Brunda Chouthoy
CSC 529: Case Study-2
Handwritten Digit recognition using SVM-kernels

Introduction

For this case study, the main objective is to apply a machine learning technique such as Support Vector Machine to classify a non-linearly separable dataset using the Kernel trick. Linear SVM cannot detect the nonlinear structures present in the dataset. Like many other kernel-based learning algorithms, SVMs work by embedding the data into a high dimensional feature space and then searching for linear relations among the embedded data points. Kernel-SVM maps the data from the input space to a feature space using a nonlinear function, the mapping is done using the kernel trick. Once the data is mapped onto the feature space (usually higher dimension than the input space), linear SVM is applied in this new mapped space.

The data consists of normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The aim of this study is to compare and assess the performance of support vector machines with RBF and polynomial kernel for classifying handwritten digits of 16*16 grayscale images. Test accuracy and misclassification error rate are used as metrics to evaluate and analyze the experimental results. Precision and recall values for each class is also examined.

Data Description and Understanding

The digits come from the handwritten ZIP codes on envelopes from U.S. postal mail. Each image is a segment from a five digit ZIP code, isolating a single digit. The images are 16×16 eight-bit grayscale maps, with each pixel raging in intensity from 0 to 255.

The original scanned digits are binary and of different sizes and orientations; the images here have been de-slanted and size normalized, resulting in 16 x 16 grayscale images. The data are in two zipped files, and each line consists of the digit id (0-9) followed by the 256 grayscale values.

There are 7291 training observations and 2007 test observations distributed as follows:

	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

As proportions:

	0	1	2	3	4	5	6	7	8	9
Train	0.16	0.14	0.1	0.09	0.09	0.08	0.09	0.09	0.07	0.09

Brunda Chouthoy
CSC 529: Case Study-2
Handwritten Digit recognition using SVM-kernels

Test	0.18	0.13	0.1	0.08	0.10	0.08	0.08	0.07	0.08	0.09

Data Source: http://statweb.stanford.edu/~tibs/ElemStatLearn/data.html

Number of instances: 7291 for training set (zip.train), 2007 for testing data set (zip.test)
Number of attributes: 256

Output variable (desired target): Y – any of the digits from 0-9
Input variables: Normalized greyscale values, 256 dimensions

Data Cleaning and Analysis:

The digits were written by many different people, using a great variety of sizes, writing styles and instruments, with widely varying amounts of care. The handwritten digit data was then linearly transformed to make the image fit in a 16*16-pixel image. Because of the linear transformation, the resulting image is not binary but has multiple gray levels, since a variable number of pixels in the original image can fall into a given pixel in the target image. The gray levels of each image are scaled and translated to fall within the range -1 to 1. Data distribution and summary for the first 20 attributes is shown in Table 1.

```
      V1              V2                V3                V4
Min.   :0.0   Min.   :-1.000   Min.   :-1.000   Min.   :-1.000
1st Qu.:1.0   1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000
Median :4.0   Median :-1.000   Median :-1.000   Median :-1.000
Mean   :3.9   Mean   :-0.996   Mean   :-0.981   Mean   :-0.951
3rd Qu.:7.0   3rd Qu.:-1.000   3rd Qu.:-1.000   3rd Qu.:-1.000
Max.   :9.0   Max.   : 0.638   Max.   : 1.000   Max.   : 1.000
      V5                V6                V7                V8
Min.   :-1.000   Min.   :-1.000   Min.   :-1.000   Min.   :-1.000
1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000
Median :-1.000   Median :-1.000   Median :-1.000   Median :-0.719
Mean   :-0.888   Mean   :-0.773   Mean   :-0.610   Mean   :-0.369
3rd Qu.:-1.000   3rd Qu.:-0.962   3rd Qu.:-0.391   3rd Qu.: 0.255
Max.   : 1.000   Max.   : 1.000   Max.   : 1.000   Max.   : 1.000
      V9               V10               V11               V12
Min.   :-1.0000   Min.   :-1.0000   Min.   :-1.000   Min.   :-1.000
1st Qu.:-0.9990   1st Qu.:-0.9500   1st Qu.:-1.000   1st Qu.:-1.000
Median : 0.0610   Median : 0.0020   Median :-0.561   Median :-0.995
Mean   :-0.0458   Mean   :-0.0526   Mean   :-0.285   Mean   :-0.504
3rd Qu.: 0.6960   3rd Qu.: 0.6745   3rd Qu.: 0.438   3rd Qu.:-0.012
Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.000   Max.   : 1.000
     V13               V14               V15               V16
Min.   :-1.000   Min.   :-1.000   Min.   :-1.000   Min.   :-1.000
1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000
Median :-1.000   Median :-1.000   Median :-1.000   Median :-1.000
Mean   :-0.686   Mean   :-0.815   Mean   :-0.906   Mean   :-0.966
3rd Qu.:-0.685   3rd Qu.:-1.000   3rd Qu.:-1.000   3rd Qu.:-1.000
Max.   : 1.000   Max.   : 1.000   Max.   : 1.000   Max.   : 1.000
     V17               V18               V19               V20
Min.   :-1.000   Min.   :-1.000   Min.   :-1.000   Min.   :-1.000
1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000   1st Qu.:-1.000
Median :-1.000   Median :-1.000   Median :-1.000   Median :-1.000
Mean   :-0.993   Mean   :-0.990   Mean   :-0.951   Mean   :-0.865
3rd Qu.:-1.000   3rd Qu.:-1.000   3rd Qu.:-1.000   3rd Qu.:-1.000
Max.   : 0.752   Max.   : 0.776   Max.   : 1.000   Max.   : 1.000
```

Table 1: Summary statistics for the train data

After initial exploration, it could be determined that the handwritten digit data is well organized and has NO missing data in the train and test data sets. (shown in Table 2).

```
> #Check if there are missing values in the Training data
> table(is.na(train_data))

  FALSE
1873787
> sapply(train_data, function(train_data) sum(is.na(train_data)))
  V1   V2   V3   V4   V5   V6   V7   V8   V9  V10  V11  V12  V13  V14  V15  V16  V17  V18  V19  V20  V21  V22
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 V23  V24  V25  V26  V27  V28  V29  V30  V31  V32  V33  V34  V35  V36  V37  V38  V39  V40  V41  V42  V43  V44
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 V45  V46  V47  V48  V49  V50  V51  V52  V53  V54  V55  V56  V57  V58  V59  V60  V61  V62  V63  V64  V65  V66
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 V67  V68  V69  V70  V71  V72  V73  V74  V75  V76  V77  V78  V79  V80  V81  V82  V83  V84  V85  V86  V87  V88
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 V89  V90  V91  V92  V93  V94  V95  V96  V97  V98  V99 V100 V101 V102 V103 V104 V105 V106 V107 V108 V109 V110
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V111 V112 V113 V114 V115 V116 V117 V118 V119 V120 V121 V122 V123 V124 V125 V126 V127 V128 V129 V130 V131 V132
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V133 V134 V135 V136 V137 V138 V139 V140 V141 V142 V143 V144 V145 V146 V147 V148 V149 V150 V151 V152 V153 V154
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V155 V156 V157 V158 V159 V160 V161 V162 V163 V164 V165 V166 V167 V168 V169 V170 V171 V172 V173 V174 V175 V176
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V177 V178 V179 V180 V181 V182 V183 V184 V185 V186 V187 V188 V189 V190 V191 V192 V193 V194 V195 V196 V197 V198
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V199 V200 V201 V202 V203 V204 V205 V206 V207 V208 V209 V210 V211 V212 V213 V214 V215 V216 V217 V218 V219 V220
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V221 V222 V223 V224 V225 V226 V227 V228 V229 V230 V231 V232 V233 V234 V235 V236 V237 V238 V239 V240 V241 V242
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
V243 V244 V245 V246 V247 V248 V249 V250 V251 V252 V253 V254 V255 V256 V257
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

Table 2:Missing data

Data Transformation:

- The class label/target variable was transformed from numeric to factor datatype (categorical) for both training and the testing datasets.
- Column names were assigned for all the attributes.

The snapshot after the transformation of the variables is shown in Table 3.

```
> #For Train Data --> Transform Label as Factor(categorical) and change column names to D.1 to D.256
> train_data[, 1] <- as.factor(train_data[, 1])  # As Category
> colnames(train_data) <- c("Y", paste("D.", 1:256, sep = ""))
> class(train_data[, 1])
[1] "factor"
> levels(train_data[, 1])
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
> ##Check datatypes of each variable after label transformation
> sapply(train_data[1, ], class)
        Y       D.1       D.2       D.3       D.4       D.5       D.6       D.7       D.8       D.9      D.10      D.11
 "factor" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.12      D.13      D.14      D.15      D.16      D.17      D.18      D.19      D.20      D.21      D.22      D.23
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.24      D.25      D.26      D.27      D.28      D.29      D.30      D.31      D.32      D.33      D.34      D.35
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.36      D.37      D.38      D.39      D.40      D.41      D.42      D.43      D.44      D.45      D.46      D.47
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.48      D.49      D.50      D.51      D.52      D.53      D.54      D.55      D.56      D.57      D.58      D.59
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.60      D.61      D.62      D.63      D.64      D.65      D.66      D.67      D.68      D.69      D.70      D.71
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.72      D.73      D.74      D.75      D.76      D.77      D.78      D.79      D.80      D.81      D.82      D.83
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
     D.84      D.85      D.86      D.87      D.88      D.89      D.90      D.91      D.92      D.93      D.94      D.95
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

Table 3: Datatype transformation for the target variable

Brunda Chouthoy
CSC 529: Case Study-2
Handwritten Digit recognition using SVM-kernels

Displaying digits of the training dataset: For exploratory analysis, I started by visualizing all the training data labels by rendering some sort of an average of each character.
An average of each digit was computed and displayed in Figure 1.

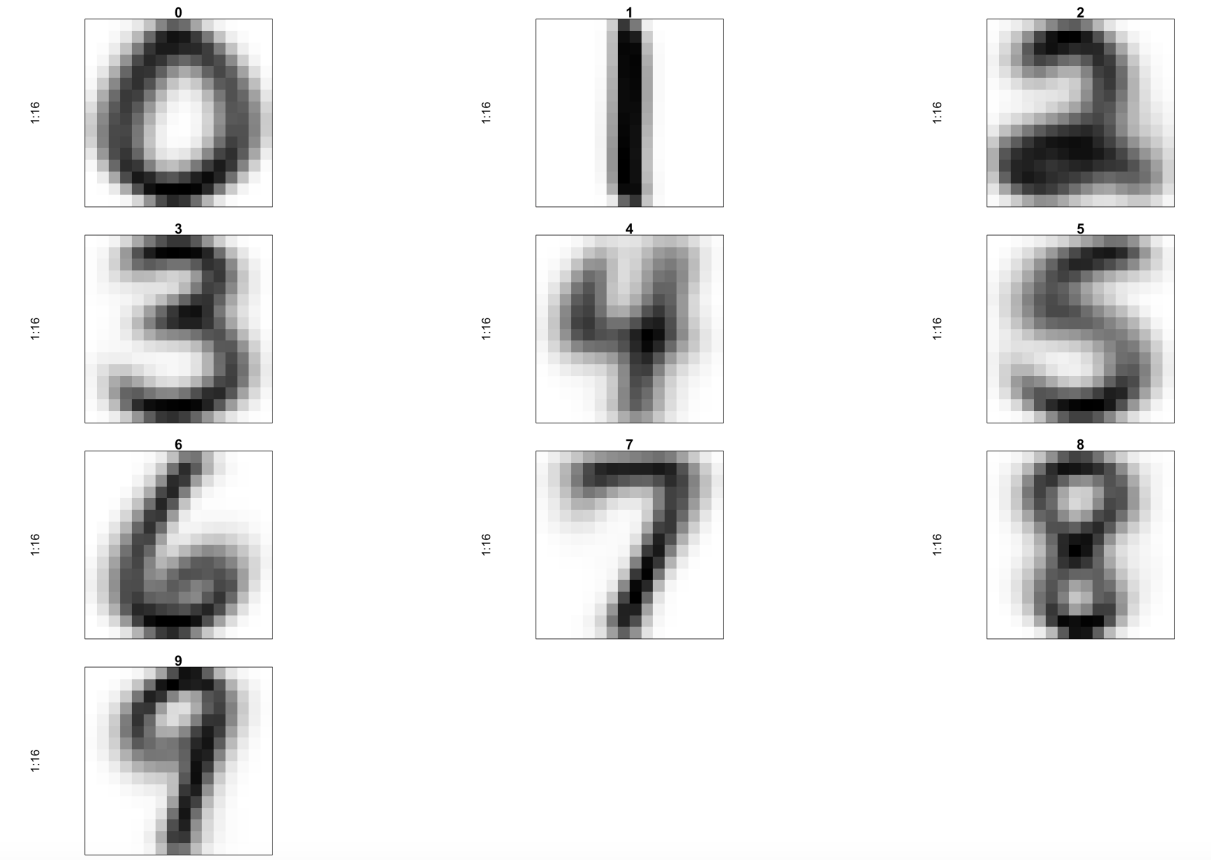


Figure 1: Display of each digit

A PDF file is also generated with digits of all the rows in the training data set to get an idea of what kind of anonymous instances should be expected for further analysis and classification – File Attached with this submission.

Analysis of the target variable:
Bar plots are used to check the total number of digits across all categories for the training and the test data set.

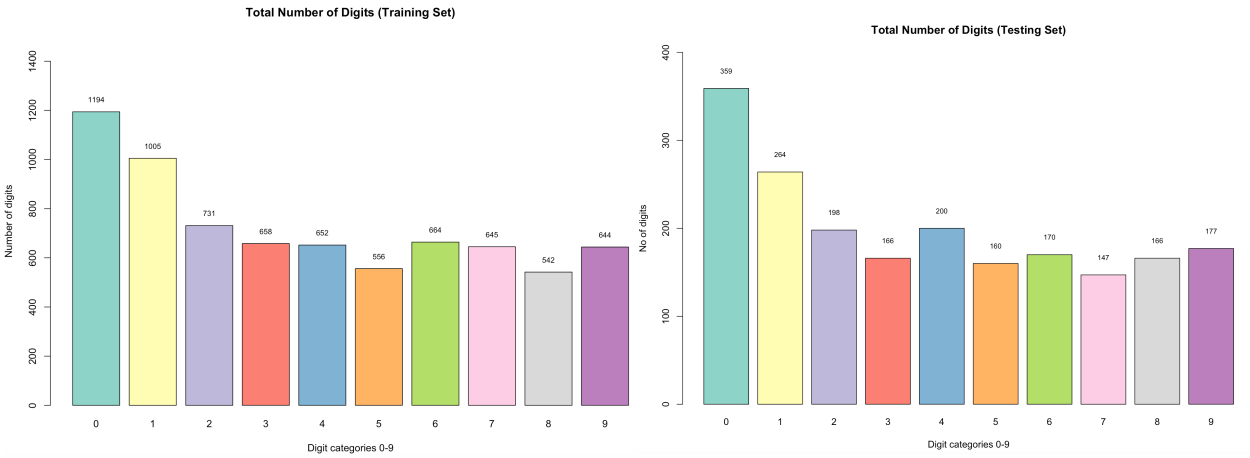


Figure 2: Bar plots to depict the total number of digits in each category

Experimental Results:

Two models are applied for recognizing and classifying handwritten digit dataset –

i. **Support Vector Machine with RBF kernel**
ii. **Support Vector Machine with polynomial kernel function**

Technical discussion:

- Linear SVM cannot detect the nonlinear structures present in the dataset.
- Kernel-SVM maps the data from the input space to a feature space using a nonlinear function, the mapping is done using the kernel trick. Once the data is mapped onto the feature space (usually higher dimension than the input space), linear SVM is applied in this new mapped space.
- Performance of the SVM algorithm with different kernel functions depends on the choice of parameters. The optimal selection of these parameters is a nontrivial issue.
- After training SVM, the best value C and g can be used to classify children with handwriting problems.
- To use the RBF kernel with SVM, we need to find parameter C (cost) and g (gamma).
- To use the polynomial kernel function with SVM, we need to determine the degree of the polynomial and co-efficient0.
- The penalty factor C – the cost for constraints violation, is used to improve generalized capability when C is increasing while gamma and degree are the adjustable parameters of study in the experiment and they are used to adjust experienced error value.
- The tuning of parameters is important to understand the influence of these parameters, because the accuracy of an SVM model is largely dependent on the selection them. For example, if C is too large, we have a high penalty for non-

separable points and we may store many support vectors and over fit the data. However, if the C parameter is too small, it might result in under fitting the data.

Results: **Support Vector Machine with RBF kernel**

- The tune.svm() function in R is used to do a grid search over the supplied parameter ranges (C cost, gamma), using the train data set. The range of gamma parameter is between 0.000001 and 0.1. For cost parameter, the range is from 0.1 until 10.
- The parameters gamma and cost were tuned with several different values manually and with the tune function in R. The best values for the handwritten digits' dataset – gamma: 0.01 and cost:5
- Summary of the model is shown in Figure 3. Number of support vectors: 1767

```
> pc <- proc.time()
> model.svm <- svm(Y ~ ., kernel = "radial", method = "class", data = train_data, gamma = 0.001, cost = 10)
> proc.time() - pc
   user  system elapsed
 14.059   0.228  14.336
> summary(model.svm)

Call:
svm(formula = Y ~ ., data = train_data, kernel = "radial", method = "class", gamma = 0.001, cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.001

Number of Support Vectors:  1767

 ( 148 235 227 147 193 46 156 201 236 178 )


Number of Classes:  10

Levels:
 0 1 2 3 4 5 6 7 8 9
```

Figure 3: Summary of SVM with RBF kernel

Brunda Chouthoy
CSC 529: Case Study-2
Handwritten Digit recognition using SVM-kernels

Confusion matrix and test Accuracy for RBF kernel:

```
> predict.svm <- predict(model.svm, newdata = test_data, type = "class")
> table(`Actual Class` = test_data$Y, `Predicted Class` = predict.svm)
            Predicted Class
Actual Class   0   1   2   3   4   5   6   7   8   9
           0 352   0   2   0   3   0   0   0   2   0
           1   0 257   0   0   4   0   3   0   0   0
           2   0   0 183   5   4   2   0   1   3   0
           3   1   0   3 150   0   8   0   1   3   0
           4   0   1   4   0 186   2   2   1   0   4
           5   4   0   0   5   2 146   0   0   2   1
           6   0   0   3   0   3   1 163   0   0   0
           7   0   0   1   0   6   1   0 137   0   2
           8   4   0   0   2   0   4   0   0 153   3
           9   0   0   0   0   5   1   0   1   0 170
>
> error.rate.svm <- sum(test_data$Y != predict.svm)/nrow(test_data)
> print(paste0("Accuary (Precision): ", 1 - error.rate.svm))
[1] "Accuary (Precision): 0.9451918285999"
```

Test Accuracy of the model: **94.57**
Total number of misclassified digits (Error numbers): **110**
Misclassification error rate on the test data set: **0.054 or 5.43%**

Recall and Precision values for each class using the polynomial kernel:

```
> #Precision for each class - RBF kernel
> precision<-(precision <- diag(mat.radial) / rowSums(mat.radial))
> print(paste0("Precision: ", precision))
 [1] "Precision: 0.98050139275766"  "Precision: 0.973484848484849" "Precision: 0.924242424242424"
 [4] "Precision: 0.903614457831325" "Precision: 0.93"              "Precision: 0.9125"
 [7] "Precision: 0.958823529411765" "Precision: 0.931972789115646" "Precision: 0.921686746987952"
[10] "Precision: 0.96045197740113"
>
> #Recall for each class
> recall <- (diag(mat.radial) / colSums(mat.radial))
> print(paste0("Recall: ", recall))
 [1] "Recall: 0.975069252077562" "Recall: 0.996124031007752" "Recall: 0.933673469387755" "Recall: 0.925925925925926"
 [5] "Recall: 0.873239436619718" "Recall: 0.884848484848485" "Recall: 0.970238095238095" "Recall: 0.971631205673759"
 [9] "Recall: 0.938650306748466" "Recall: 0.944444444444444"
>
```

Some examples of the misclassified cases on test data:

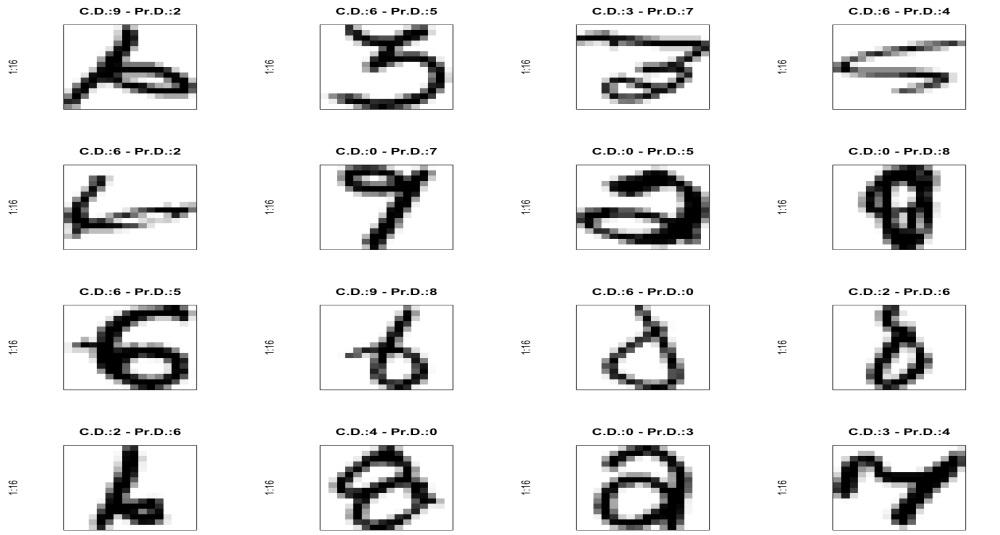


Figure 4: Misclassified digits on the test data

Support Vector Machine with Polynomial kernel

- The tune.svm() function in R is used to do a grid search over the supplied parameter ranges for the degree and coef0 parameters using the training data set. The range of degree parameter is between 1 and 4. For coef0 parameter, the range is from -1 until 4.
- The parameters degree and coef0 for the polynomial kernel were tuned with several different values manually and with the tune function in R. The best values for the handwritten digits' dataset – degree: 3 and coef0: 1
- Summary of the model is shown in Figure 5. Number of support vectors: 1891

```
> pc <- proc.time()
> model.svm.polynomial <- svm(Y ~ ., method = "class", data = train_data, kernel = 'polynomial', degree = 3,coef0 = 1)
> proc.time() - pc
   user  system elapsed
 16.053   0.406  16.584
> summary(model.svm.polynomial)

Call:
svm(formula = Y ~ ., data = train_data, method = "class", kernel = "polynomial", degree = 3, coef0 = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  0.00390625
     coef.0:  1

Number of Support Vectors:  1891

 ( 166 240 251 152 210 51 159 219 253 190 )


Number of Classes:  10

Levels:
 0 1 2 3 4 5 6 7 8 9
```

Figure 5:Summary of the model with Polynomial kernel

Brunda Chouthoy
CSC 529: Case Study-2
Handwritten Digit recognition using SVM-kernels

Confusion matrix and test Accuracy for the polynomial kernel:

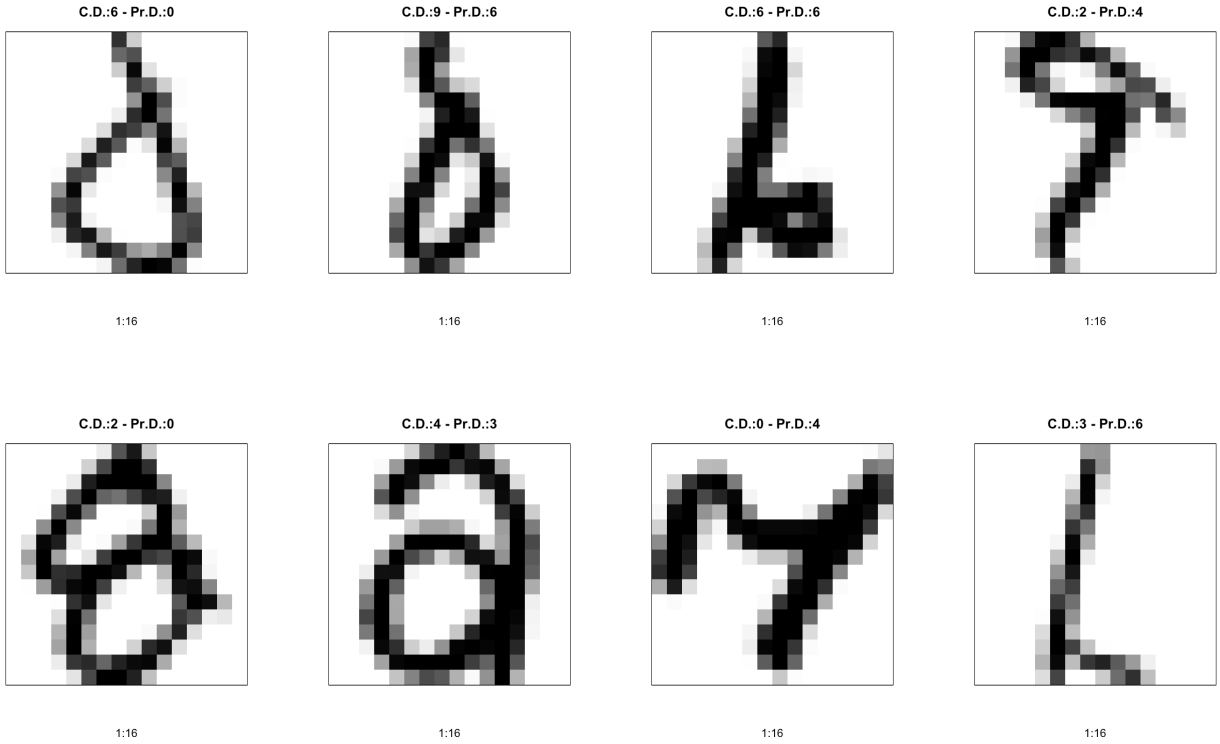
```
> ##Confusion Matrix for SVM - polynomial kernel
> predict.svm.poly <- predict(model.svm.polynomial, newdata = test_data, type = "class")
> matrix<-table(`Actual Class` = test_data$Y, `Predicted Class` = predict.svm.poly)
> matrix
            Predicted Class
Actual Class   0   1   2   3   4   5   6   7   8   9
           0 352   0   2   0   2   1   1   0   1   0
           1   0 257   0   0   4   0   3   0   0   0
           2   0   0 185   3   2   1   0   1   6   0
           3   1   0   2 151   0  10   0   1   1   0
           4   0   1   3   0 190   1   1   1   0   3
           5   2   0   0   3   1 149   0   0   3   2
           6   0   1   2   0   1   1 164   0   1   0
           7   0   0   1   0  10   1   0 134   0   1
           8   4   0   0   2   0   2   1   0 155   2
           9   0   0   0   0   5   1   0   0   1 170
> error.rate.svm <- sum(test_data$Y != predict.svm.poly)/nrow(test_data)
> print(paste0("Accuary (Precision): ", 1 - error.rate.svm))
[1] "Accuary (Precision): 0.950174389636273"
```

Test Accuracy of the model**: 0.95017 or 95.017%**
Total number of misclassified digits (Error numbers): **100**
Misclassification error rate on the test data set: **0.049 or 4.9%**

Recall and Precision values for each class using the polynomial kernel:

```
> #Precision for each class
> precision<-(precision <- diag(matrix) / rowSums(matrix))
> print(paste0("Precision: ", precision))
 [1] "Precision: 0.98050139275766"  "Precision: 0.973484848484849" "Precision: 0.934343434343434"
 [4] "Precision: 0.909638554216867" "Precision: 0.95"              "Precision: 0.93125"
 [7] "Precision: 0.964705882352941" "Precision: 0.91156462585034"  "Precision: 0.933734939759036"
[10] "Precision: 0.96045197740113"
>
> #Recall for each class
> recall <- (diag(matrix) / colSums(matrix))
> print(paste0("Recall: ", recall))
 [1] "Recall: 0.98050139275766"  "Recall: 0.992277992277992" "Recall: 0.948717948717949" "Recall: 0.949685534591195"
 [5] "Recall: 0.883720930232558" "Recall: 0.892215568862275" "Recall: 0.964705882352941" "Recall: 0.978102189781022"
 [9] "Recall: 0.922619047619048" "Recall: 0.955056179775281"
>
```

Some examples of the misclassified cases on test data:



Experimental Analysis:

Table 12 shows the results and comparison consolidated from both the models.

Testing Data	Accuracy	Number of misclassified digits	Misclassification Error rate
1. SVM with RBF kernel	0.9457 or 94.57%	110	0.054 or 5.43%
2. SVM with Polynomial kernel	0.9501 or 95.01%	100	0.049 or 4.9%

Table 4: Performance comparison

Table 5 and Table 6 depict the precision and recall values for both the models:

SVM with RBF kernel	Precision	Recall
Class 0:	0.9805	0.975
Class 1:	0.9734	0.9961
Class 2:	0.9242	0.9336
Class 3:	0.9036	0.9259

Class 4:	0.93	0.8732
Class 5:	0.9125	0.8848
Class 6:	0.9588	0.9702
Class 7:	0.9319	0.9716
Class 8:	0.9216	0.9386
Class 9:	0.9604	0.9444

Table 5:Precision and Recall values for the SVM model using RBF kernel

SVM with polynomial kernel	Precision	Recall
Class 0:	0.9805	0.9805
Class 1:	0.9734	0.9922
Class 2:	0.9343	0.9487
Class 3:	0.9096	0.9496
Class 4:	0.95	0.8837
Class 5:	0.9312	0.8922
Class 6:	0.9647	0.9647
Class 7:	0.9115	0.9781
Class 8:	0.9337	0.9226
Class 9:	0.9604	0.955

Table 6: Precision and recall values for the SVM model using polynomial kernel

- As per the results from the table, the accuracy of the models range from 94.57%-95.01% for the testing dataset. Precision and Recall for each class along with misclassification error rate (rejection of genuine category) and accuracy as evaluation metrics for the model are used to compare the two SVM models.
- The total number of misclassified digits or error digits were 110 for the model with RBF kernel and 100 for the model with polynomial kernel which is a significant improvement. The misclassification error rate for the polynomial kernel is 0.049 or 4.9% which is better than the one with RBF kernel.
- For the model with RBF kernel, the recall or the True positive rate was highest for class 1 meaning 99.61% of the test data that belonged to Class 1 was predicted correctly and TPR for Class 4 was the lowest meaning only 87% of the cases were correctly predicted.
- Similarly, for the model with polynomial kernel, the recall or the True positive rate(TPR) was highest for class 1 meaning 99.22% of the test data that belonged to Class 1 was predicted correctly and TPR for Class 4 was the lowest meaning only 88.4% of the cases were correctly predicted.
- Furthermore, Precision values for Class 0 was the highest and for Class 3 was the lowest for both the model variations. Precision is the probability that a (randomly selected) retrieved digit is relevant.

- From the above analysis, it can be established that the performance of SVM with polynomial kernel is better than the one with RBF kernel for recognizing handwritten digit data by USPS.

Conclusion:

- In this project, R code was developed to recognize handwritten digits with kernel SVM technique. Results obtained prove that the classification using kernel SVM was achieved.
- Kernel-SVM maps the data from the input space to a feature space using a nonlinear function, the mapping is done using the kernel trick. Once the data is mapped onto the feature space (usually higher dimension than the input space), linear SVM is applied in this new mapped space.
- The data consists of normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The ZIP CODE data by the USPS was already split into training and test data sets. Number of instances: 7291 for training set and 2007 instances for testing data set.
- The data consists of images that are 16×16 eight-bit grayscale maps, with each pixel ranging in intensity from 0 to 255. There is a total of 256 dimensions/predictor variables and a class variable consisting one of the digits 0-9.
- The aim of this study is to compare and assess the performance of support vector machines with RBF and polynomial kernel for classifying handwritten digits of 16*16 grayscale images.
- After initial analysis, it could be determined that that the bank telemarketing data is well organized and has NO missing data and many different variables were examined as a part of the exploratory analysis of data using visualizations.
- The class label/target variable was transformed from numeric to factor datatype (categorical) for both training and the testing datasets.
- The optimal selection of the parameters for both the models is a nontrivial issue and that was done using the tune.svm function in R.
- Experimental results were then analyzed considering different performance metrics, the accuracy of the models range from 94.57%-95.01% for the testing dataset. Precision and Recall for each class along with misclassification rate and accuracy for the model are used to compare the two SVM models.
- The experimentation gave a better insight to using machine learning algorithms and kernel trick in digit recognition domain.
- It can be testified that the performance of SVM with polynomial kernel is better than the one with RBF kernel for recognizing handwritten digit data by USPS.