Classification of MNIST Handwritten Digits Dataset using SVM

Jayesh Mahapatra
Veer Surendra Sai University of Technology
jayeshmahapatra@gmail.com

Harshit Mahapatra
College of Engineering and Technology,Bhubaneswar
harshitmahapatra95@gmail.com

Abstract

Optical character recognition is an important application of machine learning where an algorithm is trained on a labeled dataset so that it learns to classify unlabeled letters/digits. A variety of algorithms have performed very well for the problem of classification of handwritten digits. In this project we have trained a Support Vector Machine using a Radial Basis Function Kernel to classify the digits in the dataset. The parameter 'C' of the kernel was selected using model selection procedure.

This project report is divided into six sections: follows:

- In Section 1 we give a brief introduction to the problem of Optical Character Recognition and Handwritten Digit Recognition.
- In Section 2 we discuss the theory behind Support Vector Machine and RBF Kernel.
- In Section 3 we give a brief summary about the dataset used in the project.
- In Section 4 we describe our training and model selection procedure for the classifier.
- In Section 5 we draw conclusions from the observations obtained during the course of the project .
- In Section 6 we present the code used in our project.

I. Introduction

Optical Character Recognition is a technology that allows us to recognize characters through an optical mechanism. For humans,the eyes act as the optical mechanism i.e. the images seen by eyes are inputs to the brain. OCR is a technology that functions like the human reading ability.

Handwritten Digit Recognition is an important application of the OCR technology. One of the most relevant ideas in the field was put forward by Y. Le Cun in the paper titled "Handwritten digit recognition: applications of neural network chips and automatic learning "[3], in the paper he described methods to perform handwritten digit recongnition. It was followed by numerous papers studying the efficiency of neural networks in classifying handwritten digits. However there was a growing interest in the field regarding the performance of non-neural network classifiers on the task. One such study comparing the performance of various classifiers such as K-Nearest Neighbour, Radial Basis Function and Neural Networks was put forward by Y. Le Cun in 1991.[4].This paper was followed by many similar comparision studies such as the paper by LeCun, Yann and Jackel.[1]

It was noticed that Radial Basis Function Network performed fairly well in classifying the digits with fairly low test set errors (3-4%).The objective of this project is to study the perfomance of RBF in classification of MNIST digits and to achieve a similar level of accuracy.

II. Support Vector Machines and Radial Basis Function

Support Vector Machines were developed by Cortes and Vapnik in 1995.[2] The SVM conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the learning machine.

More roughly the SVM approach was sketched by David Meyer[5] as follows :

1. Class Separation: The objective of SVM is to look for optimal separating hyperplane between two classes by maximizing the margin between the classes' closest points.
2. Overlapping Classes: Points on the "wrong" side of the discriminant margin are weighted down to reduce their influence.
3. Nonlinearity: When linear separation is not possible, the data points are projected into a higher dimensional plane where they are linearly separable. This projection is realized using *kernel techniques.*
4. Problem Solution: The whole process can be represented as a quadratic optimization problem solvable by know techniques. Mathematically the optimization objective of a SVM with "soft margin" can be represented as follows:

$$\begin{aligned} &\underset{w\in R^d,\xi_i\in R^+}{\text{minimize}} && \|w\|^2 + C\sum_i^N \xi_i \\ &\text{subject to} && y_i(w^T x_i + b) \geq 1 - \xi_i \\ &\text{for i=1..N} \end{aligned} \tag{1}$$

Where "w" represents the weight vector. Here every constraint can be satisfied if ξ_i, the "slack" variable is sufficiently large. The parameter "C" is called the regularization parameter which affects the "width" of the margin.

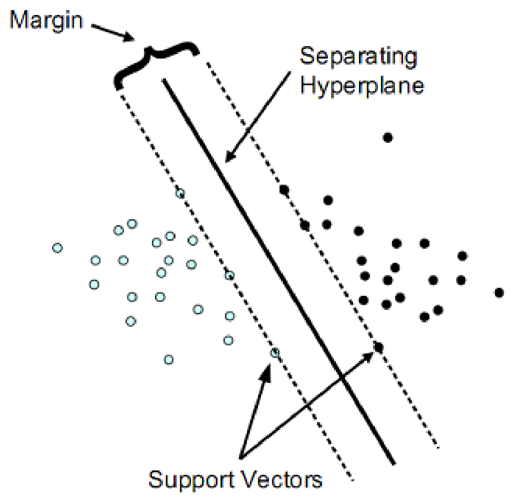


Figure 1: Classification by SVM in linearly separable case.

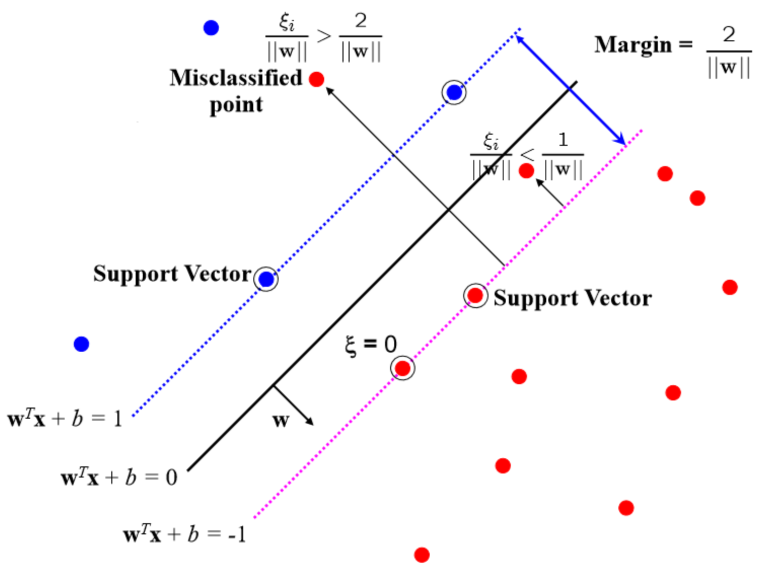

$\frac{\xi_i}{||\mathbf{w}||} > \frac{2}{||\mathbf{w}||}$
Misclassified point
Margin = $\frac{2}{||\mathbf{w}||}$
$\frac{\xi_i}{||\mathbf{w}||} < \frac{1}{||\mathbf{w}||}$
Support Vector
Support Vector
ξ = 0
w
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$

Figure 2: The varitation of points with respect to weights and "slack" variable.

Radial Basis Function: RBF is based on the belief that a classification problem cast into a higher dimensional space becomes more likely separable than in a lower dimensional space. Here the learning is equivalent to ïňĄnding a surface in high dimensional space that provides the best ïňĄt to training data. The RBF functions map the samples from the input space to hidden space, in which the same samples are supposedly separable. Generally the Gaussian RBF function is used in SVM. Mathematically RBF kernel on two samples u and v in some input is defined as:

$$\Phi(\gamma, u, v) = exp(-\gamma|u - v|^2) \tag{2}$$

The value of the RBF kernel decreases with distance and ranges between 0 (in the limit) and 1 (when $u = v$), it has a ready interpretation as a similarity measure. When we implement SVM using RBF kernel the we calculate new features f from the existing features. We replace a given observation $x^{(i)}$ with calculated feature vector $f^{(i)}$ which is defined as follows:

$$x^{(i)} \Longrightarrow f^{(i)} = \{\Phi(\gamma, x^{(i)}, x^{(1)}), \Phi(\gamma, x^{(i)}, x^{(2)}), \\ ..., \Phi(\gamma, x^{(i)}, x^{(m)})\} \tag{3}$$

Here "m" is the total number of observations in the dataset. We then feed the new dataset consisting of our calculated feature vectors to SVM for classification.

III. MNIST Dataset

In this project we have used a subset of the MNIST database of handwritten digits[10]. We obtained a CSV format of the dataset Y LeCun's from Joseph Chet Redmon's website[8]. The datset consists of 60000 training examples and 10000 test examples.Each image has dimensions 28x28 and represents a digit between 0 to 9.The digits have been size-normalized and centered in a fixed-size image. It has been made sure that the set of authors of the training set digits and that of the test set digits were disjoint. Figure 3: shows a sample of the dataset.

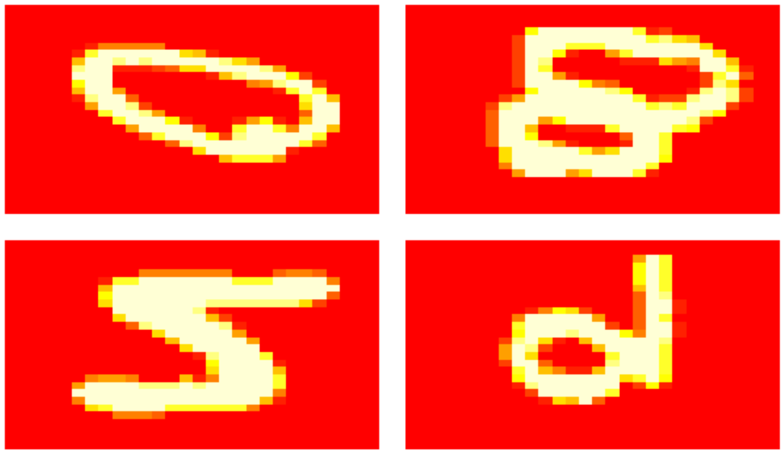


Figure 3: MNIST Dataset

IV. Training and Model Selection

The classifier of this project was trained using the e1071 package[6] of R language. The procedure followed is as follows:

1. First the given training data was divided into two sets by random sampling:
    - Training set: The set on which the classifier was trained for various values of the parameters namely the number of nodes in the hidden layer and the maximum number of iterations allowed.
    - Cross Validation Set: The set on which the accuracy of our classifier was tested.
2. Both the sets were then feauture normalized i.e. each element in both sets was divided by the range of its column (In this case the range was found to be 255). This was done to ensure that our training algorithm converges in optimal time. the formula used for feature normalization is as follows:

$$x_i = \frac{x_i}{(\text{Range of } x_i)} \text{ : for every } x_i \in f$$

where f is the set of features in the dataset.

3. Since the intial number of features was very high (784 in this case), dimensionality reduction was performed on the training set using Principal Component Analysis. The procedure followed for doing so is described below:
    (a) First the *covariance matrix* of the feature normalized training set was obtained. The mathematical formula for obtaining the covariance matrix of a given matrix X is:

$$\sigma = \frac{1}{m}\sum_{i=1}^{m}(X^{(i)})(X^{(i)})^T \qquad (4)$$

where "m" is the total no of rows in the matrix and σ is the covariance matrix. We however, used the $cov()$ function from the R package "stats"[7] to obtain our covariance matrix.

    (b) Principal Component Analysis was performed on the covariance matrix to obtain matrices containing the principal components and standard deviations of the its features.
    (c) A datatable containing the following columns was then formed:
        - Number of Principal Components
        - Individual Variance Explained

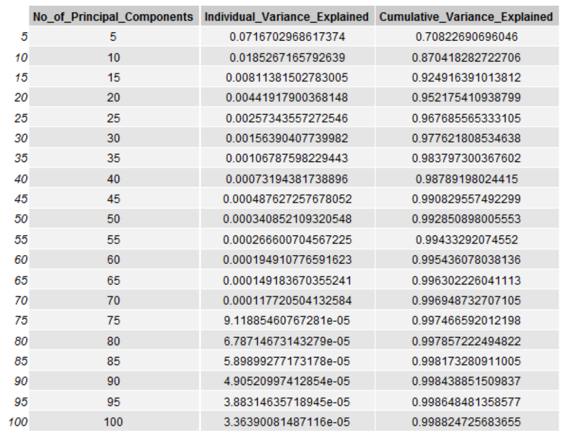
- Cumulative Variance Expalined

This was done to analyze the variance explained by principal components.

(d) The datatable was then filled using the following formulae:

$$V_i = \frac{\sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2} \qquad (5)$$

where V_i is the variance explained by ith principal component. The next column was then obtained by performing cumulative addition on the individual variance explained column. Figure 4 shows a sample of the final datatable obtained. Figure 5 shows the plot between the number of principal components and the cumulative variance explained by them.



	No_of_Principal_Components	Individual_Variance_Explained	Cumulative_Variance_Explained
5	5	0.0716702968617374	0.70822690696046
10	10	0.0185267165792639	0.870418282722706
15	15	0.00811381502783005	0.924916391013812
20	20	0.00441917900368148	0.952175410938799
25	25	0.00257343557272546	0.967685565333105
30	30	0.00156390407739982	0.977621808534638
35	35	0.00106787598229443	0.983797300367602
40	40	0.00073194381738896	0.98789198024415
45	45	0.000487627257678052	0.990829557492299
50	50	0.000340852109320548	0.992850898005553
55	55	0.000266600704567225	0.99433292074552
60	60	0.000194910776591623	0.995436078038136
65	65	0.000149183670355241	0.996302226041113
70	70	0.000117720504132584	0.996948732707105
75	75	9.11885460767281e-05	0.997466592012198
80	80	6.78714673143279e-05	0.997857222494822
85	85	5.89899277173178e-05	0.998173280911005
90	90	4.90520997412854e-05	0.998438851509837
95	95	3.88314635718945e-05	0.998648481358577
100	100	3.36390081487116e-05	0.998824725683655

Figure 4: Datatable containing the values of the individual and cumulative variances explained by the principal components.

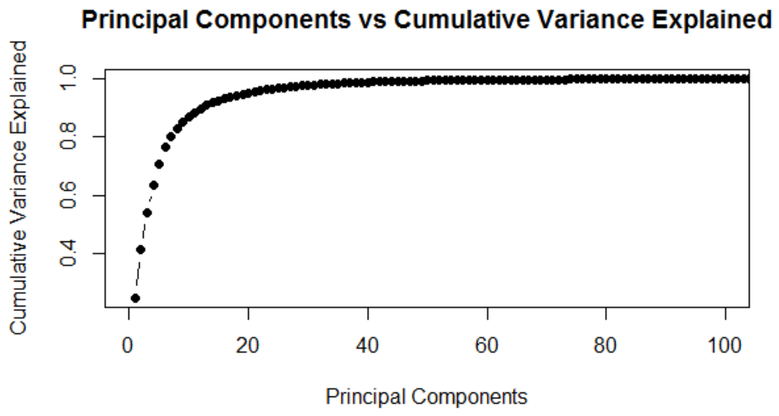


Figure 5: Plot showing the cumulative variance explained by the no of principal components.

(e) It was observed that the first 45 principal components explained 99% variance of our training set. Hence a new training set was constructed using only the first 45 principal components. The formula used for the process was :

$$TrainingSet_{new} = TrainingSet * \text{First 45 Principal Components} \tag{6}$$

This new training set consisted of only 45 features as opposed to 784 features in the original set.

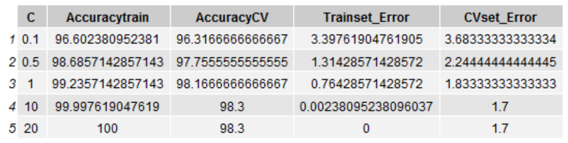
(f) Similar transformation was applied to the cross validation set i.e. the feature normalized CV set was multiplied with the first 45 principal components of the covariance matrix of training set to obtain a new CV set with reduced number of features.

4. A datatable consisting of the different values of parameter "C was formed. The table initially consisted of the following columns:

   - C
   - Trainset Accuracy
   - Testset Accuracy

5. The classifier was trained for different values of parameter "C" taken from the datatable and the resultant training set and test set accuracy values were stored in their corresponding columns.

6. The training set error and test set error were calculated for each observation in the datatable and were plotted against the number of nodes in the hidden layer using the R package ggplot2[9]. The datatable is shown in figure 4 and the corresponding plot is shown in figure 5.



	C	Accuracytrain	AccuracyCV	Trainset_Error	CVset_Error
1	0.1	96.602380952381	96.3166666666667	3.39761904761905	3.68333333333334
2	0.5	98.6857142857143	97.7555555555555	1.31428571428572	2.24444444444445
3	1	99.2357142857143	98.1666666666667	0.76428571428572	1.83333333333333
4	10	99.997619047619	98.3	0.0023809523809603 7	1.7
5	20	100	98.3	0	1.7

Figure 6: Datatable containing the error values obtained during the training of classifier with various values of parameter "C".

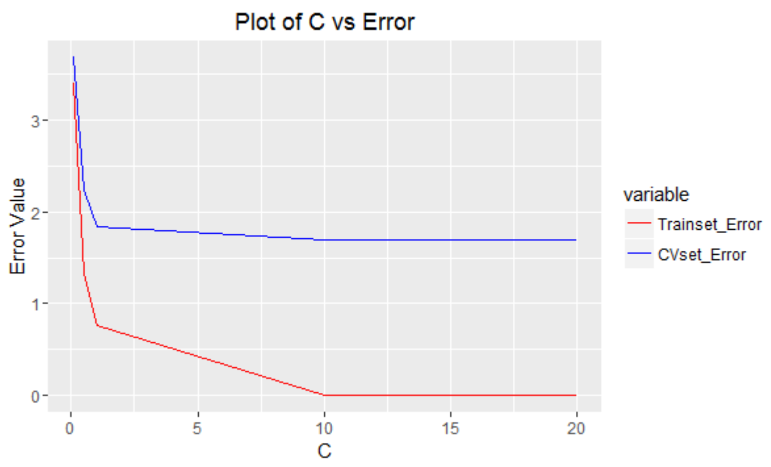


Figure 7: Graph showing the variation of training and test set errors with the parameter "C".

7. We then chose the optimal parameter for training our classifier using the plot (in this case the optimal value of C was found to be 10).

8. The classifier was then trained on the total training data i.e. training set ∪ cross validation set.

9. The test set was then loaded into memory,feature normalized and multiplied with first 45 principal components to obtain a new test set with reduced number of features.

10. Finally prediction were made on the test set and were compared with the test set labels to obtain the final test set accuracy. The formula used to calculate accuracy is as follows:

$$Accuracy = \frac{\text{No. of correct predictions}}{\text{Total No. of examples}}$$

The final test set acuuracy was found to be 98.44%.

V. Conclusion

This project explored the performance and implementation of Support Vector Machine (with Radial Basis Function Kernel) in handwritten digit classification. It was found that the classifier performed fairly well given that its parameter was chosen using model selection procedure. It should be noted that though the SVM algorithm performed desirably it has significant memory and time requirements which should be taken into account before implementation.

VI. R Code

Given below is the R code we used for our project:

```
#Reading the data and splitting it into training and
#cross validation set
train <- read.csv("mnist_train.csv",header= FALSE)
splittingseed <- 100
set.seed(splittingseed )
selsize <- floor(0.70*nrow(train))
sel_ind <- sample(seq_len(nrow(train)),size=selsize)
trainset <- train[sel_ind,]
cvset <- train[-sel_ind,]
X <- trainset[,-1]
Y <- trainset[,1]
trainlabel <- trainset[,1]
cvlabel <- cvset[,1]

#Visualizing the training set images
attach(X)
par(mfrow=c(2,2),mai=c(0.1,0.1,0.1,0.1))
m <- matrix(unlist(X[3,]),nrow = 28,ncol = 28 )
image(m, axes=FALSE)
m <- matrix(unlist(X[12,]),nrow = 28,ncol = 28 )
image(m, axes=FALSE)
m <- matrix(unlist(X[48,]),nrow = 28,ncol = 28 )
image(m, axes=FALSE)
```

```
m <- matrix(unlist(X[212,]),nrow = 28,ncol = 28 )
image(m, axes=FALSE)


#Reducing Train and CV using PCA
Xreduced <- X/255
Xcov <- cov(Xreduced)
pcaX <- prcomp(Xcov)

# Creating a datatable to store and plot the
# No of Principal Components vs Cumulative Variance Explained
vexplained <- as.data.frame(pcaX$sdev^2/sum(pcaX$sdev^2))
vexplained <- cbind(c(1:784),vexplained,cumsum(vexplained[,1]))
colnames(vexplained) <- c("No_of_Principal_Components",
"Individual_Variance_Explained","Cumulative_Variance_Explained")

#Plotting the curve using the datatable obtained
plot(vexplained$No_of_Principal_Components,
vexplained$Cumulative_Variance_Explained,
xlim = c(0,100),type='b',pch=16,xlab =
"Principal␣Components",ylab = "Cumulative␣Variance␣Explained",
main = 'Principal␣Components␣vs␣Cumulative␣Variance␣Explained')

#Datatable to store the summary of the datatable obtained
vexplainedsummary <- vexplained[seq(0,100,5),]
vexplainedsummary

#Storing the vexplainedsummary datatable in png
#format for future reference.
library(gridExtra)
png("datatablevaraince␣explained.png",height = 800,width =1000)
p <-tableGrob(vexplainedsummary)
grid.arrange(p)
dev.off()


Xfinal <- as.matrix(Xreduced) %*% pcaX$x[,1:45]
cvreduced<- cvset[,-1]/255
cvfinal <- as.matrix(cvreduced) %*% pcaX$x[,1:45]

#Making training and cvset labels as factors
cvlabel <- as.factor(cvlabel)
trainlabel <- as.factor(trainlabel)

#Making a datatable to store errors for choosing the parameter
#C suing model selection
datatable_model_selection <- data.frame(c(0.1,0.5,1,10,20),
c(NA,NA,NA,NA,NA),c(NA,NA,NA,NA,NA))
```

```
colnames(datatable_model_selection) <-
c("C","Accuracytrain","AccuracyCV")
View(datatable_model_selection)

#Function to calculate accuracy for various C and store resultant
#train and cv accuracy in datatable
calculate_accuracy<- function(variancefactor)
{
require(e1071)
svm.model <- svm(Xfinal,as.factor(trainlabel),cost = variancefactor)
prediction <- predict(svm.model,Xfinal)
table(prediction,trainlabel)
correct <- prediction==trainlabel
AccuracyTrain <- (sum(correct)/nrow(Xfinal))*100
cat(sprintf("Accuracytrain:%f\n",AccuracyTrain))
prediction2 <- predict(svm.model,cvfinal)
table(prediction2,cvlabel)
correct2<- prediction2==cvlabel
AccuracyCV <- (sum(correct2)/nrow(cvfinal))*100
cat(sprintf("Accuracycv:%f\n",AccuracyCV))
return(c(AccuracyTrain,AccuracyCV))
}

#Applying the function to all the rows of datatable
for(j in 1:5)
{
temp <-calculate_accuracy(datatable_model_selection[j,1])
datatable_model_selection[j,2]<-temp[1]
datatable_model_selection[j,3]<-temp[2]
}

#Adding columns for errors in datatable
datatable_model_selection$Trainset_Error <-
(100-datatable_model_selection$Accuracytrain)
datatable_model_selection$CVset_Error <-
(100-datatable_model_selection$AccuracyCV)
datatable_model_selection

#Storing the datatable in png format for future reference.
library(gridExtra)
png("datatable.png",height = 300,width = 4000)
p <-tableGrob(datatable_model_selection)
grid.arrange(p)
dev.off()

#Ploting the graph of C vs Errors
#Getting the subset of datatable that is to be plotted
plotdataframe <- datatable_no_of_nodes[,c(1,4,5)]
```

```
#melting the dataframe to feed to ggplot() function
library(reshape2)
meltedframe <- melt(plotdataframe,id="C")

#Applying ggplot() function
library(ggplot2)
finalplot<- ggplot(meltedframe,aes(C,value,color=variable))
+geom_line()+scale_colour_manual(values = c("red","blue"))
finalplot <- finalplot+xlab("C")+ylab("Error␣Value")
+ggtitle("Plot␣of␣C␣vs␣Error")
finalplot

#Hence the optimal value of C is 10
#Training SVM on total training set with C=10
#Applying PCA on total training set
totaltrain <- as.matrix(train[,-1])
totaltrainlabel <- as.matrix(train[,1])
totaltrainreduced <- totaltrain/255
totaltraincov <- cov(totaltrainreduced)
pcatotaltrain <- prcomp(totaltraincov)

#Applying SVM on total training set and calculating accuracy
totaltrainlabel <- as.factor(totaltrainlabel)
svm.model.final <- svm(totaltrainfinal,totaltrainlabel,cost = 10)
predictionfinaltrain <- predict(svm.model.final,totaltrainfinal)
correcttrainfinal <- predictionfinaltrain==totaltrainlabel
Accuracytrainfinal <-
(sum(correcttrainfinal)/nrow(totaltrainfinal))*100
Accuracytrainfinal

#Load test to reduced and normalize it for predictions
test<- read.csv("mnist_test.csv",header=FALSE)
testlabel <- as.factor(test[,1])

#Applying PCA to test set
testreduced <- test[,-1]/255
testfinal <- as.matrix(testreduced) %*% pcatotaltrain$x[,1:45]

#Predicitng on the test set using trained SVM model
predictionfinaltest <- predict(svm.model.final,testfinal)

#Calculating test set accuracy
correcttestfinal <- predictionfinaltest==testlabel
Accuracytestfinal <- (sum(correcttestfinal)/nrow(testfinal))*100
#Final Train Accuracy =99.98%
#Final Test Accuracy = 98.44%
```

References

[1] Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Lawrence D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *International Conference on Pattern Recognition*, pages 77–77. IEEE Computer Society Press, 1994.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[3] Y Le Cun, LD Jackel, B Boser, JS Denker, HP Graf, I Guyon, D Henderson, RE Howard, and W Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *Communications Magazine, IEEE*, 27(11):41–46, 1989.

[4] Yuchun Lee. Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural computation*, 3(3):440–449, 1991.

[5] David Meyer and FH Technikum Wien. Support vector machines.

[6] David Meyer and Technische Universitat Wien. Support vector machines.The interface to libsvm in package e1071.

[7] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[8] Joseph Chet Redmon. MNIST in CSV. `http://pjreddie.com/projects/mnist-in-csv/`.

[9] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009.

[10] Corinna Cortes Y LeCun. MNIST digit database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.