CSI 873/MATH 689
Final Project
Ajay Kulkarni (G01024139)

Hand Written Digits Recognition Problem Using Support Vector Machine

Abstract: Support Vector Machine with the dual soft-margin is implemented in Matlab. The dual soft-margin equation is optimized using "quadprog" function in Matlab. Support Vector Machine is tested on various conditions, and it is observed that it gives good results for binary classification. For binary classifications first, we classified 3s and 6s for 500 training data points, and then we reduced the pixels uniformly as well as using SVD. We also have implemented SVM to classify even vs. odd numbers. In the end, we executed 10 SVMs to detect a particular digit against the rest digits.

Introduction

Support Vector Machine with the dual soft-margin is implemented in Matlab with Radial Basis Function. In this project, we have used SVM for binary classification as well as for multiclass classification. In binary classification, we have classified 3s vs. 6s for three conditions using 500 data points for training as well as for test the model. In the first condition, we used all the data, i.e., 500 data points for training. For the second condition we reduced the pixels (784) uniformly by 50%, 75%, 90% and 95% and in the third condition, we reduced the pixels (784) using SVD by 50%, 75%, 90% and 95%. In addition to that, we also have checked the results after reducing training examples by 50%, 75%, 90% and 95%. Further, we used SVM to classify even vs. odd numbers. For even-odd classification, we have used 1000 data points (100 data points for every digit) for training as well as for test data. In the end, we performed multi-class classification using SVM. For multi-class classification, we generated 10 SVMs, and we used 1000 data points for training as well as for test the model.

This report is divided into four parts. The first part explains the data preprocessing, second part explains about the implementation of Support Vector Machine. The third part will be results as well as discussion of results, and the fourth part contains the implemented code.

Part -1: Data Preprocessing

Data preprocessing, is performed in R and details about data preprocessing for every condition are explained below,

- **3s vs. 6s classification**

1) First 250 rows from data files were extracted for training data and stored in two different variables. After that, both the variables were converted into a data frame and concatenated. The concatenated data then stored in a variable.

2) Scaling is performed on data by dividing maximum value from each column to every value in the same column. The results generated from scaling was in between 0 and 1. We stored the scaled data in one variable, and then checked for NaN values. In the data, label 3 is replaced by -1 and label 6 is replaced by 1.

3) After doing all the above steps, we saved the data into a train_data.csv file, which will be the training data for our model.

4) Similar preprocessing is performed for the test data, and the test data is selected from test3.txt and test6. txt. After performing above preprocessing, we stored the test data in test_data.csv file.

- **Even vs. odd classification**

1) For even and odd classifications we extracted 0, 2, 4, 6 and 8 digits as even numbers from their respective train and test data files. In case of odd numbers, we extracted 1, 3, 5, 7 and 9 digits from their respective train and test data files. For this classification, we have extracted 100 data points for every digit. So our test and training data both will have 1000 data points each.

2) After extracting all the data points, we combined the data and stored it in one data frame. We performed all the preprocessing steps as mentioned above for training and test data. In the end, we replace all even number labels by -1 and all odd number labels by 1.

- **Multiclass classification**

1) For multiclass classification, we used 100 data points for every digit in training as well as in test data. We also performed all the preprocessing steps as mentioned above with one change.

2) For this type of classification, we created 10 independent data files for training data and one common test data file. In training data suppose if you want to detect a particular digit say 0 then we replaced the label 0 as +1 and other labels (1, 2, 3, .. , 9) to -1. So in this way, we prepared 10 training data files.

Part 2: Support Vector Machine implementation

We have prepared data by performing all the preprocessing in R which we have explained in Part 1. In this part, we have mentioned some implementation details which we did in Matlab. The implementation details are as follows,

- In Matlab we have imported data using "csvread" command and the command which we have used is as follows,

```
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
```

So the above commands imported the train as well as test data and stored them in variables. The above command will start importing data from row 1. So the data doesn't have column titles.

- To implement SVM we have created two functions "svmcf" and "rbf". The details about these functions are given below,

a. rbf function

We have built a function rbf to represent Radial Basis Function. We have implemented Radial Basis Function using Euclidean distance and the implementation of the function is given below,

```
function [dist] = rbf(x,y)
        d1 = pdist2(x,y);
        d2 = d1.^2;
        dist = exp(-0.05.*d2);
end
```

In the above function "pdist2" will calculate the Euclidean distance between two matrices and will store it in variable d1. Further d1 is processed to find the output for Radial Basis Function.

b. svmcf function

We have created svmcf function for SVM classification. In this function, we are using rbf function for calculating Radial Basis kernel. After calculating kernel, we have formulated all the required matrices and vectors for dual soft-margin SVM. The formulation of all the matrices and vectors are given below,

```
% Radial basis function
k = rbf(x_train, x_train);

% Other matrices and vectors
H = (y_train*y_train').*k;
f = -1*ones(1, nrow);
A = [];
b = [];
Aeq = y_train';
beq = 0;
LB = zeros(1,nrow);
UB = 100*ones(1,nrow);

% function call to quadprog
alpha = quadprog(H,f,A,b,Aeq,beq,LB,UB);
```

In our implementation, we have kept lower bound as 0 and upper bound as 100. This will give us results for alpha. After getting results for alpha we are defining a small number 1e-6 and replacing the values of alpha with 0 if alpha < 1e-6. After replacing small values with 0, we are selecting the index of a first support vector using “find” function. We are using the index from find function to extract the corresponding data label and columns of that support vector. Using those values, we are calculating “b” and then predicting results. Matlab statements are given below for calculating bias and predicted values.

```
b = yi0 - (alpha.*y_train)'*rbf(x_train, xi0);
f = (alpha.*y_train)'*rbf(x_test, x_train)' + b;
```

After implementing above formulas, we are using “sign” function to understand the predicted class label. The accuracy of the model is calculated by computing confusion matrix. To compute confusion matrix, we have used “confusionmat” command and then based on those results we are calculating the percentage accuracy as well as percentage error.

Part 3: Results and conclusion

In this section, we are going to discuss results for different conditions. First, we will explain results for 3s vs. 6s classification with different conditions, and then we will explain other cases like even vs. odd and multiclass classification.

- **3s vs. 6s classification**

In 3s vs. 6s classification, we have replaced 3 as -1 and 6 as 1. But in confusion matrix, we are representing results in terms of 3 and 6. The results for different conditions are as follows,

- **Using 500 training data points (250 3s and 250 6s)**

3	**6**
250	0
11	239

The above confusion matrix shows that SVM was able to classify 250 correct 3s and 239 correct 6s. From the above confusion matrix percentage accuracy is calculated as 97.80% and percentage error is 2.2%.

- **Uniformly reduction of pixels by 50%, 75%, 90% and 95%**

Reduction by 50%	
3	**6**
248	2
1	249

Reduction by 75%	
3	**6**
244	6
5	245

Reduction by 90%	
3	**6**
241	9
7	243

Reduction by 95%	
3	**6**
235	15
17	233

The uniform reduction of the pixels is performed in R using “uniform.select” function from “bigpca” library. The confusion matrices for different conditions are mentioned above. It can be observed that after uniformly reducing 50% of the pixels we are getting 99.40% percentage accuracy and 0.60% error. In case of 75% reduction of the pixels, we are getting 97.80% accuracy and 2.20% error. For 90% reduction of pixels,

we can observe that the accuracy is 96.80% and error is 3.20%. Finally, for 95% of reduction of pixels, the percentage accuracy is 93.60% and error is 6.40%. It can be easily observed that as the pixel percentage reduces the percentage accuracy is decreasing. Also, maximum accuracy is observed for 50% of the pixel reduction which is 99.40%.

- **Using SVD for reduction of pixels by 50%, 75%, 90% and 95%**

Reduction by 50%	
3	**6**
250	0
9	241

Reduction by 75%	
3	**6**
249	1
3	247

Reduction by 90%	
3	**6**
249	1
2	248

Reduction by 95%	
3	**6**
249	1
1	249

SVD can be used for the dimensionality reduction and for selecting important features in the data. The percentage accuracy for the data with 50% of the pixel reduction is 98.20% and percentage error is 1.80%. For 75% of the pixel reduction, the percentage accuracy is 99.20% and percentage error is 0.80%. After reducing 90% pixels, the percentage accuracy is 99.40% and percentage error is 0.60%. In last case where we are reducing pixels by 95% the percentage accuracy is 99.60% and percentage error is 0.40%. So, it can be observed that as pixel percentage reduces the percentage accuracy is increasing. Thus, it means that if we use 5% of the pixels after performing SVD then also we will get the maximum accuracy.

- **Reducing original number of training examples by 50%, 75%, 90% and 95%**

Reduction by 50%	
3	**6**
248	2
15	235

Reduction by 75%	
3	**6**
250	0
47	203

Reduction by 90%	
3	**6**
250	0
79	171

Reduction by 95%	
3	**6**
250	0
140	110

For this condition, we are randomly reducing the 50%, 75%, 90% and 95% training examples. If we randomly reduce 50% of the training examples, then the percentage accuracy is calculated as 99.60% and error is 3.40%. For random reduction of 75% of the training examples, percentage accuracy is 90.60% and error is 9.40%. In case of 90% reduction of training examples, the percentage accuracy is 84.20% and error is 15.80%. In the last case where we are reducing the training examples by 95%, the percentage accuracy is 72% and error is 28%. So it can be observed that as the percentage of training examples decreases the percentage accuracy is also decreasing. It is possible because of the lack of training data for the SVM. Also, it is observed that SVM was able to classify 3s correctly and misclassifies more 6s as there is increasing in percentage of reduction of data.

- **Even vs. odd classification**

Even	Odd
474	26
40	460

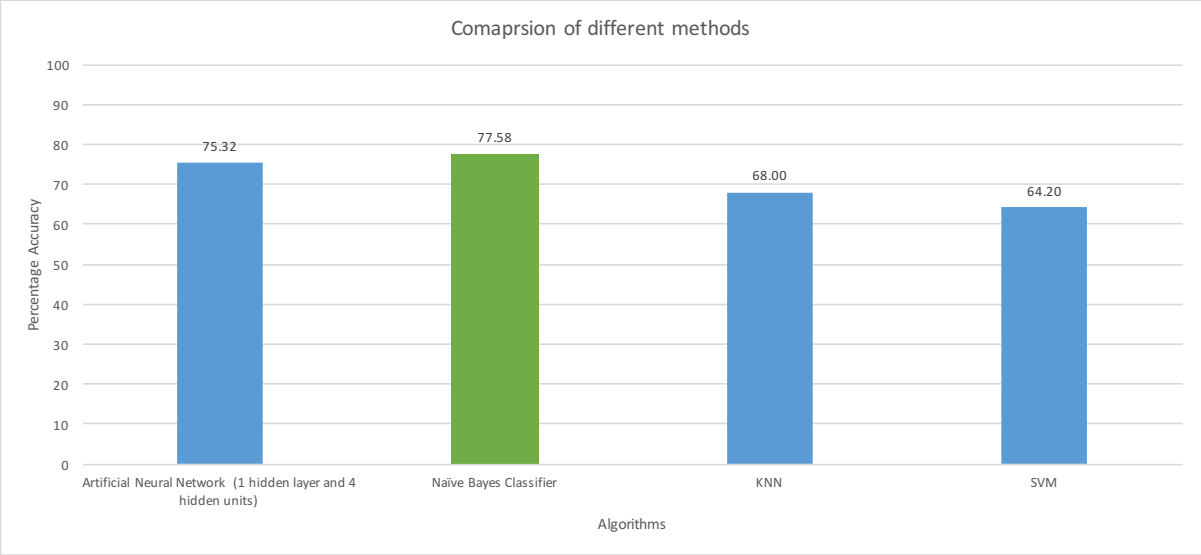
For this classification, we label all the even numbers as -1 and all the odd numbers 1. It can be observed from the confusion matrix that SVM was able to classify 474 even numbers and 460 odd numbers correctly. The percentage accuracy is calculated as 93.40% and error is 6.60%.

- **Multiclass classification**

Digit	Correct predicted labels
0	69
1	95
2	49
3	46
4	77
5	33
6	70
7	79
8	55
9	69

It can be observed from the above table that SVM was able to predict a maximum number of correct labels for digit ‘1’ and minimum number for digit ‘5’. From 1000 labels SVM was able to predict 642 labels correctly. So the percentage accuracy calculated as 64.2% and percentage error is 35.8%.

- **Comparison of results with other methods**



From the above bar plot, it can be observed that Naïve Bayes classifier have more accuracy as compared to other methods. The least accuracy can be observed for SVM which is about 64.20%.

Part 4: R code

In this section, we have included the R code for data preprocessing and the implemented Matlab code for SVM.

- **Data preprocessing scripts**

- **Data preprocessing script for classification of 3s and 6s**

```
# Data Preprocessing for Training data
t1 = read.table("train3.txt")
t11 = t1[1:250,]
dim(t11)

t2 = read.table("train6.txt")
t12 = t2[1:250,]
dim(t12)

# Converting data into dataframe
t11 = data.frame(t11)
t12 = data.frame(t12)

# Combining data into variable training_data
training_data = rbind(t11,t12)
d = dim(training_data)
d

# Scaling
for (i in 2:785) {
  k = max(training_data[,i])
  if(k!=0) {
    training_data[,i] = training_data[,i]/k
  }
}

# Labeling 3 as -1 and 6 as 1
for (i in 1:500) {
  if(training_data[i,1]==3){
    training_data[i,1] = -1
  }
  else
  {
    training_data[i,1] = 1
  }
}

# Checking NaN values
sum(is.na(training_data))

# Storing training data in csv file
```

```
write.csv(training_data, file = "train_data.csv", row.names = FALSE)

# Data Preprocessing for Test data
ts1 = read.table("test3.txt")
ts11 = ts1[1:250,]
dim(ts11)

ts2 = read.table("test6.txt")
ts12 = ts2[1:250,]
dim(ts12)

# Converting data into dataframe
ts11 = data.frame(ts11)
ts12 = data.frame(ts12)

# Combining data into variable test_data
test_data = rbind(ts11,ts12)
d = dim(test_data)

# Scaling
for (i in 2:785) {
  k = max(test_data[,i])
  if(k!=0) {
    test_data[,i] = test_data[,i]/k
  }
}

# Labeling 3 as -1 and 6 as 1
for (i in 1:500) {
  if(test_data[i,1]==3){
    test_data[i,1] = -1
  }
  else
  {
    test_data[i,1] = 1
  }
}

# Checking NaN values
sum(is.na(test_data))

# Storing test data in csv file
write.csv(test_data, file = "test_data.csv", row.names = FALSE)
```

- **Data preprocessing script for uniformly reduction of pixels**

```
install.package("bigpca")
library(bigpca)

train = read.csv("train_data.csv")
test = read.csv("test_data.csv")
train_new = train[,2:785]
test_new = test[,2:785]

# ------------------------------------------------------------------------
```

```
# Uniform reduction of 50% of pixels
u1 = uniform.select(train_new, keep = 0.5, rows = FALSE, random = FALSE)

train_50 = as.data.frame(train_new[,u1$order.c])
train_c = as.data.frame(train[,1])
train_50 = cbind(train_c, train_50)
colnames(train_50)[1] = "V1"
View(train_50)

test_50 = as.data.frame(test_new[,u1$order.c])
test_c = as.data.frame(test[,1])
test_50 = cbind(test_c, test_50)
colnames(test_50)[1] = "V1"
View(test_50)

write.csv(train_50, file = "train_50.csv", row.names = FALSE)
write.csv(test_50, file = "test_50.csv", row.names = FALSE)
# ---------------------------------------------------------------------
# Uniform reduction of 75% of pixels
u2 = uniform.select(train_new, keep = 0.25, rows = FALSE, random = FALSE)

train_25 = as.data.frame(train_new[,u2$order.c])
train_c = as.data.frame(train[,1])
train_25 = cbind(train_c, train_25)
colnames(train_25)[1] = "V1"
View(train_25)

test_25 = as.data.frame(test_new[,u2$order.c])
test_c = as.data.frame(test[,1])
test_25 = cbind(test_c, test_25)
colnames(test_25)[1] = "V1"
View(test_25)

write.csv(train_25, file = "train_25.csv", row.names = FALSE)
write.csv(test_25, file = "test_25.csv", row.names = FALSE)
# ---------------------------------------------------------------------
# Uniform reduction of 90% of pixels
u3 = uniform.select(train_new, keep = 0.10, rows = FALSE, random = FALSE)

train_10 = as.data.frame(train_new[,u3$order.c])
train_c = as.data.frame(train[,1])
train_10 = cbind(train_c, train_10)
colnames(train_10)[1] = "V1"
View(train_10)

test_10 = as.data.frame(test_new[,u3$order.c])
test_c = as.data.frame(test[,1])
test_10 = cbind(test_c, test_10)
colnames(test_10)[1] = "V1"
View(test_10)

write.csv(train_10, file = "train_10.csv", row.names = FALSE)
write.csv(test_10, file = "test_10.csv", row.names = FALSE)
# ---------------------------------------------------------------------
# Uniform reduction of 95% of pixels
```

```
u4 = uniform.select(train_new, keep = 0.05, rows = FALSE, random = FALSE)

train_5 = as.data.frame(train_new[,u4$order.c])
train_c = as.data.frame(train[,1])
train_5 = cbind(train_c, train_5)
colnames(train_5)[1] = "V1"
View(train_5)

test_5 = as.data.frame(test_new[,u4$order.c])
test_c = as.data.frame(test[,1])
test_5 = cbind(test_c, test_5)
colnames(test_5)[1] = "V1"
View(test_5)

write.csv(train_5, file = "train_5.csv", row.names = FALSE)
write.csv(test_5, file = "test_5.csv", row.names = FALSE)
```

- **Data preprocessing script for classification even and odd numbers**

```
# Data Preprocessing for Training data
t1 = read.table("train0.txt")
t11 = t1[1:100,]
dim(t11)

t2 = read.table("train2.txt")
t12 = t2[1:100,]
dim(t12)

t3 = read.table("train4.txt")
t13 = t3[1:100,]
dim(t13)

t4 = read.table("train6.txt")
t14 = t4[1:100,]
dim(t14)

t5 = read.table("train8.txt")
t15 = t5[1:100,]
dim(t15)

to1 = read.table("train1.txt")
to11 = to1[1:100,]
dim(to11)

to2 = read.table("train3.txt")
to12 = to2[1:100,]
dim(to12)

to3 = read.table("train5.txt")
to13 = to3[1:100,]
dim(to13)

to4 = read.table("train7.txt")
to14 = to4[1:100,]
dim(to14)
```

```
to5 = read.table("train9.txt")
to15 = to5[1:100,]
dim(to15)

# Converting data into dataframe
t11 = data.frame(t11)
t12 = data.frame(t12)
t13 = data.frame(t13)
t14 = data.frame(t14)
t15 = data.frame(t15)
to11 = data.frame(to11)
to12 = data.frame(to12)
to13 = data.frame(to13)
to14 = data.frame(to14)
to15 = data.frame(to15)

# Combining data into variable training_data
training_data = rbind(t11,t12,t13,t14,t15,to11,to12,to13,to14,to15)
d = dim(training_data)
d

# Scaling
for (i in 2:785) {
  k = max(training_data[,i])
  if(k!=0) {
    training_data[,i] = training_data[,i]/k
  }
}

# Labeling even as -1 and odd as 1
for (i in 1:1000) {
  if((training_data[i,1]%%2)==0){
    training_data[i,1] = -1
  }
  else
  {
    training_data[i,1] = 1
  }
}

# Checking NaN values
sum(is.na(training_data))

# Storing training data in csv file
write.csv(training_data, file = "train_data_even_odd.csv", row.names =
FALSE)

# Data Preprocessing for Test data
t1 = read.table("test0.txt")
t11 = t1[1:100,]
dim(t11)

t2 = read.table("test2.txt")
t12 = t2[1:100,]
```

```
dim(t12)

t3 = read.table("test4.txt")
t13 = t3[1:100,]
dim(t13)

t4 = read.table("test6.txt")
t14 = t4[1:100,]
dim(t14)

t5 = read.table("test8.txt")
t15 = t5[1:100,]
dim(t15)

to1 = read.table("test1.txt")
to11 = to1[1:100,]
dim(to11)

to2 = read.table("test3.txt")
to12 = to2[1:100,]
dim(to12)

to3 = read.table("test5.txt")
to13 = to3[1:100,]
dim(to13)

to4 = read.table("test7.txt")
to14 = to4[1:100,]
dim(to14)

to5 = read.table("test9.txt")
to15 = to5[1:100,]
dim(to15)


# Converting data into dataframe
t11 = data.frame(t11)
t12 = data.frame(t12)
t13 = data.frame(t13)
t14 = data.frame(t14)
t15 = data.frame(t15)
to11 = data.frame(to11)
to12 = data.frame(to12)
to13 = data.frame(to13)
to14 = data.frame(to14)
to15 = data.frame(to15)

# Combining data into variable training_data
test_data = rbind(t11,t12,t13,t14,t15,to11,to12,to13,to14,to15)
d = dim(test_data)
d

# Scaling
for (i in 2:785) {
  k = max(test_data[,i])
```

```
  if(k!=0) {
    test_data[,i] = test_data[,i]/k
  }}

# Labeling even as -1 and odd as 1
for (i in 1:1000) {
  if((test_data[i,1]%%2)==0){
    test_data[i,1] = -1
  }
  else
  {
    test_data[i,1] = 1
  }
}

# Checking NaN values
sum(is.na(test_data))

# Storing training data in csv file
write.csv(test_data, file = "testdata_even_odd.csv", row.names = FALSE)
```

- **Data preprocessing script for classification multiclass classification**

```
# Data Preprocessing for Training data
t1 = read.table("train0.txt")
t11 = t1[1:100,]
dim(t11)

t2 = read.table("train1.txt")
t12 = t2[1:100,]
dim(t12)

t3 = read.table("train2.txt")
t13 = t3[1:100,]
dim(t13)

t4 = read.table("train3.txt")
t14 = t4[1:100,]
dim(t14)

t5 = read.table("train4.txt")
t15 = t5[1:100,]
dim(t15)

t6 = read.table("train5.txt")
t16 = t6[1:100,]
dim(t16)

t7 = read.table("train6.txt")
t17 = t7[1:100,]
dim(t17)

t8 = read.table("train7.txt")
t18 = t8[1:100,]
dim(t18)
```

```
t9 = read.table("train8.txt")
t19 = t9[1:100,]
dim(t17)

t10 = read.table("train9.txt")
t10 = t10[1:100,]
dim(t10)


# Converting data into dataframe
t11 = data.frame(t11)
t12 = data.frame(t12)
t13 = data.frame(t13)
t14 = data.frame(t14)
t15 = data.frame(t15)
t16 = data.frame(t16)
t17 = data.frame(t17)
t18 = data.frame(t18)
t19 = data.frame(t19)
t10 = data.frame(t10)

# Combining data into variable training_data
training_data = rbind(t11,t12,t13,t14,t15,t16,t17,t18,t19,t10)
d = dim(training_data)
d

# Scaling
for (i in 2:785) {
  k = max(training_data[,i])
  if(k!=0) {
    training_data[,i] = training_data[,i]/k
  }
}

for (i in 1:1000) {
  if(training_data[i,1]==7){
    training_data[i,1] = 1
  }
  else
  {
    training_data[i,1] = -1
  }
}

# Checking NaN values
sum(is.na(training_data))

# Storing training data in csv file
write.csv(training_data, file = "train_data_7.csv", row.names = FALSE)

# Data Preprocessing for Test data
tr1 = read.table("test0.txt")
t21 = tr1[1:100,]
dim(t21)
```

```
tr2 = read.table("test1.txt")
t22 = tr2[1:100,]
dim(t22)

tr3 = read.table("test2.txt")
t23 = tr3[1:100,]
dim(t23)

tr4 = read.table("test3.txt")
t24 = tr4[1:100,]
dim(t24)

tr5 = read.table("test4.txt")
t25 = tr5[1:100,]
dim(t25)

tr6 = read.table("test5.txt")
t26 = tr6[1:100,]
dim(t26)

tr7 = read.table("test6.txt")
t27 = tr7[1:100,]
dim(t27)

tr8 = read.table("test7.txt")
t28 = tr8[1:100,]
dim(t28)

tr9 = read.table("test8.txt")
t29 = tr9[1:100,]
dim(t29)

tr10 = read.table("test9.txt")
t20 = tr10[1:100,]
dim(t20)

# Converting data into dataframe
t21 = data.frame(t21)
t22 = data.frame(t22)
t23 = data.frame(t23)
t24 = data.frame(t24)
t25 = data.frame(t25)
t26 = data.frame(t26)
t27 = data.frame(t27)
t28 = data.frame(t28)
t29 = data.frame(t29)
t20 = data.frame(t20)

# Combining data into variable test_data
test_data = rbind(t21,t22,t23,t24,t25,t26,t27,t28,t29,t20)
d = dim(test_data)


# Scaling
```

```
for (i in 2:785) {
  k = max(test_data[,i])
  if(k!=0) {
    test_data[,i] = test_data[,i]/k
  }
}

# Checking NaN values
sum(is.na(test_data))

# Storing test data in csv file
write.csv(test_data, file = "test_data_new_0_9.csv", row.names = FALSE)
```

- **Matlab scripts for SVM**

- **Radial Basis Function (rbf.m)**

```
function [dist] = rbf(x,y)
    d1 = pdist2(x,y);
    d2 = d1.^2;
    dist = exp(-0.05.*d2);
end
```

- **SVM implementation (svmcf.m)**

```
function [C, order, Acc] = svmcf(train, test)
% Total number of rows and columns in training data
[nrow,ncol] = size(train);
[nrow1,ncol1] = size(test);

% Storing labels and data in different variables
x_train = train(:,[2:ncol]);
y_train = train(:,1);

% Storing labels and data in different variables
x_test = test(:,[2:ncol1]);
y_test = test(:,1);

% Radial basis function
k = rbf(x_train, x_train);

% Other matrices and vectors
H = (y_train*y_train').*k;
f = -1*ones(1, nrow);
A = [];
b = [];
Aeq = y_train';
beq = 0;
LB = zeros(1,nrow);
UB = 1000*ones(1,nrow);

% function call to quadprog
alpha = quadprog(H,f,A,b,Aeq,beq,LB,UB);

% Replacing small number by 0 (alpha*)
e1 = 1e-6;
alpha(alpha < e1) = 0;
ind = find((alpha > e1)&(alpha < 100));
xi0 = x_train(ind(1),:);
```

```
yi0 = y_train(ind(1));


% Calculating bias
b = yi0 - (alpha.*y_train)'*rbf(x_train, xi0);
% Decision rule
f = (alpha.*y_train)'*rbf(x_test, x_train)' + b;
pred = sign(f);
[C, order] = confusionmat(y_test,pred);
Acc = (C(1,1)+C(2,2))/size(y_test,1);
Acc = Acc * 100;
end
```

- **Matlab script for execution (Question 1 – Question 8)**

```
clear;
% SVM classification for 3s vs. 6s
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)
% ------------------------------------------------------------------------
% Reducing orignal number of pixels by 50%
train = csvread('train_50.csv',1);
test = csvread('test_50.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Reducing orignal number of pixels by 75%
train = csvread('train_25.csv',1);
test = csvread('test_25.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Reducing orignal number of pixels by 90%
train = csvread('train_10.csv',1);
test = csvread('test_10.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Reducing orignal number of pixels by 95%
train = csvread('train_5.csv',1);
test = csvread('test_5.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)
%------------------------------------------------------------------------
% Using SVD Reducing dimension by 50%
```

```
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
train_new = train(:,2:785);
test_new = test(:,2:785);
[U,S,V] = svd(train_new);
% Selecting first 50% columns
S1 = S(:,1:392);
V1 = V(:,1:392);
train_1 = U*S1;
test_1 = test_new*V1;
train_2 = [train(:,1),train_1];
test_2 = [test(:,1), test_1];
[C, order, Acc] = svmcf(train_2, test_2);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Using SVD Reducing dimension by 75%
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
train_new = train(:,2:785);
test_new = test(:,2:785);
[U,S,V] = svd(train_new);
% Selecting first 25% columns
S1 = S(:,1:196);
V1 = V(:,1:196);
train_1 = U*S1;
test_1 = test_new*V1;
train_2 = [train(:,1),train_1];
test_2 = [test(:,1), test_1];
[C, order, Acc] = svmcf(train_2, test_2);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Using SVD Reducing dimension by 90%
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
train_new = train(:,2:785);
test_new = test(:,2:785);
[U,S,V] = svd(train_new);
% Selecting first 10% columns
S1 = S(:,1:78);
V1 = V(:,1:78);
train_1 = U*S1;
test_1 = test_new*V1;
train_2 = [train(:,1),train_1];
test_2 = [test(:,1), test_1];
[C, order, Acc] = svmcf(train_2, test_2);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

% Using SVD Reducing dimension by 95%
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
train_new = train(:,2:785);
test_new = test(:,2:785);
[U,S,V] = svd(train_new);
% Selecting first 5% columns
```

```
S1 = S(:,1:39);
V1 = V(:,1:39);
train_1 = U*S1;
test_1 = test_new*V1;
train_2 = [train(:,1),train_1];
test_2 = [test(:,1), test_1];
[C, order, Acc] = svmcf(train_2, test_2);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)
%-------------------------------------------------------------------------
train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
% Randomly selecting 50% training examples
rng(1);
c = randperm(500);
c = c(1:250);
train_new = train(c,:);
[C, order, Acc] = svmcf(train_new, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
% Randomly selecting 25% rows
rng(1);
c=randperm(500);
c = c(1:125);
train_new = train(c,:);
[C, order, Acc] = svmcf(train_new, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
% Randomly selecting 10% rows
rng(1);
c=randperm(500);
c = c(1:50);
train_new = train(c,:);
[C, order, Acc] = svmcf(train_new, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)

train = csvread('train_data.csv',1);
test = csvread('test_data.csv',1);
% Randomly selecting 5% rows
rng(1);
c=randperm(500);
c = c(1:25);
train_new = train(c,:);
[C, order, Acc] = svmcf(train_new, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)
```

```
%-------------------------------------------------------------------------
train = csvread('train_data_even_odd.csv',1);
test = csvread('test_data_even_odd.csv',1);
[C, order, Acc] = svmcf(train, test);
disp('Confusion matrix is')
C
fprintf('Percentage accuracy is %f%%\n', Acc)
fprintf('Percentage error is %f%%\n', 100-Acc)
```

- **Matlab script for execution (Question 9-10)**

```
clear;
clc;
% Data for 10 SVMs
train = csvread('train_data_0.csv',1);
%train = csvread('train_data_1.csv',1);
%train = csvread('train_data_2.csv',1);
%train = csvread('train_data_3.csv',1);
%train = csvread('train_data_4.csv',1);
%train = csvread('train_data_5.csv',1);
%train = csvread('train_data_6.csv',1);
%train = csvread('train_data_7.csv',1);
%train = csvread('train_data_8.csv',1);
%train = csvread('train_data_9.csv',1);
test = csvread('test_data_new_0_9.csv',1);

% Total number of rows and columns in training data
[nrow,ncol] = size(train);
[nrow1,ncol1] = size(test);

% Storing labels and data in different variables
x_train = train(:,[2:ncol]);
y_train = train(:,1);

% Storing labels and data in different variables
x_test = test(:,[2:ncol1]);
y_test = test(:,1);

% Radial basis function
k = rbf(x_train, x_train);

% Other matrices and vectors
H = (y_train*y_train').*k;
f = -1*ones(1, nrow);
A = [];
b = [];
Aeq = y_train';
beq = 0;
LB = zeros(1,nrow);
UB = 100*ones(1,nrow);

% function call to quadprog
alpha = quadprog(H,f,A,b,Aeq,beq,LB,UB);

% Replacing small number by 0 (alpha*)
e1 = 1e-6;
alpha(alpha < e1) = 0;
ind = find((alpha > e1)&(alpha < 100));
xi0 = x_train(ind(1),:);
yi0 = y_train(ind(1));

% Calculating bias
b = yi0 - (alpha.*y_train)'*rbf(x_train, xi0);
```

```
% Decision rule
f = (alpha.*y_train)'*rbf(x_test, x_train) + b;
pred = sign(f);

% Change this value according to digit you want to classify
% in test data
% For example -> if you want to classify 0 then pred(1:100)
% If you want to classify 7 then pred(701:800)
acc = sum(pred(1:100)==1);
acc
```

References

1) Lecture - 11 Slides, Dr. I. Griva, George Mason University
2) Learning: Support Vector Machines, Artificial Intelligence, Patrick H. Winston, MIT
3) Support Vector Machines, T.P. Runarsson and S. Sigurdsson (sven@hi.is)
4) An introduction to Support Vector Machine implementation in Matlab, David Lindsay
5) Linear SVM for two class separable data, Stephane Canu
6) https://www.mathworks.com/help/optim/ug/quadprog.html
7) https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html