Received ; | Revised ; | Accepted

DOI: xxx/xxxx

ARTICLE TYPE

Evaluating Classification Accuracy for Modern Learning Approaches

Jialiang Li[1,2,3] | Ming Gao[4,5] | Ralph D'Agostino[6]

[1] Department of statistics and Applied Probability,National University of Singapore, Singapore
[2] Duke University-NUS Graduate Medical School, Singapore
[3] Singapore Eye Research Institute, Singapore
[4] Department of Mathematics,Shanghai Jiao Tong University, China
[5] Department of Statistics,University of Michigan, USA
[6]Department of Mathematics and Statistics, Boston University, USA

Correspondence
*Jialiang Li, Email: stalj@nus.edu.sg

Present Address
6 Science Drive 2, Singapore 117546

Summary

Deep learning neural network models such as multilayer perceptron (MLP) and convolutional neural network (CNN) are novel and attractive artificial intelligence computing tools. However, evaluation of the performance of these methods is not readily available for practitioners yet. We provide a tutorial for evaluating classification accuracy for various state-of-the-art learning approaches, including familiar shallow and deep learning methods. For qualitative response variables with more than two categories, many traditional accuracy measures such as sensitivity, specificity and area under the ROC curve are not applicable and we have to consider their extensions properly. In this paper, a few important statistical concepts for multi-category classification accuracy are reviewed and their utilities for various learning algorithms are demonstrated with real medical examples. We offer problem-based R code to illustrate how to perform these statistical computations step by step. We expect such analysis tools will become more familiar to practitioners and receive broader applications in biostatistics.

KEYWORDS:
neural network, deep learning, multilayer perceptron, convolutional neural net, mxnet, R package

1 | INTRODUCTION

Deep learning methods can be used to design sophisticated neural networks for health care and medical research applications: from real-time pathology assessment to point-of-care interventions to predictive analytics for clinical decision-making. These innovations are advancing the future of precision medicine and population health management in astonishing ways. In particular, deep learning substantially boosts medical classification tasks. Using this new approach one may construct computational models that are composed of multiple processing layers to learn representations of data with high degree of abstraction. These machine learning methods have dramatically improved the state-of-the-art in scientific and business studies. Despite their popularity, there is little account in the statistical literature on how to evaluate and report the learning performance of these methods. We will address this issue in this tutorial and focus specifically on supervised classification tasks.

Statistical classification is needed particularly in clinical settings where the accurate diagnosis of a subject's status is crucial for proper treatment. An assessment of patient disease conditions and evaluation of the prognosis of patients can be achieved by analysing clinical and laboratory data using appropriate learning tools. For two-category classification (e.g. diseased and non-diseased conditions), Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC) measure have been widely adopted for evaluating the accuracy of numerical diagnostic tests [1,2]. Medical decision making sometimes

may involve more than two categories. In recent decades many authors contributed new development to extend standard metrics for multi-category classification, including class-specific Correct Classification Probabilities (CCP)[3], Hypervolume Under the ROC Manifold (HUM)[4], multi-category Integrated Discrimination Improvement (IDI)[3], multi-category Net Reclassification Improvement (NRI)[3], Polytomous Discrimination Index (PDI)[5] and R-squared (RSQ)[3]. These measures complement their counterparts for binary classification and are now available in statistical packages too. We will review these quantities in the tutorial and illustrate their utility with real examples.

We note that in a medical classification problem it is often necessary to involve more than one biomarkers. A statistical model is learned from a training sample and using the learned model one can produce predictive probability for each individual. For categorical outcome, the traditional learning programs include logistic regression, decision tree, linear discrimination analysis, and support vector machines among others. These procedures are well-known for their applicability for many small scale problems. However, for more complicated data analysis such as image data or other massive data sets with huge number of observations and variables, these shallow learning methods may be rather limited. More complicated learning methods such as multilayer perceptron (MLP) and convolutional neural network (CNN) may achieve more satisfactory classification results. We will illustrate in this paper how to evaluate learning accuracy for both shallow and deep learning approaches.

The paper is structured as follows. In Section 2, we selectively review a few modern shallow and deep learning methods, along with their computer softwares. In Section 3, we review six important accuracy measures to evaluate the performance of multi-category classifiers. In Section 4, we provide three case studies to illustrate the calculation.

2 | CLASSIFICATION MODELS AND METHODS

We first introduce some mathematical notation. Consider a set of predictors $\Omega = \{X_1, \cdots, X_p\}$, where $X_j \in \mathbb{R}$ $(j = 1, \cdots, p)$ is the jth predictor which can be discrete or continuous. Suppose we have a training sample of n subjects with measurements $\{x_{ij}, i = 1, \cdots, n; j = 1, \cdots, p\}$ and their class labels $\{y_i, i = 1, \cdots, n\}$. Denote $x_i = (x_{i1}, \cdots, x_{ip})$ as the ith subject. Researchers want to make use of the markers to accurately classify or predict the categorical outcome y. Suppose the multi-category outcome y takes values from $\mathcal{Y} = \{1, 2, \cdots, M\}$. These categories are known and in general not ordered. Of course it is also easy to imagine classifying patients into low-, medium-, and high-risk groups, for instance. We define the binary random variable $\delta_m = I(y = m)$ and let the prevalence for the mth category be $\rho_m = E(\delta_m) = P(y = m)$.

In practice in order to incorporate multiple markers we have to construct statistical models and make classification decision based on the model-based risk prediction. Suppose a model $\mathcal{M}_1$ is learned based on a set of predictors $\Omega_1 \subset \Omega$. Such a model $\mathcal{M}_1$ can generate an $M-$dimensional probability assessment vector $p_i(\mathcal{M}_1) = (p_{i1}(\mathcal{M}_1), \cdots, p_{iM}(\mathcal{M}_1))^T$ for the ith subject such that $\sum_{m=1}^{M} p_{im}(\mathcal{M}_1) = 1$. Each component $p_{im}(\mathcal{M}_1)$ in the vector is also commonly referred to as the *risk score* and indicates the predicted probability of the mth class membership. A greater value of $p_{im}(\mathcal{M}_1)$ suggests by the model $\mathcal{M}_1$ that the ith subject is more likely to be in the mth category. A classifier thus may want to assign the ith subject into the category corresponding to the greatest value of $p_{im}(\mathcal{M}_1)$. This is a very common take-the-winner classification rule. Different learning methods in general may generate quite different probability assessment vector for the same set of predictors Ω_1 and thus lead to different classification results with varied accuracy.

For the sample of n subjects, we can stack the vectors $p_i(\mathcal{M}_1)$ to form an $n \times p$ probability assessment matrix $\mathbf{p}(\mathcal{M}_1) = [p_1(\mathcal{M}_1) \cdots p_n(\mathcal{M}_1)]^T$, which is the input value for the various accuracy evaluation methods in Section 3. In this section we first review a few familiar learning approaches that produce $\mathbf{p}(\mathcal{M}_1)$ for the sample observations.

2.1 | Shallow learning methods

2.1.1 | Multinomial logistic regression

There are abundant research developments to compute the vectors of class probability estimates. Among them, the simplest method perhaps is the multinomial logistic regression model[6] by using the multiple category indicator variable as the response and using the markers involved in Ω_1 as the regressor variables. From the fitted model we may then evaluate the model-based prediction on the probability scale.

Using the first category as the reference, a multinomial logistic regression model is given by

$$\log \frac{P(y_i = m|x_i)}{P(y_i = 1|x_i)} = \beta_{m0} + \beta_{m1}^T x_i, \qquad m = 2, ..., M, \tag{1}$$

where the regression coefficient β_{m1} characterizes the dependence of the outcome on the predictor variables. In general, for a response with M categories, we need $M-1$ log-odds equations defined above to formulate the regression model. In fact the $M-1$ multinomial logit equations contrast each of categories 2, 3,...,M with the reference category 1. Thus, the model-based risk score can be computed by

$$p_{ik} = P(y_i = k|x_i) = \left\{ \sum_{m=1}^{M} \exp(\beta_m^T C_i) \right\}^{-1} \exp(\beta_k^T C_i), \qquad i = 1, \cdots, M, \tag{2}$$

where $C_i = (1, x_i^T)^T$, $\beta_k = (\beta_{k0}, \beta_{k1}^T)^T$ and $\beta_1 = \mathbf{0}$. In practice the regression coefficients β_k can be estimated from the maximum likelihood estimation. To implement the multinomial logistic regression, we may use the **R** package **nnet**. Suppose **y** is the $n-$dimensional vector of class labels and **d** is the $n \times p$ matrix of predictors. We can use the following code to obtain a fitted regression model $\mathcal{M}_1$.

```
> M1 <- nnet::multinom(y~d)
```

We may then use the command **nnet::predict(M1,type='probs')** to yield the probability assessment matrix $\mathbf{p}(\mathcal{M}_1)$ for the whole data set.

Besides logistic regression, other regression methods such as support vector machines, linear discriminant analysis, and classification tree are also extended in the literature to address the multi-category response variable. Thus they can be similarly adapted to produce suitable classification results.

2.1.2 | Support vector machine

The support vector machines (SVM) are based upon the idea of maximizing the margin, i.e., maximizing the minimum distance from the separating hyperplane to the nearest class member[7,8]. In addition to nice theoretical properties, SVMs give exceptionally good performance on classification tasks. The basic SVM supports only binary classification, but extensions[9,10] have been proposed to handle the multi-category classification as well. In these extensions, additional parameters and constraints are added to the optimization problem to handle the separation of the different classes.

The solution of an SVM is usually obtained from minimizing a regularized hinge loss function where a tuning parameter λ determines the trade-off between increasing the margin-size and ensuring that the data features x_i lie on the correct side of the margin. The computation can be solved efficiently with existing software. However, there is a nonparametric form for the estimated risk score based on functional approximation. We usually only focus on the numerical output for risk prediction from the fitted model.

To implement SVM, we may use the **R** package **e1071**. Suppose **y** is the class label and **d** is the matrix of predictors. In SVM we usually have to specify a kernel function which defines an inner product for the original data. When a radial basis function (RBF) is chosen as the kernel, we may use the following code to obtain the fitted model.

```
> M1 <- e1071::svm(y~d,kernel="radial",probability = T)
```

We can then use **e1071::attr(predict(M1,d,probability = T),"probabilities")** to extract the probability assessment matrix $\mathbf{p}(\mathcal{M}_1)$.

SVM can be viewed as a penalization method where the slope coefficients of the basis expansions are shrunk toward zero. Consider the binary case and code the outcome as $y_i \in \{1, -1\}$ and a hyperplane model $f(x_i) = w^T \phi(x_i) + b$, where $\phi(x_i)$ is a fixed transformation and b is an intercept term. SVM aims to find the vector w that minimizes

$$\sum_{i=1}^{n} \left[1 - y_i f(x_i)\right]_+ + \lambda \|w\|^2 \tag{3}$$

where $[x]_+$ represents the positive part of x and λ is a tuning parameter. In general, other margin maximizing loss functions can be used to create similar classifiers. One can apply what we introduce in this paper to carry out a similar investigation on their classification accuracy.

2.1.3 | Decision tree

Another well-known classifier is the classification tree[11] algorithm. The algorithm usually includes two steps. Firstly we grow a large tree T_0 via recursive partitioning, stopping the splitting process only when some minimum node size is reached. Secondly we conduct a cost-complexity pruning to refine the tree structure.

Algorithms for constructing large decision tree T_0 usually work top-down, by choosing a splitting variable and a threshold that best splits the set of items. After that, we find the desired tree T by deleting nodes of the large tree T_0 via balancing misclassification error and tree complexity. The tuning parameters in the actual computing can be set by cross validation. For a constructed tree model, we can then compute the posterior probability of all the leaf nodes to obtain the risk score for all the subjects.

To implement the tree method in this paper, we consider the **R** package **rpart**. Suppose again **y** is the label and **d** is the matrix of predictors. We can apply the following code to construct a classification tree.

```
> fit <- rpart::rpart(y~d)
```

We may then use **rpart::predict(fit,type="prob")** to obtain the probability assessment matrix $\mathbf{p}(\mathcal{M}_1)$.

We note that in practice a single tree model may not perform very well with relatively low or moderate accuracy. To improve the performance, readers may consider random forests, bagging, bootstrapping, adaptive boosting and other extended versions. Those advanced classifiers may also be evaluated using the same procedure in this paper.

2.1.4 | Linear discriminant analysis

Linear discriminant analysis [12] (LDA) is one of the oldest classification approaches and still finds applications in plenty of real world problems. LDA seeks a linear combination of predictor variables that best separates two or more classes of subjects. The resulting combination may be used as a linear classifier.

Typically features x_i's in the mth class are assumed to be normally distributed with mean μ_m, covariance Σ_m, and probability density function $\phi_m(x_i)$. Under such an assumption, we may use the maximum likelihood estimation to estimate the parameters and then use the Bayes' theorem to derive the posterior probability for the ith subject:

$$p_{ik} = P(y_i = k|x_i) = \frac{\rho_k \phi_k(x_i)}{\sum_{m=1}^{M} \rho_m \phi_m(x_i)} \qquad k = 1, \cdots, M. \tag{4}$$

To implement LDA, we use the **R** package **MASS** in this tutorial. The following code gives the solution of an estimated LDA model.

```
> M1 <- MASS::lda(y~d)
```

We then use **MASS::predict(M1)$posterior** to obtain the probability assessment matrix.

Like other aforementioned learning approaches, LDA also has various generalized versions such as quadratic discriminant analysis (QDA) and robust parameter estimation. These methods are quite fundamental to the traditional ways classification was done, partially because they are relatively easy to understand and computationally straight forward. However, when p is large, the sample size required to estimate the parameters reasonably well can be prohibitive. One must consider more modernized variant of LDA for such problems.

2.2 | Deep learning

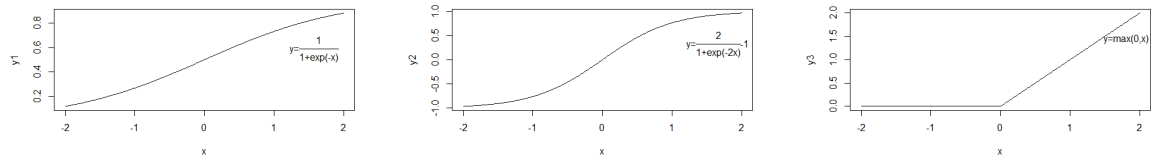
Artificial neural networks [13] (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains. Deep learning is a special type of ANN and is currently gaining a lot of attention for its utilization with big healthcare data. It is based on a collection of connected units or nodes called artificial neurons or perceptrons. Often, neurons are organized in layers which consist of connections, each connection transferring the output of a neuron i to the input of a neuron j.

There are some common components of neural network that are important for setting up the computation works. First, a so-called activation function $f(x)$ should be non-linear, differentiable and monotony. Commonly used activation functions include the Sigmod function $f(x) = 1/(1 + e^{-x})$, the Tanh function $f(x) = 2\ sigmod(2x) - 1$ and the Relu function $f(x) = max(0, x)$. We display these functions in Figure 1 .

Secondly, a loss function $l(y, y')$ is specified to measure the loss of the true value y and the predicted value y' returned from the function f. For binary category outcome, we usually use the cross entropy [14] as the loss function.

$$l(y, y') = -\left[y\ log(y') + (1 - y)\ log(1 - y')\right]. \tag{5}$$

The model parameters in the networks may be estimated by minimizing the total loss function for all the samples. The computation can be carried out using the iterative gradient descent algorithms.



(a) Sigmod function (b) Tanh function (c) Relu function

FIGURE 1 Activation functions.

We may treat the multinomial logistic regression reviewed earlier as a special kind of deep learning method with only one layer. From equation (2) we may find that the activation function for logistic regression is actually a generalization of the sigmod function, usually called softmax function:

$$f_k(x) = \text{softmax}_k(x) = \frac{\exp(\beta_k^T x)}{\sum_{m=1}^{M} \exp(\beta_m^T x)}, \qquad k = 1, \cdots, M. \tag{6}$$

For a general multi-category problem, the cross entropy loss function for a subject may be given as

$$l(y, y') = -\sum_{m=1}^{M} y_m log\ y'_m \tag{7}$$

where y'_m is the predicted probability that the subject is in the mth class based on the deep learning classifier.

We next consider two specific deep learners which are now incorporated in standard softwares.

2.2.1 | Multilayer perceptron

A multilayer perceptron[15] (MLP) is a class of feed-forward artificial neural network where connections between the units do not form a cycle. It can be viewed as a logistic regression classifier where the input is first transformed using a learned non-linear transformation f. This transformation projects the input data into a space where it becomes linearly separable. Figure 2 provides the flow sheet of this approach.

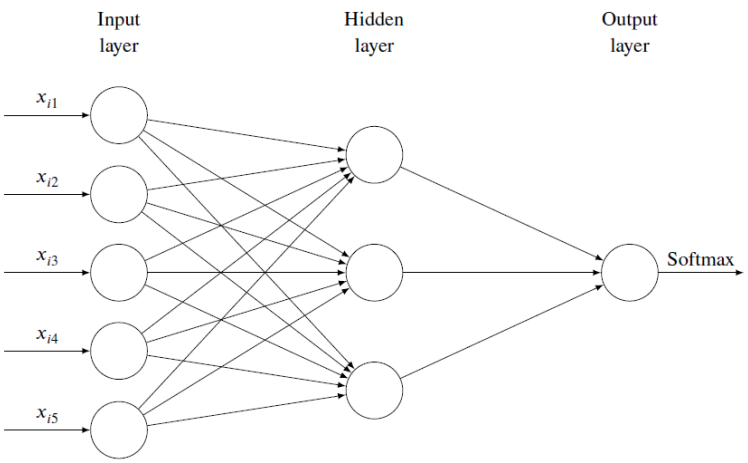


FIGURE 2 Flowsheet of multilayer perceptron. Suppose the input is $x_i = (x_{i1}, \cdots, x_{i5})$. Each circle represents a neuron and each arrow is a connection between layers. The input layer consists of a set of neurons representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation, followed by a non-linear activation function. The output layer receives the values from the last hidden layer and such values are transformed into output values. The number of neurons in the output softmax layer is the same as the category numbers.

To take a closer look at the structure, we may ignore the hidden layer and set the dimension of input x_i to be $p = 3$ and show the single layer computation in Figure 3 . Mathematically, a neuron in the $(t + 1)$th layer $x(t + 1)$ is adjusted from $x(t)$ via

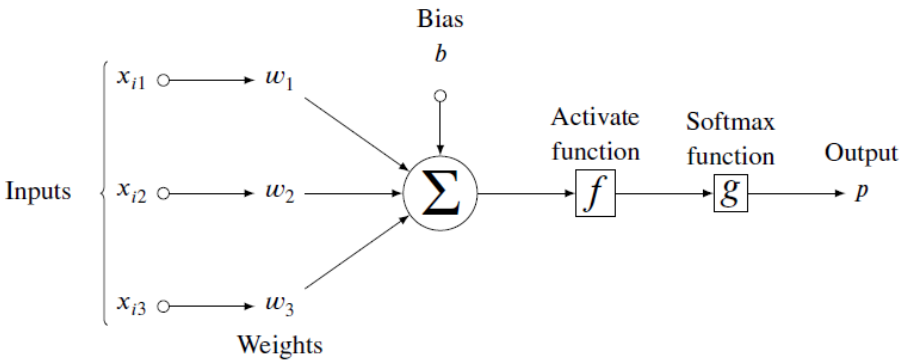


FIGURE 3 Flowsheet of a simplified multilayer perceptron with one hidden layer. Suppose the input is $x_i = (x_{i1}, x_{i2}, x_{i3})$. The output of each neuron is calculated by a non-linear function f of the sum of its input plus a bias b as in equation (8).

$$x(t + 1) = f[b(t) + w(t)^T x(t)], \tag{8}$$

where f is the activation function, $b(t)$is the so-called bias term at the tth layer, and $w(t)$ is the weight vector of the tth layer.

An MLP algorithm repeats the above calculation for a number of layers and stops at the Tth layer. In the output layer or the Tth layer, class-membership probabilities can be obtained from the softmax function.

The reason why MLP works well in practice is that it learns multiple levels of abstraction of data representations through multiple processing layers. The goal of representation learning or feature learning is to find an appropriate representation of data in order to perform a machine learning task. In particular, in MLP, each hidden layer maps its input data to an inner representation that tends to capture a higher level of abstraction. These learned features are increasingly more informative through layers towards the machine learning task. The backpropagation algorithm has a great significance during this process. We introduce a useful notation δ to represent the derivative of a loss function with respect to a dot product of weights and neurons. Then backpropagation formula suggest that the value of δ for a particular hidden unit can be obtained by propagating the δs backwards from units higher up in the network. Consequently we can recursively calculate all the derivatives.

To train an MLP, we need to learn all the parameters of the model including the weight w and the bias b. The estimation is usually carried out under the stochastic gradient descent algorithm with minibatches [16]. Evaluating the gradients can be achieved through the backpropagation algorithm [17] (a special case of the chain-rule of derivation). Software development for deep learning has been abundant and is rapidly evolving in the recent decade. In particular, open source libraries such as `mxnet` enable non-experts to easily design, train and implement deep neural networks. We will carry out all the computation in this paper using `mxnet` since it supports languages such as `Python` or `R` and can train quickly on multiple GPUs. For more information about the software visit http://mxnet.incubator.apache.org/index.html. We may download and install `mxnet` package using the following code. If there is an error about DiagrammeR during the installation, we recommend to manually install earlier version of DiagrammeR in advance.

```
> install.packages("drat", repos="https://cran.rstudio.com")
> drat:::addRepo("dmlc")
> install.packages("mxnet")
> require(mxnet)
```

In practice, the following operational parameters for the MLP need to be tuned by the user:

Layers Number of layers and number of neurons in each layer. Usually the number of neurons can be very large while the number of layers may be relatively moderate or small. The number of neurons in different layers may also differ.

Activation functions Activation functions in each layer. All three functions in Figure 1 can be used while the Relu function is usually preferred for the simplicity of computation. For example, if we want to fill in two hidden layers, first containing 500 nodes and second containing 100 nodes, and choose both of the activation functions to be "Relu", we can use the following function series **mx.symbol** for specification.

```
#construct MLP symbol
> mydata <- mx.symbol.Variable("data")
> fc1 <- mx.symbol.FullyConnected(mydata, num_hidden=500)
> act1 <- mx.symbol.Activation(fc1,  act_type="relu")
> fc2 <- mx.symbol.FullyConnected(act1,  num_hidden=100)
> act2 <- mx.symbol.Activation(fc2,  act_type="relu")
> fc3 <- mx.symbol.FullyConnected(act2,  num_hidden=2)
> softmax <- mx.symbol.SoftmaxOutput(fc3)
```

Loss function Loss function in the output layer. Usually we choose the softmax function defined in equation (6).

Number of round or epoch The number of iterations over the sample data to train the model parameters. Often we need very large number of round to achieve satisfactory numeric accuracy, similar to other nonlinear programming problems. One may specify the option `num.round` in the software.

Learning rate The step size in gradient descent method. This tuning parameter can be optimized via the cross validation. Alternatively many practitioners recommended to use a small value such as 0.1 or 0.01. One may specify the option `learning.rate` in the software.

Initializer The initialization scheme for parameters which specifies the unknown weight at the beginning, usually drawn from a uniform design. One may specify the option `initializer` in the software.

Array batch size The batch size used for array training. The whole training data is usually divided into batches to facilitate the computing. For example, if we set 41 rounds, 40 batch size, 0.08 learning rate, uniform distribution (0.07) as the initial weight, we may use function **mx.model.FeedForward.create** to create the following model specification.

Here we need to encode the first level of y to be 1 and thus denote it as yy and we normalize each column of feature matrix d to obtain data matrix dd.

```
#set model
> devices <- mx.cpu()
> mx.set.seed(0)
> M1 <- mx.model.FeedForward.create(softmax, X=dd, y=yy,
+                       ctx=devices, num.round=41, array.batch.size=40,
+                       learning.rate=0.08, momentum=0.9,  eval.metric=mx.metric.accuracy,
+                       initializer=mx.init.uniform(0.07),
+                       epoch.end.callback=mx.callback.log.train.metric(100))
```

We use the previously constructed MLP symbol "softmax" as the input of **mx.model.FeedForward.create**. It returns the fitted model `M1` from which we can obtain the probability assessment matrix.

There is still limited discussion on how to set all these options to optimize the classification performance. Depending on the scale of the problem and complexity of the data, settings of real world examples using MLP need to be addressed case by case.

After the model architectures are constructed and all the parameters are learned, every input x_i can lead to a probability vector of length M through the softmax loss. After such evaluation for all the n subjects, we obtain the probability assessment matrix $\mathbf{p}(\mathcal{M}_1)$ as in the previous shallow learning methods.

2.2.2 | Convolutional neural net

Convolutional Neural Nets[18] (CNN) are made up of neuron-like computational connections involving learnable weights and biases. Though the main hierarchical structures are similar to MLP, we highlight the following key differences.

First, the input for CNN are usually images or multi-dimensional data arrays rather than a vector in MLP. In fact, the overall learning process of CNN simulates the organization of the animal visual cortex. Thus the layers of a CNN for a 2D colored image typically have neurons arranged in 3 dimensions: width, height, depth. The width and height can be the same as the dimension of the input image matrix. On the other hand, the depth would be 3 corresponding to Red, Green, Blue channels for a colored 2D picture.

Secondly, we need three types of layers to build CNN architectures: the convolutional layer, the pooling layer (not always necessary to specify), and the fully-connected layer which is the same as the hidden layer in MLP. The primary purpose of a convolutional layer is to detect distinctive local motif-like edges, lines, and other visual elements. Parameters including number of filters, their spatial extents and stride should be specified. The pooling layer operates independently on every depth slice of the input and resizes it spatially, using the maximum (MAX) operation. For example, every MAX operation would be taking a max over 4 numbers if filter size is 2×2 with a stride of 2. Parameter such as filters sizes and stride should be settled at this stage. A CNN architecture contains a list of layers that transform the image volume into a probability vector. The initial volume stores the raw image pixels and the last volume stores the class risk scores.

To implement CNN, we may use `mxnet` package as well. There are a few key parameters to be specified by the user. The number of filters (or kernels) is the number of neurons, since each neuron performs a different convolution on the input to the layer (more precisely, the neurons' input weights form convolution kernels). Filter spatial extent is size of a filter (or a kernel). Usually we use 3 times 3 kernel or 5 times 5 kernel. Every time we will connect each neuron to a local region which is the kernel size of the input volume. We specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 then the filters jump 2 pixels at a time as we slide them around which will produce smaller output volumes spatially.

For example, suppose we want to fill in two convolution layer ensembles, first containing 20 filters and second containing 50 filters. Both of the activation functions are specified as "tanh". Also we use 5 times 5 kernel for convolution, 2 times 2 kernel for max pooling, and 2 times 2 stride for both of ensembles. After constructing convolution layer ensembles, we move on to construct one single fully connected layer with 500 hidden nodes and "tanh" to activate it. We can use function series **mx.symbol** to construct it as follows.

```
#construct CNN symbol
> mydata <- mx.symbol.Variable("data")
# first convolutional layer
> conv1 <- mx.symbol.Convolution(data=mydata, kernel=c(5,5),num_filter=20)
> tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
> pool1 <- mx.symbol.Pooling(data=tanh1,pool_type="max",kernel=c(2,2), stride=c(2,2))
# second convolutional layer
> conv2 <- mx.symbol.Convolution(data=pool1, kernel=c(5,5),num_filter=50)
> tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
> pool2 <- mx.symbol.Pooling(data=tanh2,pool_type="max",kernel=c(2,2), stride=c(2,2))
# fullyconnected layer (hidden layer)
> flatten <- mx.symbol.Flatten(data=pool2)
> fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
> tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
# output layer
> fc2 <- mx.symbol.FullyConnected(data=tanh3, num_hidden=10)
> lenet <- mx.symbol.SoftmaxOutput(data=fc2)
```

Other modeling options are similar to those for MLP discussed earlier. For example, if we specify 41 rounds, 40 batch size, 0.08 learning rate, uniform distribution (0.07) as the initial weight, we may use function **mx.model.FeedForward.create** to create the CNN model as follows.

```
#set model
> devices <- mx.cpu()
> mx.set.seed(0)
> M1 <- mx.model.FeedForward.create(lenet, X=dd, y=yy,
+                 ctx=devices, num.round=41, array.batch.size=40,
```

```
+                 learning.rate=0.08, momentum=0.9,  eval.metric=mx.metric.accuracy,
+                 initializer=mx.init.uniform(0.07),
+                 epoch.end.callback=mx.callback.log.train.metric(100))
```

We use the previously constructed CNN symbol "lenet" as the input of **mx.model.FeedForward.create**. It returns the fitted model `M1` which we can use to make classification. Every input x_i can lead to a probability vector of length M after the model architectures are constructed and the parameters are settled and learned. Computing such risk scores for all the n objects, we may then obtain the probability assessment matrix $\mathbf{p}(\mathcal{M}_1)$ for the downstream evaluation.

3 | ACCURACY METRICS

All the learning methods reviewed in the preceding section will be examined by the accuracy metrics reviewed in this section. We provide a conceptual summary for each classification accuracy measure and introduce the associated software. These measures were originally proposed to assess the diagnostic accuracy of medical tests and widely adopted in epidemiology studies. We will see that they are equally applicable to evaluate classification accuracy for various learning methods.

3.1 | Single model evaluation

First we consider a few measures that are useful to describe the classification performance of a particular statistical model. For these measures, a greater value indicates a more satisfactory classification performance.

3.1.1 | Hypervolume under the manifold

Hypervolume Under the ROC Manifold (HUM)[19] has been proposed as a generalization of AUC[1,2] value for multi-category classification. Unlike the correct classification probability to be defined in the next subsection, HUM does not depend on the class prevalence and thus reflects the intrinsic accuracy of the classifier.

We consider extending the definition of 2D ROC curve. Recall that a statistical learning method $\mathcal{M}_1$ can generate a probability assessment vector $p_i(\mathcal{M}_1) = (p_{i1}(\mathcal{M}_1), \cdots, p_{iM}(\mathcal{M}_1))^T$ for the ith subject. To make classification decision, one may apply the following sequential thresholding rule: If $p_{i1}(\mathcal{M}_1) > c_1$, then classify the ith subject as class 1; otherwise if $p_{i2}(\mathcal{M}_1) > c_2$, then classify the subject as class 2; $\cdots$; otherwise if $p_{i,M-1}(\mathcal{M}_1) > c_{M-1}$, then classify the subject as class $M-1$; otherwise classify as class M. With M classes, this procedure requires $M-1$ thresholds $c_1, \cdots, c_{M-1}$ and different threshold values produce different decision rules. At a fixed set of $M-1$ thresholds, the probability of correctly classifying the mth class is denoted by $t_m (m = 1, ..., M)$. For example, $t_1 = P(p_{i1}(\mathcal{M}_1) > c_1 | y_i = 1)$, $t_2 = P(p_{i1}(\mathcal{M}_1) \leq c_1, p_{i2}(\mathcal{M}_1) > c_2 | y_i = 2)$ and so forth. One plots $(t_1, t_2, \cdots, t_M)$ in M-dimensional space for all decision rules (i.e. all possible threshold values) and then connects these points to form an ROC manifold. When $M = 2$, t_1 and t_2 are simply the sensitivity and specificity and the manifold reduces to the familiar ROC curve. When $M = 3$, the manifold reduces to the 3D ROC surface. The ROC manifold itself may not be of much interest when $M \geq 4$. A more relevant quantity is the hypervolume under the ROC manifold which summarizes the overall accuracy for the model $\mathcal{M}_1$.

Novoselova et al.[20] developed an **R** package `HUM` which can be directly downloaded from CRAN website. The corresponding Shiny web application (https://public.ostfalia.de/ klawonn/HUM.htm.) allows graphical display of 3D ROC surface. Figure 4 presents an example, displaying 3-dimensional view of three ROC surfaces, where we consider a leukemia data[21] to be described in Section 4.4 and plot the ROC manifolds for three selected genetic markers in the data. Clearly the volume for the left panel is much greater than the other two panels, indicating the corresponding marker is a stronger classifier.

The aforementioned geometric definition of HUM may not be directly relevant to numerical evaluation. In fact, there is an appealing alternative probabilistic definition of HUM. Suppose there is a random sample involving M subjects and each subject is chosen from one of the M distinct categories. Without loss of generality, we assume that the mth subject is from the mth category, $m = 1, \cdots, M$. Then HUM for a classification model $\mathcal{M}_1$ corresponds to the probability that all the M subjects are correctly classified by the model $\mathcal{M}_1$. When $M = 2$, we recall that AUC is the probability of correctly differentiating a pair of randomly chosen diseased and non-diseased subjects. Thus, HUM provides a direct generalization of AUC for multiple classes. The null value for HUM is $1/M!$ and corresponds to a random guess.

The inference procedure for HUM has been discussed for ordered polychotomous responses[22,23] with a single marker (i.e. $p = 1$). The calculation method has been implemented in `R` package `HUM`. For the more general case of unordered polychotomous

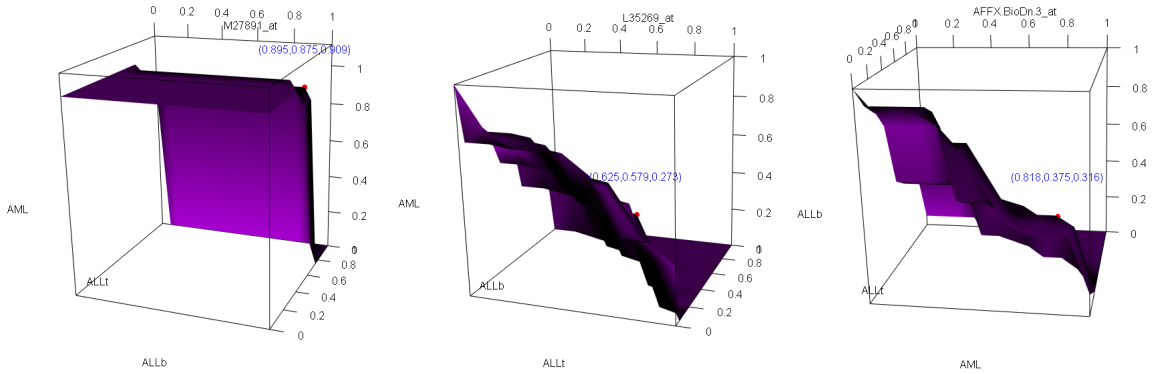


FIGURE 4 3D ROC surfaces for three gene expressions from leukemia data.

responses and for model-based multivariate predictors (i.e. $p > 1$) one needs to consider the methods in Li and Fine (2008)[24]. The method is available in R package `mcca` and will be covered with details in this tutorial.

3.1.2 | Correct classification probability

Another very popular multi-category classification accuracy measure is the correct classification probability (CCP). To fix the notation, we write $\hat{y}_i(\mathcal{M}_1)$ to be the predicted class membership for the ith subject using the machine learning model $\mathcal{M}_1$. The CCP is thus defined to be the probability that the predicted class agrees with the true class for the subject:

$$CCP = P(y_i = \hat{y}_i(\mathcal{M}_1)). \tag{9}$$

Using the law of total probability, the equation above can be re-written as

$$CCP = \sum_{m=1}^{M} P(\hat{y}_i(\mathcal{M}_1) = m | y_i = m) P(y_i = m) = \sum_{m=1}^{M} \rho_m P(\hat{y}_i(\mathcal{M}_1) = m | y_i = m), \tag{10}$$

where the probability $P(\hat{y}_i(\mathcal{M}_1) = m | y_i = m)$ can be regarded as a class-specific CCP for the mth category. When $M = 2$, the two class-specific CCP values are commonly referred to as the sensitivity and the specificity[1,2]. CCP directly assesses whether the mode-based classification for a subject is identical to his true class status. Its empirical version is simply the proportion of correctly classified subjects in the sample.

$$\widehat{CCP} = \frac{1}{n} \sum_{i=1}^{n} I(y_i = \hat{y}(\mathcal{M}_1)). \tag{11}$$

Such a simple formula facilitates the application of CCP in many real problems, especially when M is large. In fact, the computation time for CCP does not increase as the number of categories increases. CCP and its complement, misclassification rate, are commonly used in machine learning literature.

Machine classifiers usually output the probability assessment vector $p(\mathcal{M}_1)$ directly and a decision maker needs to take one more step to convert the probability into an actual class prediction $\hat{y}(\mathcal{M}_1)$. In general we may follow two types of decision rules to carry out the conversion.

The first type is the thresholding rule introduced in the preceding section. Based on such a rule, we may define the probability of correctly classifying the mth class to be $t_m = P(\hat{y}(\mathcal{M}_1) = m | y = m)$, $m = 1, ..., M$. The overall CCP for a model $\mathcal{M}_1$ is just a prevalence-weighted average

$$CCP = \sum_{m=1}^{M} \rho_m t_m. \tag{12}$$

Besides the thresholding rule, we note that t_m may also be defined without using any threshold. In fact, we may consider the take-the-winner rule: Recall that a classifier $\mathcal{M}_1$ generates a probability assessment vector $p_i(\mathcal{M}_1) = (p_{i1}(\mathcal{M}_1), \cdots, p_{iM}(\mathcal{M}_1))^T$ for the ith subject. Decision makers may assign a subject to one of the M categories according to the greatest component in the probability vector $p_i(\mathcal{M}_1)$. Under this rule, we may obtain for the mth class

$$CCP_m = t_m = P(p_{im}(\mathcal{M}_1) > p_{ik}(\mathcal{M}_1),\ k \neq m | y_i = m), \qquad m = 1, ..., M. \tag{13}$$

We note that when $M = 2$, thresholding rule and take-the-winner rule yield the same definition of t_1 and t_2 (sensitivity and specificity)[1,2].

Though it provides a straightforward assessment on the performance of a fixed sample, the estimated overall CCP value is quite sensitive to the distribution of different classes in the particular data sample and thus cannot lend support to external validity for some studies where prevalence information is unavailable. For example, CCP values obtained for low-risk disease groups may be dominated by the specificity.

The implementation of CCP is quite easy after a classification is done. We will use the function `CCP` in `R` package `mcca` in the following illustration.

3.1.3 | R-squared value

R-squared value (RSQ)[25,26,27,28], also called a coefficient of determination, is extensively studied in linear regression and has been discussed for binary logistic regression models. Simply speaking, the value of R^2 is the fraction of the total variance of the response (continuous or categorical) explained by the model. A greater percentage of R-squared value may indicate that a higher proportion of the variation of the response may be attributable to the markers included in the model and the classification based on such a model may thus lead to more meaningful results. The estimated R-squared value, though not directly rating the accuracy of classification, is still considered an important model evaluation statistic. There is also no probabilistic interpretation for R-squared value even though it lies between 0 and 1.

For the mth class, the R-squared value[29] for a particular model $\mathcal{M}_1$ is defined as

$$R_m^2 = \frac{var(\delta_m) - E(var(\delta_m|\mathcal{M}_1))}{var(\delta_m)} = \frac{var(p_{im})}{\rho_m(1-\rho_m)}, \tag{14}$$

where the second equality follows as $E(Y_i = m|\mathcal{M}_1) = p_{im}(\mathcal{M}_1)$, and p_{im} is the mth element of the probability vector $\mathbf{p}_i(\mathcal{M}_1)$ defined earlier. The overall R-squared value is simply an average of the class-specific R-squared values:

$$RSQ = \frac{1}{M}\sum_{m=1}^{M} R_m^2. \tag{15}$$

The applications of R^2 for multi-category classification is not as common as the two measures in the preceding subsections in practice, mainly because it is not widely regarded as an accuracy measure but a model fitting summary statistic. It has recently attracted more attention from biostatistical researchers due to its close connection to the IDI[29]. The computation of multi-class RSQ is implemented in function `RSQ` in `R` package `mcca`.

3.1.4 | Polytomous discrimination index

Polytomous discrimination index (PDI)[30,31 32] is a recently proposed diagnostic accuracy measure for multi-category classification. Similar to HUM, PDI is also evaluating the probability of an event related to simultaneously classifying M subjects from M categories. While HUM is pertaining to the event that all M subjects are correctly identified to their corresponding categories, PDI is pertaining to the number of subjects in the set of M subjects that are correctly identified to his/her category.

To define the PDI, we consider a random sample that involves M subjects and each subject is chosen from one of the M distinct categories. Without loss of generality, we assume that the ith subject is from the ith category. The classification decision is achieved via a joint comparison of the M subjects. Using earlier notations, for a classification model $\mathcal{M}_1$, we may denote the probability of placing a subject from category i into category j by $p_{ij}(\mathcal{M}_1)$. A class i subject can be correctly classified if $p_{ii}(\mathcal{M}_1) > p_{ji}(\mathcal{M}_1)$ for all $j \neq i$. For a fixed category i, we may define the class-specific PDI to be

$$PDI_i = p(p_{ii}(\mathcal{M}_1) > p_{ji}(\mathcal{M}_1)\ j \neq i | y_i = i) \tag{16}$$

and the overall PDI to be

$$PDI = \frac{1}{M}\sum_{m=1}^{M} PDI_m. \tag{17}$$

When $M = 2$, PDI also reduces to AUC and thus can be viewed as a generalization of AUC for the multi-class problem. However, when $M \geq 3$, PDI value is always greater than HUM for a classifier $\mathcal{M}_1$. Models or diagnostic biomarkers with poor PDI values usually also have poor HUM values. The lower bound for PDI is $1/M$, corresponding to random guess.

PDI is not as widely applied as HUM and CCP to assess the diagnostic and classification accuracy. We suggest its value should also be reported along with other major accuracy measures. Specifically, we would recommend computing PDI for big

data studies and screen out unnecessary biomarkers whose PDI values are below a satisfactory level. The computation of PDI is implemented in function `PDI` in `R` package `mcca`.

3.2 | Model comparison

When comparing two models used to make prediction or classification for the same data, it is usually not advisable to only check the difference of a single accuracy measure. For example, it is noted by many authors that the difference of HUM between two models may not be informative when the baseline model is already quite accurate. We next consider two more appropriate measures for comparing two models.

We need more notations. Now suppose that in addition to model $\mathcal{M}_1$ introduced earlier, more variable(s) are included and we construct a model$\mathcal{M}_2$ that is based on a set of predictors $\Omega_2 \supset \Omega_1$. We use the nested-structure notations as they are widely discussed in the literature. We note that there are studies where the accuracy improvement occurs among non-nested models as well. Our proposed methods can apply with slight modification. The newly constructed model $\mathcal{M}_2$ generates another probability vector $p_i(\mathcal{M}_2) = (p_{i1}(\mathcal{M}_2), ..., p_{iM}(\mathcal{M}_2))$ for the ith subject.

3.2.1 | Net reclassification improvement

Net reclassification improvement (NRI)[33] is a numeric characterizations of accuracy improvement for diagnostic tests or classification models and were shown to have certain advantage over analyses based on ROC curves.

While ROC-based measures have been widely adopted, it has been argued by many authors[34] that such measures may not be good criteria to quantify improvements in diagnostic or classification accuracy when the added value of a new predictor to an existing model is of interest. NRI can address these limitations since it essentially calculates the increase in correct classification between models.

The multi-category NRI from a baseline model $\mathcal{M}_1$ to a more complicated model $\mathcal{M}_2$ is

$$NRI = \sum_{m=1}^{M} \rho_m \{CCP_m(\mathcal{M}_2) - CCP_m(\mathcal{M}_1)\}. \tag{18}$$

NRI directly reflects how often the new model corrects the incorrectly classified cases in the old model and is therefore very appealing to practitioners. We note another interpretation for NRI is the difference of Youden's index of the two models.

For additional discussion of these recent developments, we refer the reader to materials[35,36,37,38]. We note that these metrics provide different perspectives for accuracy studies and there are also critiques in the literature (see for example[39,40,41]). In particular, Hilden[40] pointed out that NRI sometimes may inflate the prognostic performance of added biomarkers and Kerr[41] argued that NRI may perform poorly under some nonlinear data generating mechanisms. Thus users of such popular metrics should also exercise caution in practice.

The computation of multi-class NRI is implemented in function `NRI` in `R` package `mcca`. There are also a few other packages that allow the calculation of different versions of NRI.

3.2.2 | Integrated discrimination improvement

Integrated discrimination improvement (IDI)[42] is originally defined as the improvement of the integration for sensitivity and specificity over all thresholds for binary classification. Recently it has been extended to multi-class version by noting its strong connection to R-squared values[33,25,26,28]. In fact, IDI may be regarded simply as the increase in R^2 defined in (15). Specifically, we define the multi-class IDI to be

$$IDI = \frac{1}{M} \sum_{m=1}^{M} (R_m^2(\mathcal{M}_2) - R_m^2(\mathcal{M}_1)). \tag{19}$$

IDI thus reflects how many more percentage of variation of the multi-class response the new model explains. Unlike NRI, this metric does not correspond to a probability of a random event and thus has no probabilistic interpretation. However, the IDI value certainly indicates the added explanation power of the new model relative to the old model and therefore also widely adopted in biomedical studies. In the literature IDI has the same critique as the NRI[43] and we suggest practitioners to use these improvement measures with caution.

The computation of multi-class IDI is implemented in function `IDI` in `R` package `mcca`. There are also a few other packages that allow the calculation of different versions of IDI.

4 | CASE STUDIES

In this tutorial, we use **R** as the programming language. In order to analyze all the following examples, we need **mcca** package which is specifically constructed for multi-category classification accuracy. It contains functions to evaluate the six accuracy methods introduced in section 3 for binary and multi-category classifiers. It can be download directly from CRAN website https://CRAN.R-project.org/package=mcca.

```
> install.packages("mcca")
> require(mcca)
```

4.1 | Guideline

In this section, we provide a step-by-step guideline on how to analzye the classification accuracy for a real data set.

We first consider a single model evaluation. The following is the general procedure.

Single model evaluation:

Step 1 First, we pre-process the raw data. After downloading data to the computer, we clean the data by removing or imputing missing entries. We then form a data matrix of $p+1$ columns including p columns of the predictor features and 1 column of the M-class response.

Step 2 Then we may divide the whole dataset randomly into two parts: a training data and a test data. We use the training data to build the statistical model and the test data to assess the out-of-sample performance.

Step 3 For single model evaluation, we choose one set of predictors Ω_1 and build a statistical model based on these variables. The statistical model can be based on any of the shallow learning methods (multinomial logistic regression, classification tree, support vector machine or linear discriminant analysis) or the deep learning methods (MLP or CNN). Pay attention that some of the procedures require sophisticated parameters tuning to arrive at a satisfactory model.

Step 4 After the model is built, we apply this model to training and test data, respectively, to obtain the in-sample and out-of-sample performance assessment. As discussed in Section 2, we may obtain a probability assessment matrix from the fitted model and make classification decision based on the risk scores. Thus, we may feed this matrix into the accuracy methods CCP, PDI, RSQ, HUM reviewed in Section 3 and obtain the accuracy metric values accordingly.

When the goal is to evaluate the accuracy improvement, we need to consider two models with different degree of complexity. In this tutorial we mainly consider two nested models built from the same learning program where the simpler model can be reduced from the more complex model. In general our discussion can also be applied to any two classification models with non-nested structure. We provide a simple guideline on paired model comparison in the following.

Model comparison:

Step 1 First, we pre-process the data as in the single model evaluation and divide the whole data into training and test sets. For paired model comparison, we choose two sets of predictors Ω_1 and Ω_2. We aim to evaluate the improvement of the model constructed by the two sets of predictors.

Step 2 We then use either a shallow learning or a deep learning method to train the statistical model for both sets of predictors, separately. Consequently we arrive at two models $\mathcal{M}_1$ and $\mathcal{M}_2$, based on Ω_1 and Ω_2, respectively.

Step 3 After the models are built, we apply the two fitted models to training and test data to obtain the in-sample and out-of-sample performance assessment. As discussed in Section 2, we may obtain probability assessment matrices for the two models and make classification decision accordingly. We may then compute NRI and IDI reviewed in Section 3 and interpret the values accordingly.

In the following R illustration, one only needs to specify the two sets Ω_1 and Ω_2 to acquire the training set accuracy. To compute the test set accuracy, one needs to explicitly extract the probability matrix from the fitted models and then feed such a matrix to the R functions.

4.2 | Breast Cancer

Breast cancer is the most common cancer and the second greatest cause of cancer mortality for women. The diagnostic system at University of Wisconsin Hospitals analyzed digital morphometry samples from 569 patients where 212 malignant cases were identified. We revisit this Wisconsin breast cancer data which is publicly available at the UCI website:

$$https://archive.ics.uci.edu/ml/datasets/.$$

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These measurements describe characteristics of the cell nuclei present in the image. The data set contains 30 real-valued features computed for cell nucleus. The diagnosis outcome is either malignant or benign. Each input variable represents certain feature of the cell nucleus such as radius, perimeter, area, smoothness, concavity and symmetry. These real values are standardized before the deep learning calculation.

Following the guideline given in the preceding subsection, we carry out the following step-by-step calculation to this dataset.

Step 1 Check whether the dataset is appropriate. After we download the dataset from UCI website, we save the file **wpbc.data** to the desktop. Then, we assign proper variable names for the data file.

```
d <- read.table("wpbc.data",sep=",")
#get variable names
> name10 <- c("radius","texture","perimeter","area","smoothness","compactness","concavity",
"concave points","symmetry","fractal dimension")
> vbnames <- c("ID","Outcome","Time",paste(name10,"MEAN"),paste(name10,"SE"),
paste(name10,"LARGEST"),"Tumor size","Lymph node status")
> colnames(d) <- vbnames
> mydata0=d[,-1]
> str(mydata0)
'data.frame': 194 obs. of  34 variables:
$ Outcome                 : Factor w/ 2 levels "N","R": 1 1 1 1 2 2 2 1 1 1 ...
$ Time                    : int  31 61 116 123 27 77 77 119 76 123 ...
$ radius MEAN             : num  18 18 21.4 11.4 20.3 ...
$ texture MEAN            : num  27.6 10.4 17.4 20.4 14.3 ...
$ perimeter MEAN          : num  117.5 122.8 137.5 77.6 135.1 ...
......
```

After the above operation, we obtain a data file with 194 observations and 34 variables where the first column is the binary outcome variable indicating the disease status.

Step 2 We consider a sample with only the first ten variables (**Outcome**, **Time**, **radius MEAN**, **texture MEAN**, **perimeter MEAN**, etc) in the original data file. Then we randomly split this data frame into training and test data in a $2:1$ ratio.

```
> mydata <- mydata0[,1:10]
> set.seed(1)
> idx<-sample(nrow(mydata),2/3*nrow(mydata))
> train<-mydata[idx,]
> test<-mydata[-idx,]
```

Step 3 Carry out the classification procedure and evaluate the accuracy measures. One may simply choose a function for the accuracy evaluation methods (`hum, ccp, rsq, pdi`) with an option `method` specifying the classifier used. Available options for `method` include `"multinom", "tree", "lda", "svm", "prob"`, corresponding respectively to multinomial logistic regression, decision tree, linear discrimination analysis, support vector machine and arbitrary probability assessment matrix obtained externally. The following code is to evaluate HUM of a multinomial logistic regression and R-squared value of a decision tree for the training data created before.

```
#for training data
> hum(y=train[,1],d=train[,c(2:10)],method="multinom",k=2)
```

```
[1] 0.7910354
> rsq(y=train[,1],d=train[,c(2:10)],method="tree",k=2)
[1] 0.2518744
```

In this case, using logistic regression achieves over 79% HUM value, suggesting that the two disease categories can be differentiated by the markers with a moderately high probability.

```
#for training data
> result <- ests(y=train[,1],d=train[,c(2:10)],acc="hum",level=0.95,method="multinom",k=2)
> result$se
[1] 0.03785291
> result$interval
[1] 0.7502646 0.8895315
```

The above code computes the standard error and 95% confidence interval for HUM value using the bootstrap method.

All accuracy measures for the training sample and the test sample are summarized in Table 1 along with their standard errors and 95% confidence intervals. We notice that in-sample accuracy is usually higher than the out-of-sample accuracy. In particular, MLP attains the top in-sample accuracy among all classification methods under all the four evaluation criteria. To evaluate the accuracy for the test sample, one needs to carry out the model fitting explicitly and generate the

			CCP		
	multinom	tree	svm	lda	mlp
train	0.81 (0.04)	0.81 (0.02)	0.80 (0.03)	0.80 (0.04)	0.81 (0.03)
	[0.72, 0.88]	[0.73, 0.91]	[0.76, 0.88]	[0.74, 0.87]	[0.74, 0.86]
test	0.78 (0.05)	0.66 (0.05)	0.80 (0.05)	0.75 (0.05)	0.75 (0.05)
	[0.71, 0.88]	[0.55, 0.75]	[0.71, 0.90]	[0.66, 0.83]	[0.63, 0.85]
			R-squres		
	multinom	tree	svm	lda	mlp
train	0.22 (0.07)	0.25 (0.09)	0.09 (0.10)	0.22 (0.07)	0.49 (0.06)
	[0.15, 0.42]	[0.27, 0.63]	[0.06, 0.45]	[0.14, 0.41]	[0.39, 0.61]
test	0.25 (0.09)	0.21 (0.07)	0.04 (0.01)	0.22 (0.08)	0.52 (0.13)
	[0.16, 0.45]	[0.19, 0.49]	[0.01, 0.08]	[0.13, 0.40]	[0.35, 0.82]
			HUM (PDI)		
	multinom	tree	svm	lda	mlp
train	0.79 (0.04)	0.80 (0.09)	0.86 (0.03)	0.79 (0.04)	0.89 (0.03)
	[0.75, 0.89]	[0.70, 0.94]	[0.85, 0.96]	[0.74, 0.89]	[0.82, 0.93]
test	0.76 (0.06)	0.70 (0.13)	0.68 (0.10)	0.76 (0.08)	0.69 (0.09)
	[0.66, 0.88]	[0.35, 0.83]	[0.43, 0.81]	[0.64, 0.87]	[0.51, 0.86]

TABLE 1 Accuracy for training and test samples for breast cancer data along with their standard errors (in the parenthesis) and 95% confidence intervals (in the brackets).

probability assessment matrix for the test sample. The following is an example to obtain the probability assessment matrix for the test sample when using a multinomial logistic regression model.

```
#for test data
> fit <- nnet::multinom(train$Outcome~.,train)
> predict.test.probs <- predict(fit,type='probs',newdata=test[,c(2:10)])
> pm2_m <- data.frame(1-predict.test.probs,predict.test.probs)
```

After the probability assessment matrix pm2_m is obtained, we may then evaluate the accuracy measure such as the HUM as follows.

```
> hum(test[,1],pm2_m,"prob",2)
[1] 0.7647929
> result <- ests(y=test[,1],d=pm2_m,acc="hum",level=0.95,method="prob",k=2)
> result$se
[1] 0.06167885
> result$interval
[1] 0.6568878 0.8757062
```

The HUM for this model is 0.765 with a standard error 0.062. The 95% confidence interval for HUM is [0.657, 0.876].

We provide more details on the implementation of multiple layer perceptron (MLP) method. In this example, we use two hidden layers with 50 and 10 nodes, respectively. The activation function is chosen to be the Relu function. We specify learning.rate=0.08, batch.size=40, num.round=40. The following code specifies the necessary symbol for MLP computing and implements the multi-layer model fitting.

```
> require(mxnet)
#for training data
> y=train[,1]; d=train[,c(2:10)]
> yy <- as.numeric(y)-1
> scaled <- scale(d); dd <- data.matrix(scaled)
#construct MLP symbol
> mydata <- mx.symbol.Variable("data")
> fc1 <- mx.symbol.FullyConnected(mydata, num_hidden=50)
> act1 <- mx.symbol.Activation(fc1,  act_type="relu")
> fc2 <- mx.symbol.FullyConnected(act1,  num_hidden=10)
> act2 <- mx.symbol.Activation(fc2,  act_type="relu")
> fc3 <- mx.symbol.FullyConnected(act2,  num_hidden=2)
> softmax <- mx.symbol.SoftmaxOutput(fc3)
#set model
> model <- mx.model.FeedForward.create(softmax, X=dd, y=yy,
+             ctx=devices, num.round=40, array.batch.size=40,
+             learning.rate=0.08, momentum=0.9,  eval.metric=mx.metric.accuracy,
+             initializer=mx.init.uniform(0.07),
+             epoch.end.callback=mx.callback.log.train.metric(100))
Start training with 1 devices
...
[40] Train-accuracy=0.8125
```

After the above model is fitted using MLP method, we may then extract the probability assessment matrix pp as follows. The accuracy measures such as CCP and HUM can be computed accordingly for the training sample. and the test sample.

```
#obtain probability matrix
preds = predict(model, dd)
pp <- data.frame(t(preds))
#obtain accuracy value for training data
ccp(y=y,d=pp,method="prob",k=2)
[1] 0.8062016
```

For the test sample, we can feed the fitted model into the predict function to obtain the probability assessment matrix pm2_mlp and then evaluate the test accuracy using functions such as hum.

```
#for test data
> y=test[,1]; d=test[,c(2:10)]
> dd <- data.matrix(scale(d,attr(scaled, "scaled:center"), attr(scaled, "scaled:scale")))
> preds = predict(model, dd)
> pm2_mlp <- data.frame(t(preds))
> hum(y=y,d=pm2_mlp,method="prob",k=2)
[1] 0.6945624
```

In this case $M = 2$, the ROC manifold is actually a 2D ROC curve for each classifier. We plot the ROC curves of the five classifiers in Figure 5 for training (top) and test (bottom) data respectively. We observe that MLP outperforms other classifiers for the training sample while its performance is not as good as LDA for the test sample.

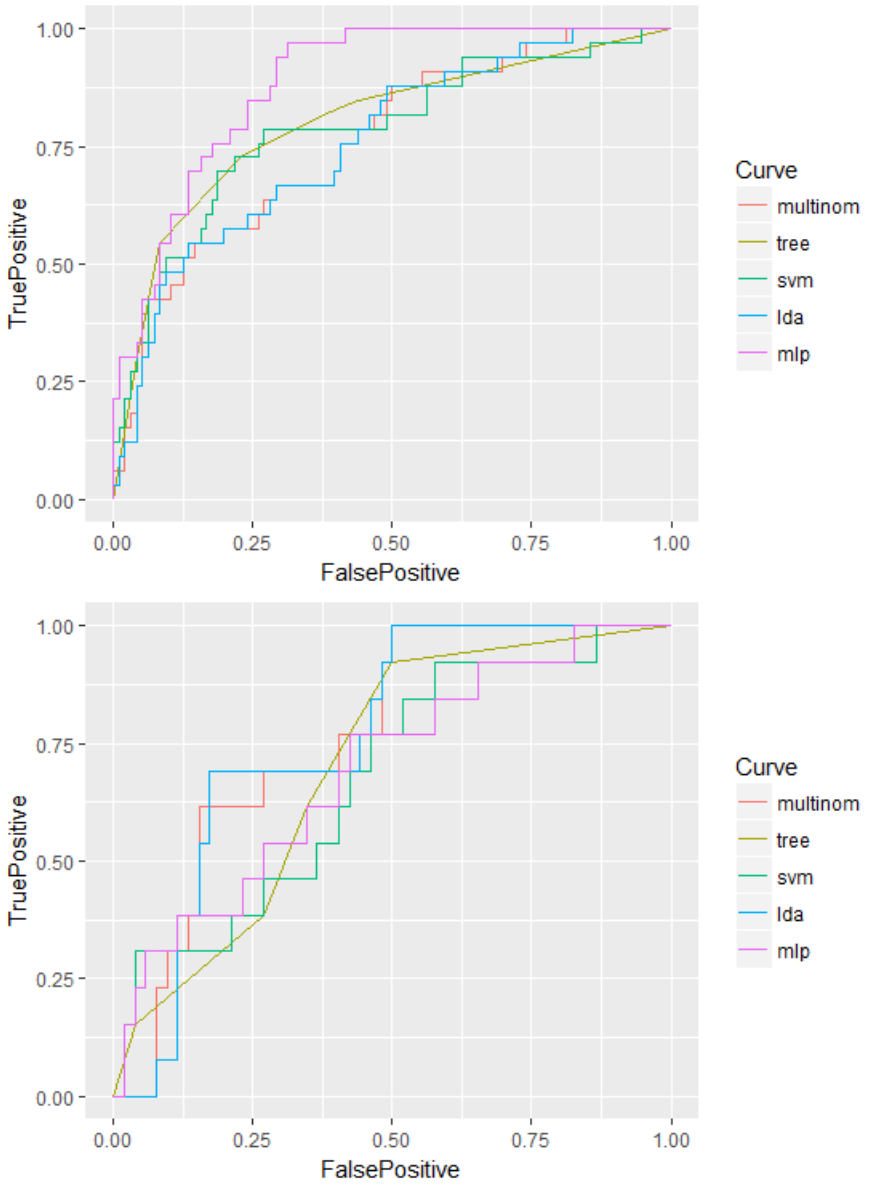


FIGURE 5 ROC curve of five classifiers for training (top) and test (bottom) data.

Step 4 We now consider an accuracy improvement study and compare models based on the above dataset with a larger data containing from 23^{rd} to 27^{th} variables on top of the first nine markers. We use **mydatal** to represent the new dataset in the following.

```
> mydatal <- data.1[,c(1:10,23:27)]
> trainl<-mydatal[idx,]
> testl<-mydatal[-idx,]
```

We may then evaluate the accuracy improvement for the training data. We exemplify the calculation by setting the shallow learning classifier to be multinomial logistic regression. Other classification methods can be used by changing the `method` option in the following `nri` and `idi` functions.

```
#for training data
> idi(y=trainl[,1],m1=trainl[,2:10],m2=trainl[,11:15], method="multinom", k=2)
[1] 0.09115631
> nri(y=trainl[,1],m1=trainl[,2:10],m2=trainl[,11:15], method="multinom", k=2)
[1] 0.03875969
```

The test sample can be similarly evaluated using the following code.

```
#for test data
> fit <- nnet::multinom(trainl$Outcome~.,trainl)
> predict.test.probs <- predict(fit,type='probs',newdata=testl[,c(2:15)])
> pml2_m <- data.frame(1-predict.test.probs,predict.test.probs)
> idi(y=testl[,1],m1=pm2_m,m2=pml2_m,method="prob",k=2)
[1] 0.0707272
> nri(y=testl[,1],m1=pm2_m,m2=pml2_m,method="prob",k=2)
[1] 0
> result <- estp(y=testl[,1],m1=pm2_m,m2=pml2_m,acc="idi",level=0.95,method="prob",k=2)
> result$se
[1] 0.05952015
> result$interval
[1] -0.0595499  0.1750748
```

We next consider the deep learning classifier MLP. Using the same model settings as in Step 3, we apply the following code to obtain the IDI and NRI values for the accuracy improvement using MLP.

```
#for training data
> y=trainl[,1]; d=trainl[,c(2:15)]
> yy <- as.numeric(y)-1
> scaled <- scale(d)
> dd <- data.matrix(scaled)
#construct MLP symbol
...(the same in step 3)
#set model
...(the same in step 3)
[40] Train-accuracy=0.94375
> preds = predict(model, dd)
> predict.train.df <- data.frame(t(preds))
> pml1_mlp=predict.train.df
#accuracy improvement value for training data
> idi(y=trainl[,1],m1=pm1_mlp,m2=pml1_mlp, method="prob", k=2)
```

```
[1] 0.1711035
> nri(y=trainl[,1],m1=pm1_mlp,m2=pml1_mlp, method="prob", k=2)
[1] 0.1472868
#for test data
> d=testl[,c(2:15)]
> dd <- data.matrix(scale(d,attr(scaled, "scaled:center"), attr(scaled, "scaled:scale")))
> preds = predict(model, dd)
> predict.train.df <- data.frame(t(preds))
> pml2_mlp=predict.train.df
> idi(y=testl[,1],m1=pm2_mlp,m2=pml2_mlp, method="prob", k=2)
[1] 0.1370151
> nri(y=testl[,1],m1=pm2_mlp,m2=pml2_mlp, method="prob", k=2)
[1] 0
```

Figure 6 plots the ROC curves of small and large dataset using training and test data respectively. One can observe that switching from the smaller set to the larger set leads to a slightly higher ROC curve for the multinomial logistic regression and the MLP method.

Figure 7 compares the NRI and IDI values of both training and test datasets for five machine learning models.

4.3 | Leukemia

We next consider an example with $M = 3$ categories. Golub et al. (1999)[21] analyzed a leukemia data set using microarray gene expression. The data included three types of acute leukemias: acute lymphoblastic leukemia arising from T-cells (ALL T-cell), acute lymphoblastic leukemia arising from B-cells (ALL B-cell) and acute myeloid leukemia (AML). The data set contains 8 ALL T-cell samples, 19 ALL B-cell samples and 11 AML samples. Each sample contains 3916 gene expression values obtained from Affymetrix high-density oligonucleotide microarrays. The dataset to be analyzed is downloaded from http://www.broad.mit.edu/cgi-bin/cancer/datasets.cgi.

The training and test data were prepared by the authors in this case and we will thus use these data directly.

Step 1 Input the training data. In this case we only consider six gene expression for the simplicity and therefore attain a training data with 38 subjects and 7 variables. The first column `st` indicates the three types of disease status `ALLb, ALLt, AML`.

```
> traindata=read.table('leuk7.txt', head=T)
> str(traindata)
'data.frame': 38 obs. of  7 variables:
$ st            : Factor w/ 3 levels "ALLb","ALLt",..: 1 2 2 1 1 2 1 1 2 2 ...
$ AFFX.BioB.5_at: int  -214 -139 -76 -135 -106 -138 -72 -413 5 -88 ...
$ AFFX.BioB.M_at: int  -153 -73 -49 -114 -125 -85 -144 -260 -127 -105 ...
$ AFFX.BioB.3_at: int  -58 -1 -307 265 -76 215 238 7 106 42 ...
$ AFFX.BioC.5_at: int  88 283 309 12 168 71 55 -2 268 219 ...
$ AB000114_at   : int  72 21 -27 61 16 85 -10 25 -38 65 ...
$ AC000062_at   : int  20 13 -33 -9 -17 -42 6 34 -11 -2 ...
```

Step 2 Input the test data in the same way. These 35 subjects were used by Golub et al. (1999) as the test samples and we thus follow the same analysis as the original authors.

```
> testdata=read.table('test-mit7.txt', head=T)
> str(testdata)
'data.frame': 35 obs. of  7 variables:
$ st            : Factor w/ 3 levels "ALLb","ALLt",..: 1 1 1 1 1 1 1 1 1 1 ...
$ AFFX.BioB.5_at: int  -342 -87 22 -243 -130 -256 -62 86 -146 -187 ...
$ AFFX.BioB.M_at: int  -200 -248 -153 -218 -177 -249 -23 -36 -74 -187 ...
```

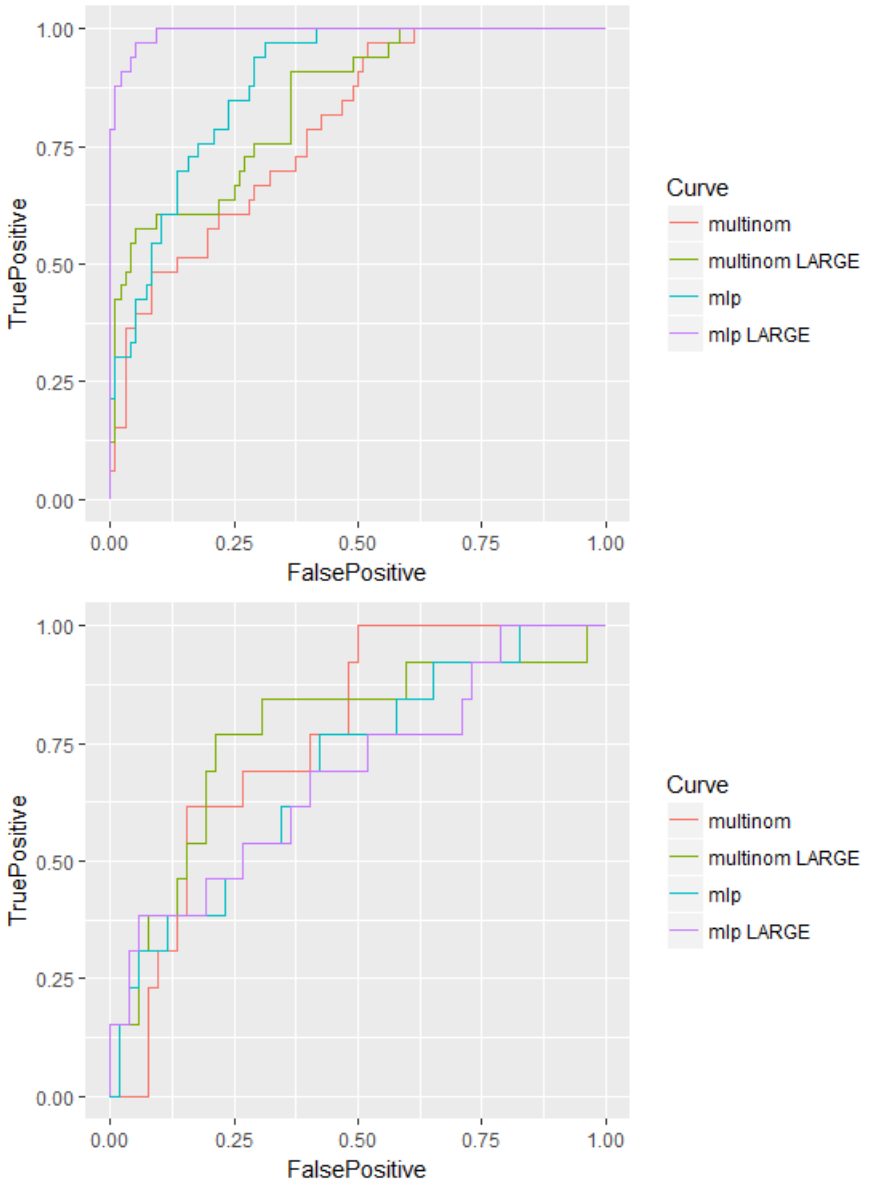


FIGURE 6 ROC curves for training (top) and test (bottom) data. Curves "multinom" and "mlp" are for the baseline model and curves "multinom LARGE" and "mlp LARGE" are for the model with additional markers.

```
$ AFFX.BioB.3_at: int  41 262 17 -163 -28 -410 -7 -141 170 312 ...
$ AFFX.BioC.5_at: int  328 295 276 182 266 24 142 252 174 142 ...
$ AB000114_at   : int  -38 4 31 -212 28 68 45 168 34 249 ...
$ AC000062_at   : int  -14 30 -27 -27 -16 -49 14 19 -66 -81 ...
```

Step 3 We construct a baseline model with the first 4 gene expressions. We used similar `R` code as in the preceding subsection except that we need to set the number of outcome **k=3** in all the functions. We use the following code to compute HUM for a multinomial logistic regression model and R-squared value for a tree model. Other accuracy measures for all kinds of classifiers can be similarly obtained.

```
> train<-traindata[,1:5]
> test<-testdata[,1:5]
```

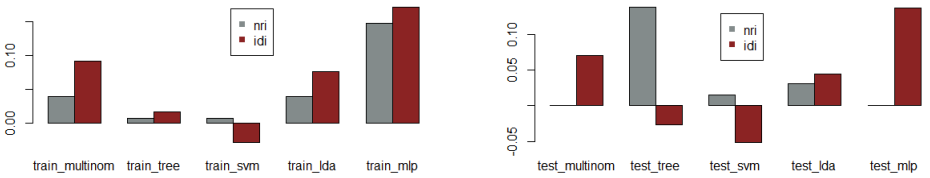


FIGURE 7 NRI and IDI values for all the five machine learning models to predict the breast cancer outcomes: multinomial logistic regression, tree, support vector machine, LDA and MLP. The left panel is for the training dataset while the right panel is for the test dataset. The baseline model contains 9 markers and the new model contains 5 more genes.

```
> hum(y=train[,1],d=train[,c(2:5)],method="multinom",k=3)
[1] 0.4013158
> result <- ests(y=train[,1],d=train[,c(2:5)],acc="hum",level=0.95,method="multinom",k=3)
> result$se
[1] 0.136734
> result$interval
[1] 0.3556999 0.8536842
> rsq(y=train[,1],d=train[,c(2:5)],method="tree",k=3)
[1] 0.4294227
```

We next illustrate the deep learning computation using MLP. In this case, we set the number output nodes to be 3 for the three-category classification.

```
> y=train[,1]; d=train[,c(2:5)]
> yy <- as.numeric(y)-1
> scaled <- scale(d); dd <- data.matrix(scaled)
#construct MLP symbol
> mydata <- mx.symbol.Variable("data")
> fc1 <- mx.symbol.FullyConnected(mydata, num_hidden=500)
> act1 <- mx.symbol.Activation(fc1,  act_type="relu")
> fc2 <- mx.symbol.FullyConnected(act1,  num_hidden=100)
> act2 <- mx.symbol.Activation(fc2,  act_type="relu")
> fc3 <- mx.symbol.FullyConnected(act2,  num_hidden=3)
> softmax <- mx.symbol.SoftmaxOutput(fc3)
> mx.set.seed(0)
#set model
> model <- mx.model.FeedForward.create(softmax, X=dd, y=yy,
+           ctx=mx.cpu(), num.round=41, array.batch.size=40,
+           learning.rate=0.08, momentum=0.9,  eval.metric=mx.metric.accuracy,
+           initializer=mx.init.uniform(0.07),
+           epoch.end.callback=mx.callback.log.train.metric(100))
Start training with 1 devices
......
[41] Train-accuracy=0.710526315789474
```

After the model is fitted with the above procedure, we may then obtain the probability assessment matrix and evaluate the accuracy measures using the following code.

```
#obtain probability matrix
```

```
> preds = predict(model, dd)
> pm1_mlp <- data.frame(t(preds))
> ccp(y=y,d=pm1_mlp,method="prob",k=3)
[1] 0.6543062
> rsq(y=y,d=pm1_mlp,method="prob",k=3)
[1] 0.2306806
> hum(y=y,d=pm1_mlp,method="prob",k=3)
[1] 0.6226077
> pdi(y=y,d=pm1_mlp,method="prob",k=3)
[1] 0.7282695
```

Test sample can be similarly evaluated using the following code.

```
#for test data of MLP
> y1=test[,1]; d=test[,c(2:5)]
> dd <- data.matrix(scale(d,attr(scaled, "scaled:center"), attr(scaled, "scaled:scale")))
> preds = predict(model, dd)
> pm2_mlp <- data.frame(t(preds))
> ccp(y=y1,d=pm2_mlp,method="prob",k=3)
[1] 0.3471178
> rsq(y=y1,d=pm2_mlp,method="prob",k=3)
[1] 0.1863409
> hum(y=y1,d=pm2_mlp,method="prob",k=3)
[1] 0.2030075
> pdi(y=y1,d=pm2_mlp,method="prob",k=3)
[1] 0.3972431
```

Accuracy measures for all the classifiers are reported in Table 2 along with their standard errors (in the parenthesis) and 95% confidence intervals (in the brackets). In this example tree method attains the best in-sample classification accuracy. For example, applying tree method to the four selected gene expression, we can differentiate the three types of tumors with a 86% probability according to the reported HUM value. The out-of-sample performance is much worse for all methods, mainly due to the small sample size.

Step 4 We then compare the above models with more complicated models involving additional markers. To this end, we include two more genetic biomarkers from the data file.

```
> trainl <- traindata[,1:7]
> testl <- testdata[,1:7]
```

We first evaluate NRI and IDI for multinomial logistic regression models, comparing the model with 4 markers and the model with 6 markers.

```
#for training data
> idi(y=trainl[,1],m1=trainl[,2:5],m2=trainl[,6:7], method="multinom", k=3)
[1] 0.07506785
> nri(y=trainl[,1],m1=trainl[,2:5],m2=trainl[,6:7], method="multinom", k=3)
[1] 0.02412281
> result <- estp(y=trainl[,1],m1=trainl[,2:5],m2=trainl[,6:7],acc="nri",level=0.95,
+ method="multinom", k=3)
> result$se
[1] 0.1197613
> result$interval
[1] -0.0331825  0.4055556
```

	CCP				
	multinom	tree	svm	lda	mlp
train	0.52 (0.12)	0.75 (0.10)	0.61 (0.11)	0.51 (0.09)	0.65 (0.07)
	[0.44, 0.86]	[0.48, 0.84]	[0.50, 0.92]	[0.39, 0.76]	[0.52, 0.78]
test	0.36 (0.05)	0.44 (0.12)	0.35 (0.05)	0.36 (0.06)	0.35 (0.06)
	[0.27, 0.46]	[0.18, 0.64]	[0.25, 0.44]	[0.26, 0.48]	[0.23, 0.45]
	R-squres				
	multinom	tree	svm	lda	mlp
train	0.18 (0.12)	0.43 (0.13)	0.01 (0.11)	0.16 (0.10)	0.23 (0.03)
	[0.14, 0.59]	[0.16, 0.62]	[0.02, 0.44]	[0.11, 0.47]	[0.17, 0.29]
test	0.21 (0.04)	0.51 (0.06)	0.04 (0.01)	0.17 (0.03)	0.19 (0.03)
	[0.13, 0.27]	[0.35, 0.58]	[0.02, 0.05]	[0.10, 0.23]	[0.12, 0.25]
	HUM				
	multinom	tree	svm	lda	mlp
train	0.40 (0.14)	0.86 (0.13)	0.04 (0.25)	0.43 (0.12)	0.62 (0.12)
	[0.36, 0.85]	[0.46, 1.00]	[0.09, 0.98]	[0.36, 0.79]	[0.37, 0.87]
test	0.13 (0.11)	0.41 (0.18)	0.05 (0.04)	0.14 (0.11)	0.20 (0.10)
	[0.00, 0.34]	[0.04, 0.75]	[0.00, 0.12]	[0.00, 0.41]	[0.06, 0.42]
	PDI				
	multinom	tree	svm	lda	mlp
train	0.55 (0.10)	0.61 (0.17)	0.37 (0.17)	0.55 (0.09)	0.73 (0.08)
	[0.53, 0.92]	[0.21, 0.74]	[0.41, 0.99]	[0.52, 0.84]	[0.56, 0.87]
test	0.29 (0.09)	0.24 (0.11)	0.21 (0.06)	0.30 (0.10)	0.40 (0.11)
	[0.14, 0.51]	[0.06, 0.48]	[0.11, 0.34]	[0.15, 0.48]	[0.17, 0.62]

TABLE 2 Accuracy for training and test samples in Leukemia data along with their standard errors (in the parenthesis) and 95% confidence intervals (in the brackets).

```
#for test data
> fit <- nnet::multinom(trainl$st~.,trainl)
> predict.test.probs <- predict(fit,type='probs',newdata=testl[,c(2:7)])
> pml2_m <- data.frame(predict.test.probs)
> idi(y=testl[,1],m1=pm2_m,m2=pml2_m, method="prob", k=3)
[1] 0.1926137
> nri(y=testl[,1],m1=pm2_m,m2=pml2_m, method="prob", k=3)
[1] -0.1002506
```

We next consider evaluating NRI and IDI for deep learning classifier MLP.

```
#for training data of MLP
> y=trainl[,1]; d=trainl[,c(2:7)]
> yy <- as.numeric(y)-1
> scaled <- scale(d); dd <- data.matrix(scaled)
#construct MLP symbol
> mydata <- mx.symbol.Variable("data")
> fc1 <- mx.symbol.FullyConnected(mydata, num_hidden=500)
> act1 <- mx.symbol.Activation(fc1,  act_type="relu")
> fc2 <- mx.symbol.FullyConnected(act1,  num_hidden=100)
> act2 <- mx.symbol.Activation(fc2,  act_type="relu")
> fc3 <- mx.symbol.FullyConnected(act2,  num_hidden=3)
```

```
> softmax <- mx.symbol.SoftmaxOutput(fc3)
> mx.set.seed(0)
> model <- mx.model.FeedForward.create(softmax, X=dd, y=yy,
+            ctx=devices, num.round=41, array.batch.size=40,
+            learning.rate=0.08, momentum=0.9,  eval.metric=mx.metric.accuracy,
+            initializer=mx.init.uniform(0.07),
+            epoch.end.callback=mx.callback.log.train.metric(100))
Start training with 1 devices
......
[41] Train-accuracy=0.894736842105263
> preds = predict(model, dd)
> pml1_mlp <- data.frame(t(preds))
> idi(y=trainl[,1],m1=pm1_mlp,m2=pml1_mlp, method="prob", k=3)
[1] 0.2489909
> nri(y=trainl[,1],m1=pm1_mlp,m2=pml1_mlp, method="prob", k=3)
[1] 0.1903907

#for test data of MLP
> d=testl[,c(2:7)]
> dd <- data.matrix(scale(d,attr(scaled, "scaled:center"), attr(scaled, "scaled:scale")))
> preds = predict(model, dd)
> pml2_mlp <- data.frame(t(preds))
> idi(y=testl[,1],m1=pm2_mlp,m2=pml2_mlp, method="prob", k=3)
[1] 0.3865431
> nri(y=testl[,1],m1=pm2_mlp,m2=pml2_mlp, method="prob", k=3)
[1] -0.1002506
```

4.4 | ADHD data

Next we analyze the attention deficit hyperactivity disorder (ADHD) data from the ADHD-200 Sample Initiative ($http://fcon_1000.projects.nitrc.org/indi/adhd200/$). ADHD is a common childhood disorder and can continue through adolescence and adulthood. Symptoms include difficulty in staying focused and paying attention, difficulty in controlling behavior, and overactivity. The dataset that we used is part of the ADHD- 200 Global Competition datasets and can be requested from the ADHD website for research purpose. It consists of 406 subjects, with 221 normal controls and 185 combined ADHD subjects.

Resting state fMRIs and T1-weighted images were acquired for each subject. RAVENS methodology is based on a volume-preserving spatial transformation. Figures 8 and 9 display two typical individuals where the first row is an ADHD patient and the second row is a normal control. The learning task is then to differentiate the two types of patients using the complicated neural image data.

In this case it is infeasible to run any of the shallow learning classifier directly on the image input. The classification is achieved with convolutional neural network (CNN) introduced earlier. For this example, the input data tensor A is of dimension $160\times160\times128$, which can be viewed as a stack of 160 by 160 size images with channel equal to 128. We apply different kernels of sizes $2\times2\times128$, $3\times3\times128$ and $4\times4\times128$, respectively. The pooling layers in the CNN model are all 2×2 $Pool_2$.

The calculation is implemented using the `mxnet` package in `R`. First we need to prepare the data in an appropriate format to be loaded into the training network. The raw image data are stored in NIfTI form under two folders for AD and NC, respectively and can be read in `R` using function `readNIfTI` from library `oro.nifti`.

```
> library(oro.nifti)
> setwd("..\\ADunzip")
#can change "AD" to "NC" to read from the NC folder
#setwd("..\\NCunzip")
> lst<-list.files(getwd())
> len<-length(lst)
```

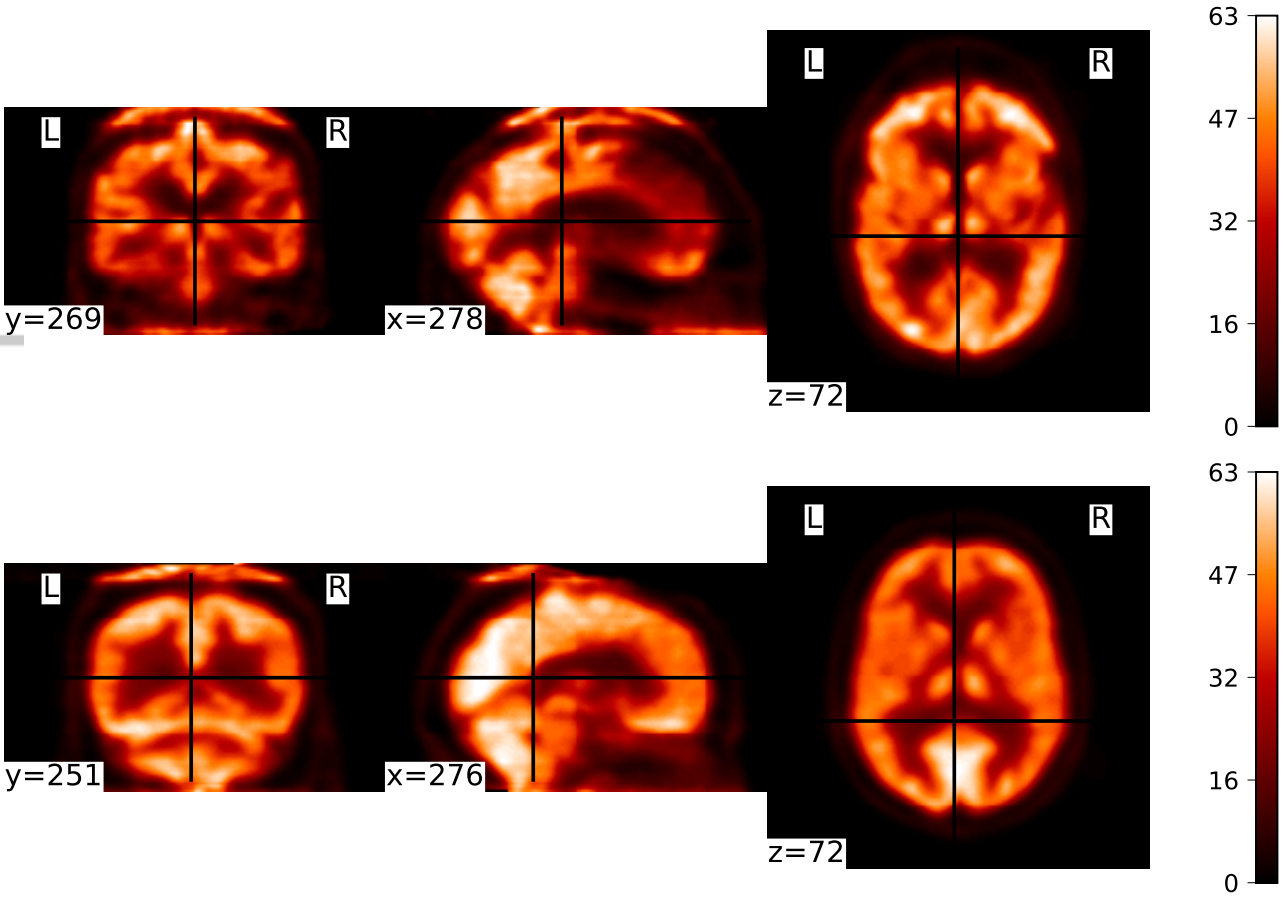


FIGURE 8 Axial, Sagittal, Coronal view at the center of the brain MRI for two selected subjects, one with AD (1st row) and one without AD (2nd row).

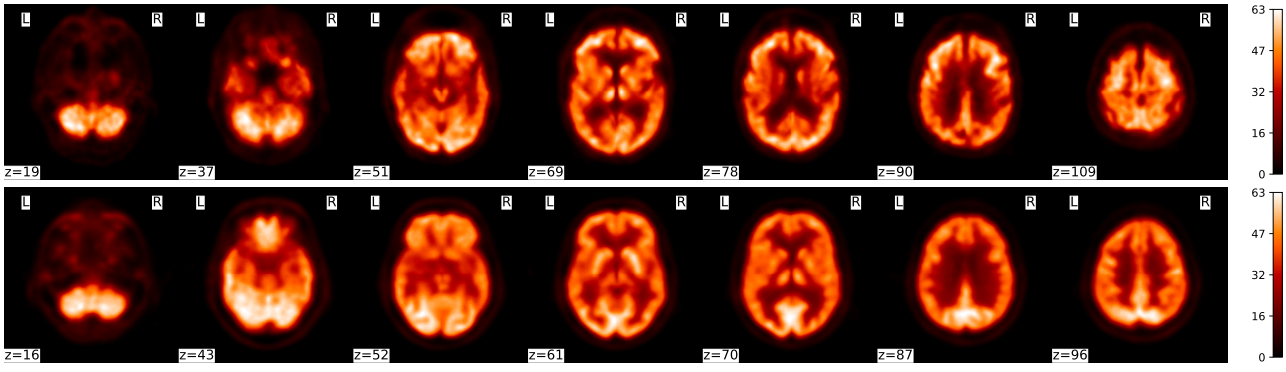


FIGURE 9 MRI images for two selected brains, one with AD (1st row) and one without AD (2nd row).

```
> datlst<-list()
> for(i in 1:len){
>  datlst[[i]]<-readNIfTI(lst[[i]])@.Data[,,,1]
> }
> ADlst<-datlst
```

```
>trainSampleSize<-300
>testSampleSize<-106
>trainADsize<-137 #300*185/406
>trainNCsize<-163 #300-137
>biglist<-c(ADlst,NClst)
>bigres<-c(rep(1,length(ADlst)),rep(2,length(NClst)))
>trainID<-c(sample(1:185,trainADsize,replace=F),sample(186:406,trainNCsize,replace=F))
>testID <-seq(length(biglist))[-trainID]
>trainres<-bigres[trainID]
>testres<-bigres[testID]
```

We arrive at a list object `biglist` including both AD and NC subjects. The above code also randomly split the whole data into a training set ($n = 300$) and a test set ($n = 106$). The data to be fed into the CNN algorithm is a four dimensional array, with the first three dimensions defining the size of the tensor images and the last dimension being the sample size. Next we create the network structure as follows:

```
> library(mxnet)
> data <- mx.symbol.Variable('data')
> conv_1 <- mx.symbol.Convolution(data = data, kernel = c(2,2), num_filter = 8)
> tanh_1 <- mx.symbol.Activation(data = conv_1, act_type = "relu")
> pool_1 <- mx.symbol.Pooling(data = tanh_1, pool_type = "max", kernel = c(2,2),
>  stride = c(2, 2))
> conv_2 <- mx.symbol.Convolution(data = pool_1, kernel = c(2,2), num_filter =8)
> tanh_2 <- mx.symbol.Activation(data = conv_2, act_type = "relu")
> pool_2 <- mx.symbol.Pooling(data=tanh_2, pool_type = "max", kernel = c(2, 2),
>  stride = c(2, 2))
> flatten <- mx.symbol.Flatten(data = pool_2)
> fc_1 <- mx.symbol.FullyConnected(data = flatten, num_hidden = 1000)
> train_model <- mx.symbol.SoftmaxOutput(data = fc_1)
> device <- mx.gpu()
> mx.set.seed(30)
> model <- mx.model.FeedForward.create(train_model, X = traindata, y = trainres,
>  ctx = device, num.round = 1000,  learning.rate = 0.0001, momentum = 0.9,
>  array.batch.size = 1, wd = 0.001, eval.metric = mx.metric.accuracy,
>  epoch.end.callback = mx.callback.log.train.metric(100))
```

The above construction of the network with `mxnet` in `R` is lengthy but intuitive, like other deep learning approaches. Type `relu` forces all the negative value to zero, providing relatively faster training speed than *sigmoid* or *tanh* function. In the pooling function, we have a parameter `stride` which defines how the sliding window on each layer is moving on both horizontal and vertical directions.

We specify the device for the training method to be GPU with `device <- mx.gpu()` command. Note that to facilitate the graphical computing, the user's graphic card needs to support CUDA and has the CUDA Toolkit installed. Otherwise only CPU is supported and this line of code needs to be modified as `device <- mx.cpu()`.

After the model is trained, we may use the trained classifier to predict the test data.

```
> predictProb<- predict(model, testdata)
```

From `predictProb` one can obtain the predicted probabilities and we can further calculate accuracy measures such as HUM and CCR. Such computation can be done in the same manner as in the previous two examples and we do not repeat the evaluation in this tutorial for space consideration.

ACKNOWLEDGMENTS

We thank Adrian Roellin for helpful comments that improve the clarity of the presentation on deep learning. The work was partly supported by Academic Research Funds R-155-000-205-114, R-155-000-195-114 and Tier 2 MOE funds in Singapore MOE2017-T2-2-082: R-155-000-197-112 (Direct cost) and R-155-000-197-113 (IRC).

Financial disclosure

None reported.

Conflict of interest

The authors declare no potential conflict of interests.

References

1. Zhou X H, Obuchowski N A, McClish D K. *Statistical Methods in Diagnostic Medicine*. John Wiley & Sons: New York.; 2002.
2. Pepe M S. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford University Press; 2003.
3. Li Jialiang, Jiang Binyan, Fine Jason P.. Multicategory reclassification statistics for assessing improvements in diagnostic accuracy. *Biostatistics.* 2013;14(2):382-394.
4. Li Jialiang, Fine Jason P.. ROC analysis with multiple classes and multiple tests: methodology and its application in microarray studies. *Biostatistics.* 2008;9(3):566-576.
5. Van Calster Ben, Vergouwe Yvonne, Looman Caspar WN, Van Belle Vanya, Timmerman Dirk, Steyerberg Ewout W. Assessing the discriminative ability of risk models for more than two outcome categories. *European journal of epidemiology.* 2012;27(10):761–770.
6. Kwak Chanyeong, Clayton-Matthews Alan. Multinomial logistic regression. *Nursing research.* 2002;51(6):404–410.
7. Vapnik V. *Statistical Learning Theory*. Wiley, New York; 1998.
8. Suykens Johan AK, Vandewalle Joos. Least squares support vector machine classifiers. *Neural processing letters.* 1999;9(3):293–300.
9. Crammer K, Singer Y. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research.* 2001;2:265-292.
10. Lee Y., Lin Y., Wahba G. Multicategory Support Vector Machines, Theory, and Application to the Classification of Microarray Data and Satellite Radiance Data. *Journal of the American Statistical Association.* 2004;99:67-81.
11. Liaw Andy, Wiener Matthew, others . Classification and regression by randomForest. *R news.* 2002;2(3):18–22.
12. Mika Sebastian, Ratsch Gunnar, Weston Jason, Scholkopf Bernhard, Mullers Klaus-Robert. Fisher discriminant analysis with kernels. In: :41–48IEEE; 1999.
13. Schalkoff Robert J. *Artificial neural networks*. McGraw-Hill New York; 1997.
14. Shore John, Johnson Rodney. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on information theory.* 1980;26(1):26–37.
15. Gardner Matt W, Dorling SR. Artificial neural networks (the multilayer perceptron)?a review of applications in the atmospheric sciences. *Atmospheric environment.* 1998;32(14):2627–2636.

16. Zhao Kai, Huang Liang. Minibatch and Parallelization for Online Large Margin Structured Learning.. In: :370–379; 2013.

17. Hecht-Nielsen Robert, others . Theory of the backpropagation neural network.. *Neural Networks.* 1988;1(Supplement-1):445–448.

18. Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. Imagenet classification with deep convolutional neural networks. In: :1097–1105; 2012.

19. Li Jialiang, Chow Yanyu, Wong Weng Kee, Wong Tien Yin. Sorting multiple classes in multi-dimensional ROC analysis: parametric and nonparametric approaches. *Biomarkers.* 2014;19(1):1–8.

20. Novoselova Natalia, Beffa Cristina Della, Wang Junxi, Li Jialiang, Pessler Frank, Klawonn Frank. HUM calculator and HUM package for R: easy-to-use software tools for multicategory receiver operating characteristic analysis. *Bioinformatics.* 2014;30(11):1635.

21. Golub TR, Slonim DK, TAMAYO P, et al. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science.* 1999;286:531-537.

22. Nakas C T, Yiannoutsos C T. Ordered multiple-class ROC analysis with continuous measurements. *Statistics in Medicine.* 2004;23:3437-3449.

23. Li J, Chow Y, Wong W K, Wong T Y. Sorting Multiple Classes in Multi-dimensional ROC Analysis: Parametric and Nonparametric Approaches. *Biomarkers.* 2014;19(1):1-8.

24. Li J, Fine J P. ROC analysis with multiple tests and multiple classes: methodology and applications in microarray studies. *Biostatistics.* 2008;9:566-576.

25. Cox D R, Wermuth N. A comment on the Coefficient of determination for binary response. *The American Statisticians.* 1992;46:1-4.

26. Menard S. Coefficients of determination for multiple logistic regression analysis. *The American Statisticians.* 2000;54:17-24.

27. Hu B, Palta M, Shao J. Properties of R2 statistics for logistic regression. *Statistics in Medicine.* 2006;25:1383-1395.

28. Tjur T. Coefficients of determination in logistic regression models - a new proposal: The coefficient of discrimination. *The American Statistician.* 2009;64:366-372.

29. Li J, Jiang B, Fine J P. Multicategory reclassification statistics for assessing improvements in diagnostic accuracy. *Biostatistics.* 2013;14(2):382-394.

30. Van Calster B., Van Belle V., Vergouwe Y., Timmerman D., Van Huffel S., Steyerberg E.W.. Extending the c-statistic to nominal polytomous outcomes: The Polytomous Discrimination Index. *Statistics in Medicine.* 2012;31:2610-2626.

31. Van Calster B., Vergouwe Y., Looman C. W. N., Van Belle V., Timmerman D., Steyerberg E.W.. Assessing the discriminative ability of risk models for more than two outcome categories: a perspective. *European Journal of Epidemiology.* 2012;27:761-770.

32. Li Jialiang, Feng Qunqiang, Fine Jason P, Pencina Michael J, Van Calster Ben. Nonparametric estimation and inference for polytomous discrimination index. *Statistical Methods in Medical Research.* 2017;:0962280217692830.

33. Pencina M. J., D'Agostino Sr R. B., D'Agostino Jr R. B., Vasan R. S.. Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond. *Statistics in Medicine.* 2008;27:157–172.

34. Pepe Margaret Sullivan, Janes Holly, Longton Gary, Leisenring Wendy, Newcomb Polly. Limitations of the odds ratio in gauging the performance of a diagnostic, prognostic, or screening marker. *American journal of epidemiology.* 2004;159(9):882–890.

35. Steyerberg EW, Vickers AJ, Cook NR, et al. Assessing the Performance of Prediction Models, A Framework for Traditional and Novel Measures. *Epidemiology.* 2010;21:128-138.

36. Pencina M. J., D'Agostino Sr R. B., Steyerberg E. W.. Extensions of net reclassification improvement calculations to measure usefulness of new biomarkers. *Statistics in Medicine.* 2011;30:11–21.

37. Pencina M J, D'Agostino Sr. R B, Demler O V. Novel metrics for evaluating improvement in discrimination: net reclassification and integrated discrimination improvements for normal variables and nested models. *Statistics in Medicine.* 2012;31:101-113.

38. Austin PC, Steyerberg EW. Predictive accuracy of risk factors and markers: a simulation study of the effect of novel markers on different performance measures for logistic regression models. *Statistics in Medicine.* 2013;32:661-672.

39. Pepe M S, Feng Z, Gu J W. Comments on 'Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond' by M. J. Pencina et al., Statistics in Medicine (DOI: 10.1002/sim. 2929). *Statistics in Medicine.* 2008;27(2):173–181.

40. Hilden Jørgen, Gerds Thomas A. A note on the evaluation of novel biomarkers: do not rely on integrated discrimination improvement and net reclassification index. *Statistics in Medicine.* 2014;33(19):3405–3414.

41. Kerr Kathleen F, Wang Zheyu, Janes Holly, McClelland Robyn L, Psaty Bruce M, Pepe Margaret S. Net reclassification indices for evaluating risk prediction instruments: a critical review. *Epidemiology.* 2014;25(1):114–121.

42. Pepe M S, Feng Z, Gu J W. Comments on 'Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond' by M. J. Pencina et al.. *Statistics in Medicine.* 2008;27:173-181.

43. Hilden Jørgen. Commentary: On NRI, IDI, and good-looking statistics with nothing underneath. *Epidemiology.* 2014;25(2):265–267.