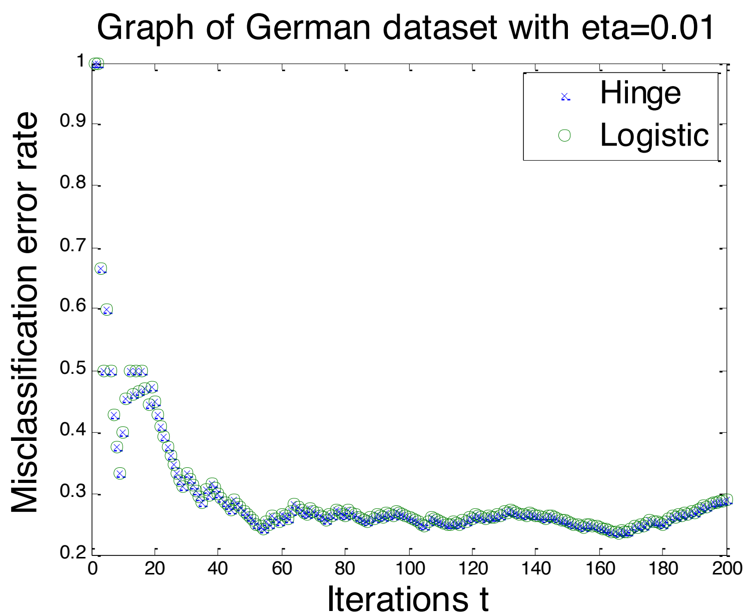

Graph of Magic dataset with eta=0.1
Misclassification error rate
Iterations t
Hinge
Logistic
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20
40
60
80
100
120
140
160
180
200

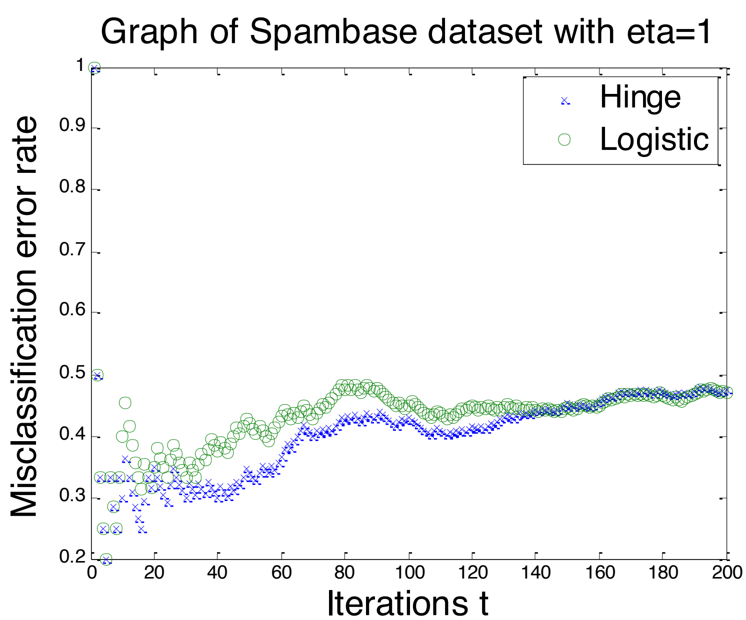


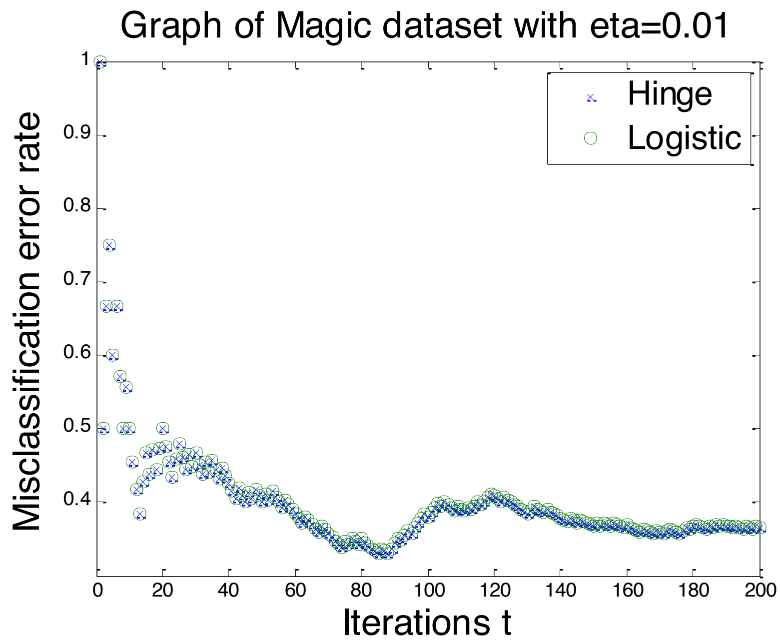
Below is the table of misclassification error rates for the four datasets using hinge and logistic function with learning rate 0.01, 0.1, 1, 10, $\frac{1}{\sqrt{t}}$ and **with projection**. We project function parameter of each iteration on an l_2 ball of unit radius. The misclassification error rate is calculated in the same way as before.

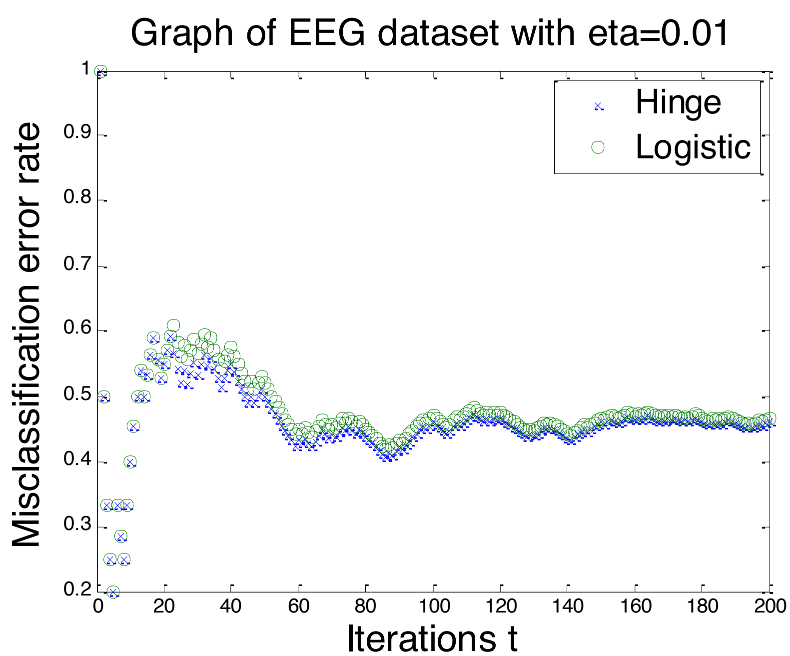
	T	Eta = 0.01	Eta = 0.1	Eta = 1	Eta = 10	Eta=1/sqrt(t)
German, Hinge	1000	0.3007	0.3007	0.3716	0.4306	0.3007
German, Logistic	1000	0.3007	0.3027	0.3566	0.4306	0.3007
Spambase, Hinge	4601	0.3942	0.3974	0.4694	0.4718	0.3948
Spambase, Logistic	4601	0.3942	0.3970	0.4561	0.4724	0.3839
Magic, Hinge	19020	0.3511	0.3512	0.4166	0.4578	0.3515
Magic, Logistic	19020	0.3380	0.3373	0.3943	0.4578	0.3377
EEG, Hinge	14980	0.4494	0.4590	0.4829	0.4940	0.4489
EEG, Logistic	14980	0.4493	0.4677	0.4882	0.4940	0.4496

Below are the graphs of misclassification error rates with eta which gives the smallest error rate for each of the four dataset:









Summary: from the tables and graphs we conclude that the misclassification error rates with projection and without projection are very similar. The only exception is Magic dataset since the error rates without projection are smaller than those with projection. Generally, when eta equals to 0.01, the error rates are relatively small. Error rates using Hinge and Logistic differ the most when eta is 1 or 10, while less when eta is 0.01 or $\frac{1}{\sqrt{t}}$.

2. Online Gradient Descent for Learning Kernel Based Non-Linear Classification Function with No Budget Restriction

Since the kernel function is calculated using each pair of <x_i, α_i, η_i>, we need to store each instance and scalar, which is an issue for memory and computational efficiency. In this algorithm, we assume there is no memory and computational constraint. The scalar α has different form for Hinge loss and Logistic functions. The kernel functions we use here are Gaussian and polynomial.

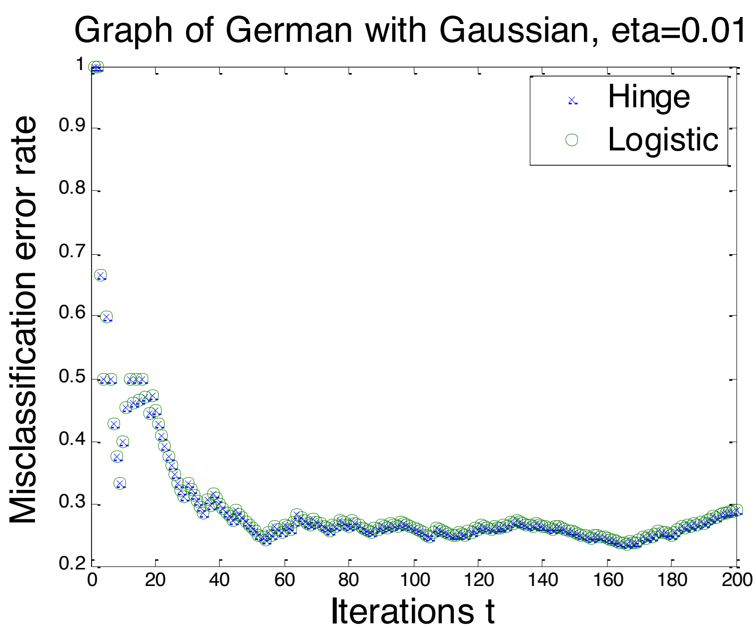
Below is the table of misclassification error rates for the four datasets using hinge and logistic function with learning rate 0.01, 0.1, 1, $\frac{1}{\sqrt{t}}$, and with **Gaussian and Polynomial kernel function** (Gaussian kernel parameter = 1).

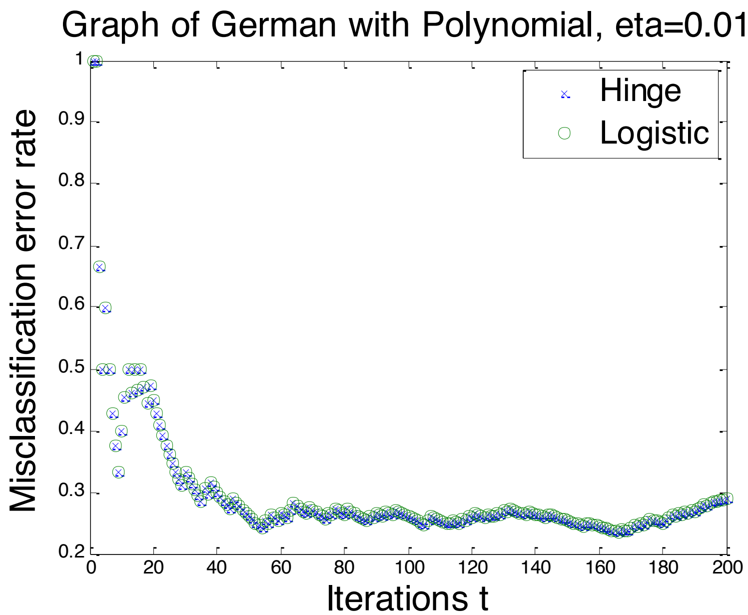
	Kernel	Eta = 0.01	Eta = 0.1	Eta = 1	Eta = 1/sqrt(t)
German, Hinge	Gaussian	0.3007	0.3007	0.3207	0.3007
	Polynomial	0.3007	0.3177	0.3616	0.3097
German, Logistic	Gaussian	0.3007	0.3017	0.3157	0.3037
	Polynomial	0.3007	0.3127	0.3516	0.3067
Spambase, Hinge	Gaussian	0.3942	0.3577	0.3533	0.3872
	Polynomial	0.3946	0.3746	0.3818	0.3629
Spambase, Logistic	Gaussian	0.3945	0.3501	0.3438	0.3683
	Polynomial	0.3872	0.3618	0.3831	0.3585
Magic, Hinge	Gaussian	0.3028	0.2607	0.2678	0.2873
	Polynomial	0.2885	0.2696	0.3148	0.2785
Magic, Logistic	Gaussian	0.3046	0.2701	0.2564	0.2891
	Polynomial	0.2932	0.2719	0.3120	0.2843
EEG, Hinge	Gaussian	0.4496	0.4587	0.4847	0.4506
	Polynomial	0.4493	0.4845	0.4871	0.4520
EEG, Logistic	Gaussian	0.4492	0.4638	0.4873	0.4515
	Polynomial	0.4563	0.4845	0.4914	0.4619

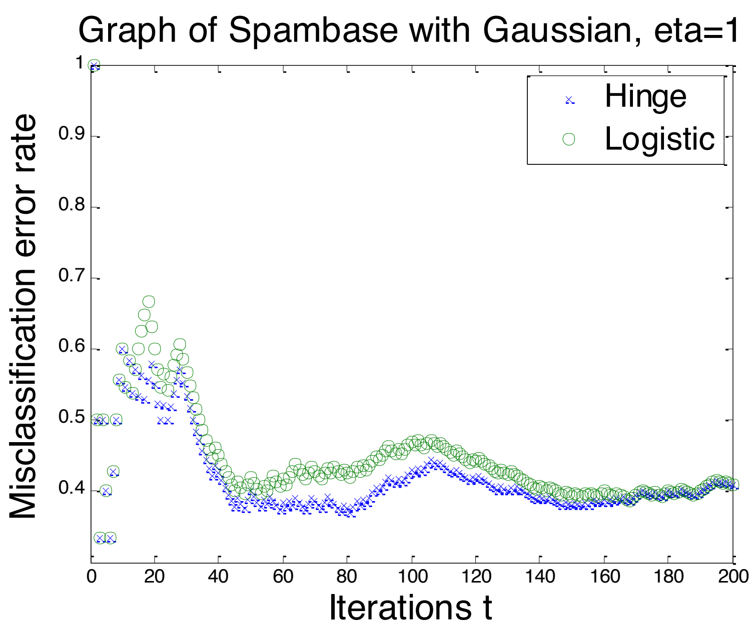
Training time table is as below (in seconds):

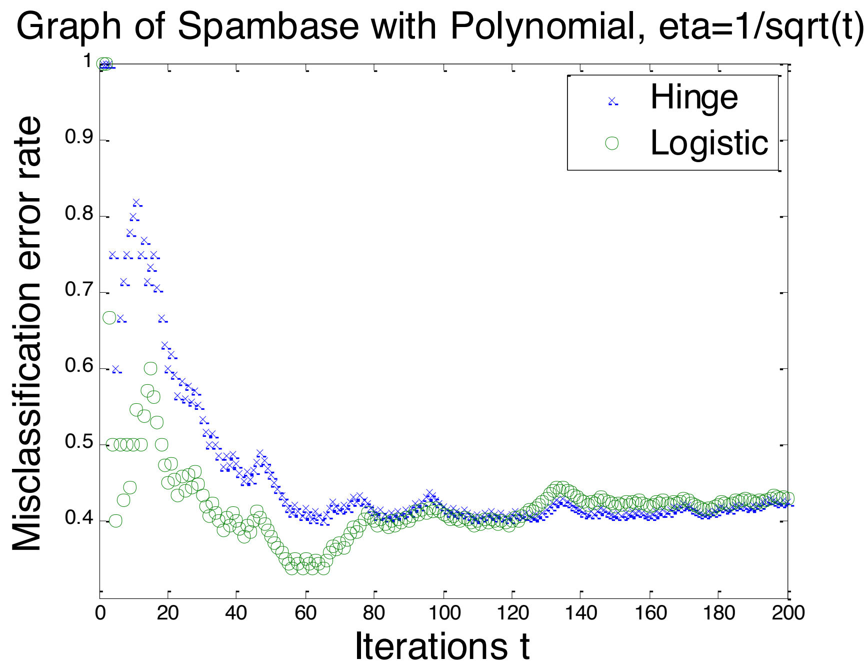
	Kernel	Eta = 0.01	Eta = 0.1	Eta = 1	Eta = 1/sqrt(t)
German	Gaussian	7.377850	7.251028	7.360134	7.336239
	Polynomial	27.809096	27.709677	27.597569	27.991671
Spambase	Gaussian	82.773811	82.985134	82.599790	82.725483
	Polynomial	511.313790	510.557001	530.717847	531.338944
Magic	Gaussian	1309.814441	1314.569120	1308.950468	1293.261339
	Polynomial	3348.883527	3456.564090	3363.526899	3407.951616
EEG	Gaussian	881.004934	876.245041	857.471990	875.480789
	Polynomial	2819.752156	2389.510764	2787.654218	2798.046620

Below are the graphs of misclassification error rates with eta which gives the smallest error rate for each of the four dataset:

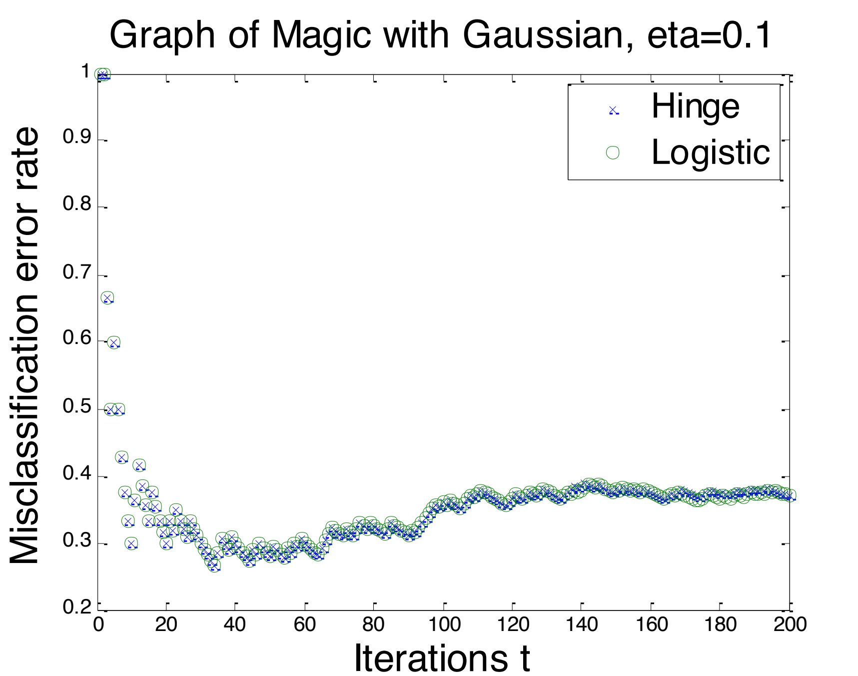



Graph of German with Polynomial, eta=0.01
Hinge
Logistic
Misclassification error rate
Iterations t
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
120
140
160
180
200

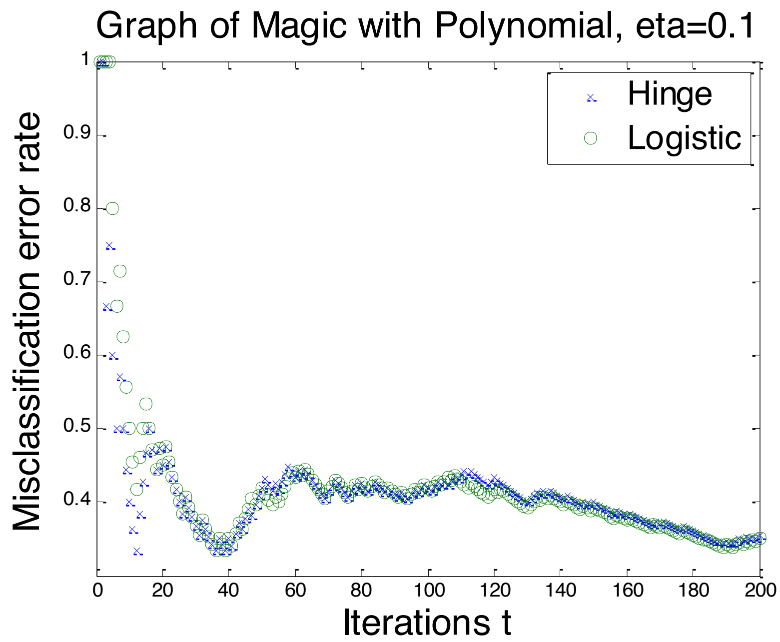

Graph of Spambase with Gaussian, eta=1
Hinge
Logistic
Misclassification error rate
Iterations t
1
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120
140
160
180
200

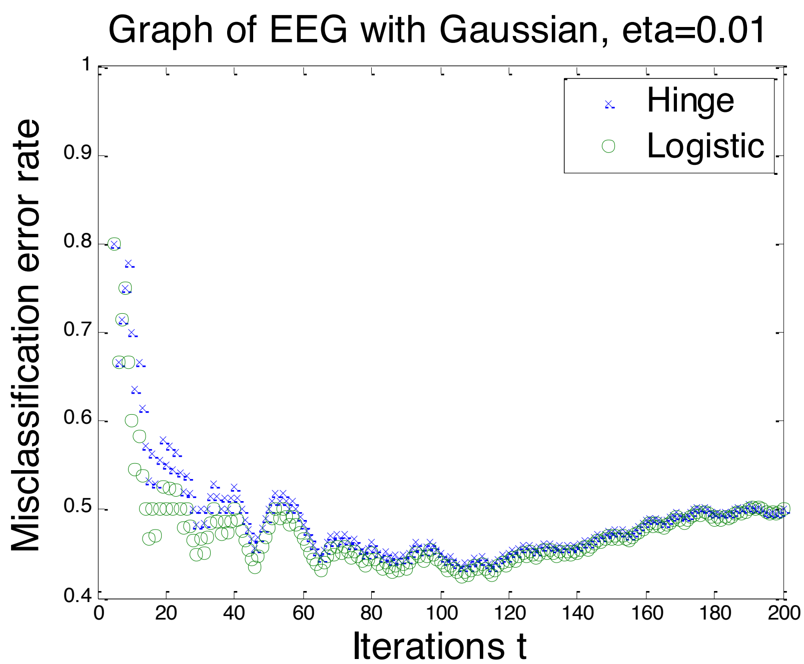


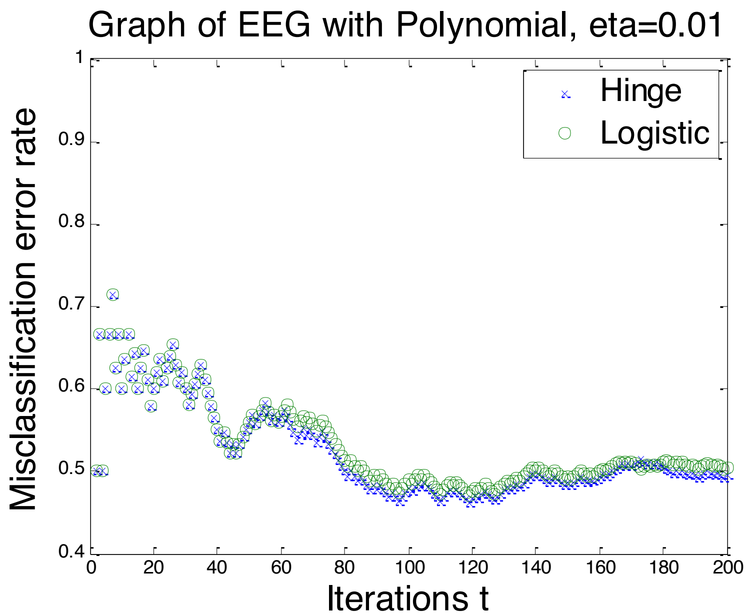
Graph of Spambase with Polynomial, eta=1/sqrt(t)
Hinge
Logistic
Misclassification error rate
Iterations t



Graph of Magic with Gaussian, eta=0.1
Hinge
Logistic
Misclassification error rate
Iterations t


Graph of Magic with Polynomial, eta=0.1
Misclassification error rate
Iterations t
Hinge
Logistic
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20
40
60
80
100
120
140
160
180
200


Graph of EEG with Gaussian, eta=0.01
Misclassification error rate
Iterations t
Hinge
Logistic
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20
40
60
80
100
120
140
160
180
200



Summary: The classification error rates using Gaussian kernel is smaller than using Polynomial kernel. Generally, this algorithm performs better when eta is 0.01. The error rates for German dataset are larger using non-budget kernel function than linear function, while for Magic dataset those using non-budget kernel function are smaller than linear function. The Gaussian algorithms perform 3 to 4 times faster than Polynomial algorithms.

3. Online Gradient Descent for Learning Kernel Based Non-Linear Classification Function with Budget Restriction

Using kernel function, we need to store the instances $\{x_t\}$ (and scalar η_t , α_t) at the end of every round. This becomes a problem for memory and computational efficiency. Here we introduce memory budget to deal with this problem.

We assume the size of memory budget is S, and only S tuples can be stored in budget. When the budget is full, we use two methods to discard one of the tuples in budget and include new tuple: Random Discard, Oldest Tuple Discard. When a new tuple has to be stored, the random discarding strategy is to discard a randomly chosen tuple from budget, which the oldest tuple discarding strategy is to discard the oldest tuple from budget. We use the learning rate eta of 0.1, since it gives the best result. S varies for different datasets, because the total number of instances for each dataset is different.

Below is the table of misclassification rates for the four datasets using hinge and logistic loss function with learning rate eta 0.1, Gaussian and polynomial kernel function, and **Random Discard and Oldest Tuple Discard strategies**.

	Kernel	S=100, Random	S=100, Oldest	S=400, Random	S=400, Oldest	S=500, Random	S=500, Oldest
German, Hinge	Gaussian	0.3067	0.3027	0.3037	0.3017	0.3007	0.3007
	Polynomial	0.3347	0.3277	0.3317	0.3227	0.3197	0.3287
German, Logistic	Gaussian	0.3067	0.3057	0.3027	0.3027	0.3017	0.3017
	Polynomial	0.3457	0.3197	0.3107	0.3177	0.3107	0.3157

	Kernel	S=100, Random	S=100, Oldest	S=1000, Random	S=1000, Oldest	S=2000, Random	S=2000, Oldest
Spambase, Hinge	Gaussian	0.4126	0.4129	0.3837	0.3890	0.3751	0.3787
	Polynomial	0.4342	0.4261	0.4046	0.4053	0.3931	0.3848
Spambase, Logistic	Gaussian	0.4137	0.4159	0.3879	0.3787	0.3703	0.3720
	Polynomial	0.4333	0.4381	0.3992	0.3980	0.3757	0.3772

	Kernel	S=100, Random	S=100, Oldest	S=1500, Random	S=1500, Oldest	S=10000, Random	S=10000, Oldest
Magic, Hinge	Gaussian	0.3454	0.3452	0.2922	0.2886	0.2683	0.2684
	Polynomial	0.3652	0.3655	0.3101	0.3078	0.2863	0.2815
Magic, Logistic	Gaussian	0.3452	0.3445	0.2977	0.2972	0.2762	0.2755
	Polynomial	0.3613	0.3581	0.3029	0.3035	0.2786	0.2793

	Kernel	S=100, Random	S=100, Oldest	S=1500, Random	S=1500, Oldest	S=7500, Random	S=7500, Oldest
EEG, Hinge	Gaussian	0.4740	0.4752	0.4663	0.4658	0.4628	0.4641
	Polynomial	0.4829	0.4837	0.4825	0.4759	0.4749	0.4767
EEG, Logistic	Gaussian	0.4859	0.4835	0.4703	0.4749	0.4679	0.4741
	Polynomial	0.4902	0.4862	0.4831	0.4825	0.4788	0.4809

Training time table is as below (in seconds):

	Kernel	S=100, Random	S=100, Oldest	S=400, Random	S=400, Oldest	S=500, Random	S=500, Oldest
German	Gaussian	0.677464	0.678596	2.087489	2.112098	3.122506	3.097645
	Polynomial	2.227651	2.227617	7.290480	7.385192	10.915067	10.852339

	Kernel	S=100, Random	S=100, Oldest	S=1000, Random	S=1000, Oldest	S=2000, Random	S=2000, Oldest
Spambase	Gaussian	4.111473	4.129727	31.392597	31.749946	50.366228	50.718951
	Polynomial	11.859072	11.791451	101.515842	100.243279	258.031435	256.061256

	Kernel	S=100, Random	S=100, Oldest	S=1500, Random	S=1500, Oldest	S=10000, Random	S=10000, Oldest
Magic	Gaussian	15.127512	15.517319	198.108581	197.135372	974.516355	961.951610
	Polynomial	35.493801	34.899790	492.563144	481.319542	3092.643502	4488.062764

	Kernel	S=100, Random	S=100, Oldest	S=1500, Random	S=1500, Oldest	S=7500, Random	S=7500, Oldest
EEG	Gaussian	12.568357	12.489083	153.908331	152.995260	590.489152	574.621233
	Polynomial	38.327984	37.918678	529.323398	528.761239	1711.573384	1718.156244

Summary: The error rates become smaller as the size of budget gets larger. The performances of algorithms using Random Discard technique and Oldest Tuple Discard technique are similar. The performance of Magic dataset improves the most as budget increases. The performances using Gaussian kernels are 2 - 3 times faster than using Polynomial kernels.

4. Fast Bounded OGD Algorithm for Scalable Kernel Based Online Learning – ICML 2012

This algorithm is from the ICML paper that learns Kernel based functions with restricted budget. It uses Uniform and Non-uniform sampling scheme for discarding and employs projection strategy. We used eta = 2^{-3}, lambda = $\frac{2^{-3}}{T^2}$, gamma = 2^0 = 1, because these are the parameters given in the paper. The budget size S varies for different datasets. The conditions of this algorithm are: Hinge and Logistic loss function, Gaussian and Polynomial Kernel, Uniform and Non-uniform sampling discard technique.

Below is the table of misclassification error rates for the four datasets based on these conditions (some results are missing because it was taking too long).

German		S = 100	S = 150	S = 200
Gaussian, uniform	Hinge	0.3247	0.3317	0.3497
	Logistic	0.3257	0.3127	0.3327
Gaussian, non-uniform	Hinge	0.3676	0.3407	0.3636
	Logistic	0.9211	0.8841	0.8531
Polynomial, uniform	Hinge	0.3846	0.3966	0.3896
	Logistic	0.3556	0.3666	0.3586
Polynomial, non-uniform	Hinge	0.4281	0.4271	0.4182
	Logistic	0.9361	0.8922	0.8703

Spambase		S = 100	S = 200	S = 300
Gaussian, uniform	Hinge	0.4118	0.4107	0.4011
	Logistic	0.4092	0.4044	0.3998
Gaussian, non-uniform	Hinge	0.4368	0.4250	0.4148
	Logistic	0.9859	0.9761	0.9607
Polynomial, uniform	Hinge	0.4376	0.4368	0.4218
	Logistic	0.4394	0.4396	0.4203
Polynomial, non-uniform	Hinge	0.4591	0.4442	0.4322
	Logistic	0.9863	0.9767	0.9605

Magic		S = 500	S = 1000	S = 1500
Gaussian, uniform	Hinge	0.3108	0.3012	0.2908
	Logistic	0.3112	0.2936	0.2875
Gaussian, non-uniform	Hinge	0.3277	0.3277	N/A
	Logistic	0.9811	0.9811	N/A

Polynomial, uniform	Hinge	0.3611	0.3496	0.3380
	Logistic	0.3629	0.3533	0.3430
Polynomial, non-uniform	Hinge	0.3717	N/A	N/A
	Logistic	0.9829	N/A	N/A

Training time table is as below (in seconds):

German	S = 100	S = 150	S = 200
Gaussian, uniform	1.354158	1.946180	2.487288
Gaussian, non-unif	4.403808	4.423398	7.624706
Poly, uniform	5.222308	9.652918	9.867392
Poly, non-uniform	23.477671	38.328848	59.432683

Spambase	S = 100	S = 200	S = 300
Gaussian, uniform	26.559091	26.335963	26.689395
Gaussian, non-unif	38.484068	127.391326	265.525025
Poly, uniform	10.467371	19.840022	29.142045
Poly, non-uniform	100.577733	349.353676	734.266628

Magic	S = 500	S = 1000	S = 1500
Gaussian, uniform	71.039601	135.948865	202.129344
Gaussian, non-unif	2462.301618	2462.301618	N/A
Poly, uniform	193.490654	372.809591	549.162968
Poly, non-uniform	7970.532109	N/A	N/A

Summary: We observe that when using Hinge loss and Gaussian kernel for German dataset, non-uniform does worse than uniform sampling. This is the same as using Hinge loss and Polynomial kernel for German dataset, but Polynomial kernel is not used in the paper. These situations also happen for Spambase, Magic and EEG datasets. This shows that the non-uniform sampling scheme is worse than uniform sampling scheme, which is opposed to the theory in the paper.

The Logistic loss with non-uniform sampling gives high error rates, possibly because the parameter $\{\alpha\}$ become zero. This happens because somewhere in the non-uniform probabilities, something is going wrong. It is unclear what is going wrong. The paper also did not use Logistic loss.

5. Online Boosting – ICML 2012

Two datasets are used for empirical evaluation in this algorithm: German and Australian datasets. The information of Australian dataset is as below:

- Australian Credit Approval dataset (14 attributes, 690 instances, class: 0, 1)

Below is the table of misclassification error rates for these two datasets using different values of theta and gamma.

	Theta=0.03, gamma=0.05, N=100	Theta=0.03, gamma=0.05, N=400	Theta=0.01, gamma=0.02, N=100	Theta=0.01, gamma=0.02, N=400
German	0.4260	0.4276	0.4276	0.4284
Australian	0.4359	0.4275	0.4368	0.4342

Code for the these five algorithms are attached in the following pages:

Code of OGD for learning linear classification function:

```
% Online Gradient Descent Algorithm for Learning Linear Classification
Function
% Main function for learning linear classifier via OGD. In this function, we
read the text file
% one instance-label pair at a time, mimicking the online setting. This
% also reduces stress in memory as we dont need to load the entire
% instance-label file on main memory
function OGDLinear()

% Setting the variables
% Since we are running on the german dataset, we know featuresize is 24. We
% need to change the variables for other datasets.
iter=1; featuresize=14;instsize=14980; count=0;
wHinge=repmat(0,featuresize,1);wLogistic=repmat(0,featuresize,1);
cumlosshinge=0; cumlosslogistic=0;
eta=0.01;
Hloss=repmat(0,instsize,1);Lloss=repmat(0,instsize,1);
% iter indicate number of times (iterations) the entire data file will be
scanned. iter=2 means each instance
% in the data file will be seen twice and so on.

% The spambase.data file has instances of the 2 classes separated. That is,
% all spams occur first and then all emails. Though we do not assume any
% distribution on the data and hence the performance should not be
% affected, at least theoretically, it makes sense to permute the rows in
spambase file.
% That is, all rows will be read into a matrix, permuted in a random way
% and then written back in a new text file, from which instances would be
% read 1 line at a time. This should be followed for all datasets where
% such a problem occurs.
M=csvread('C:\Users\hxinyan\Documents\MATLAB\eeg.txt');

% for magic04 dataset
% f=fopen('C:\Users\hxinyan\Documents\MATLAB\magic04.data','r');
% s=repmat('%f',1,10);
% s=strcat(s,'%s');
% A=textscan(f,s,'delimiter',',');
% D=cell2mat(A(1:10));
% C=cell2mat(A{11});
% E=repmat(0,19020,1);
% for i=1:19020
%    if(C(i)=='g')
%        E(i)=1;
%    else
%        E(i)=-1;
%    end
% end
% M=horzcat(D,E);

rowperm=randperm(size(M,1));
M=M(rowperm,:);
dlmwrite('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt',M);

for it=1:iter
```

```
    f = fopen('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt');
    while 1
        count=count+1;
        l = fgetl(f);
        if ~ischar(l), break; end;
        % Reading the joint instance-label pair, one at a time
        xy= sscanf(l,'%f,');

        % Separating instance and label. We dont need to store the
        % instances over time,
        x=xy(1:end-1);
        y=xy(end);
        % 1 indicates spam, 0 indicates email. Hence, the following
conversions
        if(y==0)
            y=-1;
        else
            y=1;
        end

        % Normalizing x
        x=x/norm(x);
        % Running the OGD, with different choice of losses.
        % First for hinge loss
%        eta=1/sqrt(count);
        [predhinge, gradhinge]= hinge(wHinge,x,y);
        cumlosshinge=cumlosshinge+ (predhinge~=y);
        Hloss(count)=cumlosshinge/count;
        wHinge=wHinge- eta*gradhinge;
        % The projection step is optional. We are using projection on an
        % $l-2$ norm ball of unit radius (U=1).
%        wHinge=min(norm(wHinge),1)*(wHinge/norm(wHinge));

        % Next for logistic loss
%        eta=1/sqrt(count);
        [predlogistic, gradlogistic]= logistic(wLogistic,x,y);
        cumlosslogistic=cumlosslogistic+ (predlogistic~=y);
        Lloss(count)=cumlosslogistic/count;
        wLogistic=wLogistic- eta*gradlogistic;
        % The projection step is optional. We are using projection on an
        % $l-2$ norm ball of unit radius
%       wLogistic=min(norm(wLogistic),1)*(wLogistic/norm(wLogistic));

    end
end
disp(cumlosshinge/count);
disp(cumlosslogistic/count);
num=1:200;Hloss=Hloss(num); Lloss=Lloss(num);
plot(num,Hloss,'x',num,Lloss,'o');
lg=legend('Hinge','Logistic'); set(lg,'FontSize',16)
xlabel('Iterations t','FontSize',18); ylabel('Misclassification error
rate','FontSize',18);
title('Graph of EEG dataset with eta=0.01','FontSize',18);
end

% Function to calculate prediction and gradient based on hinge loss
function [predhinge,gradhinge]=hinge(w,x,y)
```

```
predhinge=sign(dot(w,x));
gradhinge=((1-y*dot(w,x))>=0)*(-1*y*x);
end

%Function to calculate prediction and gradient based on logistic loss
function [predlogistic,gradlogistic]=logistic(w,x,y)
predlogistic=sign(dot(w,x));
gradlogistic=(exp(-1*y*dot(w,x))/(1+exp(-1*y*dot(w,x))))*(-1*y*x);
end
```

Code of OGD for learning kernel based non-linear classification function with no budget restriction:

```
% Online Gradient Descent Algorithm for Learning Kernel Classification
% Function with no Budget Restriction. In this function, we read the text
file
% one instance-label pair at a time, mimicking the online setting.

function KernelOGDnB()
tic
% Setting the variables
% Since we are running on the german dataset, we know featuresize is 24. We
% need to change the variables for other datasets. instsize is the size of
% the budget. Since there is no budget restriction, this is an upper limit
% on the number of instances the program is going to see.
iter=1; instsize=14980; count=0; cumlosshinge=0; cumlosslogistic=0;
Bhinge=cell(instsize,2); Blogistic=cell(instsize,2);
eta=0.01;
Hloss=repmat(0,instsize,1);Lloss=repmat(0,instsize,1);
M=csvread('C:\Users\hxinyan\Documents\MATLAB\eeg.txt');

% for magic04 dataset
% f=fopen('C:\Users\hxinyan\Documents\MATLAB\magic04.data','r');
% s=repmat('%f',1,10);
% s=strcat(s,'%s');
% A=textscan(f,s,'delimiter',',');
% D=cell2mat(A(1:10));
% C=cell2mat(A{11});
% E=repmat(0,19020,1);
% for i=1:19020
%    if(C(i)=='g')
%        E(i)=1;
%    else
%        E(i)=-1;
%    end
% end
% M=horzcat(D,E);

rowperm=randperm(size(M,1));
M=M(rowperm,:);
dlmwrite('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt',M);

for it=1:iter
    f = fopen('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt');
    while 1
        count=count+1;
        l = fgetl(f); %Read line from file, removing newline characters
        if ~ischar(l), break; end;
        % Reading the joint instance-label pair, one at a time
        xy= sscanf(l,'%f,');

        % store normalized x into B
        Bhinge{count,2}=xy(1:end-1)/norm(xy(1:end-1));
        Blogistic{count,2}=xy(1:end-1)/norm(xy(1:end-1));
```

```
        y=xy(end);
        % 1 indicates spam, 0 indicates email. Hence, the following
conversions
        if(y==0)
            y=-1;
        else
            y=1;
        end
        display(count);

        % Running the OGD, with different choice of losses.
        % First for hinge loss. choice here indicates the type of kernel
        % choice=1 means gaussian, choice=2 means polynomial
        choice=1;
%       eta=1/sqrt(count);
        [predhinge, alpha]= hinge(y,Bhinge,count,choice);
        cumlosshinge=cumlosshinge+ (predhinge~=y);
        Hloss(count)=cumlosshinge/count;
        Bhinge{count,1}=eta*alpha;

        % Next for logistic loss. choice here indicates the type of kernel
        % choice=1 means gaussian, choice=2 means polynomial
        choice=1;
%       eta=1/sqrt(count);
        [predlogistic, alpha]= logistic(y,Blogistic,count,choice);
        cumlosslogistic=cumlosslogistic+ (predlogistic~=y);
        Lloss(count)=cumlosslogistic/count;
        Blogistic{count,1}= eta*alpha;

    end
end
disp(cumlosshinge/count);
disp(cumlosslogistic/count);
toc
num=1:200;Hloss=Hloss(num); Lloss=Lloss(num);
plot(num,Hloss,'x',num,Lloss,'o');
lg=legend('Hinge','Logistic'); set(lg,'FontSize',16)
xlabel('Iterations t','FontSize',18); ylabel('Misclassification error
rate','FontSize',18);
title('Graph of EEG with Gaussian, eta=0.01','FontSize',18);
end



% Function to calculate prediction and gradient based on hinge loss
function [predhinge,alpha]=hinge(y,B,count,choice)
%calculate the sum of scalar times kernels
sumker=0;
for i=1:count-1
    if(choice==1)
        sumker=sumker+B{i,1}*gaussiankernel(B{i,2}, B{count,2});
    else
        sumker=sumker+B{i,1}*polykernel(B{i,2}, B{count,2});
    end
end
sumker=-1*sumker;
% Calculating gradient and prediction
```

```
alpha=((1-y*sumker)>0)*(-y);
predhinge=sign(sumker);
end
%Function to calculate prediction and gradient based on logistic loss
function [predlogistic, alpha]= logistic(y,B,count,choice)
%calculate the sum of scaler times kernels
sumker=0;
for i=1:count-1
    if(choice==1)
        sumker=sumker+B{i,1}*gaussiankernel(B{i,2}, B{count,2});
    else
        sumker=sumker+B{i,1}*polykernel(B{i,2}, B{count,2});
    end
end
sumker=-1*sumker;

% Calculating gradient and prediction
alpha=(-y)*exp(-1*y*sumker)/(1+exp(-1*y*sumker));
predlogistic=sign(sumker);

end

% Defining the Gaussian Kernel Function
function [val]= gaussiankernel(w,x)
gamma=1;
val= exp(-1*gamma * norm(w-x)^2);
end

% Defining the Polynomial Kernel Function
function [val]= polykernel(w,x)
val= ( 1 + dot(w,x))^2;
end
```

Code of OGD for learning kernel based non-linear classification function with budget restriction:

```
% Online Gradient Descent Algorithm for Learning Kernel Classification
% Function with Budget Restriction. In this function, we read the text file
% one instance-label pair at a time, mimicking the online setting.

function kernelOGDwB()
tic
% Setting the variables. Since we are running on the spambase dataset, we
know featuresize is 57.
% We need to change the variables for other datasets.

S=7500; %size of budget
iter=1; count=0; cumlosshinge=0; cumlosslogistic=0;
Bhinge=cell(S,2); Blogistic=cell(S,2);  %create budget
eta=0.1;
% Hloss=repmat(0,instsize,1);Lloss=repmat(0,instsize,1);
M=csvread('C:\Users\hxinyan\Documents\MATLAB\eeg.txt');

% for magic04 dataset
% f=fopen('C:\Users\hxinyan\Documents\MATLAB\magic04.data','r');
% s=repmat('%f',1,10);
% s=strcat(s,'%s');
% A=textscan(f,s,'delimiter',',');
% D=cell2mat(A(1:10));
% C=cell2mat(A{11});
% E=repmat(0,19020,1);
% for i=1:19020
%    if(C(i)=='g')
%        E(i)=1;
%    else
%        E(i)=-1;
%    end
% end
% M=horzcat(D,E);

rowperm=randperm(size(M,1));
M=M(rowperm,:);
dlmwrite('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt',M);

for it=1:iter
    f = fopen('C:\Users\hxinyan\Documents\MATLAB\eeg1.txt');

    while 1
        count=count+1;
        l = fgetl(f); %Read line from file, removing newline characters
        if ~ischar(l), break; end;
        % Reading the joint instance-label pair, one at a time
        xy= sscanf(l,'%f,');
        x=xy(1:end-1)/norm(xy(1:end-1));
        y=xy(end);
        % 1 indicates spam, 0 indicates email. Hence, the following
conversions
```

```
        if(y==0)
            y=-1;
        else
            y=1;
        end

        % Running the OGD, with different choice of losses.
        % choice=1 means gaussian, choice=2 means polynomial
        % Discard strategy: 1 is random discard, 2 is oldest tuple discard
        % First for hinge loss
        choice=2; discard=1;
%        eta=1/sqrt(count);
        [predhinge, alpha]= hinge(x,y,Bhinge,count,S,choice);
        cumlosshinge=cumlosshinge+ (predhinge~=y);
        % updata B based on budget size S
        if (count<=S)
            Bhinge{count,1}=eta*alpha;
            Bhinge{count,2}=x;
        else
            if (discard==1)
                %Random budgeting technique
                I=ceil(S*rand);
            else
                I=mod(count,S);%discard the oldest tuple
                if(I==0)
                    I=S;
                end
            end
            Bhinge{I,1}=eta*alpha;
            Bhinge{I,2}=x;
        end

        % Next for logistic loss
%        eta=1/sqrt(count);
        [predlogistic, alpha]= logistic(x,y,Blogistic,count,S,choice);
        cumlosslogistic=cumlosslogistic+ (predlogistic~=y);
        % updata B based on budget size S
        if (count<=S)
            Blogistic{count,1}=eta*alpha;
            Blogistic{count,2}=x;
        else
            if (discard==1)
                %Random budgeting technique
                I=ceil(S*rand);
            else
                I=mod(count,S);%discard the oldest tuple
                if(I==0)
                    I=S;
                end
            end
            Blogistic{I,1}=eta*alpha;
            Blogistic{I,2}=x;
        end

    end
end
disp(cumlosshinge/count);
```

```
disp(cumlosslogistic/count);
toc
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to calculate prediction and gradient based on hinge loss
function [predhinge,alpha]=hinge(x,y,B,count,S,choice)
%Calculate the sum of scaler times kernels
sumker=0;
if (count>S)
    maximum=S;
else
    maximum=count-1;
end
for i=1:maximum
    %sumker=sumker+sum(B(i,1)*exp((-1)*sqrt((B(i,2:end)-transpose(x)).^2)));
    if(choice==1)
        sumker=sumker+B{i,1}*gaussiankernel(B{i,2}, x);
    else
        sumker=sumker+B{i,1}*polykernel(B{i,2}, x);
    end
end
sumker=-1*sumker;
alpha=((1-y*sumker)>0)*(-y);
predhinge=sign(sumker);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function to calculate prediction and gradient based on logistic loss
function [predlogistic, alpha]= logistic(x,y,B,count,S,choice)
%calculate the sum of scaler times kernels

sumker=0;
if (count>S)
    maximum=S;
else
    maximum=count-1;
end
for i=1:maximum
    %sumker=sumker+sum(B(i,1)*exp((-1)*sqrt((B(i,2:end)-transpose(x)).^2)));
    if(choice==1)
        sumker=sumker+B{i,1}*gaussiankernel(B{i,2}, x);
    else
        sumker=sumker+B{i,1}*polykernel(B{i,2}, x);
    end
end
sumker=-1*sumker;
alpha=(-y)*exp(-1*y*sumker)/(1+exp(-1*y*sumker));
predlogistic=sign(sumker);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Defining the Gaussian Kernel Function
function [val]= gaussiankernel(w,x)
gamma=1;
val= exp(-1*gamma * norm(w-x)^2);
end

% Defining the Polynomial Kernel Function
function [val]= polykernel(w,x)
val= ( 1 + dot(w,x))^2;
end
```

Code of fast bounded OGD algorithm for scalable kernel based online learning – ICML 2012:

```
% Online Gradient Descent Algorithm for Learning Kernel Classification
% Function with Budget Restriction

% Main function for learning kernel classifier via OGD with no budget
% restriction
function fastBkernelOGDwB()
tic
% Setting the variables
S=1500; %size of budget
iter=1; instsize=19020; count=0; cumlosshinge=0; cumlosslogistic=0;
eta=2^(-3);
Bhinge=cell(S,3); Blogistic=cell(S,3); %create budget
lambda=2^(-3)/instsize^2; gamma=2^(0);

% M=csvread('C:\Users\hxinyan\Documents\MATLAB\spambase.data');

%for magic04 dataset
f=fopen('C:\Users\hxinyan\Documents\MATLAB\magic04.data','r');
s=repmat('%f',1,10);
s=strcat(s,'%s');
A=textscan(f,s,'delimiter',',');
D=cell2mat(A(1:10));
C=cell2mat(A{11});
E=repmat(0,19020,1);
for i=1:19020
   if(C(i)=='g')
       E(i)=1;
   else
       E(i)=-1;
   end
end
M=horzcat(D,E);

rowperm=randperm(size(M,1));
M=M(rowperm,:);
dlmwrite('C:\Users\hxinyan\Documents\MATLAB\magic041.data',M);


for it=1:iter
    f = fopen('C:\Users\hxinyan\Documents\MATLAB\magic041.data');
    max1=0; max2=0;
    while 1
        count=count+1;
        l = fgetl(f); %Read line from file, removing newline characters
        if ~ischar(l), break; end;
        % Reading the joint instance-label pair, one at a time
%        xy= sscanf(l,'%f');
        xy=strread(l,'%f,');
        x=xy(1:end-1)/norm(xy(1:end-1));
        y=xy(end);

%         if (y==0)
%             y=-1;
%         else
%             y=1;
%         end
```

```
        % Running the OGD, with different choice of losses.
        % Kernels: choice=1 means gaussian, choice=2 means polynomial
        % Discard strategy: 1 is uniform sampling, 2 is non-uniform
        % sampling
        % First for hinge loss
        choice=1; discard=2;
        [predhinge,alpha1,s1]= hinge(x,y,Bhinge,eta,choice,max1);
        cumlosshinge=cumlosshinge+ (predhinge~=y);

        if (1-s1<0 && max1~=0) %do not updata budget
            for i=1:max1
                Bhinge{i,1}=Bhinge{i,1}*(1-eta*lambda);
            end
        else
            %updata B based on budget size S
            if (max1<S)
                for i=1:max1
                    Bhinge{i,1}=Bhinge{i,1}*(1-eta*lambda);
                end
                max1=max1+1;
                Bhinge{max1,1}=alpha1;
                Bhinge{max1,2}=y;
                Bhinge{max1,3}=x;
            else
                if(discard==1) %take p to be uniformly distributed over budget
                    I=ceil(S*rand);
                else %take p to be non-uniformly distributed
                    num=0;
                    sump=0; %sum of p_i
                    random=rand;
                    p=0;
                    while (sump<random)
                        num=num+1;
                        if (choice==1) %Gaussian kernel
                            z=0;
                            for j=1:S
                                z=z+Bhinge{j,1}*sqrt(gaussiankernel
(Bhinge{j,3},Bhinge{j,3}));
                            end
                            z=(S-1)/z;
                            p=(1-z*Bhinge{num,1}*sqrt(gaussiankernel
(Bhinge{num,3},Bhinge{num,3})));
                            sump=sump+p;
                        else %Polynomial kernel
                            z=0;
                            for j=1:S
                                z=z+Bhinge{j,1}*sqrt(polykernel
(Bhinge{j,3},Bhinge{j,3}));
                            end
                            z=(S-1)/z;
                            p=(1-z*Bhinge{num,1}*sqrt(polykernel
(Bhinge{num,3},Bhinge{num,3})));
                            sump=sump+p;
                        end
                    end
                    I=num;
                    for i=1:S
                       if (i~=I)
                           Bhinge{i,1}=min((1-lambda*eta)*Bhinge{i,1}/(1-
p),eta*gamma);
                       end
                    end
                end
```

```
                Bhinge{I,1}=alpha1;
                Bhinge{I,2}=y;
                Bhinge{I,3}=x;
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%%% Next for logistic loss
        [predlogistic,alpha2,s2]= logistic(x,y,Blogistic,eta,choice,max2);
        cumlosslogistic=cumlosslogistic+ (predlogistic~=y);
        if (-exp(-s2)/(1+exp(-s2))==0 && max2~=0)
            for i=1:max2
                Blogistic{i,1}=Blogistic{i,1}*(1-eta*lambda);
            end
        else
            %updata B based on budget size S
            if (max2<S)
                for i=1:max2
                    Blogistic{i,1}=Blogistic{i,1}*(1-eta*lambda);
                end
                max2=max2+1;
                Blogistic{max2,1}=alpha2;
                Blogistic{max2,2}=y;
                Blogistic{max2,3}=x;
            else
                if(discard==1) %take p to be uniformly distributed over budget
                    I=ceil(S*rand);
                else %take p to be non-uniformly distributed
                    num=0;
                    sump=0; %sum of p_i
                    random=rand;
                    p=0;
                    while (sump<random)
                        num=num+1;
                        if (choice==1) %Gaussian kernel
                            z=0;
                            for j=1:S
                                z=z+Blogistic{j,1}*sqrt(gaussiankernel
(Blogistic{j,3},Blogistic{j,3}));
                            end
                            z=(S-1)/z;
                            p=(1-z*Blogistic{num,1}*sqrt(gaussiankernel
(Blogistic{num,3},Blogistic{num,3})));
                            sump=sump+p;
                        else %Polynomial kernel
                            z=0;
                            for j=1:S
                                z=z+Blogistic{j,1}*sqrt(polykernel
(Blogistic{j,3},Blogistic{j,3}));
                            end
                            z=(S-1)/z;
                            p=(1-z*Blogistic{num,1}*sqrt(polykernel
(Blogistic{num,3},Blogistic{num,3})));
                            sump=sump+p;
                        end
                    end
                    I=num;
                    for i=1:S
                       if (i~=I)
                           Blogistic{i,1}=min((1-lambda*eta)*Blogistic{i,1}/(1-
p),eta*gamma);
                       end
                    end
                end
```

```
                Blogistic{I,1}=alpha2;
                Blogistic{I,2}=y;
                Blogistic{I,3}=x;
            end
        end
    end
end
disp(cumlosshinge/count);
disp(cumlosslogistic/count);
toc
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to calculate prediction and gradient based on hinge loss
function [predhinge,alpha,s]=hinge(x,y,B,eta,choice,max)
%calculate the sum of scaler times kernels
if (max~=0)
    sumker=0;
    for i=1:max
        if(choice==1)
            sumker=sumker+B{i,1}*B{i,2}*gaussiankernel(B{i,3}, x);
        else
            sumker=sumker+B{i,1}*B{i,2}*polykernel(B{i,3}, x);
        end
    end
else
    sumker=0;
end

predhinge=sign(-1*sumker); %predictive y
s=-y*sumker;
% get the value of alpha
if (1-s<0)
    alpha=0;
else
    alpha=-eta;
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function to calculate prediction and gradient based on logistic loss
function [predlogistic,alpha,s]= logistic(x,y,B,eta,choice,max)
%calculate the sum of scaler times kernels
if (max~=0)
    sumker=0;
    for i=1:max
        if(choice==1)
            sumker=sumker+B{i,1}*B{i,2}*gaussiankernel(B{i,3}, x);
        else
            sumker=sumker+B{i,1}*B{i,2}*polykernel(B{i,3}, x);
        end
    end
else
    sumker=0;
end

predlogistic=sign(-1*sumker);
s=-y*sumker;
% get the value of alpha
alpha=-eta*exp(-y*(-sumker))/(1+exp(-y*(-sumker)));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Defining the Gaussian Kernel Function
function [val]= gaussiankernel(w,x)
gamma=1;
val= exp(-1*gamma * norm(w-x).^2);
end

% Defining the Polynomial Kernel Function
function [val]= polykernel(w,x)
val= ( 1 + dot(w,x)).^2;
end
```

Code of online boosting – ICML 2012:

```
% Online Boosting Algorithm coupled with Online Convex Programming
% for Binary Classification
% In this function, we read the text file one instance-label pair
% at a time, mimicking the online setting.

function OSBoost()
% Setting the variables
% Since we are running on the german dataset, we know featuresize is 24. We
% need to change the variables for other datasets.
iter=5; T=1000; d=24;
% We will try to replicate the experiments in the original paper.
theta=0.03; gamma=0.05; N=400;count=0; cum=0;
% a=zeros(N,1);
alpha=(1/N)*ones(N,1);
h=zeros(N,d); %N rows of d dimensional vectors, all initialized to 0.
for it=1:iter
    % initilize step for z, w, alpha, h
    %z=zeros(T,1);
    %w=ones(T,N+1);
    file = fopen('C:\Users\hxinyan\Documents\MATLAB\german.data-numeric');
    for t=1:T
        count=count+1;
        l = fgetl(file); %Read line from file, removing newline characters
        if ~ischar(l), break; end;
        % Reading the joint instance-label pair, one at a time
        xy= sscanf(l,'%f');
        x=xy(1:end-1); x=x/norm(x);
        y=xy(end);
        % 2 indicates bad credit, 1 indicates good credit. Hence, the
following conversions
        if(y==2)
            y=-1;
        else
            y=1;
        end
        % Running the OSBoost, with different choice of losses.
        % First for hinge loss.
        eta=0.1;
        %define f_t(x_t)
        f=0;
        for i=1:N
            %disp(dot(h(i,:),x));
            f=f+alpha(i)*dot(h(i,:),x);
        end

        y_pred=sign(f);

        cum=cum+(y_pred~=y);
        if (y*f<theta)
            for i=1:N
                alpha(i)=alpha(i)+eta*y*dot(h(i,:),x);
            end
            %projection step
%             v=sort(alpha(:),'descend');
```

```
%              p=0;maximum=0;
%              for j=1:N
%                  val= v(j)-(sum(v(1:j))-1)/j;
%                  if (val > 0 && val > maximum)
%                      p=j;
%                  end
%              end
%
%              beta=(sum(v(1:p))-1)/p;
%              for i=1:N
%                  a(i)=max(alpha(i)-beta,0);
%              end
        end
        %alpha=a;
        z=0;w=1;
        for i=1:N
           z=z + y*dot(h(i,:),x) - theta;
           h(i,:)=h(i,:)-y*w*x'; %Weak Learning Algorithm-OGD
           w= min((1-gamma)^(z/2),1);
        end

    end


end
disp(cum/count);
end
```