Martin\_Alonso\_HW6

Martin Alonso

November 19, 2018

# HW6 Instructions

The data set comes from the Kaggle Digit Recognizer competition. The goal is to recognize digits 0 to 9 in handwritten images. Because the original data set is too large to be loaded in Weka GUI, I have systematically sampled 10% of the data by selecting the 10th, 20th examples and so on. You are going to use the sampled data to construct prediction models using naïve Bayes and decision tree algorithms. Tune their parameters to get the best model (measured by cross validation) and compare which algorithms provide better model for this task.

Due to the large size of the test data, submission to Kaggle is not required for this task. However, 1 extra point will be given to successful submissions. One solution for the large test set is to separate it to several smaller test set, run prediction on each subset, and merge all prediction results to one file for submission. You can also try use the entire training data set, or re-sample a larger sample.

## Section 1: Introduction

As always, we will start by loading the data and exploring the data sets.

# Load the required packages.  
require(dplyr)

## Loading required package: dplyr

##   
## 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

require(caret)

## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

require(rpart)

## Loading required package: rpart

require(e1071)

## Loading required package: e1071

require(stringr)

## Loading required package: stringr

# Load the datasets  
trainSet <- read.csv('train.csv')  
testSet <- read.csv('test.csv')  
  
# The training set has 42,000 observations and 785 variables.   
dim(trainSet)

## [1] 42000 785

str(trainSet[, 1:10])

## 'data.frame': 42000 obs. of 10 variables:  
## $ label : int 1 0 1 4 0 0 7 3 5 3 ...  
## $ pixel0: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel1: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel2: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel3: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel4: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel5: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel6: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel7: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel8: int 0 0 0 0 0 0 0 0 0 0 ...

summary(trainSet[, 1:10])

## label pixel0 pixel1 pixel2 pixel3   
## Min. :0.000 Min. :0 Min. :0 Min. :0 Min. :0   
## 1st Qu.:2.000 1st Qu.:0 1st Qu.:0 1st Qu.:0 1st Qu.:0   
## Median :4.000 Median :0 Median :0 Median :0 Median :0   
## Mean :4.457 Mean :0 Mean :0 Mean :0 Mean :0   
## 3rd Qu.:7.000 3rd Qu.:0 3rd Qu.:0 3rd Qu.:0 3rd Qu.:0   
## Max. :9.000 Max. :0 Max. :0 Max. :0 Max. :0   
## pixel4 pixel5 pixel6 pixel7 pixel8   
## Min. :0 Min. :0 Min. :0 Min. :0 Min. :0   
## 1st Qu.:0 1st Qu.:0 1st Qu.:0 1st Qu.:0 1st Qu.:0   
## Median :0 Median :0 Median :0 Median :0 Median :0   
## Mean :0 Mean :0 Mean :0 Mean :0 Mean :0   
## 3rd Qu.:0 3rd Qu.:0 3rd Qu.:0 3rd Qu.:0 3rd Qu.:0   
## Max. :0 Max. :0 Max. :0 Max. :0 Max. :0

# The dataset is also composed of a label, going from 0 to 9, and 784 pixel variables, all being ints.   
# The same can be said of the test set, sans for the label, which it lacks.

There is very little work to be done on the datasets before we start modeling the data. But, because of the size of the datasets, we will work with a subset of both of these. We’ll use ten percent of the original training set to train our model, and ten percent of the original test set to test the model.

# We'll set the seed to assure reproducibility.  
set.seed(766)  
  
# We'll then randomly select ten percent of the training and testing sets.  
trainSplit <- sample(nrow(trainSet), nrow(trainSet) \* .1)  
testSplit <- sample(nrow(testSet), nrow(testSet) \* .1)  
  
# And now we'll select the rows and build our new sets.  
trainSubSet <- trainSet[trainSplit, ]  
testSubSet <- testSet[testSplit, ]  
  
row.names(trainSet) <- NULL  
row.names(testSet) <- NULL

Aside from that, we are going to try to use the pixel information - which pixel lights up - to determine which number is formed, and use that to try and determine the correct labels for the test dataset.

## Section 2: Decision tree

# Decision tree model  
decTreeTrain <- rpart(label ~ ., data = trainSubSet, method = 'class',   
 control = rpart.control(cp = 0), minsplit = 100, maxdepth = 10)  
  
# Testing the accuracy of the tree on the training set  
trainPred <- data.frame(predict(decTreeTrain, trainSubSet))  
  
# Select the maximum likelihood number  
trainPred <- as.data.frame(names(trainPred[apply(trainPred, 1, which.max)]))  
colnames(trainPred) <- 'prediction'  
trainPred$number <- substr(trainPred$prediction, 2, 2)  
  
  
trainPred <- trainSubSet %>% bind\_cols(trainPred) %>% select(label, number) %>% mutate(label = as.factor(label), number = as.factor(round(as.numeric(number), 0)))  
  
# Confusion matrix on the training dataset shows 86.9 percent accuracy.   
confusionMatrix(trainPred$label, trainPred$number)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 377 0 5 2 2 6 4 0 7 1  
## 1 0 434 7 7 0 5 1 5 4 0  
## 2 3 2 341 4 4 2 11 5 18 7  
## 3 4 1 20 354 5 11 9 3 22 8  
## 4 3 3 11 4 362 6 3 9 8 14  
## 5 4 2 11 17 5 311 12 1 8 1  
## 6 12 1 13 3 11 18 376 0 7 3  
## 7 1 2 9 6 4 3 1 377 7 23  
## 8 4 3 13 9 4 13 6 4 359 13  
## 9 3 2 11 9 9 12 1 8 18 326  
##   
## Overall Statistics  
##   
## Accuracy : 0.8612   
## 95% CI : (0.8504, 0.8715)  
## No Information Rate : 0.109   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8457   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.91727 0.9644 0.77324 0.85301 0.89163 0.80362  
## Specificity 0.99287 0.9923 0.98510 0.97807 0.98392 0.98400  
## Pos Pred Value 0.93317 0.9374 0.85894 0.81007 0.85579 0.83602  
## Neg Pred Value 0.99104 0.9957 0.97370 0.98379 0.98835 0.98015  
## Prevalence 0.09786 0.1071 0.10500 0.09881 0.09667 0.09214  
## Detection Rate 0.08976 0.1033 0.08119 0.08429 0.08619 0.07405  
## Detection Prevalence 0.09619 0.1102 0.09452 0.10405 0.10071 0.08857  
## Balanced Accuracy 0.95507 0.9784 0.87917 0.91554 0.93777 0.89381  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.88679 0.91505 0.78384 0.82323  
## Specificity 0.98199 0.98522 0.98156 0.98081  
## Pos Pred Value 0.84685 0.87067 0.83879 0.81704  
## Neg Pred Value 0.98722 0.99071 0.97375 0.98158  
## Prevalence 0.10095 0.09810 0.10905 0.09429  
## Detection Rate 0.08952 0.08976 0.08548 0.07762  
## Detection Prevalence 0.10571 0.10310 0.10190 0.09500  
## Balanced Accuracy 0.93439 0.95013 0.88270 0.90202

# Let's run the prediction on the test set and see what we get.   
testPred <- data.frame(predict(decTreeTrain, testSubSet))  
testPred <- as.data.frame(names(testPred[apply(testPred, 1, which.max)]))  
colnames(testPred) <- 'prediction'  
testPred$number <- substr(testPred$prediction, 2, 2)  
  
# Seems about even but we cannot test the results unless we run the decision tree on the entire test set and sumbmit it to Kaggle.   
testComplete <- data.frame(predict(decTreeTrain, testSet))  
testComplete <- as.data.frame(names(testComplete[apply(testComplete, 1, which.max)]))  
colnames(testComplete) <- 'ImageId'  
testComplete$Label <- substr(testComplete$ImageId, 2, 2)  
testComplete$ImageId <- 1:nrow(testComplete)  
  
# Export the result set and run through Kaggle  
write.csv(testComplete, 'Decision Tree Test.csv')

After submitting the test results to Kaggle, the score was 0.76071. Not bad for a first attempt but can be improved, given that the tree was built on ten percent of the testing data and has not been pruned after the initial pre-pruning process.

## Section 3: Naive bayes

Now that we have tried identifying the numbers using a decision tree, it is time to try our hand using the Naive Bayes algorithm.

# Build the classifier.   
nbTrain <- naiveBayes(as.factor(label) ~ ., data = trainSubSet)  
  
# Again, test the results on the train set before submitting to Kaggle.   
nbTrainPred <- predict(nbTrain, trainSubSet, type = 'class')  
confusionMatrix(nbTrainPred, as.factor(trainSubSet$label))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 391 0 71 94 18 89 4 38 12 6  
## 1 1 450 41 64 20 32 30 41 121 32  
## 2 1 0 71 4 1 1 1 1 1 0  
## 3 0 1 11 71 0 2 0 1 2 0  
## 4 0 0 0 2 57 2 0 2 3 4  
## 5 0 0 2 1 2 16 3 0 4 0  
## 6 7 5 121 39 38 25 400 5 13 4  
## 7 0 0 1 3 0 0 0 76 0 0  
## 8 4 5 72 130 48 173 5 17 219 17  
## 9 0 2 7 29 239 32 1 252 53 336  
##   
## Overall Statistics  
##   
## Accuracy : 0.4969   
## 95% CI : (0.4817, 0.5121)  
## No Information Rate : 0.1102   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4401   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.96782 0.9719 0.17884 0.16247 0.13475 0.043011  
## Specificity 0.91254 0.8978 0.99737 0.99548 0.99656 0.996865  
## Pos Pred Value 0.54080 0.5409 0.87654 0.80682 0.81429 0.571429  
## Neg Pred Value 0.99626 0.9961 0.92085 0.91099 0.91138 0.914669  
## Prevalence 0.09619 0.1102 0.09452 0.10405 0.10071 0.088571  
## Detection Rate 0.09310 0.1071 0.01690 0.01690 0.01357 0.003810  
## Detection Prevalence 0.17214 0.1981 0.01929 0.02095 0.01667 0.006667  
## Balanced Accuracy 0.94018 0.9349 0.58811 0.57898 0.56565 0.519938  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.90090 0.17552 0.51168 0.8421  
## Specificity 0.93158 0.99894 0.87513 0.8382  
## Pos Pred Value 0.60883 0.95000 0.31739 0.3533  
## Neg Pred Value 0.98758 0.91335 0.94046 0.9806  
## Prevalence 0.10571 0.10310 0.10190 0.0950  
## Detection Rate 0.09524 0.01810 0.05214 0.0800  
## Detection Prevalence 0.15643 0.01905 0.16429 0.2264  
## Balanced Accuracy 0.91624 0.58723 0.69341 0.8402

# The results aren't as satisfactory - only 50.8 percent accuracy. But we'll still test it against the test set and check the results in Kaggle.   
  
nbTestPred <- predict(nbTrain, testSet, type = 'class')  
nbTestPred <- data.frame(nbTestPred)  
colnames(nbTestPred)[1] <- 'Label'  
nbTestPred$ImageId <- 1:nrow(nbTestPred)  
nbTestPred <- nbTestPred %>% select(ImageId, Label)  
  
# Write the csv for Kaggle submission  
write.csv(nbTestPred, 'Naive Bayes Classifier.csv', row.names = FALSE)

There is still work to be done on the Naive Bayes algorithm. There was no tuning done which, using the caret package, might have built a more robust algorithm. As is, the results scored 0.4984 on Kaggle - not as good as the Decision Tree classifier.

## Section 4: Algorithm performance comparison

We can see by the Confusion matrices shown above that the decision tree does a better job at classifying the results as it is missing less on many numbers. The naive Bayes classifier, however, has a lot of trouble distinguishing between the number 2 and the other numbers, meaning that the algorithm needs more work. But what has to be taken into account was that, due to available memory, only ten percent of the actual training set was used. This set was selected at random so, had we used another random set, or the entire training set, results might have been different. This is something worth investigating and pursuing, as it could lead to better results, given that the algorithms will have more data points from which to learn from.

## Section 5: Kaggle test result

The Kaggle test results were not great but they were quite encouraging. Having the decision tree score 0.7607 was a great first step that can be improved upon. The naive Bayes algorithm, on the other hand, started off at 0.4984 - not a great result as it is almost a coin-toss between the actual result and the predicted result. The model needs further work, both in the data being used to build the classifier, and the tuning methods that can help with the model underfitting.