STA 242 Final Project

Junxiao Bu(999452701) Xuesi Feng(999492046) Mutian Niu(999529375)

June 7, 2015

SSH:git@bitbucket.org:shingtime/sta242-final-project.git

Overview

The intention of this project is to explore both straightforward and advanced machine learning methods for a specific dataset. We decided to implement seven different methods and apply them to a Kaggle competition data.

For this project, we use a dataset with 93 features for roughly 60000 products that was provided by Otto Group. Dataset was split into training and testing sets at 20:80, and the correct categories of the products in the training set was given. The objective is to build an optimal predictive model that is able to distinguish between major product categories.

In this project, cross validation procedure as well as different classification approaches will possibly be used to predict the categories of the products including kNN,Naïve Bayesian, ensemble meta-algorithm (Random Forest, Extra Trees and Gradient boosting), deep learning, and Support Vector Machine.

We mainly use iPython and R to finish tasks. There are many modules that can be used. We decide to use some mature modules and packages such as **scikit-learn** and **Multilayer Perceptron**.

Classification accuracy (Estimating error rates) and the reasoning and advantages of each approach will be compared and discussed.The optimal model/approach will be chosen and presented as the output of the project and also to classify Otto products into the different categories.

For this dataset and our selected tuning parameters, our methods rank as

Data Description

The dataset can be easily downloaded from Kaggle.com.There are 61878 products in the training dataset. All the 93 features are named as *feat_i* with numberc values. For the response variable,it is a categorical value with 9 different classes.There are no NA values for all variables.

In figure 1, the range of each feature is shown. We find that majority of features' value are in a relatively same range.Since we don't know what these features are, we cannot compare among different features.

In figure 2, the distribution of different classes doesn't have any pattern. It seems that there are more products that belong to *class_2* and *class_6* than other classes.

Since we don't know the units of all the features, for the accuracy of the following analysis,we choose to standardlize all the feature variables.

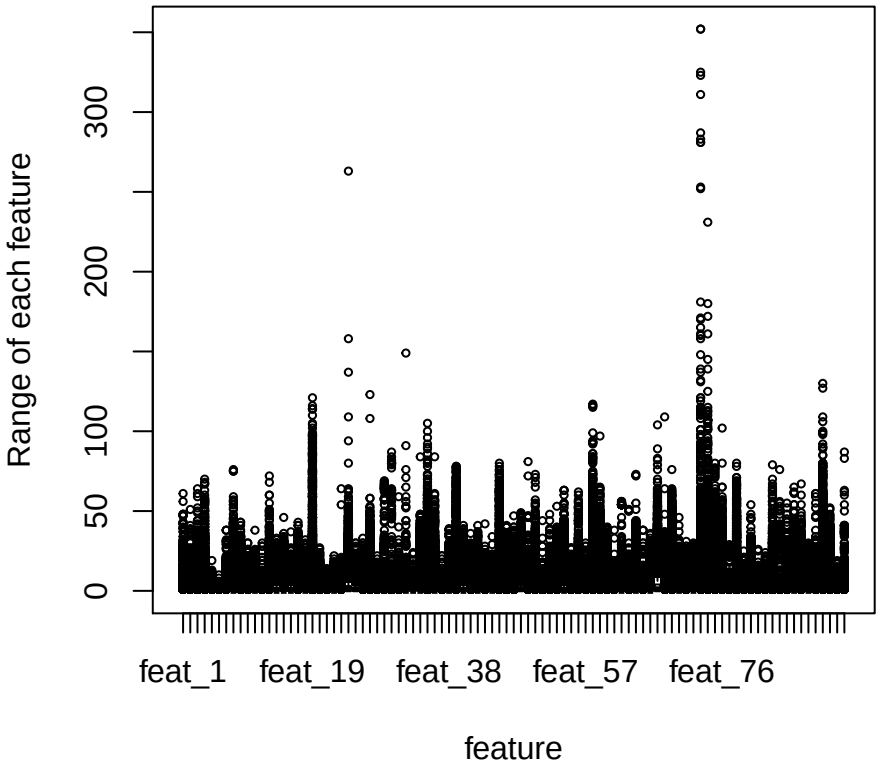

Figure 1: Range of Feature

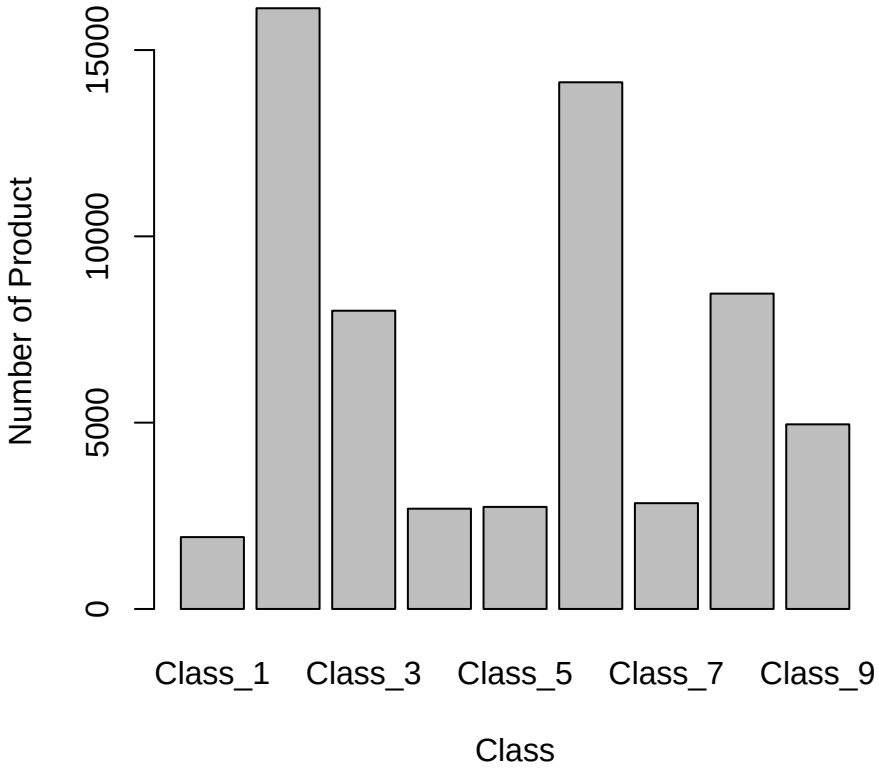

Figure 2: Count of different classes

Method Background

k Nearest Neighbor(kNN)

The **k Nearest Neighbor** algorithm is one of the most widely used classification techniques and probably the simpliest that is often applied to some pattern recognition problems. It is a relatively simple algorithm that imput all available training cases and then classifies a new case based on a majority vote of its neighbors according to the distance. A case will be assigned to the class that most of its k nearest neighbors belonging to, for instance, k = 1 means that the object is simply assgined to the class of that simgle nearest neighbors (k is a positive integer).

Along with the easy rule, different distance functions and value of k will affect the performance of the classification. There are different distance measures, such as euclidean, maximum, manhattan, canberra, and minkowski etc,. In this project, we conducted k-NN using only eculidean distance function. In addition, we were also trying different k values to find the optimal number of neighbors which help to get the smallest error rate, or mean square error.

- *Euclidean*: usual square distance between the two vectors.

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Naïve Bayes

Assume there are m features and n classes in the dataset. Also assume that in the dataset, each cell corresponds to the count of features. In such a situation, multinomial naive bayes could be applied. Let $\mathbf{x}_k$ be the feature vector, i.e. the occurances of each feature on kth class, and assume that for all the features in the feature vector $\mathbf{x}_k$, the features are independent. Then according to the Bayes Theorem, we have:

$$p(C_k|\mathbf{x}_k) = \frac{p(C_k)p(\mathbf{x}_k)}{p(\mathbf{x})} \propto p(C_k)p(\mathbf{x}_k)$$

Also assume that for any sample in the dataset, with the kth class given, then for each feature in the feature vector $\mathbf{x}_k$, the feature is independent and obeys multinomial distribution; i.e.

$$\mathbf{x}_k \sim Multinomial(p_{1k}, p_{2k}, \cdots, p_{mk})$$

Thus the posterior probability satsifies:

$$p(C_k|\mathbf{x}_k) \propto p(C_k)\frac{(\sum_{i=1}^{m} x_i)!}{\prod_{i=1}^{m} x_i!}\prod_{i=1}^{m} {p_i}^{x_i} \propto p(C_k)\prod_{i=1}^{m} {p_{ik}}^{x_{ik}}$$

Consider the whole dataset, the posterior probability is:

$$p(\mathbf{C}|\mathbf{X}) \propto \prod_{k=1}^{n}(p(C_k)\prod_{i=1}^{m} {p_{ik}}^{x_{ik}})$$

According to the maximum a posteriori rule, and take logarithm of the both side, we have:

$$p = \max_{p} \sum_{k=1}^{n} \log p(C_k) + \sum_{k=1}^{n} \sum_{i=1}^{m} x_{ik} log(p_{ik})$$

Take 1st order derivative of the target function and set it to 0, and solve the formulas using Lagrange multipliers, we have for given jth feature on kth class,

$$\hat{p}_{jk} = \frac{x_{jk}}{x_{\cdot k}}$$

Where $x_{\cdot k} = \sum_{j=1}^{m} x_{jk}$. Also, in order to deal with unobserved class, apply Laplace smoothing on the classifier, i.e.

$$\hat{p}_{jk} = \frac{x_{jk} + 1}{x_{\cdot k} + n}$$

Tree Based Methods

For this part, we mainly implment three ensemble methods to predict the classes of our dataset. There are two types of ensemble methods.

The first type is average methods. The principle of these kinds ofmethods is to build several independently estimators and then to average their predictions.Generally, the combination of estimator has better performance than the single estimator. Specificlly, for this problem, we use **random forest** and **Extra trees** to fit a number of decision tree classifiers on the dataset and use averaging to improve the predictive accuracy and control over-fitting.

The second type is boosting methods. In these kinds of methods, base estimators are built sequentially and we tries to reduce the bias of the combined estimators. The important feature is that we tries to combine several weak models to generate an better combined model.Specificlly, for this problem, we use **GradientBoosting** to build a additive decision model.We optimize the model by calculating the negative gradient of the multinomial deviance loss function.

Random Forest

The **Random Forest** algorithm provides an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees. In this method, we build a number of decision trees on bootstrappedtraining samples.But when building these decision trees, each time a random selection of m predictors is chosen as split candidates from the full set of p predictors.

Extra Trees

The **Extra Trees** Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with **Random Forest** method is that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap sample) to grow the trees.

Gradient Boosting

Unlike in bagging, in boosting we fit the tree to the entire training set, but adaptively weight the observations to encourage better predictions for points that were previously misclassified.we weight misclassified observations in such a way that they get properly classified in future iterations.

Difference between Random Forest and Gradient Boosting

Random Forest are trained with random sample of data and it trusts randomization to have better generalization performance on out of train set. On the other hand, **Gradient Boosting** tries to find optimal linear combination of trees(final model is the weighted sum of individual tree's prediction).

We know that error can be composited from bias and variance. A too complex model has low bias but large variance, while a too simple model has low variance but large bias, both leading a high error but two different reasons. As a result, two different ways to solve the problem come into people's mind, variance reduction for a complex model, or bias reduction for a simple model, which refers to **Random forest** and **Gradient Boosting**.

Besides, the computational speed of **Random Forest** is faster. **Random Forest** is easily run in parallel, whereas **Gradient Boosting** only run sequentially. If we have limitted computational resources, we may have to implement **Random Forest**.

Difference between Random Forest and Extra Trees

From the bias-variance point of view, the rationale behind the **Extra Trees** method is that the explicit randomization of the sample with ensemble averaging should be able to reduce variance more strongly than the weaker randomization schemesused by other methods. The usage of the full original learning sample rather than bootstrap samples is motivated in order to minimize bias.

Deep Learning

Multilayer Neural Network

Inspired by biological neural networks, people invented neural network technologies.

sigmoid functions, softmax functions and regressions In general, sigmoid functions are functions which has S-shape curve, if plotted on the 2D space. The most commonly used sigmoid functions include:

$$f(\mathbf{x}) = \frac{1}{1 + e^{(-\mathbf{w}^T\mathbf{x})}}$$

$$f(\mathbf{x}) = \tanh(\mathbf{w}^T\mathbf{x})$$

Both of these two functions have S-shape curve, for example:

The difference is, first function(also called logistic function) maps $[-\infty, \infty]$ to $[0, 1]$, and the second function(also called hyperbolic tangent function) maps $[-\infty, \infty]$ to $[-1, 1]$. In neural networks, these functions are also used as binary classifiers because of their S-shaped curve. However, logistic function

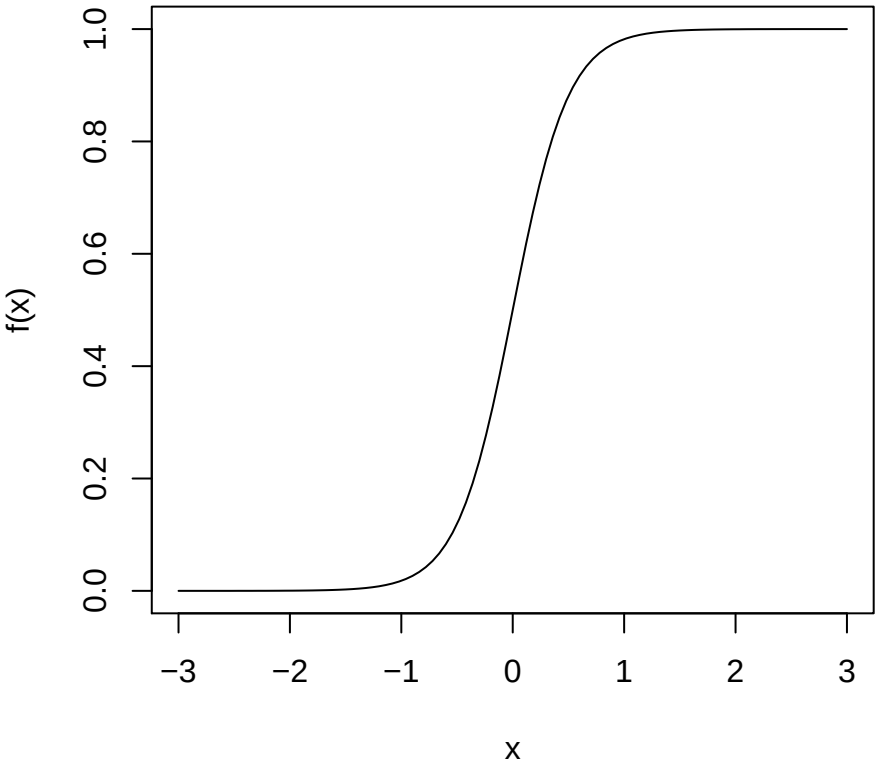

Figure 3: Logistic Shape

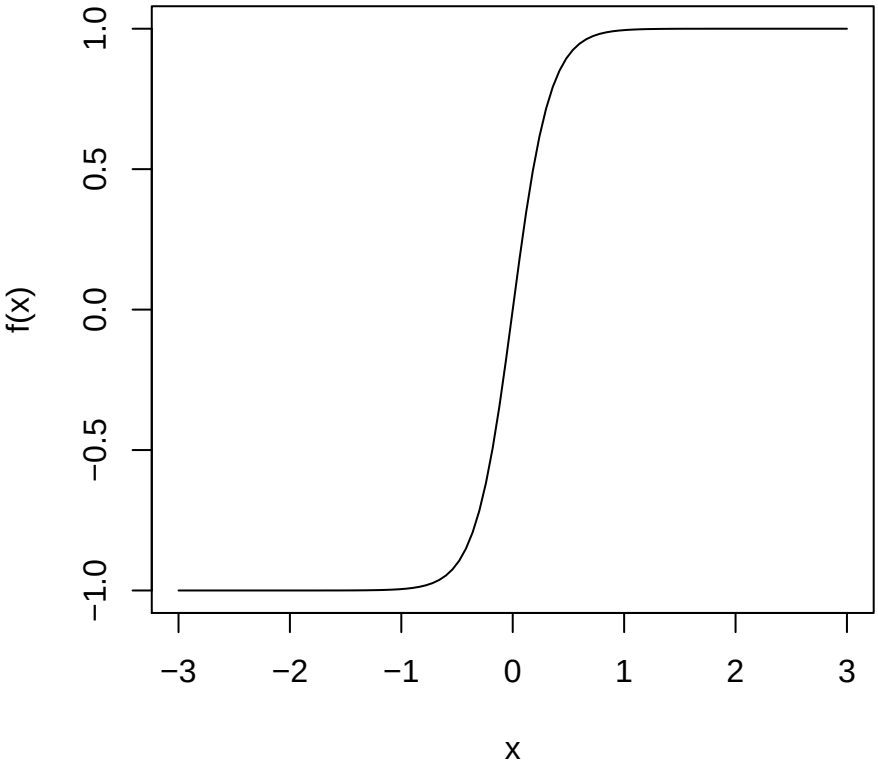

Figure 4: Hypobolic Tangent Shape

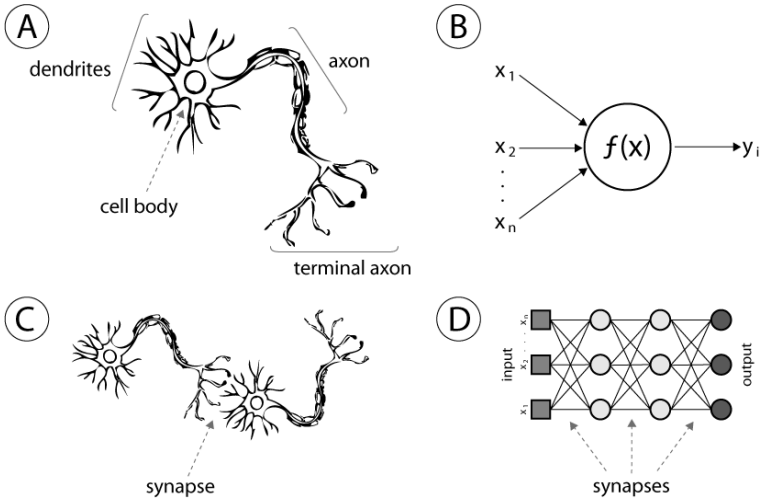

Figure 5: simarities[1]

is more commonly used, since the output can be explained as probabilities of the observation falling in some specified class. In this case, logistic regression, which aims to find the "best" parameter vector $\mathbf{w}$(determined by KL-divergence), can be used as a 1-0 classifier.

Similarily, softmax function, which is defined as:

$$f_k(\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{i=1}^{n} e^{\mathbf{w}_i^T \mathbf{x}}}$$

This function is also called "multinomial logistic regression". Aiming at finding the best parameter matrix $\mathbf{W}$ minimizing the KL divergence, tt can be applied in n-class classcification.

The difference of logistic regression and softmax regression is: logistic regression is used as 1-0 classifier; That is, to predict "whether the given point falling to some specific class". Under such circumstance, given n classes, the point can fall into k classes, $k \geq 1$, and these classes are not exclusive. But the softmax regression is used as a n classifier to predict "which exactly one class will the given point fall into", and classes are "exclusive".

Each neuron can be simulated as a logistic function, with

- dendrite as input from the outputs of other logistic functions;
- cell body as the activation function, often logistic function;
- axon as output from the logistic function;
- terminal axons as connections to neighbors of other logistic functions.

In this project, the fully connected multilayer neural network is adopted. Take the example shown in delta subplot in the above plot, the three layer fully connected multilayer neural network can be expressed as follows in a mathematical way:

$$f(\mathbf{x}) = f_3(\mathbf{W}_3 f_2(\mathbf{W}_2(f_1(\mathbf{W}_1\mathbf{x}) + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3)$$

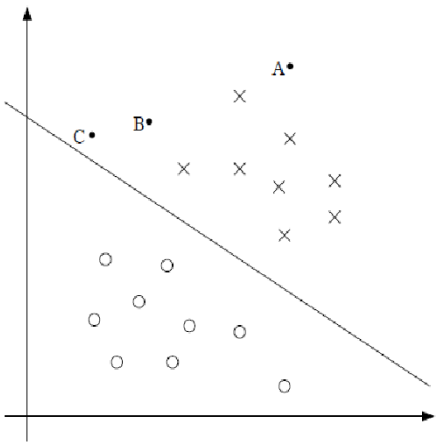

Figure 6: 2D-points Classification Separating by Straight Line (Andrew Ng)[2]

Where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$ are 3x3 weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ are 3x1 biases, f_1, f_2 are sigmoid activation functions, and f_3 is the output softmax function, used as n-classifier.

As for training algorithm, greedy back-propagation algorithm are often used. The main idea is:

- perform a feed forward pass for each level, until it reaches the output.
- calculate the errors of output layer.
- backpropagate the errors, calculated the errors that each level should be responsible for.
- using gradient descent to tune the parameters for the level that the back-propagated errors reach.

In neural network training, such a procedure is also called an epoch. Limited by the number of pages in requirement, the mathematical deductions are omitted here.

Support Vector Machine

Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane which is among the best "off-the-shelf" supervised learning algorithm. It constructs a hyperplane or set of hyperplanes in future space given labeled training data set, and it outputs the optimal hyperplane(s) which categorizes new case in order to achieve the classfication. For example, as shown on Figure.1, a linearly separable set of 2D-points which belong to one of two classes, a separating straight line is helping us label A, B, and C. (Due to the limitation of report length, we would not show the entile algorithm of SVM.)

As the enlarge of dimentional space, the calculation of feature mappings increased dramaticlly. In order keep the computational load reasonable and avoid high-dimention calculation, the feature mappings used by SVM schemes are designed to ensure that dot products being inexpensive to calculate, by defining a kernel funtion $K(x, z)$ selected to suit different problems. In this case, we were trying SVM with,

- *Linear*: $K(x, z) = \langle x, z \rangle$

- *Polynomial (quadratic)*: $K(x, z) = (\langle x, z \rangle + R)^d$
- *Radial (Gaussian, degree = 2)*: $K(x, z) = exp(-\frac{||x_1 - x_2||^2}{2\sigma^2})$

Results

kNN

Since all predictor variables are continuous, here we can apply kNN algorithm through an R packages "FNN", which runs the nearest neighbor more efficiently than the packages "class". We split the standardized dataset into training and test sets at a 80:20 ratio. Also, we made use of 5-fold cross validation in order to reduce the random errors and mostly utilize the samples.

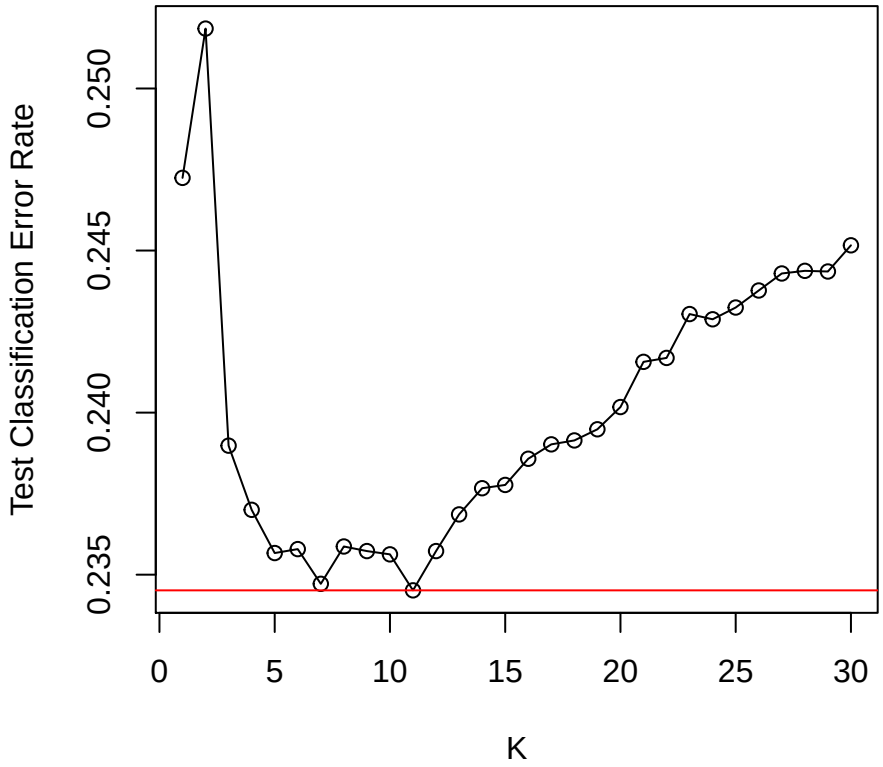

Figure 7: Misclassification Rate over Different k

K value ranges from 1 to 30 was conducted to find the optimal k. As shown on Figure 7, the misclassification rate decreases as the increasing of k value then starts increasing after reaching the trough (as maked in red line). From cross-validation, it seems that we shall select k = 11 as for the best fit model which minimize the cross validation error rate. However, k = 5 also gave a pretty close error rate. Hence, we tried to fit both k = 5 and 11 in order to pick the best model.

Then we used the cross validation result to predict the testset and we got a relatively good error rate for both k = 5 and k = 11, as shown on Table 1. Indicating that both kNN models were performing pretty stable between training set and test set, error rate around 0.23.

Table 1: Error Rate for k-nearest Neighbor

k-NN	Test Misclassification Rate
k = 5	0.2310116
k = 11	0.2313348

Naïve Bayes

In python, scikit-learn package provides mature Multinomial Naive Bayes training algorithm. Using Multinomial Naive Bayes algorithm to make the prediction, and compare that with real classfication result, we can find that the error rate is around 66.6%. That is, some assumptions in Multinomial Naive Bayes does not hold.

Tree Based Methods

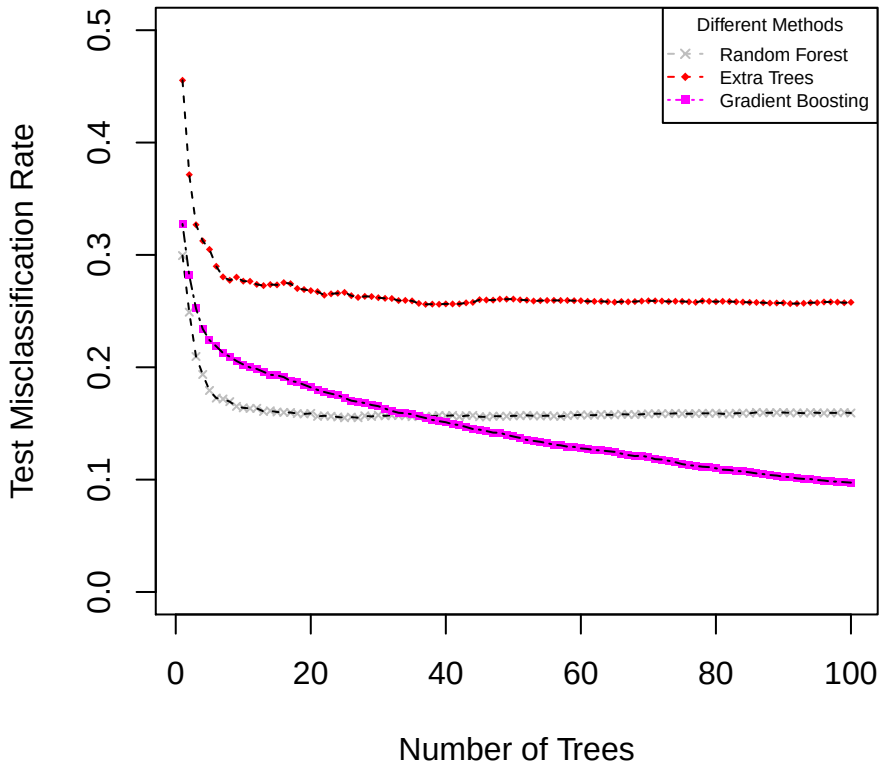

Figure 8: Test Misclassification Rate of Different Tree based Methods

From the plot, the test misclassification rate of **Random Forest** and **Extra Trees** start to converge when number of trees are close to 20. Not surprisingly, the test misclassification rate of **Gradient Boosting** continues decreasing since the algorthim decides so.

We calculate the minimum misclassification rate of each method. The best performance model's summary is in the following table.

Table 2: Summary of Best Models through Three Methods

Methods	**Tree Number**	**Tree Depth**	**Predictor Number**	**Misclassification Rate**
Random Forest	25	18	9	15.5
Extra Trees	37	18	9	25.6
Gradient Boosting	100	10	9	9.7

In this model, when tuning parameters are same across three methods, the **Gradient Boosting** gives us the best performace with a test missclassification rate of 0.097. Notice that the parameter **Tree Depth** of **Gradient Boosting** is smaller because we want to reduce the overfitting problem since this method always has this kind of risk.

Deep Learning

GPU can significantly increase the speed of multilayer neural network training; However, it is painful to use CUDA or OpenCL to build hardware-accelerated neural networks, since people need to deal with a lot of lower level details to build a robust and fast neural network. However in python, package Lasange, which based on package Theano, provides quite a good abstraction of neural network building procedures, thus makes it more friendly to build GPU accelerated neural networks.

All the neural networks are trained for 40 epochs.

Here is different accuracy rates v.s. number of epochs plot, given 4 neural networks with different number of levels:

The structures of these four neural networks are as follows:

- Net0: 1 layer neural network with 256 units and the output softmax layer;
- Net1: 2 layer neural network with 256 units in the first layer, 512 units in the second layer and the output softmax layer;
- Net2: 3 layer neural network with 256 units in the first layer, 512 units in the second layer, 256 units in the third layer and the output softmax layer;
- Net3: 3 layer neural network with 256 units in the first layer, 512 units in the second layer, 256 units in the third layer, 256 units in the 4th layer and the output softmax layer.

It is not hard to find out that as the number of layers increases, the validation accuracy is not necessary increase. From this, it is also reasonable to assume that as the number of layer increases, the performance of neural network does not necessary goes better.

In order to add some sparsity, the dropout layer is introduced. this layer is designed to dropout some "unnecessary" connections randomly, so that there won't be overfitting.

Here is different accuracy rates v.s. number of epochs plot, plotted.

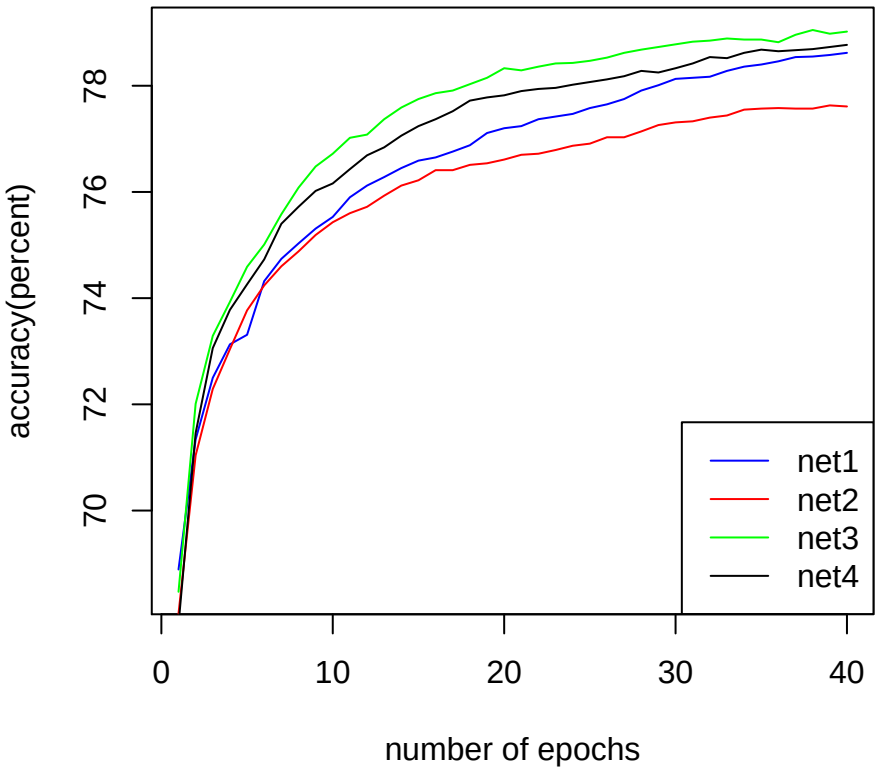

Figure 9: Valid. Acc. v.s. Num. of Epochs for NN with Different Levels

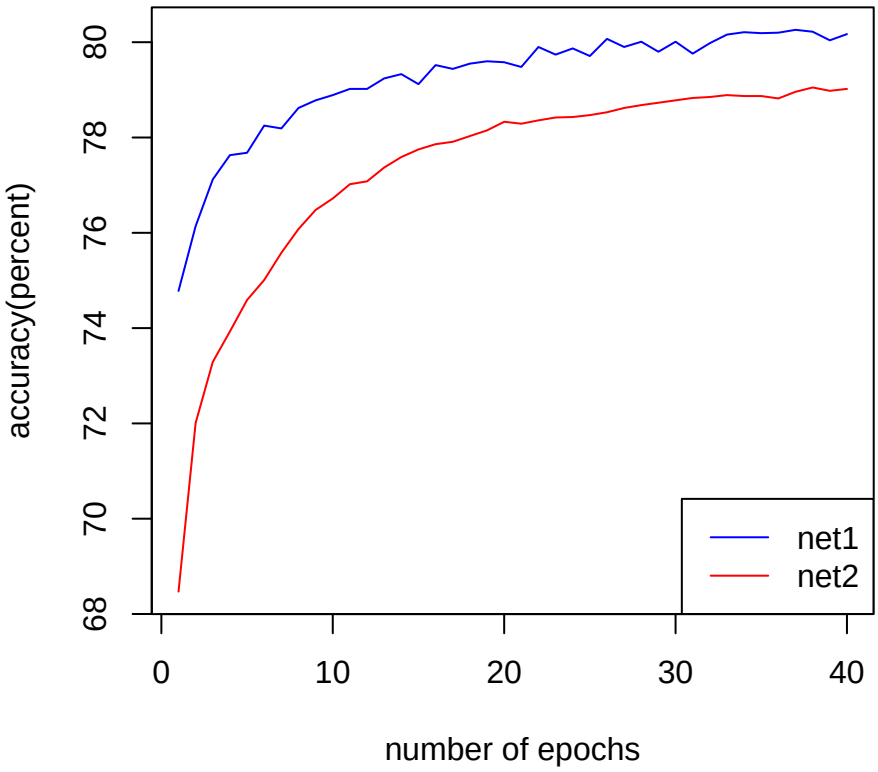

Figure 10: Valid. Acc. v.s. Num. of Epochs for NN with/without Dropout Layers

Net0 is the best model in the previous plot, and Net1 is the neural network with layer neural network with 256 units in the first layer and dropout layer, 512 units in the second layer and dropout layer, 256 units in the third layer and the output softmax layer. According to the plot, adding the dropout layer indeed improves the performance of the neural network.

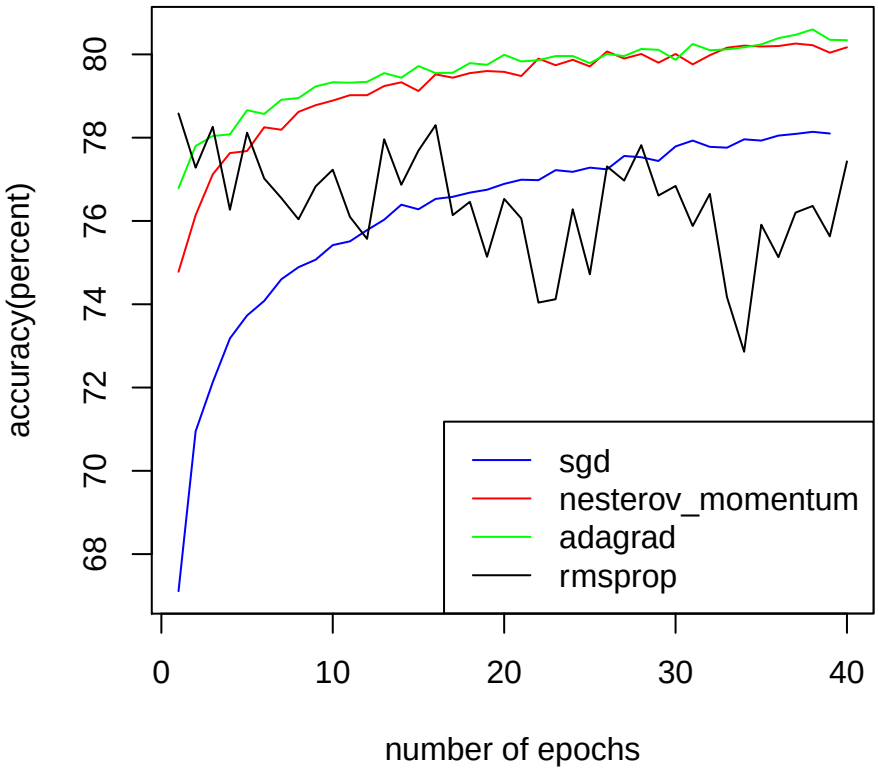

Figure 11: Valid. Acc. v.s. Num. of Epochs for NN With Different Gradient Descent Algorithm

From this plot, we can find that different gradient descent algorithm performs different. The rmsprop algorithm performs worst, both nesterov momentum and the adagrad performs best.

Support Vector Machine

The best **SVM** model with different kernals are shown in the **comparison** part. With adjusting all of the tuning parameters, we try to find those, which are the most suitable to the dataset.

Comparison

Using the remaining 20% of the total data, we implement best model of each method and calculate the test misclassification rate. All the results are in the following two tables.

Here in the table are the cross-validation results of best models using linear, Gaussian, and polynomial (quadratic) kernels which selected based on their error rate. A good numbers of parameters were tuned using 5-fold cross validation in order to reduce the random errors. Last column of the table is showing the test error rate using selected best models. Apparently, the misclassification rate of the test cases were relatively similar. Out of all three SVMs, Gaussian kernel in SVM seems to proform

Table 3: Test Misclassification Rate of Different Methods

Methods	Misclassification Rate (%)
kNN	23.3
Naïve Bayes	33.4
Random Forest	18.46
Extra Trees	29.6
Gradient Boosting	9.73
Deep learning	19.6

Table 4: Test Misclassification Rate for Support Vector Machine

SVM	Gamma	Cost	Misclassification Rate (%)
Linear	0.01	0.2	23.27
Gaussian	0.01	10	20.15
Polynomial (Quadratic)	0.1	0.3	22.22

the best, has a 0.20 error rate (with gamma = 0.01, cost = 10). However, we may not maximize the potential of SVM method 100% since we didn't train all the possible parameters. In addition, higher order polynomial kernals were not applied here which might also be a possiblity of increasing the prediction accuracy.

Discussion

In this project, the performance of different methods is heavily influenced by the selection of features and tuning parameters. For this particular dataset, with our selected tuning parameters we find that performance is ranked as **Gradient Boosting** > **Random Forest** > **Deep Learning** > **SVM(Gaussian)** > **SVM(Quadratic** > **SVM(Linear)** > **kNN** > **Extra Trees** > **Naïve Bayes**.

kNN has a smaller number of settings to adjust so we are able to find the optimal k. Due to the limitition of computation time, we only conduct the distance between each case using Euclidean distance. However, it is also interesting to try different distance functions and compare their performance in future work such as manhattan, canberra, and minkowski etc,.

For **SVM** algorithm,though **SVM** is far from being the panecea, it indeed provides intuitive representation of data partitioning and outlier detection. As for the given predictors, since we had no reason dropping any predictors, we took all the predictors into considerations.

For three of the tree based methods,the model is grown and prune under a set of tuning parameters such as **number of trees**, **depth of trees** and **number of predictors**. Each tree in the model is grown under the optimized version of parameters for this particular dataset. Generally, for **Gradient Boosting**, the many of the **tree number**, the better performance the model has.

For **Deep Learning** algorithm, since the meaning of features are unknown, the Bayesian Network method cannot be applied to the method. With the meaning of the features known, the Bayesian Network can be built, which may provide better predictions.

We enjoy this opportunity to learn about several machine learning algorithms and program with various Python modules and R packages.We wish we could dig more of this dataset and train our models to have better performancein the future.

Reference

1.http://electronicsnewsline.com/819/biological-neural-network-and-artificial-neural-network-a-comparison.html

2.CS 229 Machine Learning Course Materials, Stanford University. http://cs229.stanford.edu/materials.html

3.Google

4.http://en.wikipedia.org/wiki/Deep_learning