Hand-Written Character Recognition Using Multi-Class Support Vector Machine (SVM)

Pawan S. Sawant

University of Colorado
Boulder

MCEN 5125: Optimal Design
Prof. Shalom D. Ruben

May 7, 2024

Contents

List of Figures

List of Tables

1 Introduction

In the field of machine learning, identifying patterns and optimizing them is crucial, especially in applications like analyzing handwritten documents and sorting mail automatically. One of the key tools for character recognition in these areas is the MNIST dataset. MNIST stands for Modified National Institute of Standards and Technology. It's a comprehensive database that contains images of handwritten digits, ranging from 0 to 9. This dataset is extensively used because it helps computers learn how to recognize and interpret human handwriting. By using the MNIST dataset, researchers and developers can train machine learning models to accurately identify handwritten numbers. This capability is particularly useful in real-world applications like reading postal codes on envelopes for mail sorting or digitizing written texts. The MNIST dataset is highly valued in the tech community for its reliability and its role in advancing machine learning technologies related to character recognition.
In this project, our primary goal is to develop a character recognition system for handwritten digits using Support Vector Machines (SVM). SVM is a powerful machine learning method used for classification and regression. In character recognition, SVM helps by distinguishing between different digits based on their features extracted from images. The idea is to find the best dividing boundary (hyperplane) that separates digits into their respective categories with maximum margin. This method effectively handles the complexities of handwriting variations and improves the accuracy of digit recognition. By using SVM, we aim to create a robust system capable of accurately identifying handwritten numbers, enhancing the efficiency of processes that rely on digit recognition.

2 Background: Classification

2.1 Binary classification

There are many problems in the real world that we need to classify or categorize into different groups or classes depending on their features. Classification is a data fitting with an outcome that takes on typically non-numerical values such as True or False, Spam or Not-spam, Elephant or Giraffe. This output of classification is called as labels or Categories. This type of classification is called binary classification or Boolean classification. Due to the fact that classifiers are dependent on the numerical data, we need to represent the outcomes(i.e. True or False, Elephant or Giraffe) in terms of numerical values. In binary classification, this outcome is encoded as +1 for one class and -1 for another class.

2.2 Multi-class Classification

When the given problem has more than 2 categories or labels, it comes under Multi-class classification. In the least square multi-class classifiers, we create a least square classifier for each label or category versus all other categories. To explain this more intuitively, let us consider an example we are trying to predict an Apple, Banana or Peach. Therefore, we will have to create a classifier of an apple versus a banana or a peach, the second classifier would be a banana versus an apple or a peach and similarly a classifier for a peach versus the rest. In this project, we will be using this least square multi-class classification technique, where we will be finding a classifier for each digit versus all other digits.

2.3 Support Vector Machine(SVM)

Support Vector Machine is a machine learning algorithm which is used for classification and regression problems. An SVM divides data along a decision boundary (plane) established by a limited subset of the data (feature vectors). The data subset that supports the decision boundary is known as the support vector and the optimal choice boundary is known as a hyperplane. In this algorithm we will be finding the classifiers for each digit vs each digit like explained in the multi-class classification.

3 Methodology:

3.1 Training the Model:

3.1.1 Given Data:

The MNIST is a collection of handwritten digits from 0 to 9, each of size 28x28 pixels, these pixel values are populated in a single row of 784 columns. The data comprises 60000 training Images which also have corresponding labels that help in the development and comparison of the recognition models. Therefore, we will name the known variable for training as X which is of the size 60000x784.

3.1.2 Preprossing the Training Data

- Trimming the data: In the given data set, each image contains zero-valued pixels in the border of each image. These values do not provide essential information for the classification of the digits, therefore these pixel values may cause unnecessary noise and can increase the processing time. To solve this problem, we will be preprocessing the data by eliminating the border pixel values. The data trimming can be done for each 60000 number and the size of each number will change to 26x26. Therefore, the known variable X will have dimension 60000x676.

- Data Normalization: The original pixel value in the given dataset is represented by 8-bit unsigned integers, these values range from 0 to 255. To maintain consistency in the data, we will be Normalizing the pixel values by scaling the values that range between 0 and 1. Normalizing the data will ensure that all the features are on a similar scale which in turn results in speeding the training model.

- Sorting the training data: We need to sort the data based on the labels given in ascending order(or descending order) so that we can form the systems of equations easily. We can sort the given pixel data by merging the labels and their corresponding image pixel data and then we can sort them in ascending order based on the labels.

3.1.3 Deriving the Cost Function and Formulating into Quadratic Programming

As explained in the 2.2: Multi-class Classification, we will be using classifiers to differentiate between the digits. Instead of using 1 vs All, we will be generating 1 vs. 1 classifiers for each digit using a Directed Acyclic Graph(DAG) which we will discuss in a later section. For the two digits , we can differentiate them using just one hyperplane that can be simply defined by a equation. $y = mx + b$.

$$y = mx + b \tag{1}$$

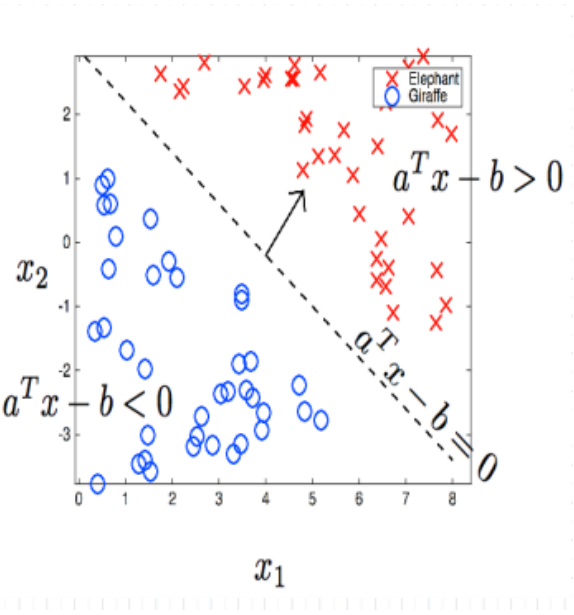


Figure 1: Hyperplane Seperating two features

Figure 1 shows a two-dimensional case, where a hyperplane (black dotted line) separates the two features. In this way, we can generate hyperplanes for all the classifiers.

In order to define a higher order, the equation of hyper plane will have vector, and the equation will be given as follows:

$$\begin{aligned} a^T x - b &= 0 \\ \text{where,} \quad a^T &= [a_1, a_2, \cdots, a_n] \\ x &= [x_1, x_2, \cdots, x_n] \end{aligned} \tag{2}$$

Since, we will be using SVM algorithm, which tries to find the hyperplanes that not only seperates the two classes but also keeps a maximum margin m. This margin is the distance from the hyperplanes to the data point of the class. To find the optimal hyperplanes, let x_1 and x_2 be the two points on the boundries of the margin on the opposite side of the hyperplanes. Therefore, the equation of the hyperplanes is given below, also we Figure 2 shows two hyperplanes at a optimal margin.

$$a^T x_1 - b = 1 \quad \text{and} \quad a^T x_2 - b = -1 \tag{3}$$

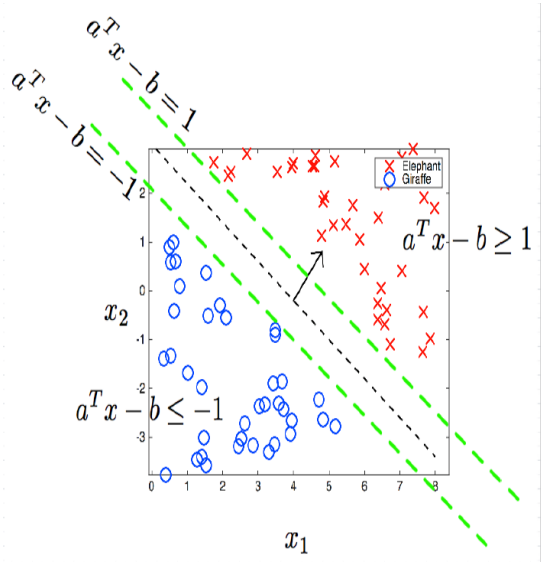


Figure 2: Two Hyperplane Seperated by a Margin

The margin m in between the two hyperplane is given as follows:

$$m = \frac{2}{||a||_2} \tag{4}$$

In practice, the best classifier may not be the one whose margin is set to a constant value. The data-to-hyperplane distance criterion is either loosened or tightened using a weighting function based on slack variables to account for the inherent unpredictability in each binary classifier. The weighting function, denoted by the variable gamma, is like a tuning knob on the margin; it's what decides the values of the hyperplane coefficients.

In practice, there wont be a best classifier whose margin is a constant value. We can alter the distance between the data and the hyperplanes using a weighting function denoted by γ. We can use this γ as a tuning knob to adjust the values of the hyperplane coefficient A and b.

In this project, we will be iterating different values of γ to find the optimal solution.

$$\gamma = [0.00001, 0.001, 0.01, 1, 100, 1000, 10000, 100000]$$

Therefore the cost function can be given as follows

$$\begin{aligned} \text{minimize} \quad & a^T a + \gamma(1^\top u + 1^\top v) \\ \text{subject to} \quad & a^T x - b \geq 1 - u, \\ & a^T x - b \leq -(1 - v), \\ & u \geq 0, \\ & v \geq 0. \end{aligned} \tag{5}$$

Now that we have have our cost function and subject to, we need to formulate this in to quadratic programming form as required in MATLAB. According to MATLAB, for quadratic programming the required form is as follows:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T Q x + q^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{6}$$

Therefore, the given form can be written as follows

$$\text{Solution} = \underbrace{\begin{bmatrix} a \\ b \\ u \\ v \end{bmatrix}^\top}_{x} \underbrace{\begin{bmatrix} I/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{Q} \begin{bmatrix} a \\ b \\ u \\ v \end{bmatrix} + \gamma \underbrace{\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}}_{q^T} \begin{bmatrix} a \\ b \\ u \\ v \end{bmatrix}^\top \tag{7}$$

The x vector has hyperplane coefficient A and b which we can use to test the accuracy of the model.

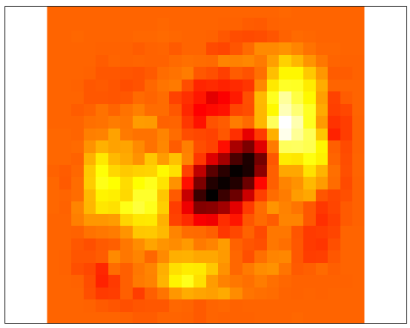
3.1.4 Feature Engineering and Parallel Computing

In MATLAB, parallel computing is the technique for carrying out many calculations at once by allocating tasks among several processors, cores, or computers. Programs that need a lot of data handling or are computationally demanding may run much more quickly using this method. Among many features provided by the Parallel Programming Toolbox in MATLAB, we will be using parallel-for-loops(parfor) in this project. This function will run the highly computationally required loops to be run on multiple cores to get the results as fast as possible. We will discuss the time difference in solving the optimization problem in the results section.

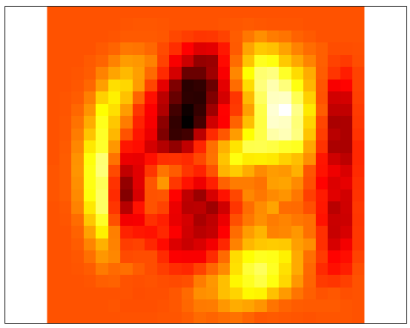
In the relm of image recoginition, feature engineering is the procedure of choosing, obtaining and modifying the features from the preprocessed picture data to improve the machine learning model's performance. In this project, we will be two types of feature engineering on the images data. First will be trimming the edges of the by significant amount so that only digits data can be used to train the model. This will eliminate the bunch of zeros in the data and can potentially reduce the computation time. Then we will be sharpening the images, in this method each pixel is assigned a value of 0 or 1 based on a threshold value by the process of iteration. Using this method, we will be able to eliminate less relevent information while keeping and enhansing the key characteristics of the digit. To show the effect of image sharpining, we can see the difference in the image quality in the Figure **??**below. The comparison of the results will be shown in the result section.

3.1.5 Displaying the classifiers

Now that we have hyperplanes for each digit with respect to each digits, we can plot the values of each pixel that the model will be using to detect the digit. This plot/image will represent the sensitivity map of each classifier to the pixel value of the predicting digit. following images show the heat map of the classifiers. Since we have 45 classifiers, we won't be including all the images for each and every classifier.



(a) Classifier values for digit 0 vs. 4



(b) Classifier values for digit 2 vs. 9

Figure 3: Classifier values(coefficients) that distinguish the digits

3.2 Testing the model

3.2.1 Given Testing Data

Similar to the Training Dataset, the MNIST also has a collection of handwritten digits from 0 to 9, each of size 28x28 pixels, these pixel values are populated in a single row of 784 columns. The data also comprises 10000 Testing Images which also have corresponding labels that will be used for the testing and verifying the accuracy of the model. We will name the known variable for testing as $\hat{X}$ which is of the size 10000x784.

3.2.2 Preprocessing the Testing Data

While training the model, we had done various preprocessing of the given data, Similarly, we will be preprocessing the given training data. First, we will trim the given 28x28 pixel data into 26x26 pixel data. Then will be Normalizing the pixel values by scaling the values that range between 0 and 1.

3.2.3 Results using Directed Acyclic Graph(DAG) and Hyperplanes

The classifier's output can be determined using DAG. Since we have 10 digits, the algorithm perform 10 comparision to find the resultant digit using process of elimination. A simple DAG can shown in the Figure 4. In this method,we builds a hierarchy of classifiers in which each decision point distinguishes between two possible classes rather than handling the issue at once. By use of successive binary tests, this approach effectively reduces the range of potential digit classifications, facilitating a faster convergence to the right class. The DAG method is thus a sensible option for

digit recognition jobs since it reduces the total computing load and complexity when compared to a direct multi-class classification.

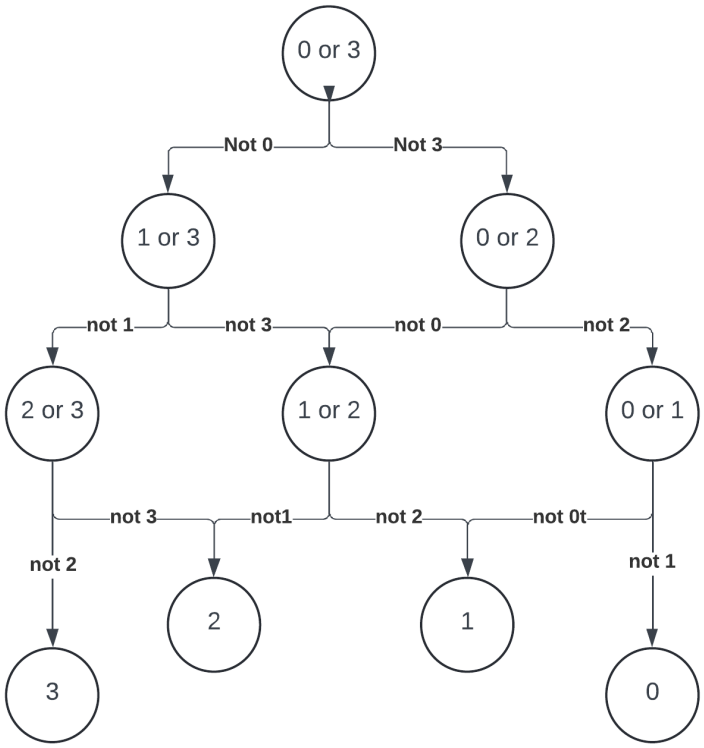


Figure 4: Simplified DAG method for digits from 0 to 3

We will be using this method to find the predicted number(labels) and then we can you this predicted labels to find the accuracy of the model.

4 Results

4.1 Understanding Prediction errors

[1] Let us we are given a set of data $x_1, x_2, ..., x_N$ and their corresponding predictions are written as $\hat{x_1}, \hat{x_2}, ..., \hat{x_N}$. There are four possible outcomes of the predictions:

- True Positive : $x_N = +1$ and $\hat{x_N} = +1$
- True Negative : $x_N = -1$ and $\hat{x_N} = -1$
- False Positive : $x_N = -1$ and $\hat{x_N} = +1$
- False Negative : $x_N = +1$ and $\hat{x_N} = -1$

So, in the first two cases, the predicted error is correct, i.e. the model predicted the digit correctly. Whereas in the last two cases, the prediction is incorrect, i.e. the model did not predict the digit correctly. These predictions can be easily tabulated into a matrix that is known as a confusion matrix.

Table 1: Understanding the Confusion matrix.

4.1.1 Finding the Error Rates

	Prediction		
Outcome	$\hat{x_N} = +1$	$\hat{x_N} = -1$	Total
$x_N = +1$	N_{tp}	N_{fn}	N_p
$x_N = -1$	N_{fp}	N_{tn}	N_n
All	N_{tp}+N_{fp}	N_{fn}+N_{tn}	N

Where N_{tp} and N_{tn} are the true positive and true negative predictions respectively. Whereas, N_{fn} and N_{fp} are false negative and false positive predictions respectively. Therefore, the off-diagonal elements of the confusion matrix are the errors in the predictions and the diagonal elements are the correct predictions.

4.2 Finding the Error Rates

[1] The performance of the model is expressed in terms of confusion matrix and error rates. Various rates are given below:

- error rate = $N_{fp} + N_{fn}/N$

- true positive rate = N_{tp}/N_p
- false positive rate = N_{fp}/N_n

4.3 Training Results

4.3.1 Accuracy for different γ value

We iterated various γ values to find the optimal solution, and we found out the the best results we achieved in the training was with the maximum γ. Following table shows the accuracy of the model in the training set data.

γ	True Positive Rate (%)
0.00001	18.38
0.001	92.31
0.01	94.56
1	96.98
100	98.32
1000	98.41
10000	98.43
1e5	98.44

Table 2: Accuracy rates for different values of γ

The figure below shows the variation of the positive rate vs γ.

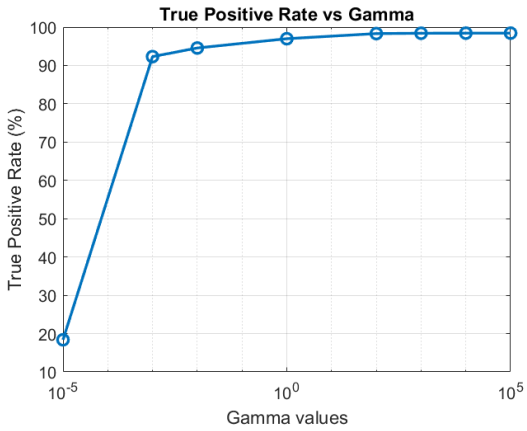


Figure 5: Variation of positive rate with respect to γ

4.3.2 Confusion Matrix for Training Dataset

We can easily plot the table of confusion matrix based on the results we got. In training data set we got maximum accuracy at $\gamma = 1e5$ Therefore confusion matrix for the optimal solution in training set in shown below.

Table 3: Confusion matrix for the training set for $\gamma = 1e5$.

	Prediction										
Digit	0	1	2	3	4	5	6	7	8	9	Total
0	5923	0	0	0	0	0	0	0	0	0	5923
1	0	6742	0	0	0	0	0	0	0	0	6742
2	0	0	5859	54	9	4	4	4	24	0	6742
3	0	0	46	5940	11	79	2	12	41	0	6131
4	0	0	0	0	5788	0	0	3	3	48	5842
5	0	0	0	95	3	5276	6	2	39	0	5421
6	0	0	0	0	0	0	5918	0	0	0	5918
7	0	0	0	0	0	0	0	6188	6	71	6265
8	0	0	20	58	8	83	15	7	5660	0	5851
9	0	0	0	0	59	2	0	110	6	5772	5949
All	5912	6785	5865	6052	5970	5567	5999	6327	5732	5791	60000

4.3.3 Error Predictions

From the predictions shown in the confusion matrix, we can easily find the error rates in the training dataset.

- error rate $= N_{fp} + N_{fn}/N$ .Therefore, the error rate is 1.56%
- true positive rate for digit $9 = N_{tp}/N_p$. Therefore, the true positive rate is 99.67%
- false positive rate for digit $0 = N_{fp}/N_n$. Therefore, the false positive rate is 2.05%

4.4 Testing Results

4.4.1 Accuracy for different γ value

Similarly in like we did in training's result, we iterated various γ values to find the optimal solution in the testing data, and we found out the the best results we achieved in the training was with the maximum γ. Following table shows the accuracy of the model in the training set data.

γ	True Positive Rate (%)
0.00001	18.78
0.001	92.44
0.01	94.13
1	92.10
100	87.73
1000	87.35
10000	87.13
1e5	87.17

Table 4: Accuracy rates for different values of γ without feature engineering

γ	True Positive Rate (%)
0.00001	15.31
0.001	93.01
0.01	94.45
1	93.33
100	92.39
1000	92.31
10000	92.34
1e5	92.35

Table 5: Accuracy rates for different values of γ with feature engineering

The figure below shows the variation of the positive rate vs γ.

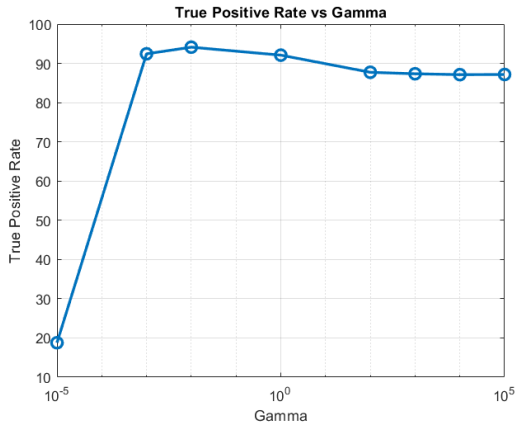


Figure 6: Variation of positive rate with respect to γ without feature engineering

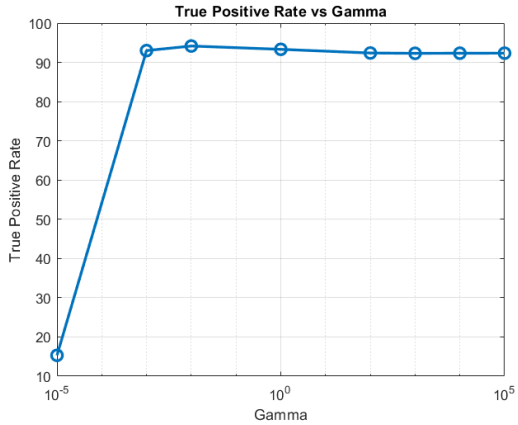


Figure 7: Variation of positive rate with respect to γ with feature engineering

4.4.2 Confusion matrix for the testing set

Based on the hyperplanes we have found for each digit, we can find the resultant predictions for the testing dataset. This resultant prediction can be populated into a confusion matrix is shown below:

Table 6: Confusion matrix for the Testing set with feature engineering.

	Prediction										
Digit	0	1	2	3	4	5	6	7	8	9	Total
0	962	0	2	2	1	5	3	1	4	0	980
1	0	1118	1	4	0	2	2	1	7	0	1135
2	6	1	962	11	11	6	9	10	13	3	1032
3	0	1	10	943	1	21	0	13	18	3	1010
4	1	0	4	2	931	2	8	2	2	30	982
5	9	1	4	39	5	797	14	3	18	2	892
6	7	2	8	2	4	6	927	0	2	0	958
7	0	5	23	6	9	3	0	958	3	21	1028
8	4	4	4	19	6	27	10	4	894	2	974
9	4	4	1	6	26	9	0	24	10	925	1009
All	993	1136	1019	1034	994	878	973	1016	971	986	10000

4.4.3 Error Predictions

From the predictions shown in the confusion matrix, we can easily find the error rates in the testing dataset.

- error rate = $N_{fp} + N_{fn}/N$ .Therefore, the error rate is 5.83%
- true positive rate for digit 9 = N_{tp}/N_p. Therefore, the true positive rate is 93.81%
- false positive rate for digit 9 = N_{fp}/N_n. Therefore, the false positive rate is 6.19%

5 Conclusion

In this project, we did a thorough analysis of hand-written character recognition using Multi-Class Support Vector Machine (SVM) on the MNIST dataset. In this project we iterated various values of γ and found accuracy for each value. It was evident that if the γ is extremely less, the accuracy would drastically decrease, the variation can be seen in the Figures 5 and 7. Although the feature engineering that we implemented did not improve the accuracy by much, but using parallel computing reduced the time required to run the training model drastically. Before using the parallel computing, the time required to run the training model was about 6.5hrs for the all the eight γ, but using the parallel computing(11 core, may change from device to device) this time reduced to about 40 minutes. In conclusion,the model achieves a high accuracy using the quadratic programming as compaired to the least squares accuracy. In future improvements we could use various feature engineering techniques such as kernal method to improve the accuracy by more percentage.

6 References

[1] Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares. (n.d.). https://web.stanford.edu/ boyd/vmls/

[2] Class Notes and slides.