Devyn Hughes HW 6/7

2024-03-02

Introduction

Machine Learning is a powerful tool that helps in various ways to promote better decisions in a more efficient and streamlined manner. But, how far can one go with the use of machine learning? There are a various number of tools and models that have been created to all try and find the “best fit” for any particular scenario. Some differ in approach than others while some of them change regarding the simple outputs and inputs into them. For this particular report we will be looking into a popular data set that is known as the Kaggle Digit Recognizer. This set presents 9 digits as the labels for thousands of hand written images. Each of the rows represent a handwritten letter while the columns define the pixels of number. With this data, we will briefly define each of the different methods on how we are going to model an algorithm to see which one best suits depicting this data set. We will be using a Decision Tree, PCA Classification, Naive Bayes, kNN, SVM, and Random Tree.

A Decision Tree model is a non-parametric method of supervised learning. This particular model can handle classifications and regressions. Also, this model is the basis of a more complicated model known as a Random Tree Model.

A Principal Component Analysis Classification is a pre-processing step used in conjunction with other algorithms. It is a way to reduce the dimensionality of a data set to provide easier and/or faster processing. Att he same time, we can potentially reduce noise and extract inappropriate features as well. For this project, we will be implementing this tactic with a Decision Tree model to see if there is a higher performance.

A Naive Bayes model is a type of machine learning that is based on the populare theorem known as Bayes Theorem. It is a probabilistic form of learning that is used for classification tasks. It takes into consideration that all the feautres are independent of each other given the label you are classifying.

A k-Nearest Neighbors model is a type of model that operates on instance or memory based learning algorithms. It evaluates the distance and finds the “nearest neighbors” to set an averaging for classification.

A Support Vector machine Model is a form of machine learning that searches for the best hyperplane that separates different classes of the feature space.This can be altered by determining the dimension space, margin, and etc.

A Random Forest Tree is a type of classification and regression model that operates by forming a multitude of decision trees and outputting the class that is the average or mode of the prediction.

With all of these different forms of machine learning, we can use each of these powerful tools to try and receive the best results possible for this experiment. Ultimately, we are looking to make models that are constructed effectively, have convincing outputs, a sufficient amount of information and details to repeat, determine if anything is irrelevant content, and who has the best accuracy!

Download Libraries

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

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

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: rpart
```

```
## naivebayes 0.9.7 loaded
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Loading the Data

```
traindata = read.csv("C:/Users/Devs PC/OneDrive/Documents/IST 707/train.csv",header = TRUE,strin
gsAsFactors = TRUE)
testdata = read.csv("C:/Users/Devs PC/OneDrive/Documents/IST 707/test.csv",header = TRUE,strings
AsFactors = TRUE)
```

Change the Data and Create a Sample

```
#Set Seed for Processs
set.seed(534)

#Create New Data Index - Regulare
index = createDataPartition(traindata$label, p = .1, list = FALSE, times = 1)

#View Dimension of New_data
new_data = traindata[index,]
dim(new_data)
```

```
## [1] 4203  785
```

```
#Create Train/Test of New_Data
index1 = createDataPartition(new_data$label, p = 0.8, list = FALSE, times = 1)
train_new = new_data[index1,]
test_new = new_data[-index1,]

#Dimensions of the Train/Test Set
dim(train_new)
```

```
## [1] 3364  785
```

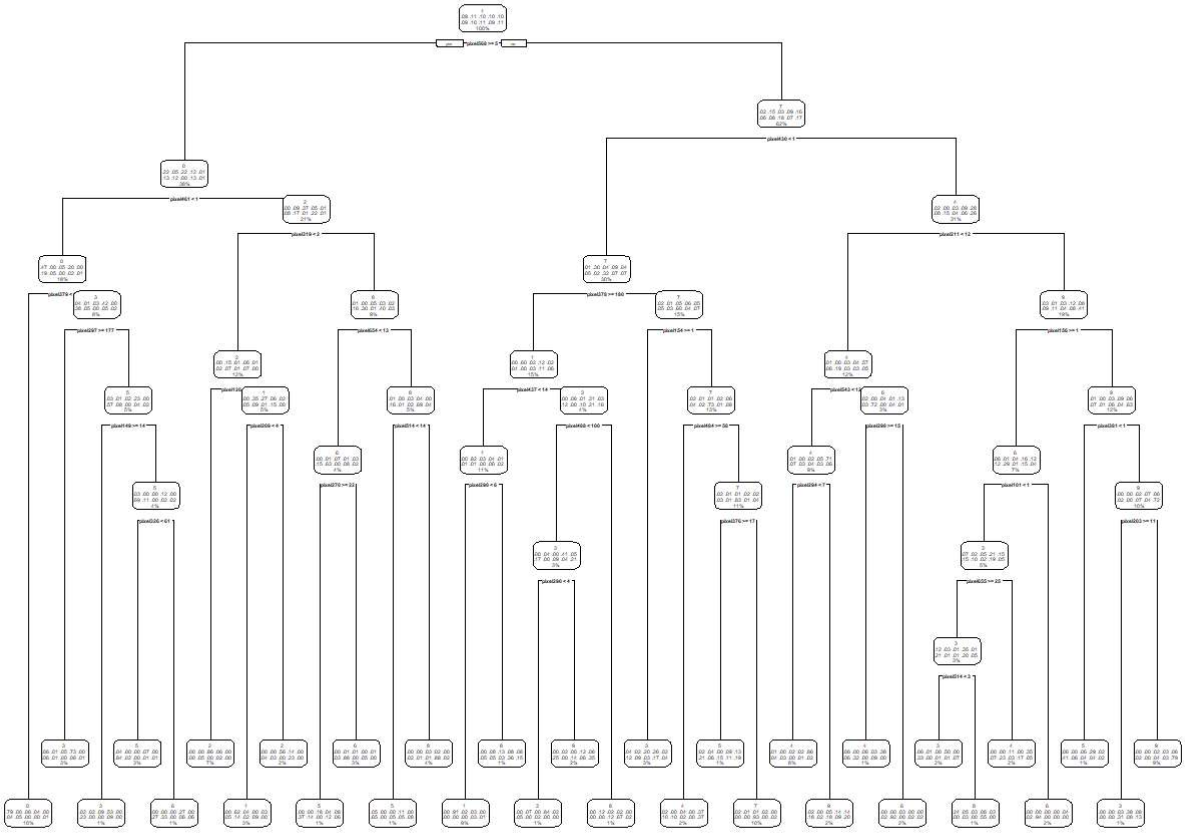
```
dim(test_new)
```

```
## [1] 839 785
```

Create the Model for Experiment

```
#Model 1
model1 = rpart(label~., data = train_new,method = "class", control = rpart.control(cp = 0, minsp
lit = 100, maxdepth = 20))
rpart.plot(model1,box.palette = 0)
```

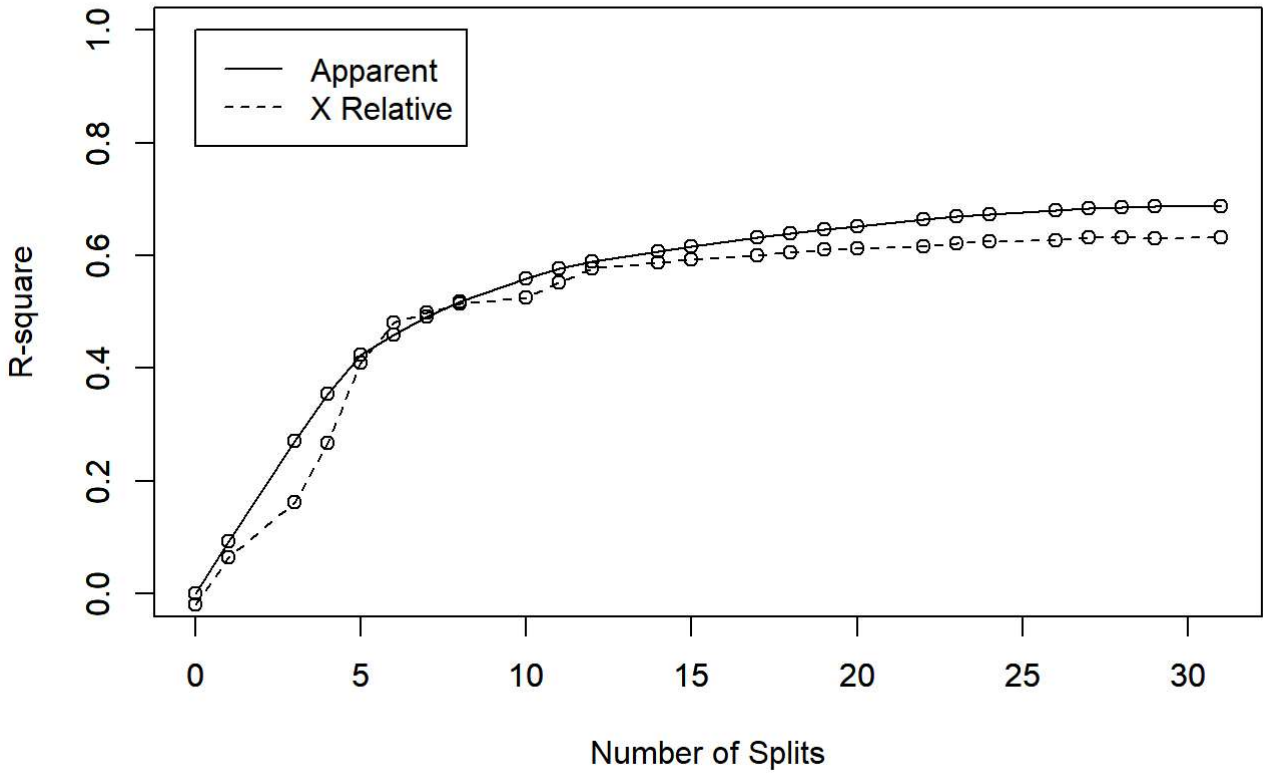
```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

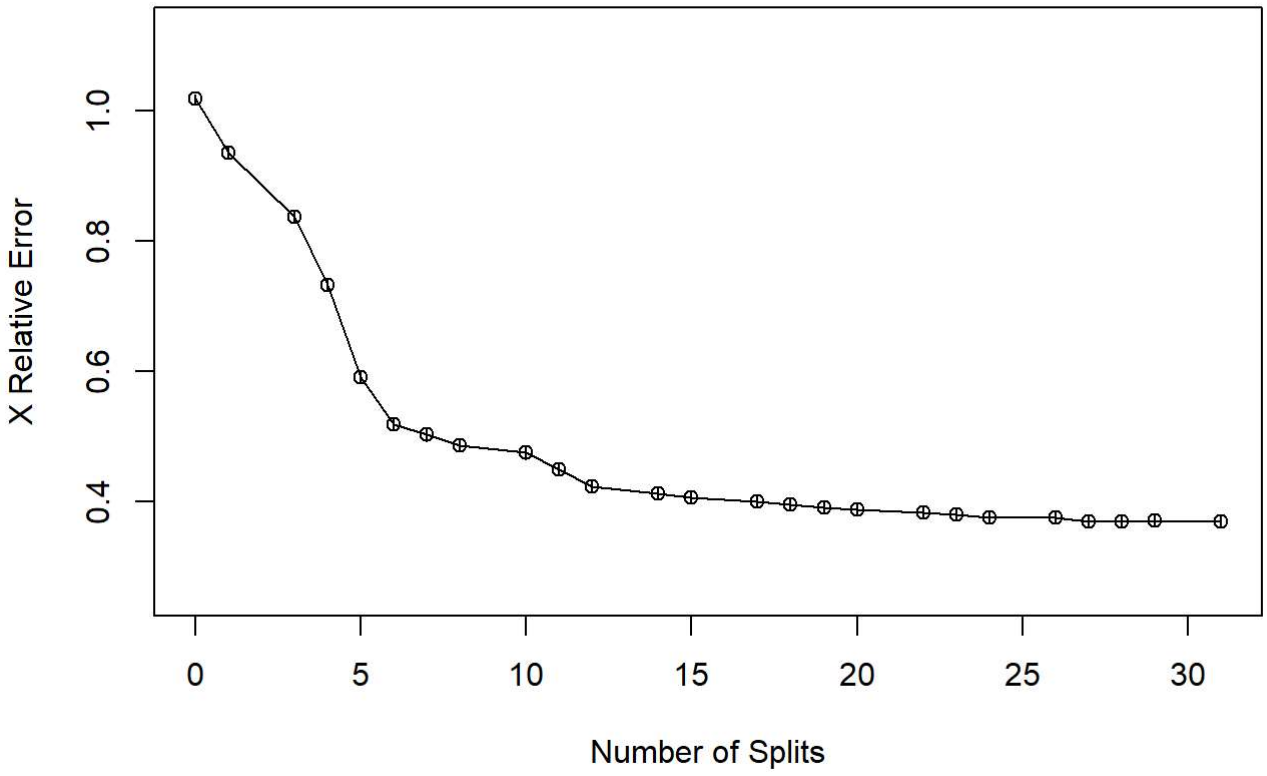


```
rsq.rpart(model1)
```

```
##
## Classification tree:
## rpart(formula = label ~ ., data = train_new, method = "class",
##     control = rpart.control(cp = 0, minsplit = 100, maxdepth = 20))
##
## Variables actually used in tree construction:
##  [1] pixel101 pixel126 pixel149 pixel154 pixel156 pixel203 pixel209 pixel211
##  [9] pixel270 pixel290 pixel294 pixel297 pixel298 pixel319 pixel326 pixel376
## [17] pixel378 pixel379 pixel381 pixel430 pixel437 pixel461 pixel484 pixel488
## [25] pixel514 pixel543 pixel568 pixel654 pixel655
##
## Root node error: 2990/3364 = 0.88882
##
## n= 3364
##
##           CP nsplit rel error  xerror      xstd
## 1  0.0923077      0   1.00000 1.01873 0.0056752
## 2  0.0896321      1   0.90769 0.93545 0.0072617
## 3  0.0836120      3   0.72843 0.83746 0.0084619
## 4  0.0688963      4   0.64482 0.73211 0.0092479
## 5  0.0354515      5   0.57592 0.59097 0.0096866
## 6  0.0327759      6   0.54047 0.51839 0.0096691
## 7  0.0267559      7   0.50769 0.50201 0.0096427
## 8  0.0197324      8   0.48094 0.48562 0.0096079
## 9  0.0187291     10   0.44147 0.47425 0.0095788
## 10 0.0123746     11   0.42274 0.44849 0.0094976
## 11 0.0088629     12   0.41037 0.42207 0.0093918
## 12 0.0086957     14   0.39264 0.41171 0.0093438
## 13 0.0076923     15   0.38395 0.40602 0.0093160
## 14 0.0073579     17   0.36856 0.39900 0.0092801
## 15 0.0070234     18   0.36120 0.39465 0.0092570
## 16 0.0063545     19   0.35418 0.38997 0.0092313
## 17 0.0060201     20   0.34783 0.38729 0.0092163
## 18 0.0050167     22   0.33579 0.38294 0.0091914
## 19 0.0043478     23   0.33077 0.37893 0.0091678
## 20 0.0036789     24   0.32642 0.37425 0.0091395
## 21 0.0033445     26   0.31906 0.37391 0.0091375
## 22 0.0013378     27   0.31572 0.36789 0.0090999
## 23 0.0010033     28   0.31438 0.36823 0.0091020
## 24 0.0006689     29   0.31338 0.36957 0.0091105
## 25 0.0000000     31   0.31204 0.36856 0.0091041
```

```
## Warning in rsq.rpart(model1): may not be applicable for this method
```


Apparent
X Relative
R-square
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Number of Splits


X Relative Error
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Number of Splits

```
pred1 = predict(model1,test_new,type = "class")
pred1 = as.factor(pred1)

#Create a Confusion Matrix
actual = test_new$label
confMatrix1 = table(Predicted = pred1,actual = actual)
print(confMatrix1)
```

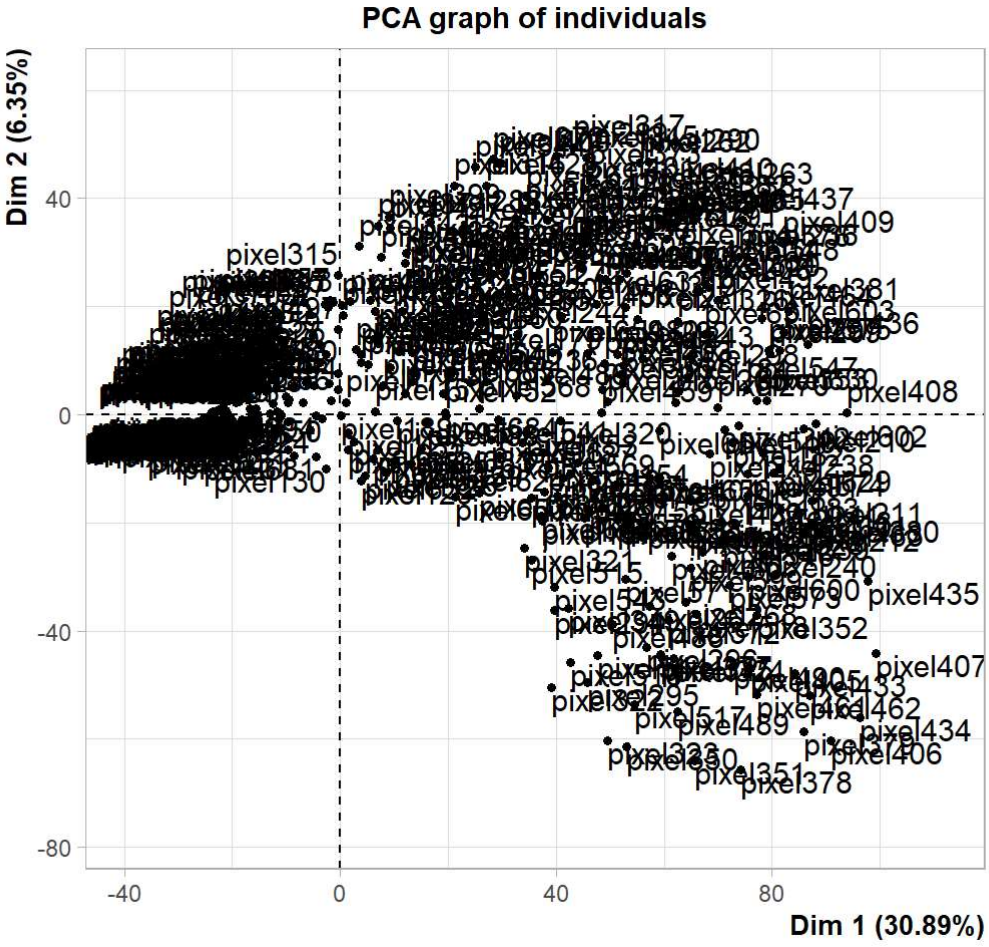
```
##          actual
## Predicted  0  1  2  3  4  5  6  7  8  9
##         0 69  0  5  4  0  4  4  0  0  1
##         1  0 91  4  1  0  4  2  0  1  2
##         2  0  2 44  8  0  2  2  1 11  2
##         3  2  4 10 55  2 23  2  2 11  5
##         4  3  1  2  4 58  2 13  3  2 11
##         5  3  2  2 11  2 44  6  4  4  4
##         6  0  0  2  3  2  0 44  0  2  1
##         7  1  5  1  1  0  0  0 59  0  3
##         8  2  1  2  3  0  3  1  4 50  1
##         9  1  0  1  5  9  9  1  7  4 50
```

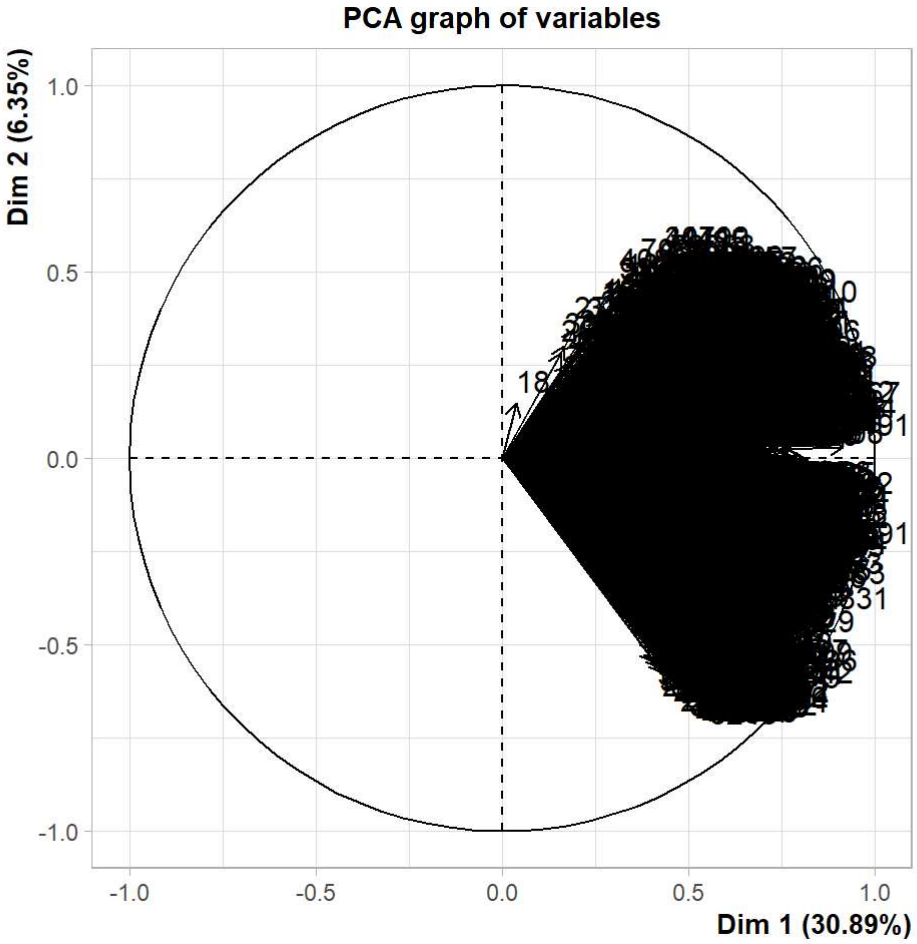
```
#Calculate Accuracy
correct_pred = sum(pred1 == actual)
total_pred = length(pred1)
accuracy = correct_pred/total_pred
print(paste("The accuracy of model 1 is",accuracy))
```

```
## [1] "The accuracy of model 1 is 0.67222884386174"
```

Attempt PCA and Reducing Dimensionality

```
pca_digits = PCA(t(select(new_data,-label)))
```


PCA graph of individuals
Dim 2 (6.35%)
Dim 1 (30.89%)
40
0
-40
-80
-40
0
40
80
pixel315
pixel130
pixel408
pixel435
pixel352
pixel407
pixel434
pixel406
pixel378

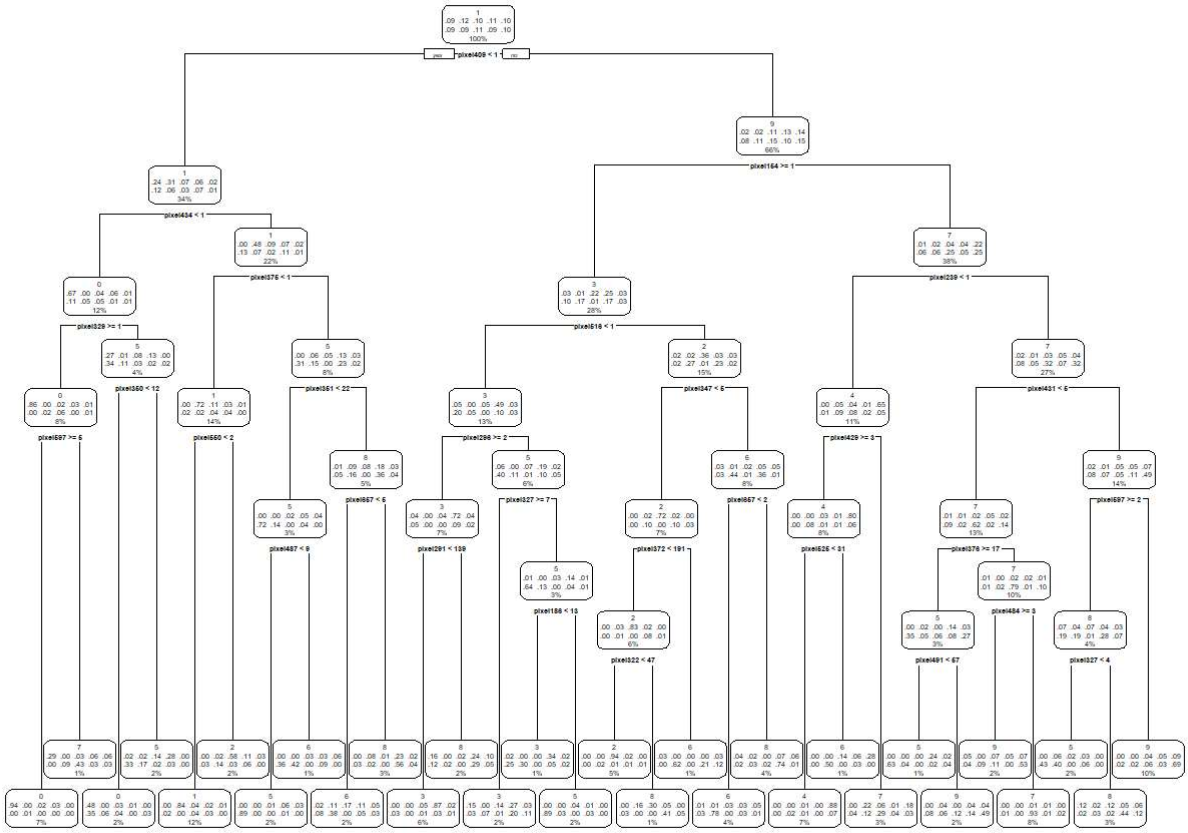

PCA graph of variables
Dim 2 (6.35%)
Dim 1 (30.89%)
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0

```
new_data2 = data.frame(new_data$label,pca_digits$var$coord)

#Create Train/Test of New_Data
index2 = createDataPartition(new_data2$new_data.label, p = 0.8, list = FALSE, times = 1)
train_new2 = new_data[index2,]
test_new2 = new_data[-index2,]
```

Model with PCA

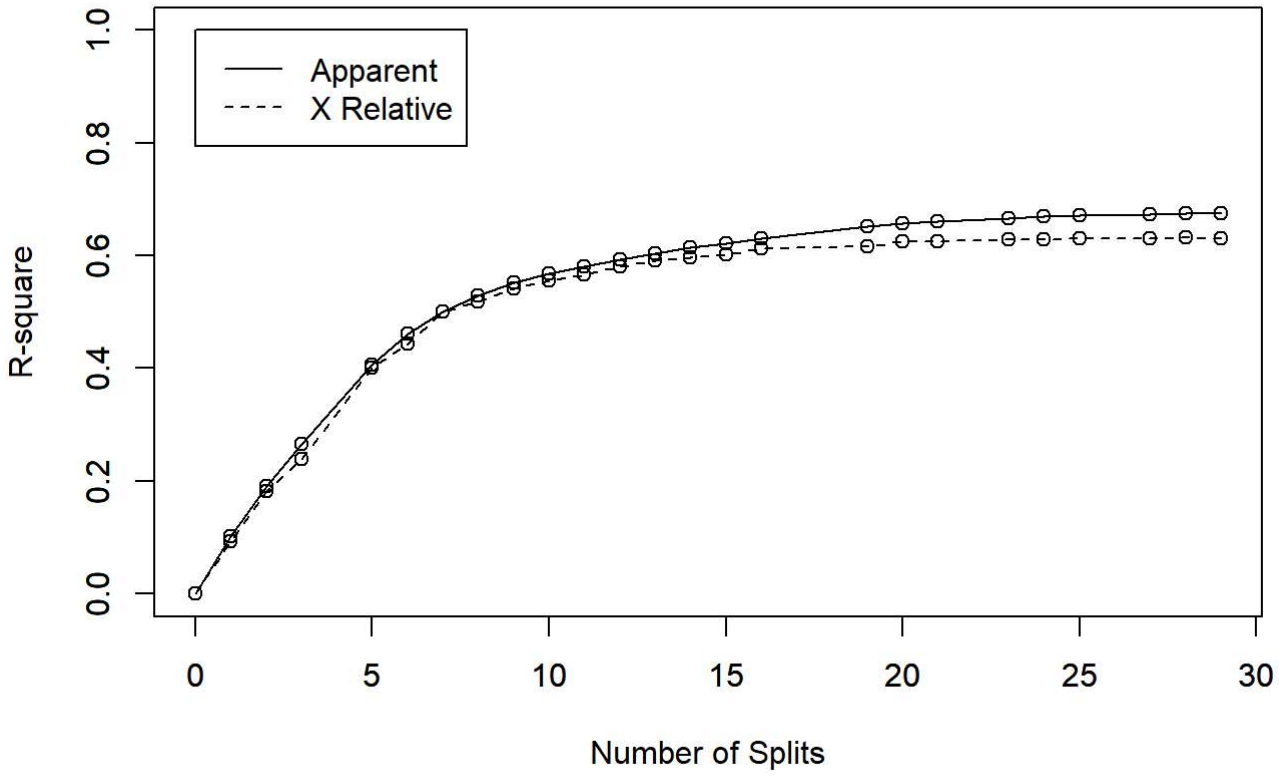
```
#Model 2
model2 = rpart(label~., data = train_new2,method = "class", control = rpart.control(cp = 0, mins
plit = 100, maxdepth = 20))
rpart.plot(model2,box.palette = 0)
```

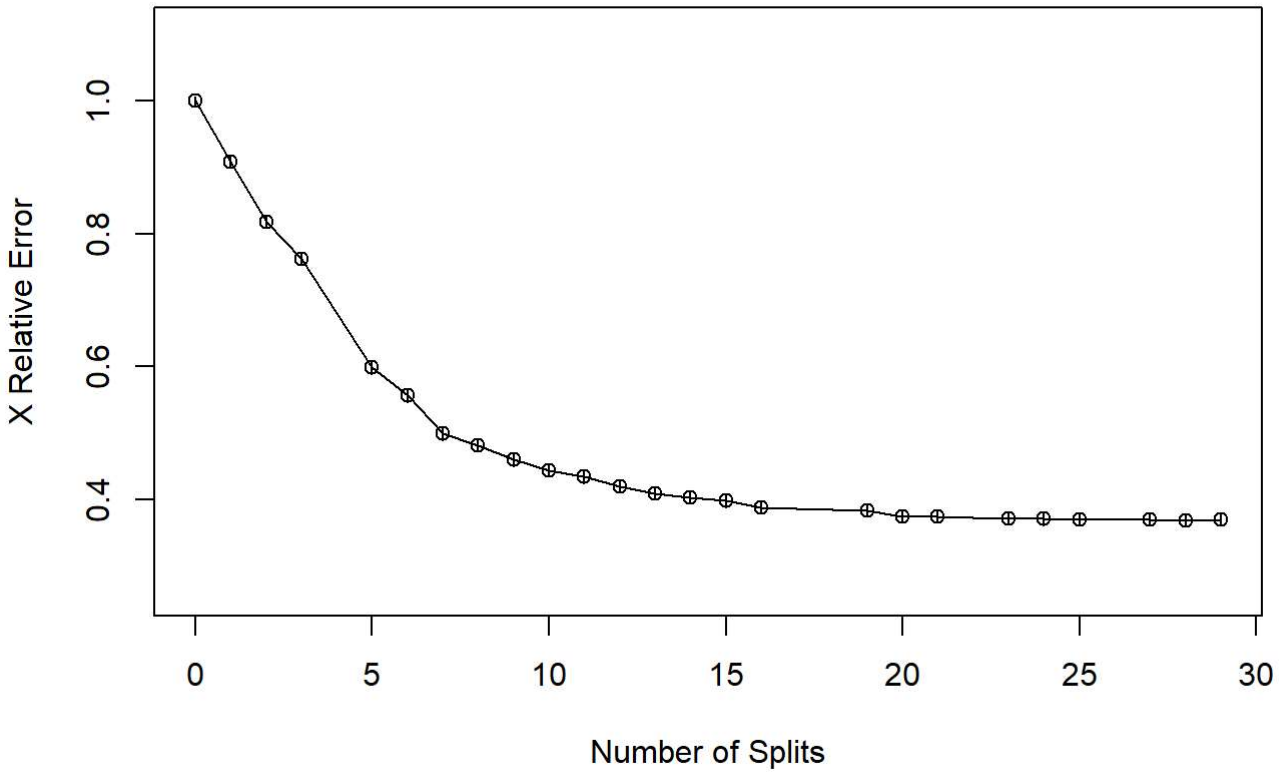


```
rsq.rpart(model2)
```

```
##
## Classification tree:
## rpart(formula = label ~ ., data = train_new2, method = "class",
##     control = rpart.control(cp = 0, minsplit = 100, maxdepth = 20))
##
## Variables actually used in tree construction:
##  [1] pixel154 pixel186 pixel239 pixel291 pixel296 pixel322 pixel327 pixel329
##  [9] pixel347 pixel350 pixel351 pixel372 pixel375 pixel376 pixel409 pixel429
## [17] pixel431 pixel434 pixel484 pixel487 pixel491 pixel516 pixel525 pixel550
## [25] pixel597 pixel657
##
## Root node error: 2969/3364 = 0.88258
##
## n= 3364
##
##            CP nsplit rel error  xerror      xstd
## 1  0.10138094      0   1.00000 1.00000 0.0062888
## 2  0.08925564      1   0.89862 0.90738 0.0078019
## 3  0.07409902      2   0.80936 0.81812 0.0087515
## 4  0.07073089      3   0.73526 0.76120 0.0091728
## 5  0.05591108      5   0.59380 0.59852 0.0097520
## 6  0.03772314      6   0.53789 0.55608 0.0097659
## 7  0.02930280      7   0.50017 0.49882 0.0096976
## 8  0.02256652      8   0.47087 0.48198 0.0096582
## 9  0.01650387      9   0.44830 0.45941 0.0095914
## 10 0.01313574     10   0.43180 0.44426 0.0095374
## 11 0.01212529     11   0.41866 0.43382 0.0094958
## 12 0.01077804     12   0.40653 0.41967 0.0094337
## 13 0.00976760     13   0.39576 0.40856 0.0093802
## 14 0.00808353     14   0.38599 0.40249 0.0093492
## 15 0.00774672     15   0.37791 0.39778 0.0093242
## 16 0.00707309     16   0.37016 0.38767 0.0092681
## 17 0.00606265     19   0.34894 0.38296 0.0092406
## 18 0.00336814     20   0.34288 0.37420 0.0091875
## 19 0.00303132     21   0.33951 0.37454 0.0091896
## 20 0.00269451     23   0.33345 0.37117 0.0091685
## 21 0.00168407     24   0.33075 0.37050 0.0091642
## 22 0.00134725     25   0.32907 0.36915 0.0091556
## 23 0.00101044     27   0.32637 0.36915 0.0091556
## 24 0.00067363     28   0.32536 0.36847 0.0091513
## 25 0.00000000     29   0.32469 0.36982 0.0091599
```

```
## Warning in rsq.rpart(model2): may not be applicable for this method
```


Apparent
X Relative
R-square
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Number of Splits


X Relative Error
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Number of Splits

```
#Prediction
pred2 = predict(model2,test_new2,type = "class")
pred2 = as.factor(pred2)

#Create a Confusion Matrix
actual2 = test_new2$label
confMatrix2 = table(Predicted = pred2,actual = actual2)
print(confMatrix2)
```

```
##          actual
## Predicted  0  1  2  3  4  5  6  7  8  9
##         0 60  0  5  1  0  9  4  1  0  0
##         1  1 69  5  0  1  2  0  6  5  1
##         2  0  2 53  3  1  1  5  0  0  1
##         3  9  0  5 44  4  5  6  0  8  3
##         4  0  0  2  0 49  0  2  1  0  4
##         5  2  0  2 13  1 46 15  4  3  0
##         6  1  0  6  0  6  3 42  0  5  1
##         7  5  6  1  4  7  1  4 59  2  5
##         8  7  7  5 13  2  5  2  3 48  7
##         9  3  1  3  3 16  4  6 10  4 68
```

```
#Calculate Accuracy
correct_pred2 = sum(pred2 == actual2)
total_pred2 = length(pred2)
accuracy2 = correct_pred2/total_pred2
print(paste("The accuracy of model 2 is",accuracy2))
```

```
## [1] "The accuracy of model 2 is 0.641239570917759"
```

Naives Bayes Model

```
#Train A Simple Naive Bayes Model
n_model = naiveBayes(label~., data = train_new,na.action = na.pass, mode = "classification")
summary(n_model)
```

```
##           Length Class  Mode
## apriori    10    table  numeric
## tables    784    -none- list
## levels     10    -none- character
## isnumeric 784    -none- logical
## call        5    -none- call
```

```
##Naive Bayes model Prediction
nb_Pred <- predict(n_model, test_new)

#Create a Confusion Matrix
actual3 = test_new$label
confMatrix3 = table(Predicted = nb_Pred,actual = actual3)
print(confMatrix3)
```

```
##          actual
## Predicted   0   1   2   3   4   5   6   7   8   9
##         0  78   0   6   8   0   7   0   4   0   4
##         1   0 104  12  16   0  14   6  22  29  12
##         2   0   0  15   1   0   0   0   0   0   0
##         3   0   0  12  52   1   5   0   1   2   0
##         4   0   0   0   0  33   9   0   0   0   1
##         5   0   0   1   0   0   6   0   0   1   0
##         6   0   1  17   4   2   4  65   0   0   1
##         7   0   0   0   0   1   1   0  20   0   2
##         8   1   1   8   7   3  31   3   0  35   0
##         9   2   0   2   7  33  14   1  33  18  60
```

```
#Calculate Accuracy
correct_pred3 = sum(nb_Pred == actual)
total_pred3 = length(nb_Pred)
accuracy3 = correct_pred3/total_pred3
print(paste("The accuracy of n_model is",accuracy3))
```

```
## [1] "The accuracy of n_model is 0.557806912991657"
```

kNN Model

```
#Scale the Data
train_knn = train_new[-1]
test_knn = test_new[-1]

test_pred <- knn(
                 train = train_knn,
                 test = test_knn,
                 cl = train_new$label,
                 k=10
                 )

#Cross Validation
actual4 <- test_new$label

cm <- table(actual4,test_pred)
cm
```

```
##        test_pred
## actual4   0   1   2   3   4   5   6   7   8   9
##       0  81   0   0   0   0   0   0   0   0   0
##       1   0 105   1   0   0   0   0   0   0   0
##       2   3   4  62   1   0   0   0   2   1   0
##       3   0   1   2  86   0   2   0   2   0   2
##       4   0   1   0   0  70   0   1   0   0   1
##       5   0   4   0   2   0  84   0   0   0   1
##       6   1   0   0   0   0   2  72   0   0   0
##       7   0   6   0   0   0   0   0  70   0   4
##       8   0   3   0   2   0   3   1   0  72   4
##       9   2   2   1   0   2   0   0   6   1  66
```

```
#Accuracy
accuracy4 <- sum(diag(cm))/length(actual4)
sprintf("Accuracy: %.2f%%", accuracy4*100)
```

```
## [1] "Accuracy: 91.54%"
```

SVM Model

```
# Separate labels and features for the training set
train_labels <- train_new[, 1]
train_features <- train_new[, -1]

# Similarly, for the test set
test_labels <- test_new[, 1]
test_features <- test_new[, -1]

# Train the SVM model
svm_model <- svm(train_features, as.factor(train_labels), kernel = "linear")

# Predict using the model
svm_predict <- predict(svm_model, test_features)

#Cross Validation
actual5 <- test_new$label

cm2 <- table(actual5,svm_predict)
cm2
```

```
##        svm_predict
## actual5   0   1   2   3   4   5   6   7   8   9
##       0  78   0   0   1   0   1   1   0   0   0
##       1   0 104   1   0   1   0   0   0   0   0
##       2   0   1  66   1   0   1   1   2   1   0
##       3   1   0   5  81   0   4   0   0   4   0
##       4   0   1   0   0  71   0   0   0   0   1
##       5   0   3   0   5   2  74   2   0   4   1
##       6   1   0   0   0   1   1  71   0   1   0
##       7   0   3   1   0   2   1   0  71   0   2
##       8   1   1   0   3   0   2   0   0  78   0
##       9   1   2   1   2   1   1   0   2   3  67
```

```
#Accuracy
accuracy5 <- sum(diag(cm2))/length(actual5)
sprintf("Accuracy: %.2f%%", accuracy5*100)
```

```
## [1] "Accuracy: 90.70%"
```

Random Forest

```
#Train the Random Forest Model
rf <- randomForest(label~., data=train_new, proximity=TRUE)

# Predict using the model
rf_predict <- predict(rf, test_new)

#Cross Validation
actual6 <- test_new$label

cm3 <- table(actual6,svm_predict)
cm3
```

```
##        svm_predict
## actual6   0   1   2   3   4   5   6   7   8   9
##       0  78   0   0   1   0   1   1   0   0   0
##       1   0 104   1   0   1   0   0   0   0   0
##       2   0   1  66   1   0   1   1   2   1   0
##       3   1   0   5  81   0   4   0   0   4   0
##       4   0   1   0   0  71   0   0   0   0   1
##       5   0   3   0   5   2  74   2   0   4   1
##       6   1   0   0   0   1   1  71   0   1   0
##       7   0   3   1   0   2   1   0  71   0   2
##       8   1   1   0   3   0   2   0   0  78   0
##       9   1   2   1   2   1   1   0   2   3  67
```

```
#Accuracy
accuracy6 <- sum(diag(cm3))/length(actual6)
sprintf("Accuracy: %.2f%%", accuracy6*100)
```

```
## [1] "Accuracy: 90.70%"
```

Conclusion

After creating 6 different models, we can surely say that the world of machine learning is always advancing! Improvements and changes are always being sought after to find that perfect model that provides the best results. However, not all models are equal as well! One model might work miraculously upon a data set and then one that is of similar information could prove different results. This means that data is always ever-shifting! This aspect is a data analyst's biggest challenge. As the saying goes, there are multiple ways to skin a cat. Relating this information to the report, all 6 of the different models either had similar results or some different results. In order to begin processing the data, we had to reduce the information down to a more manageable set. The original set has more than 100,000 cells! Thus, for this experiment, we reduced the dimensions down to a total of 4206x785. After reducing the data, we then created the train and test data from a 80/20 split. First, let's talk about the first model, the Decision Tree.

For our first model, the Decision Tree, we had to take the data and process it in an rpart code. Initially, the results calculated with about 50% accuracy without changing any of the parameters. Altering the minsplit to 100 and the maxdepth to 20, we were able to calculate an accuracy of 67.22%. This is a decent accuracy for our testing. Potentially, the model could perform better if more parameters were tested. The beauty about Decision Trees is that they are simple to set up and require little preparation for a result. However, one disadvantage to this is that they can over complicate the tree which leads to over fitting.

For the second model, the PCA and Reducing Dimensionality, we have to alter the data in a specific way to add this feature into a Decision Tree. In order for this process to work, we standardized all the data for each feature to be a mean of 0 and a variance equal to 1. Once standardized, we could use the visualizations to make sure that our information is within our limits. Adding this into an rpart model, we concluded an accuracy of 64.12%. This was lesser than the decision tree model. From doing this, we might have run into an issue with the PCA coding not recognizing the linearity of the data. Because of this, it caused a lesser percentage than improving it.

For the third model, the Naive's Bayes Model, we started by placing the training data into it. There was no need for any further alterations of the data for this classification. Reassuring the model to pass any data that was NA, we received a result of 55.78%. Looking at the correlation matrix, we were able to see that integers that were similar in shape such as 4 and 9 were misinterpreted 33 times. For this model in particular, I believe that the issue where this model receives such a low accuracy is because it assumes all the features are independent from one another.

For the fourth model, We used a k-Nearest Neighbor for this method. Before we set up a model for it, we had to determine a k value from 1 to 10. Usuing a 10 for starters, we were then able to process the model. Also, this model cannot have the labels within the data for processing. After creation, we were able to achieve and accuracy of 91.54%! This method even was able to discern a 4 from a 9 however, it did have some issues with 1 and 7.

For the fifth model, we used a Support Vector Machine. This particular model needed to have the labels listed as factors for processing. Originally, the kernel was set to "sigmoid" for that particular activation. However, since we are dealing with more linear workings of data, it was set to linear. Upon checking the accuracy of the data, it proved to be a 90.70% accuracy. This particular model didn't have any incorrect answers over a frequency of 5 however, they were more errors across the board.

For our final model, we used a Random Forest tree model. This is a more complex use of a decision tree that was utilized earlier. For this model, it can handle robust data because it utilizes its effectiveness by reducing the variance through averaging multiple trees. For this particular experiment, we were able to acquire an accuracy of _____. Though this model can improve the accuracy of harder data, a couple of things can happen. The amount of “trees” can make the model incredibly difficult to interpret and it has the possibility of having slower processing time due to the computational stress.

Ultimately, the model that proved the highest evaluation accuracy was the k-Nearest Neighbors model. With an accuracy of 91%, it took the lead over the Random Forest and SVM models. I believe that the data set, in general was much more robust and noisy to deal with than the original usages such as a Decision Tree, Naive Bayes, and PCA tuning. What we can conclude is that creating a model to interpret visual data is possible depending on the parameters set and how the information is provided. Data is always evolving and one model can work great one day and then fail another with a different testing set. Thus, I believe each method has its uses and all comes down to effective analysis and testing to try and create a “great” model.