STA 561 Final Project: Handwritten Digits Recognition

Minjung Park, Yue(April) Mu

Introduction

In this final project, our goal is to classify images of handwritten single digits using the Support Vector Machine(SVM) method. SVM was originally designed and used for binary classification. However, it is also a great tool for multi-class classification, novelty detection, and regression. In our project, we will be using packages 'e1071' and 'kernlab' for SVMs with Gaussian kernel function. We will test the accuracy of such method and present the results through visualization.

Data

The data we will be using comes from the Kaggle competition(link: https://www.kaggle.com/c/digit-recognizer). It was orginally taken from the Modified National Institute of Standards and Technology (MNIST) database, which is a classic within the Machine Learning community. This dataset contains a total of 70000 observations split into training and testing datasets. Each observation is a 28 * 28 = 784 pixels image, and each pixel contains a grey-scale value from 0 to 255. The only difference between training and testing dataset is that the training data contains a label column with values of 0-9, which indicates the actual handwritten digit.

Initial exploration of the data

We first read the data into R with each row representing a handwritten digit. For the training data, it contains a label with a value 0-9, which indicates the actual handwritten digits.

```
library(e1071)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.3
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 3.1.3
```

```
#Read training and testing datasets into R
train <- read.csv("train.csv",header=T)
test <- read.csv("test.csv",header=T)
```

As we mentioned above, this is a classification problem. Hence the label column should be treated as category variable. We used the 'as.factor' function in order to convert the label value from numeric to factor.

```
#Convert predicted column to factor
train$label <- as.factor(train$label)
```

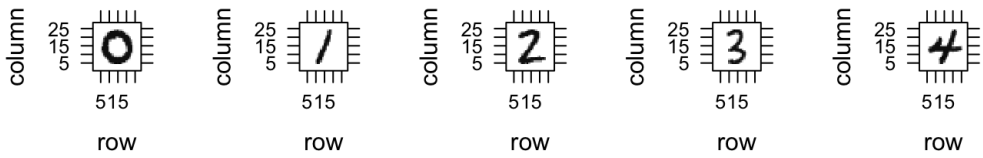
By mapping the 784 grey-scale values back into a 28 * 28 matrix, we can then recreate the digit images from the data by plotting the matrix. Here we pick a sample of 10 digits from the training dataset and plot them.

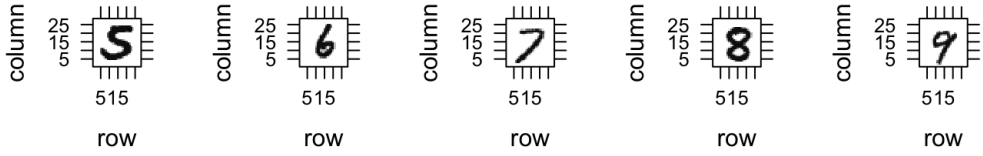
```
library(lattice)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.1.3
```

```
#Plot sample digit from 0 to 9 to see what they look like
num <- c(2,39,23,66,82,100,161,173,215,87)
grid1 <- rep(list(list()),10)
for (i in 1:10){
  digit <- matrix(train[num[i],2:785],nrow=28,ncol=28)
  digit <- digit[,nrow(digit):1]
```

```
  grid2 <- levelplot(digit,col.regions = grey(seq(1, 0, length = 256)),colorkey=FALSE)
  grid1[[i]] <- grid2
}
grid.arrange(grid1[[1]],grid1[[2]],grid1[[3]],grid1[[4]],
             grid1[[5]],grid1[[6]],grid1[[7]],grid1[[8]],
             grid1[[9]],grid1[[10]],nrow=2,ncol=5)
```





SVM Classification and Testing for Accuracy

In order to assess the accuracy of the model, we decided to use the K-fold cross-validation approach. In K-fold cross-validation, the original dataset is randomly partitioned into K subsets of equal size. One of the K subsets is retained as the validation data for testing the model, and the remaining K-1 subsets are used as training data. This cross-validation process is then repeated for K times, with each of the K subsets used exactly once as the validation data. For this dataset, we will implement cross validation on a subset of the training data with 5000 observations and K = 5 folds.

```
#Take the first 5000 observations from the training data
cv.train <- train[1:5000,]

#Create 5 folds
n <- 5
folds <- createFolds(cv.train$label , k = n, list = TRUE, returnTrain = FALSE)
```

We use the ‘ksvm’ function from the ‘kernlab’ to build the svm model. As the label column was transformed into a factor, the SVM model will use ‘C-SVM classification’ as the default type with Gaussian Kernel Function. The ‘predict’ function would be then used in order to predict the labels based on the testing set.

In order to obtain the accuracy of the prediction, the ‘confusionMatrix’ function from the ‘caret’ package would be used as well.

```
accuracy <- numeric(5)
for ( i in 1:length(folds)){

    #Split data into train/test sets
    cat("Creating training and testing data sets for fold num: ", i, "\n")
    train.dat <- cv.train[as.numeric(unlist(folds[-i])),]
    test.dat <- cv.train[as.numeric(unlist(folds[i])),]

    #Build SVM model
    cat("Buliding model for fold num: ", i, "\n")
    fit.svm <- ksvm(label~., data=train.dat, kernel="rbfdot")

    #Prediction for test data
    cat("Making prediction for fold num: ", i, "\n")
```

```
    predict.svm <- predict(fit.svm,test.dat)

    #Calculate accuracy
    cat("Calculating accuracy for fold num: ", i, "\n")
    tmp <- confusionMatrix(test.dat$label, predict.svm)
    overall <- tmp$overall
    cat(overall, "\n")
    overall.accuracy <- overall['Accuracy']
    cat(overall.accuracy, "\n")
    accuracy[i] <- as.numeric(tmp[[3]][1])
}
```

```
## Creating training and testing data sets for fold num:  1
## Buliding model for fold num:  1
## Making prediction for fold num:  1
## Calculating accuracy for fold num:  1
## 0.9419419 0.9354684 0.9255897 0.9556225 0.1121121 0 NaN
## 0.9419419
## Creating training and testing data sets for fold num:  2
## Buliding model for fold num:  2
## Making prediction for fold num:  2
## Calculating accuracy for fold num:  2
## 0.952953 0.9477051 0.9379258 0.96523 0.1151151 0 NaN
## 0.952953
## Creating training and testing data sets for fold num:  3
## Buliding model for fold num:  3
## Making prediction for fold num:  3
## Calculating accuracy for fold num:  3
## 0.9391825 0.9323955 0.9225595 0.9531636 0.114656 0 NaN
## 0.9391825
## Creating training and testing data sets for fold num:  4
## Buliding model for fold num:  4
## Making prediction for fold num:  4
## Calculating accuracy for fold num:  4
## 0.953047 0.9478058 0.9380488 0.9652998 0.1128871 0 NaN
## 0.953047
## Creating training and testing data sets for fold num:  5
## Buliding model for fold num:  5
## Making prediction for fold num:  5
## Calculating accuracy for fold num:  5
## 0.9428858 0.9365182 0.9266318 0.9564588 0.1092184 0 NaN
## 0.9428858
```

There may be some warning messages because there are many columns with zero value, meaning that the variable is constant, hence cannot be scaled to unit variance (and zero mean). However, it is not effecting the model.

```
accuracy
```

```
## [1] 0.9419419 0.9529530 0.9391825 0.9530470 0.9428858
```

```
paste("Avg accuracy: ",round(mean(accuracy)*100,1),"%",sep="")
```

```
## [1] "Avg accuracy: 94.6%"
```

The average accuracy we got from the K-fold cross validation was about 94%.

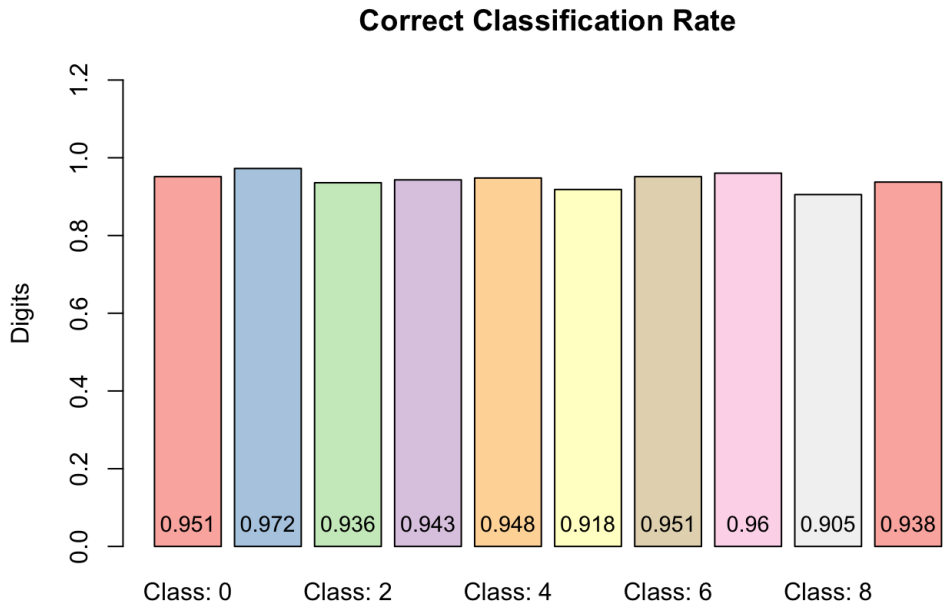
Visualization

```
library(grDevices)
library(RColorBrewer)
table <- as.matrix(tmp$table)
table
```

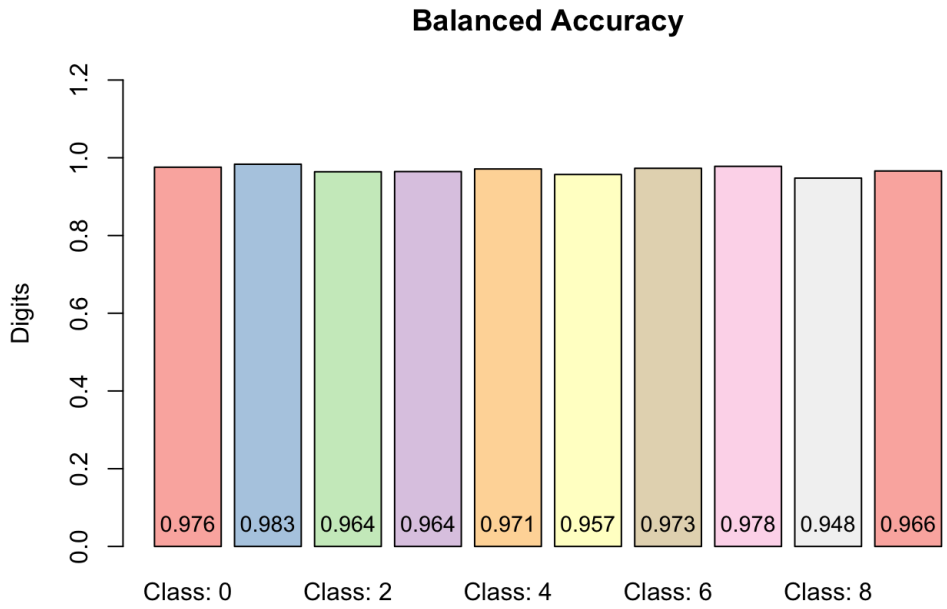
```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  98   0   0   0   0   0   0   0   0   0
##          1   0 106   2   0   0   1   1   0   1   0
##          2   2   0 102   0   3   0   0   1   1   0
##          3   2   0   0  83   0   5   0   0   5   1
##          4   0   0   0   0  91   0   1   0   0   4
##          5   0   0   1   1   0  90   2   0   0   0
##          6   0   0   2   0   2   0  98   0   1   0
##          7   1   2   0   1   0   0   0  97   0   0
##          8   0   1   2   2   0   2   1   0  86   1
##          9   0   0   0   1   0   0   0   3   1  90
```

In the table above, the columns represent the actual digits, and the rows represent predictions. Hence, the diagonal values are the numbers of correct predictions, and the off-diagonal values are the numbers of misclassifaction.

```
byclass <- as.matrix(tmp$byClass)
colors <- colorRampPalette(brewer.pal(9, "Pastel1"))
plot1 <- barplot(byclass[,1], col = colors(9), main = "Correct Classification Rate",
    ylab = "Digits",ylim=c(0,1.2))
text(plot1,0,labels = round(byclass[,1],3), cex=0.9,pos=3)
```




```
plot2 <- barplot(byclass[,8], col = colors(9), main = "Balanced Accuracy",
    ylab = "Digits",ylim=c(0,1.2))
text(plot2,0,labels = round(byclass[,8],3), cex=0.9,pos=3)
```



When looking at the Sensitivity Rate (True Positive Rate), it seems that the model was performing a bit better for some specific classes. However, for some other classes, the True Positive Rate is a bit lower.

We created another bar plot for the Balanced Accuracy, which is another useful performance measure that avoids inflated performance estimates on imbalanced datasets. After averaging sensitivity and specificity, we can see from the plot that the predictive accuracy accross the 10 digits are more similar.

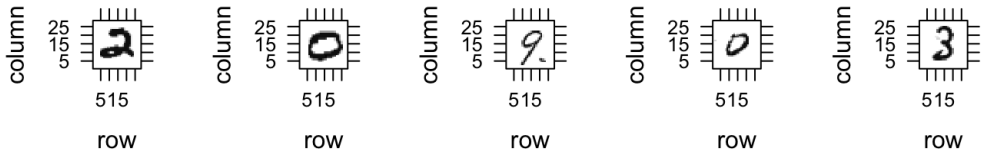
Prediction

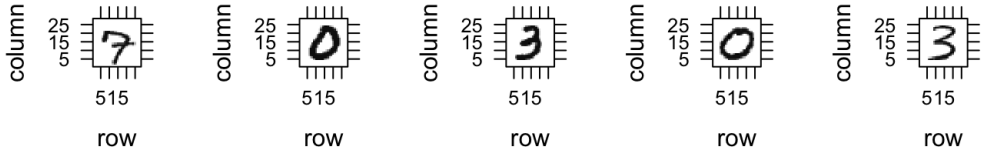
With an overall accuracy of 94%, we can then train the model on the full training dataset and then predict the label based on the full testing dataset. To shorten the time it takes to run the process, we will only use the first 10000 observations of both datasets.

```
fit.svm2 <- ksvm(label~., data=train[1:10000,], kernel="rbfdot")

## predict labels for testing data
predict.svm2 <- predict(fit.svm2,test[1:10000,])

## merge predictions for one data frame
prediction <- data.frame(label=predict.svm2,test[1:10000,])
```

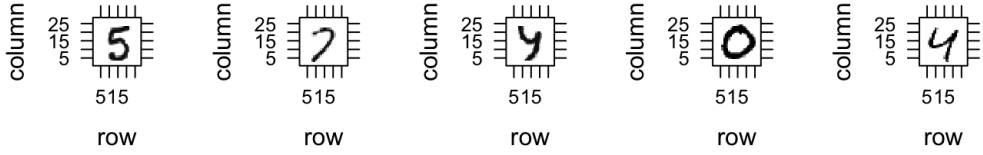
Since the test data do not have labels, it is not easy to verify the accuracy. However, we can still visualize the test data and compare them with our predictions. We cannot do this for all observations, but we can take a sample of 20 observations and check for accuracy. In this sample, except that the 4th digit '0' is misread as '9', the rest 19 predictions are correct, which roughly gives a 95% accuracy. This result is consistent with the average accuracy we calculated previously.

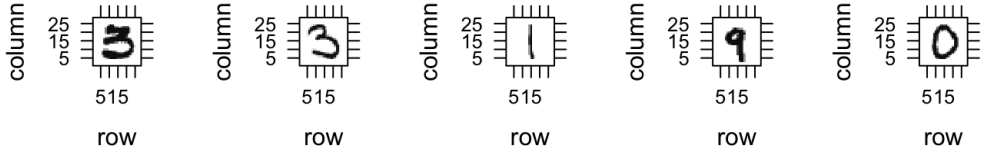
```
num=seq(1:20)
grid1=rep(list(list()),20)
for (i in 1:20){
  digit=matrix(prediction[num[i],2:785],nrow=28,ncol=28)
  digit=digit[,nrow(digit):1]
  grid2=levelplot(digit,col.regions = grey(seq(1, 0, length = 256)),colorkey=FALSE)
  grid1[[i]]=grid2
}
grid.arrange(grid1[[1]],grid1[[2]],grid1[[3]],grid1[[4]],
             grid1[[5]],grid1[[6]],grid1[[7]],grid1[[8]],
             grid1[[9]],grid1[[10]],nrow=2,ncol=5)
```






```
grid.arrange(grid1[[11]],grid1[[12]],
             grid1[[13]],grid1[[14]],grid1[[15]],grid1[[16]],
             grid1[[17]],grid1[[18]],grid1[[19]],grid1[[20]],nrow=2,ncol=5)
```






```
as.character(prediction[1:10,1])
```

```
##  [1] "2" "0" "9" "9" "3" "7" "0" "3" "0" "3"
```

```
as.character(prediction[11:20,1])
```

```
##  [1] "5" "7" "4" "0" "4" "3" "3" "1" "9" "0"
```