Machine Learning Final Project

Yi Wang, Lina Cao, Jiaheng Zhou

1. Introduction

In our studies, we are researching on the building of classifiers to identify the attributes that have more impacts on the annual income level, also by implementing the machine learning techniques to predict whether a person has income greater than $50K per year or not.

Before we start modeling, data preliminary work was conducted, with respect to data cleaning, data transformation. We removed the feature that represents the person's nationality at the very beginning, since 29,170 out of 32,561 data point (89.6%) is from the USA, which is too dominated and data from other categories may not be specific enough to stand out. Also, we used PCA techniques for dimension reduction, due to the tremendous amount of data, dimension reduction techniques would indeed largely boost the computational efficiency.

In the implement of two classifiers, our group used Support Vector Machine (SVM) and Neural Networks to model our data set. By using SVM, we firstly create our own simple SVM model, using gradient descent to update parameters. Hyper-parameters, step size ε and L2 regularization term λ were then introduced to minimize the absolute value of our parameter. As for Neural Networks, we mainly used the built-in functions in sklearn package for computation.

2. Data Mining

2.1 Data Structure

A specific dataset was used for this analysis; it was provided by University of California, Irvine.

The data source is http://www.census.gov/ftp/pub/DES/www/welcome.html. 14 features were provided to predict whether a person makes over $50K per year. The data was robust, and other

than a few specific missing variables was complete. The data was varied, ranging from categorical variables such as: relationship, race and sex to numerical variables age, education-num and hours-per-week.

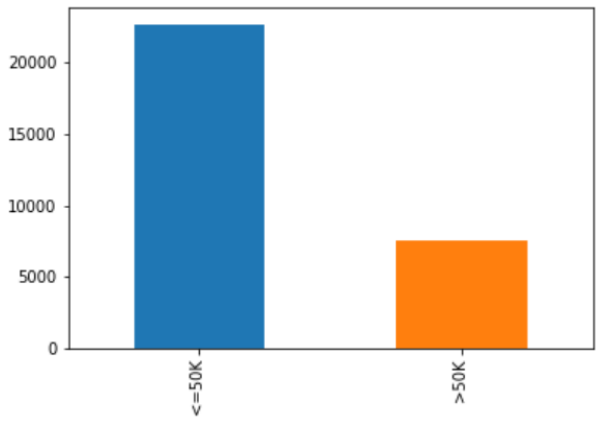

Figure 1. Labels

We used the classification label 0 for the target variable <$50K and the label 1 for the target variable >$50K. Also, from the graph we can see that the target variables are very imbalanced, and we believe it will have huge influence on our results.

2.2 Data Exploration

The adult dataset included 14 variables and represented more than 30000 observations. Before testing for the predictive power of specific variables we explored the dataset extensively to find portions that were missing and relevant patterns among the different variables. During data exploration, we learned that there are 3 major missing value variables, 'work class', 'occupation' and 'native-country'. To handle the missing value properly before model training, we decided to drop the all the missing value rows since there are less than 3% of missing value for each variable and we believe it will not significantly affect the result of our model result.

Besides that, we also dropped feature attribute native-country since over 90% of the native-country variable (see figure 2) is 'United States'. High portion of 'United States' value for native-country variable indicates that it has no predictive power for the models.

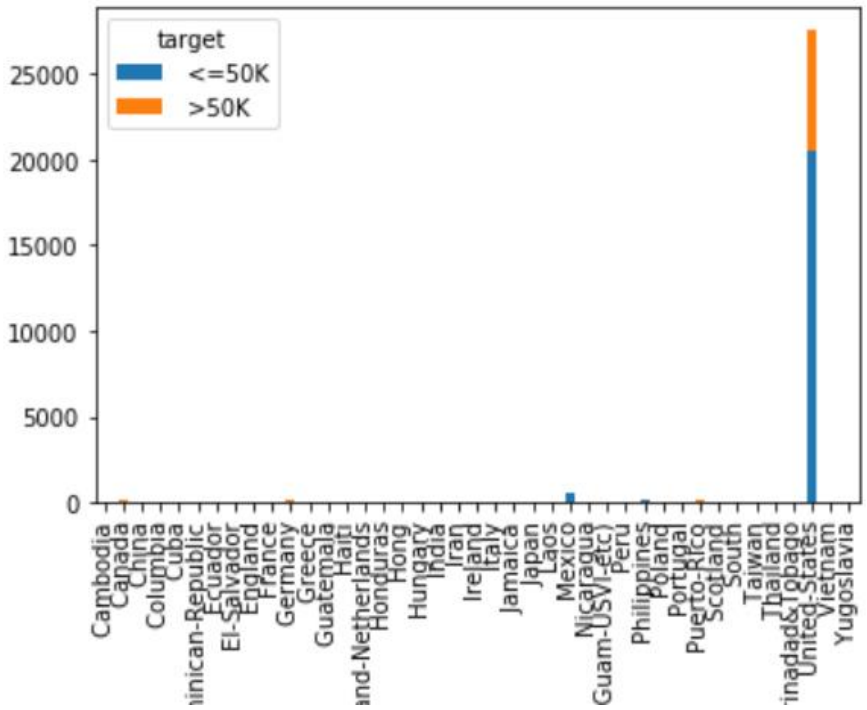

Figure 2

The figure below, indicates that those who have greater education-number ten to have higher chance of making more than 50k annually.

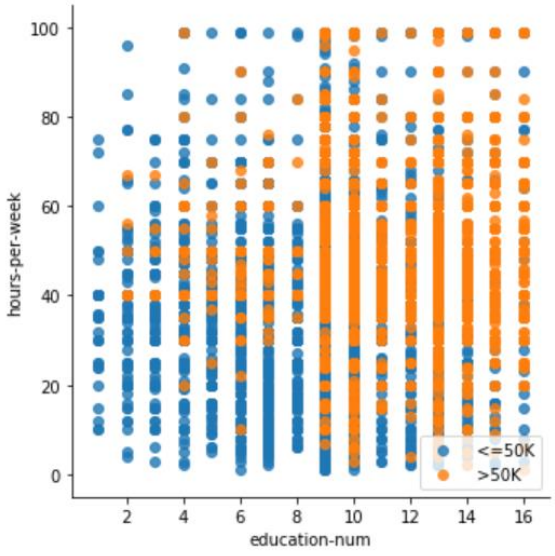

Figure 3

2.3 Data transformation

As mentioned before, majority of the attributes are categorical variables (string variables) which would not be interpreted by our chosen model. Both chosen models require numerical variables during training. To solve the issue, we turned all categorical data into dummy variables using get_dummies function in python. After data transformation, we have 63 variables in total and one target variable y in the train set.

3. Methods

3.1 Principal Components Analysis (PCA)

Due to the tremendous amount of data, dimension reduction techniques would indeed largely boost the computational efficiency. For our studies, we implemented Principal Components Analysis (PCA) techniques to normalize the data and reduce the dimensionality for increasing the computational speed in our machine learning algorithms.

Step 1: Get the data

In our studies, we have a 30162 rows * 63 columns train set matrix. It has 63 dimensions, so it would be very slow if there is no pre-processing measurement.

Step 2: Subtract the mean

For PCA to work properly, we have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension. This produces a data set whose mean is zero (the column averages are zero). Intuitively, we translate the origin to the center of gravity then we obtain the matrix B(N,n):

$$\mathrm{B} = [b_{i,j}] = [a_{i,j} - \overline{a_{*,j}}]$$

Step 3: Calculate the covariance matrix C

$$\mathrm{C(m,n)} = \frac{1}{N}[B^T(m,N) \times B(N,n)]$$

Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix

By using the eigenvector-eigenvalue calculation to get the eigenvectors and eigenvalue.

$$\Sigma \mathbf{v} = \lambda \mathbf{v}$$

where

$$\Sigma = \text{Covariance matrix}$$

$$\mathbf{v} = \text{Eigenvector}$$

$$\lambda = \text{Eigenvalue}$$

Step 5: Choosing components and forming a feature vector

Here is where the notion of data compression and reduced dimensionality comes into it. The eigenvector with the highest eigenvalue is the principle component of the data set. What needs to be done now is you need to form a *feature vector*, which is constructed by taking the eigenvectors that we want to keep from the list of eigenvectors and forming a matrix with these eigenvectors in the columns.

$$\text{FeatureVector} = (eig_1 eig_2 \ldots eig_n)$$

So, in order to decide which eigenvector(s) we want to drop for our lower-dimensional subspace, we have to take a look at the corresponding eigenvalues of the eigenvectors. Roughly speaking, the eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data, and those are the ones we want to drop.

The common approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top k eigenvectors. But in our studies, we keep all the features because of the significant proportion of categorical variables so it will lose a category if there is feature dropped.

Step 6: Deriving the new data set

This the final step in PCA and is also the easiest. Once we have chosen the components (eigenvectors) that we wish to keep in our data and formed a feature vector, we simply take the transpose of the vector and multiply it on the left of the original data set, transposed.

$$\text{FinalData} = \text{RowFeatureVector} * \text{RowDataAdjust}$$

The same as:

$$\mathbf{y} = \boldsymbol{W^T} \times \boldsymbol{x}$$

Basically, we have transformed our data so that is expressed in terms of the patterns among them, where the patterns are the lines that most closely describe the relationships between the data. This is helpful because we have now classified our data point as a combination of the contributions from each of those lines. Initially we had the simple and axes. This is fine, but the and values of each data point don't really tell us exactly how that point relates to the rest of the data. Now, the values of the data points tell us exactly where (i.e. above/below) the trend lines the data point sits. In the case of the transformation using *both* eigenvectors, we have simply altered the data so that it is in terms of those eigenvectors instead of the usual axes. The single-eigenvector decomposition could remove the contribution due to the smaller eigenvector and leave us with data that is only in terms of the other, but that's the general idea for PCA. not the case for our studies.

3.2 Support Vector Machine (SVM)

3.2.1Traditional implementation

As we learn from the lecture, we use gradient descent method to learn the model with Support Vector Machine (SVM).

Basic objective function

In order to find largest margin between boundary points (i.e the "support vectors" on both class sides. We firstly generate the formula from perceptron, that is to maximize $M = \frac{2}{\sqrt{w^T w}}$, which ends up with minimizing $\frac{1}{2}\sqrt{w^T w}$, subject to $\begin{cases} w^T x^i + b \geq 1, \ y^i = 1 \\ w^T x^i + b \leq -1, \ y^i = -1 \end{cases}$.

Note that the y_i label has the same sign with the class side, we can also simplify the condition we subject to as $y^i(w^T x^i + b) \geq 1$. And minimize $\frac{1}{2} w^T w$ will have the same result of minimize $\frac{1}{2}\sqrt{w^T w}$. Therefore, we have the basic objective function of SVM.

$$arg\min_w \frac{1}{2} w^T w\,, \qquad s.t.\ \ y^i(w^T x^i + b) \geq 1$$

"Loss" function

The error of a prediction should also be added into the objective function before minimizing. To do so, we calculate the prediction error as below:

$$diff_i = \begin{cases} 1 - \left(w^T x^i + b\right)\,, & y^i = 1 \\ \left(w^T x^i + b\right) - (-1), & y^i = -1 \end{cases}$$

The two situations when the prediction $f(x)$ is not consistent with y^i can also be rewritten as $diff_i = 1 - y^i * \left(w^T x^i + b\right)$.

Objective function

Thus, we have the objective function:

$$arg\min_w \frac{1}{2} w^T w + \sum_{i \in +1} \gamma^i \left(1 - \left(w^T x^i + b\right)\right) + \sum_{i \in -1} \gamma^i \left(1 + \left(w^T x^i + b\right)\right), \qquad (1)$$

This is also the formula that we used to derive the update rule for both w and γ .

Gradient descent

To start with a simple version, we first do the objective function with a single γ for each correction of the error, i.e. formula (2).

$$arg\min_w \frac{1}{2} w^T w + \sum_{i \in +1} \gamma \left(1 - \left(w^T x^i + b\right)\right) + \sum_{i \in -1} \gamma \left(1 + \left(w^T x^i + b\right)\right), \qquad (2)$$

When we integrate the prediction difference, the final formula turns to

$$arg\min_w \frac{1}{2} w^T w + \sum \gamma * \left(1 - y^i * \left(w^T x^i + b\right)\right), \qquad (3)$$

Update w

After taking partial derivative of (3) with respect to w,

$$\frac{\partial argmin}{\partial w} = \frac{1}{2} * 2w - \gamma * \sum_i y^i * x^i = w - \gamma * \sum_i y^i * x^i$$

We introduce the step size ε and have the updating rule for w :

$$w \leftarrow w - \varepsilon * \left(w - \gamma \sum_i y^i * x^i \right)$$

Update γ

After taking partial derivative of (1) with respect to γ,

$$\frac{\partial argmin}{\partial \gamma} = \sum_{i \in +1} \left(1 - \left(w^T x^i + b\right)\right) + \sum_{i \in -1} \left(1 + \left(w^T x^i + b\right)\right)$$

We introduce the step size ε and have the updating rule for w :

$$\gamma \leftarrow \gamma - \varepsilon * \left(\sum_{i \in +1} \left(1 - \left(w^T x^i + b\right)\right) + \sum_{i \in -1} \left(1 + \left(w^T x^i + b\right)\right) \right)$$

These steps happen after each round when we get a new w with which we have a minimum objective function in that iteration.

Quit condition

Now we have the result of each iteration after updating for the more optimal w and γ . Normally there are two thoughts that can be used when we decide that we have found the optimized w and quit at that time.

In our learning process, a) the value of the objective function (1) and b) the test sets are both considered to decide the time when we quit the iteration. In each iteration, a) will be calculated and we want it to decline after each iteration's update, while the correctly labeled data points in test set will be counted and used to further calculation of the percentage of the correct predictions and quit until we have a) no longer decrease and b) no longer increases.

3.2.2 Improvement implementation

Now we add more constraint to minimize the absolute value of each w. We chose L2 regularizer, $\frac{w^2}{\lambda}$, to perform as the "penalty" for values that have larger w. Our w update formula, are then shown as below, the last term $\frac{2w}{\lambda}$ is from the derivative of the L2 regularizer.

$$w \leftarrow w - \varepsilon * \left(w - \gamma \sum_i y^i * x^i \right) + \frac{2w}{\lambda}$$

Hyper-parameter analysis

In our model, the learning ratio ε and λ in L2 regularizer are the hyper parameters. We now focus on different value setting combinations and how they differ the performance of our model.

(Note: when we ran specific combination (e.g. $\varepsilon = 0.00001$ and $\lambda = 10$), the loop to update γ kept changing after 3,000 iterations. Our objective function did decrease all the time while the accuracy didn't change any more. For time saving purpose, we decided to quit the loop if we have already done 200 iterations).

The results of combination of hyper parameters are listed below.

λ \ ε	**0.000001 (10^-6)**	**0.00001**	**0.0001**	**0.001**	**0.01**
10	0.518083 56.04s	0.518083 56.95s	0.518083 0.54s	0.518083 0.53s	0.518083 0.54s
100	0.492264 0.56s	0.492465 0.55s	0.493068 0.7s	0.497087 0.53s	**0.520595** 0.54s
1,000	0.485333 0.58s	0.485333 0.57s	0.485232 0.64s	0.484931 0.59s	0.48473 0.54s
10,000	0.493068 3.46s	0.492867 2.71s	0.490958 2.13s	0.486337 0.84s	0.484629 0.51s

Figure 4

For the result in each cell, the first line is the accuracy using these two hyper parameters, followed by the next line showing the running time (in seconds). From the result, when $\varepsilon = 0.01$ and $\lambda = 100$, learned a better result for SVM classifier. In this case, we have a highest accuracy in our own test set is, which is 0.5206.

Test set review

Now we have the "optimized" w from the case above. Let's test it using the given test set in the original data. The result is 0.5336 in the test set. Our sensitivity is 0.4381 and specificity is 0.5631.

From the confusion matrix below, the accuracy is 0.5336, which means slightly over half of the data is correctly labeled using the model we build by SVM. At the meantime, the true negatives (TP) are 2161, in which case they are actually have greater than 50k income while we predicted they don't; while the false positives (FP) contain 5433 data points, meaning that we misclassified those who actually don't have greater than 50k income as they had.

The true positives are much less than false negatives, therefore it seems that the SVM performs better on identify people who have less than 50k income.

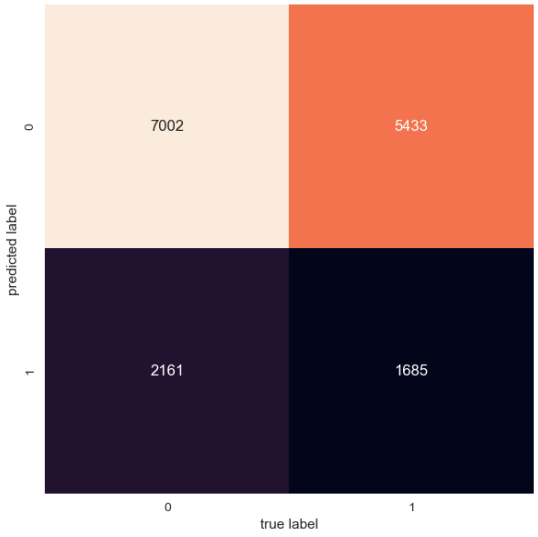

Figure 5

3.3 Neural Network

As a super powerful tool in the deep learning field, Neural network models (Supervised) has been implemented in our research by utilizing the Multi-layer Perceptron (MLP) algorithm that trains using Backpropagation in python, sklearn package.

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f(.) : $R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X $= x_1, x_2, \ldots, x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 5 shows a one hidden layer MLP with scalar output.

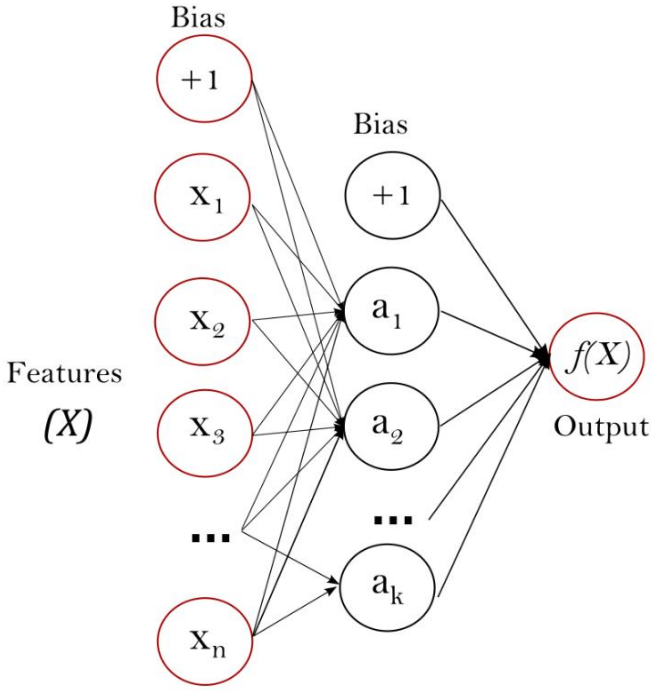

Figure 6

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, \ldots, x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$, followed by a non-linear activation function g(.) : R $\rightarrow$ R $-$ like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

The Multi-layer Perceptron classifier contains tons of different parameter settings, for the sake of simplicity, we have only chosen three settings, more specifically, we chose stochastic gradient

descent as the solver, 0.00001 as L2 penalty parameter and logistic sigmoid function as activation. Expect these specific settings, we let the rest be default settings.

The results showed an accuracy of 0.7723 and the corresponding confusion matrix as following:

	Precision	Recall	F1-score	Support
0	0.77	1.00	0.87	12435
1	0.92	0.04	0.08	3846
Avg / Total	0.81	0.77	0.68	16281

Figure 7

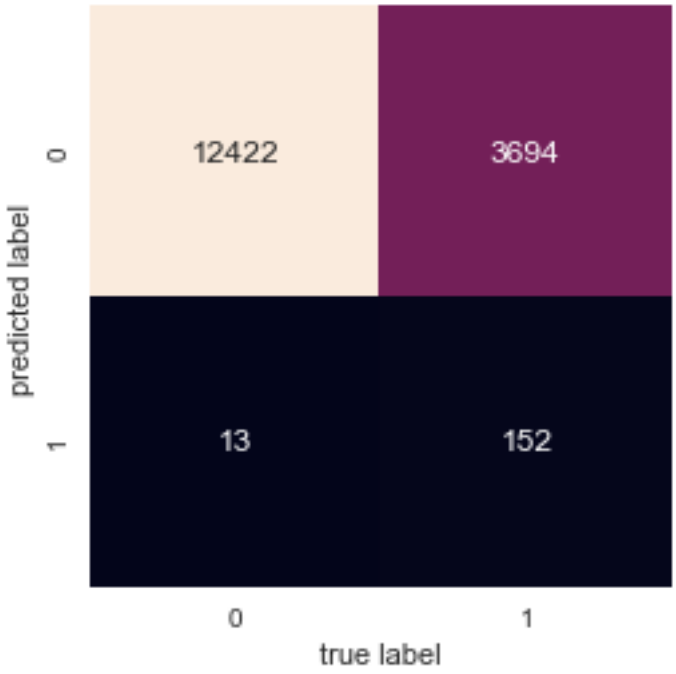

Figure 8

In terms of the relative importance among these 63 features, we can have figure to illustrate this issue as following:

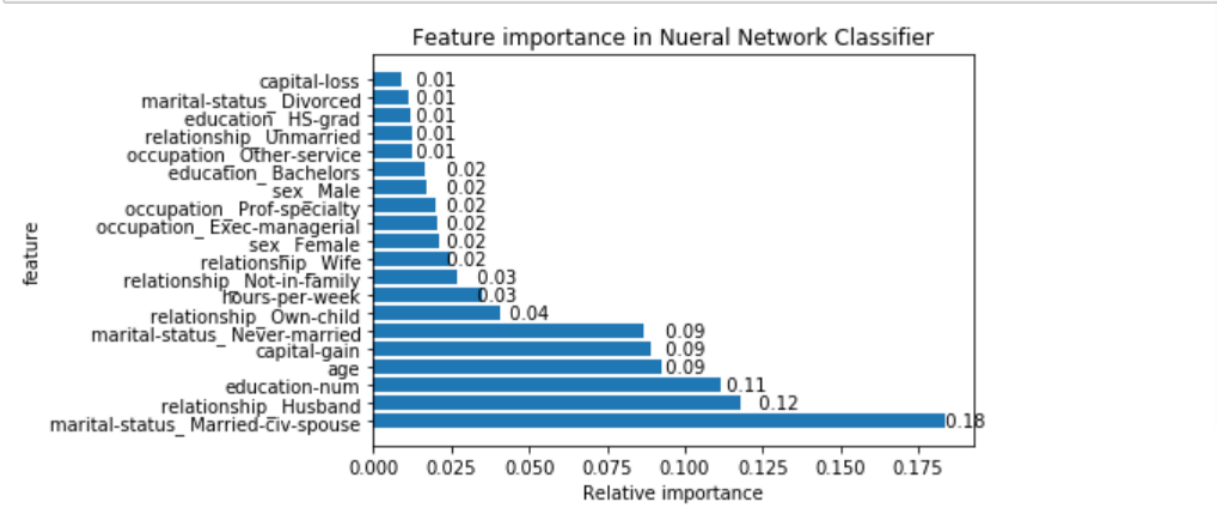

Figure 9

As we can see, the marital status of married-civ-spouse (18%) is the most essential factor, then for the numbers of education (11%) and husband relationship (12%) certainly play a vital role in as well.

4. Conclusion

In terms of SVM implement, the test set accuracy seems good when we use SVM classifier. But there are still some concerns from our SVM classifier.

- We only use a single gamma for all data points, which is not a good idea to separate the influential data points from normal ones.
- In some cases, when epsilon ε is 0.001 or bigger, gamma seems not change much and will end up with closing to the initial guess. More investigation is needed to deal with a better improvement in gamma.
- We take accuracy as one of the criteria to quit our learning process. Different parameter settings will actually change the final result w, as well as the corresponding accuracy. We may not quit at an absolute highest accuracy point.

Consequently, the learning result will not be as optimal as we want. Also, the accuracy doesn't increase much when we increase ε and λ, which may indicate more optimization method can be used in our further study.

As for Neural Networks, it works well given we implement the built-in function in sklearn package, we got a high accuracy for the test set prediction. But still, the calibration of parameters could be taken into consideration for the purpose of optimization. For current stage, we used default setting for the number of hidden layers which take a key role in the neural network algorithms, not only for boost the computational efficiency but also for the improvement on the prediction accuracy. For this issue, we should use more advanced and complex measurements to handle with in later research. Overall, the performance is quite good for the illustration of our research.

To summarized, this project enabled us to familiarize ourselves with data cleaning, model building, and troubleshooting and was extremely valuable. We gained an appreciation for understanding of the underlying data. Our experience with this project has shown us that the context behind the data is as important as the model one builds to interpret the data. We may not be able to build up the best model, but we experienced the procedures on the using machine learning techniques to tackle the issues. More techniques, like feature selection, ensemble techniques, other Machine Learning Algorithms (Random Forest, KNN, Decision Tree, etc.) can also be implemented into this research in the future works.

5. References

[1] Lindsay I Smith, A tutorial on Principal Components Analysis

[2] Wei-Chun Kao and Kai-Min Chung and Lucas Assun and Chih-Jen Lin. Decomposition Methods for Linear Support Vector Machines

[3] David R. Musicant and Alexander Feinberg. Active Set Support Vector Regression

[4] David R. Musicant. Data Mining via Mathematical Programming and Machine Learning

[5] Andrew Ng, Stanford University, CS229 Lecture Notes

[6] Jason Weston, Columbia University, Support Vector Machine and Statistical Learning Theory Tutorial.