TABLE OF CONTENTS

EXECUTIVE SUMMARY

Handwritten digit recognition is a classical machine learning problem and is one of the algorithms used by USPS to identify hand-written postal zip codes. We tried different machine learning algorithms to accurately classify the handwritten digits based on the pixel intensities.

Of the various machine learning algorithms tested, boosted trees performed best with a test classification accuracy of 0.989. SVM, k-nearest neighbors and random forest models were the next best, producing fairly close test classification accuracies ranging from 0.95-0.97. Naive Bayes and decision tree models had the least predictive power. The performance of the different machine learning algorithms is summarized in the table below:

Machine Learning Algorithm	**Test Data Set Classification Accuracy**
Decision Tree (CART)	0.618
Naïve Bayes	0.821
Random Forest	0.960
K-Nearest Neighbors	0.969
Boosted Trees	0.990
Support Vector Machines	0.977

I. INTRODUCTION

The data for our project is provided by Kaggle and is based on the famous MNIST dataset. The dataset consists of 42000 handwritten digits and each hand-written digit is represented in the form of a 28X28 pixel. So for each digit there are 784 predictor variables which indicate the pixel intensities (continuous and takes a value between 0 and 255). The dependent variable is categorical (0-9) and is the label associated with the hand-written digit. The objective of the project is to accurately classify the handwritten digits based on their pixel intensities using different supervised machine learning algorithms.

II. DATA PREPARATION

The first step in any modeling exercise is to visualize the data. Figure 1 gives a visualization of the digits in the dataset by considering the average pixel intensities of the corresponding hand-written digits (0-9).

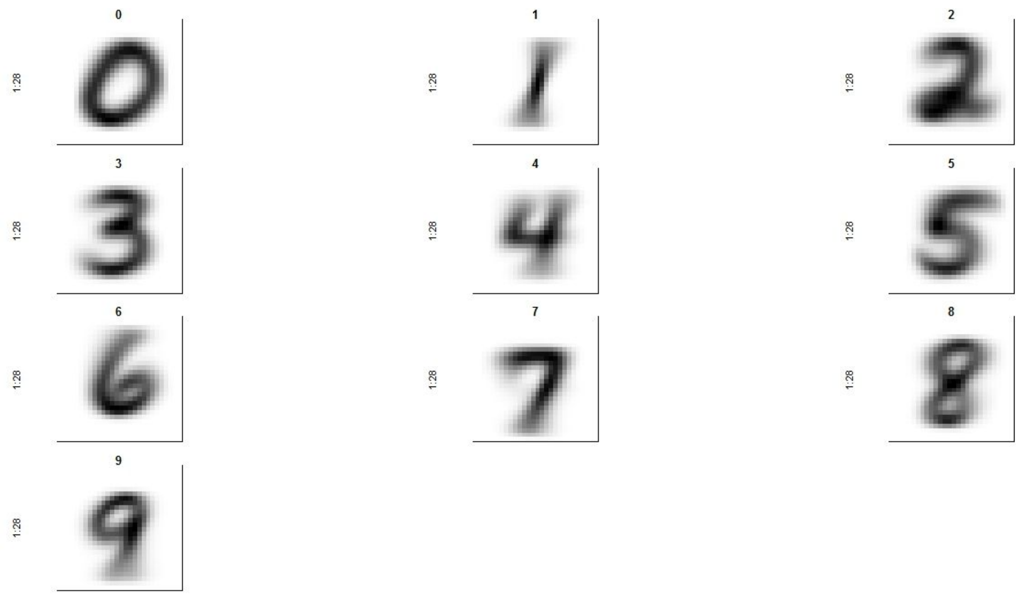


Figure 1. Visualization of the handwritten digits

Next we split the dataset into two parts: (a) training dataset based on 70% of the entire dataset - 29,400 images and (b) test dataset based on 30% of the entire dataset - 12,600 images. We used the training dataset to build our predictive models and tested the robustness of the model against the test dataset. The distribution of the different digits in the training dataset reveals that the digits are more or less equally likely to occur although there are more 1's overall in the training dataset. The table below gives a frequency distribution of the digit labels as they appear in the training dataset.

Table 1. Frequency distribution of digit labels in training dataset

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2961	3309	2891	3066	2847	2635	2896	3078	2817	2900

Since we have 784 predictors in our dataset we decided to reduce the dimensionality of the dataset by discarding predictors with almost zero variance. This reduces the dimensionality of the dataset significantly from 784 to 253. These 253 predictors are then standardized to make sure that each of the predictors have the same scale.

III. SUPERVISED MACHINE LEARNING ALGORITHMS

We tried out the following supervised machine learning algorithms to classify the handwritten digits.

1. CART
2. Naive Bayes
3. Random Forest
4. K Nearest Neighbors
5. Boosted Trees
6. Support Vector Machines
7. Artificial Neural Network

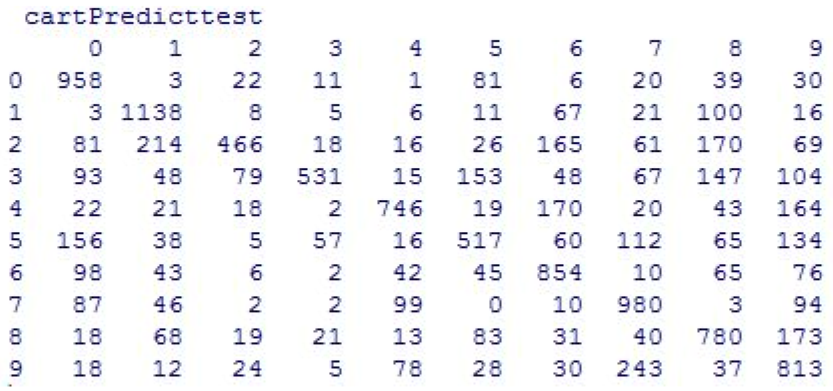
The following section describes the methodology we adopted for each of these supervised machine learning techniques and how each of them performed on the test dataset.

Decision Tree (CART)

We first decided to try decision trees due to their easy interpretability. We used the rpart package in R, and made sure to treat the 10 labels as classification variables by using the method=”class” option. We pruned the tree back using cross-validation such that it minimizes the Cp.

Our misclassification table can be viewed in Table 2 below. The tree did not perform exceptionally well with an accuracy of 0.62, however as you can see below, the largest values are along the diagonals, as they should be. The tree had a particularly difficult time in classifying the digits 2, 3, and 5, but did well with 0, 1, and 7.

Table 2. CART Misclassification table for test dataset

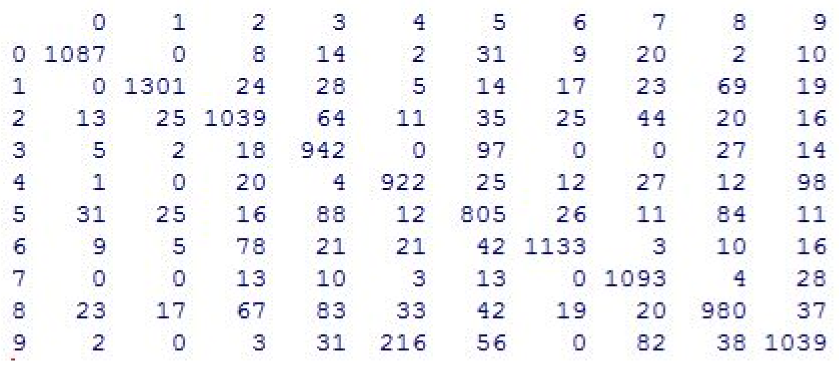


```
  cartPredicttest
       0    1    2    3    4    5    6    7    8    9
  0  958    3   22   11    1   81    6   20   39   30
  1    3 1138    8    5    6   11   67   21  100   16
  2   81  214  466   18   16   26  165   61  170   69
  3   93   48   79  531   15  153   48   67  147  104
  4   22   21   18    2  746   19  170   20   43  164
  5  156   38    5   57   16  517   60  112   65  134
  6   98   43    6    2   42   45  854   10   65   76
  7   87   46    2    2   99    0   10  980    3   94
  8   18   68   19   21   13   83   31   40  780  173
  9   18   12   24    5   78   28   30  243   37  813
```

Naive Bayes

Then we looked at fitting a Naive Bayes model which is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions. This is one of the models we decided to try which has not yet been covered in class. The prior probabilities of each of the digit labels is estimated by dividing the frequency counts of each label by the total size of the training dataset. We used the e1071 R package to build the naive Bayes classifier. The misclassification matrix after predicting the test dataset is given below. Overall we get an accuracy of 0.821 on the test dataset, which is significantly higher than the decision tree tested earlier.

Table 3. Naive Bayes misclassification table for test dataset



```
      0    1    2    3    4    5    6    7    8    9
0  1087    0    8   14    2   31    9   20    2   10
1     0 1301   24   28    5   14   17   23   69   19
2    13   25 1039   64   11   35   25   44   20   16
3     5    2   18  942    0   97    0    0   27   14
4     1    0   20    4  922   25   12   27   12   98
5    31   25   16   88   12  805   26   11   84   11
6     9    5   78   21   21   42 1133    3   10   16
7     0    0   13   10    3   13    0 1093    4   28
8    23   17   67   83   33   42   19   20  980   37
9     2    0    3   31  216   56    0   82   38 1039
```

Random Forest

We next fit a random forest model on the handwritten digit recognition dataset using the randomForest package in R. The random forest is an ensemble model and is fit by bagging simple decision trees without pruning them. Each of the trees vote for classification of the handwritten digit to one of the labels and based on the majority vote of all trees, the handwritten digit is classified into the chosen label. The randomForest package in R tries to find an optimum value of the mtry parameter, which stands for the subset of variables randomly sampled from candidates for split. The optimal value of the mtry parameter obtained using cross-validation so as to minimize the OOB error is 30, and a graph of the mtry versus the error is shown in Figure 2 below.

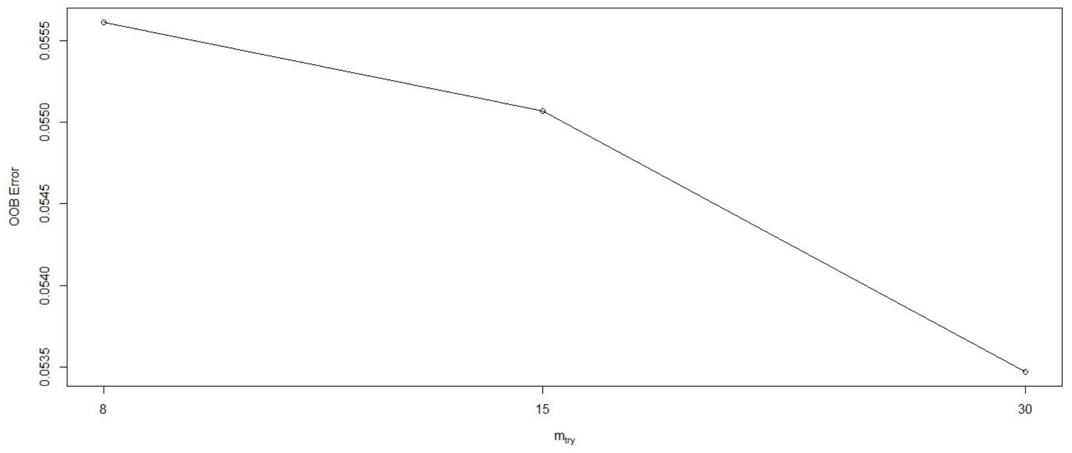
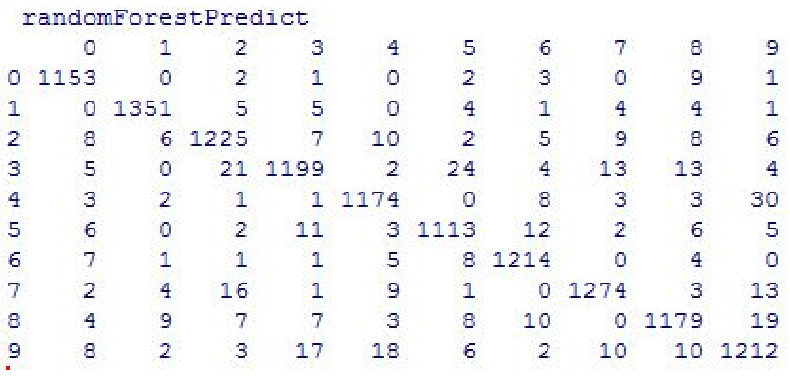


Figure 2. Optimum value of mtry parameter for Random Forest

The random forest model is fitted with mtry=30 and 500 decision trees. The classification accuracy of the model on the test dataset is 0.960. The random forest model performs pretty well in classifying the handwritten digits, but we feel that we can get better classification accuracy with other supervised machine learning algorithms such as SVM and boosted trees. The table below shows the misclassification table on the test dataset for the fitted random forest model.

Table 4. Random forest misclassification table for test dataset



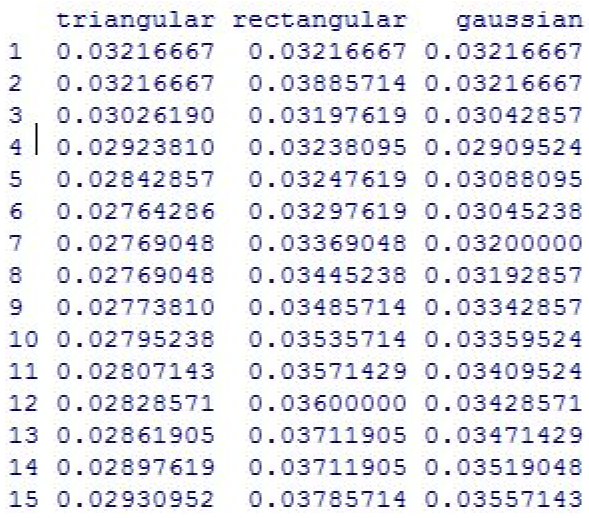
```
  randomForestPredict
       0    1    2    3    4    5    6    7    8    9
0 1153    0    2    1    0    2    3    0    9    1
1    0 1351    5    5    0    4    1    4    4    1
2    8    6 1225    7   10    2    5    9    8    6
3    5    0   21 1199    2   24    4   13   13    4
4    3    2    1    1 1174    0    8    3    3   30
5    6    0    2   11    3 1113   12    2    6    5
6    7    1    1    1    5    8 1214    0    4    0
7    2    4   16    1    9    1    0 1274    3   13
8    4    9    7    7    3    8   10    0 1179   19
9    8    2    3   17   18    6    2   10   10 1212
```

K-Nearest Neighbors

The K-nearest neighbor algorithm computes the Euclidean distance of all other observations from the unlabeled data point and selects K neighbors which are closest to the unlabeled data point based on the Euclidean distances. The unlabeled data point is then classified into one of the 10 classes based on the class which has the maximum predicted probability based on the voting of the K-nearest neighbors. By default the K-NN algorithms gives equal weight to each of the K neighbors.

We considered a weighted K-Nearest Neighbor approach using the kknn package in R, so that neighbors closest to the unlabeled data point will be weighted higher than neighbors far away from the unlabeled data point. We considered 3 different kernels, namely: **1. triangular** - weight assigned decreases linearly with further distance from the unlabeled data point, **2. rectangular** - equal weight assigned to all neighbors, and **3. gaussian** - weight assigned to the neighbor decays along the gaussian curve further the distance from the unlabeled data point. The optimal value of K and the choice of the kernel is obtained through cross-validation. The table below gives the cross-validation misclassification rates for different choices of K and kernels.

Table 5. Optimal Choice of K for 3 different kernels based on cross-validation

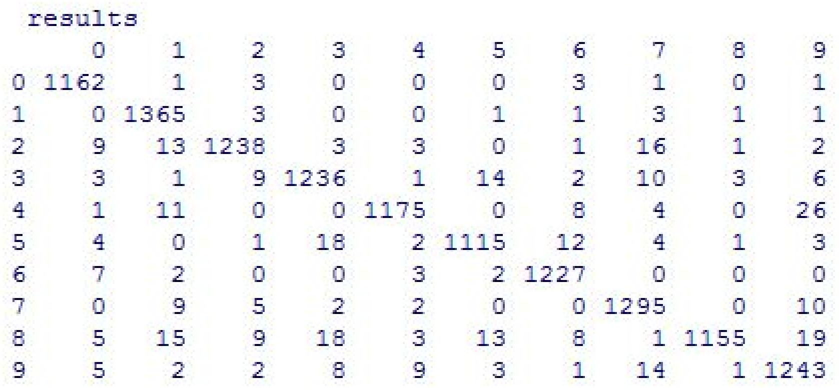


```
   triangular rectangular   gaussian
1  0.03216667  0.03216667 0.03216667
2  0.03216667  0.03885714 0.03216667
3  0.03026190  0.03197619 0.03042857
4  0.02923810  0.03238095 0.02909524
5  0.02842857  0.03247619 0.03088095
6  0.02764286  0.03297619 0.03045238
7  0.02769048  0.03369048 0.03200000
8  0.02769048  0.03445238 0.03192857
9  0.02773810  0.03485714 0.03342857
10 0.02795238  0.03535714 0.03359524
11 0.02807143  0.03571429 0.03409524
12 0.02828571  0.03600000 0.03428571
13 0.02861905  0.03711905 0.03471429
14 0.02897619  0.03711905 0.03519048
15 0.02930952  0.03785714 0.03557143
```

From the table the optimum value of K = 6 and kernel is triangular kernel. The K-NN model is fitted with the above parameters and the model results in a classification accuracy of 0.969 on the test dataset.

The table below gives the misclassification table for the weighted K-NN model.

Table 6. Misclassification table for the Weighted K-NN model



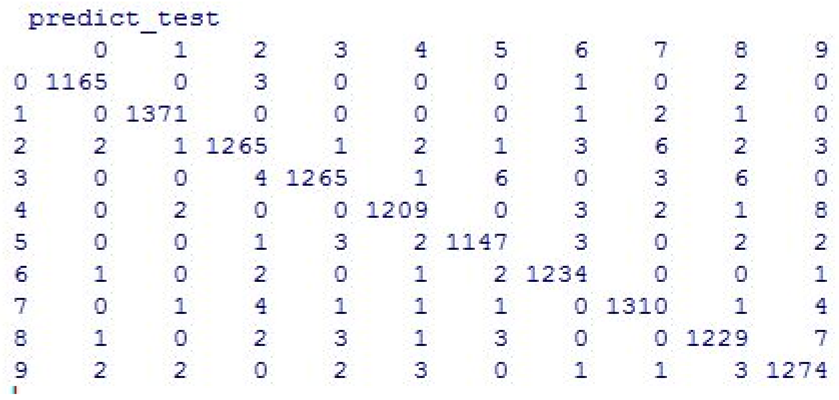
```
 results
      0    1    2    3    4    5    6    7    8    9
0 1162    1    3    0    0    0    3    1    0    1
1    0 1365    3    0    0    1    1    3    1    1
2    9   13 1238    3    3    0    1   16    1    2
3    3    1    9 1236    1   14    2   10    3    6
4    1   11    0    0 1175    0    8    4    0   26
5    4    0    1   18    2 1115   12    4    1    3
6    7    2    0    0    3    2 1227    0    0    0
7    0    9    5    2    2    0    0 1295    0   10
8    5   15    9   18    3   13    8    1 1155   19
9    5    2    2    8    9    3    1   14    1 1243
```

Boosted Trees

We next tried a boosted tree model hoping to improve upon the benchmark set by Random Forest and K-NN. We used the gbm and caret packages in R to fit the boosted tree model.
The optimal parameters for number of trees and shrinkage parameter is decided by 5-fold cross-validation. The optimal number of trees was 1000, and the value of the shrinkage parameter is 0.1. We fixed the interaction depth at 4.

The boosted tree model led to a prediction classification accuracy of 0.990 on the test dataset which beats both Random Forest and Weighted K-NN model. The table below shows the misclassification table on the test dataset for the fitted Boosted Trees model.

Table 7. Misclassification table for the Boosted trees model



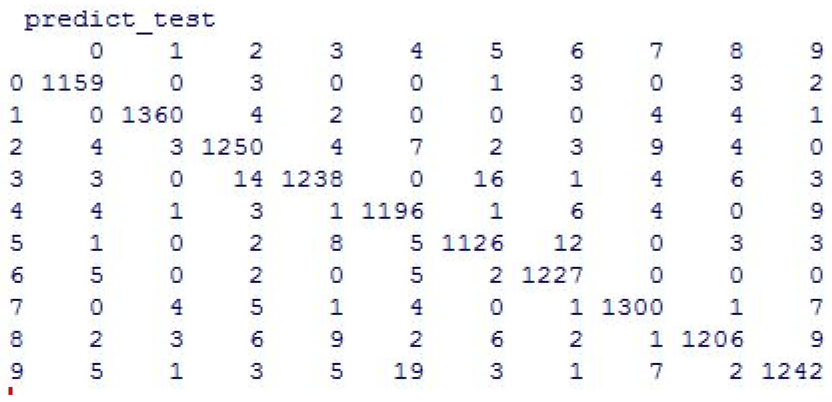
```
 predict_test
      0    1    2    3    4    5    6    7    8    9
0 1165    0    3    0    0    0    1    0    2    0
1    0 1371    0    0    0    0    1    2    1    0
2    2    1 1265    1    2    1    3    6    2    3
3    0    0    4 1265    1    6    0    3    6    0
4    0    2    0    0 1209    0    3    2    1    8
5    0    0    1    3    2 1147    3    0    2    2
6    1    0    2    0    1    2 1234    0    0    1
7    0    1    4    1    1    1    0 1310    1    4
8    1    0    2    3    1    3    0    0 1229    7
9    2    2    0    2    3    0    1    1    3 1274
```

Support Vector Machines

Support vector machines perform predictions by finding a hyperplane that separates points into groups. The support vectors form a “soft” margin surrounding the dividing hyperplane that allows for some flexibility in the classification. If the data is not easily grouped through a linear hyperplane, support vector machines allow for analysis in higher dimensional spaces through the usage of kernels. We used the caret and kernlab package in R to fit the SVM model. We used an RBF kernel and the optimal parameters C and sigma are obtained using a 5 fold cross-validation. We find the optimal value of C as 8 and the optimal value of sigma as 0.0022.

The fitted SVM model has a prediction classification accuracy of 0.976 in the test dataset.
The table below gives the misclassification table for the fitted SVM model.

Table 7. Misclassification table for the SVM model



```
 predict_test
     0    1    2    3    4    5    6    7    8    9
0 1159    0    3    0    0    1    3    0    3    2
1    0 1360    4    2    0    0    0    4    4    1
2    4    3 1250    4    7    2    3    9    4    0
3    3    0   14 1238    0   16    1    4    6    3
4    4    1    3    1 1196    1    6    4    0    9
5    1    0    2    8    5 1126   12    0    3    3
6    5    0    2    0    5    2 1227    0    0    0
7    0    4    5    1    4    0    1 1300    1    7
8    2    3    6    9    2    6    2    1 1206    9
9    5    1    3    5   19    3    1    7    2 1242
```

Artificial Neural Network

Lastly we tried Artificial Neural Network for our digit classification problem. We vary the number of hidden units over the range [1,5,10,20,50,100,200,500] and the decay parameter over the range [0,0.001,0.005,0.01,0.05,0.1,0.5,1,2]. Our objective is to choose the optimal number of hidden layers and decay parameter using 5 fold cross-validation. Unfortunately R’s nnet package is unable to scale to higher number of units in the hidden layer. A problem like handwriting digit recognition requires hundreds of units in the hidden layer of the neural network to achieve high prediction classification accuracy. Because of limitations of R’s nnet package we were unable to scale to hundreds of hidden units. A neural network implementation with 4 hidden units gave a classification accuracy of as low as ~70% in the test dataset. We concluded that unless we can scale up the number of hidden units, we can't improve the prediction classification accuracy.

IV. CONCLUSION

The list of supervised machine learning classification algorithms is long and diverse. We only tested a handful of algorithms on our handwritten digit recognition dataset, and of all the methods we tested, Boosted Trees gave the best classification prediction accuracy. We wanted to try out other algorithms as well, but the implementation of many algorithms such as Regularized Discriminant Analysis in R was very time consuming for our dataset. However, we feel that we still got an opportunity to cover the major machine learning classification algorithms for prediction of handwritten digits.

V. REFERENCES

CARET package in R: http://caret.r-forge.r-project.org/
Journal of Statistical Software: Building Predictive Models in R using the CARET package: Max Kuhn: http://www.jstatsoft.org/v28/i05/paper

VI. APPENDIX

R code:

```
## Reading the training data
train <- read.csv("train.csv", header=TRUE)
train<-as.matrix(train)

## Visualizing the hand-written digits
##Color ramp def.
colors<-c('white','black')
cus_col<-colorRampPalette(colors=colors)

## Plot the average image of each digit
par(mfrow=c(4,3),pty='s',mar=c(1,1,1,1),xaxt='n',yaxt='n')
all_img<-array(dim=c(10,28*28))
for(di in 0:9)
{
print(di)
all_img[di+1,]<-apply(train[train[,1]==di,-1],2,sum)
all_img[di+1,]<-all_img[di+1,]/max(all_img[di+1,])*255

z<-array(all_img[di+1,],dim=c(28,28))
z<-z[,28:1] ##right side up
image(1:28,1:28,z,main=di,col=cus_col(256))
}

train<-as.data.frame(train)

## Using the caret package to discard predictors with almost zero variance and standardizing the
predictors

library(caret)
badCols <- nearZeroVar(train)
train <- train[, -badCols]
train[c(-1)]<-sapply(train[c(-1)],scale)
test<-sapply(test,scale)

## Splitting the entire dataset into training and test datasets
indexes=sample(1:nrow(train),size=0.3*nrow(train))
testdigits=train[indexes,]
traindigits=train[-indexes,]

## Fitting a single decision tree
library(rpart)
cart = rpart(label ~ ., data=traindigits, method="class", control = rpart.control(minbucket=10))
pfit<- prune(cart, cp=cart$cptable[which.min(cart$cptable[,"xerror"]),"CP"])
cartPredicttrain = predict(pfit, newdata=traindigits, type="class")
cartPredicttest = predict(pfit, newdata=testdigits, type="class")

## Generating the misclassification table for training and test datasets
cartTabletrain = table(traindigits$label, cartPredicttrain)
cartTabletest = table(testdigits$label, cartPredicttest)


sum(diag(cartTabletrain))/nrow(traindigits)
sum(diag(cartTabletest))/nrow(testdigits)

## Fitting a randomForest
library(randomForest)
traindigits$label = factor(traindigits$label)
testdigits$label = factor(testdigits$label)
```

```
tunedForest=tuneRF(traindigits[,-1],traindigits[,1],ntreeTry=50,stepfactor=2)
randomForest = randomForest(label ~ ., data=traindigits, nodesize=10, mtry=30,
do.trace=TRUE)
randomForestPredict = predict(randomForest, newdata=testdigits)
head(randomForestPredict)
randomForestTable = table(testdigits$label, randomForestPredict)
randomForestTable
sum(diag(randomForestTable))/nrow(testdigits)

## Naive Bayes Digit Classifier
library(e1071)
traindigits$label = factor(traindigits$label)
testdigits$label = factor(testdigits$label)
NBclassifier<-naiveBayes(traindigits[,-1],traindigits$label)
NBtabletrain=table(predict(NBclassifier, traindigits[,-1]), traindigits[,1])
NBtabletest=table(predict(NBclassifier, testdigits[,-1]), testdigits[,1])
sum(diag(NBtabletrain))/nrow(traindigits)
sum(diag(NBtabletest))/nrow(testdigits)

## K Nearest Neighbour
# weighted k-nearest neighbors package
library(kknn)
# optimize knn for k=1:15
# and kernel=triangular, rectangular, or gaussian
model <- train.kknn(as.factor(label) ~ ., train, kmax=15,
kernel=c("triangular","rectangular","gaussian"))

# print out best parameters and prediction error
print(paste("Best parameters:", "kernel =", model$best.parameters$kernel, ", k =",
model$best.parameters$k))
print(model$MISCLASS)

# train the optimal kknn model
model <- kknn(as.factor(label) ~ ., traindigits, testdigits, k=6, kernel="triangular")
results <- model$fitted.values
##write(as.numeric(levels(results))[results], file="knn_submission.csv", ncolumns=1)
sum(results==testdigits$label)/nrow(testdigits)

## Fitting a boosted trees model
library(gbm)
library(caret)
```

```
gbmGrid <-  expand.grid(interaction.depth = 4,
                n.trees = (1:20)*50,
                shrinkage = 0.1)

fitControl <- trainControl(method="repeatedcv",
                 number=5,
                 repeats=1,
                 verboseIter=TRUE)
gbmFit <- train(as.factor(label) ~ ., data=traindigits,
           method="gbm",
           trControl=fitControl,
                   tuneGrid=gbmGrid,
           verbose=FALSE)
gbmFit
predict_train<-predict(gbmFit,newdata=traindigits)
sum(predict_train==traindigits$label)/nrow(traindigits)
predict_test<-predict(gbmFit,newdata=testdigits)
sum(predict_test==testdigits$label)/nrow(testdigits)
```

Accuracy of 1 on training dataset and 0.9643651 on test dataset

```
## Fitting SVM model
require(caret)
require(e1071)
svm.tune = tune.svm(label ~ ., data = traindigits[1:1000,], gamma = 10^(-6:-3), cost = 10^(1:2))
summary(svm.tune)
svm.model = svm(label ~ ., data = traindigits, method = "C-classification", kernel = "radial",
gamma = .001, cost = 10)
svm.pred  = predict(svm.model, testdigits[,-1])
acc = table(pred = svm.pred, true = testdigits[,1])
classAgreement(acc)
sum(diag(acc))/nrow(testdigits)
```

Parameters sigma=0.001 C=10
classification accuracy of 0.9765079 on test set

```
## Fitting a NN model
require(caret)
require(nnet)
fitControl <- trainControl(method="repeatedcv",
                 number=5,
                        repeats=1,
                        verboseIter=TRUE)

nnetFit<- train(label~., data=traindigits, method="nnet",
                             trControl=fitControl,

tuneGrid=expand.grid(.size=c(1,5,10,20,50,100,200,500),.decay=c(0,0.001,0.005,0.01,0.05,0.1,0.5
,1,2)))
predict_train<-predict(nnetFit,newdata=traindigits)
sum(predict_train==traindigits$label)/nrow(traindigits)
predict_test<-predict(nnetFit,newdata=testdigits)
sum(predict_test==testdigits$label)/nrow(testdigits)
```