Kyle\_Walter\_Homework\_7

Kyle Walter

3/7/2021

# Which Model excels at predicting Handwritten Numbers

## Introduction

As focused on the prior review, we looked at 1400 samples of training data in two different models. One being the naive bayes and the other being decision tree. These models preformed poorly at classifying numbers between 0 and 9 correctly. In order to find a system that works well for the task at hand, we’re going to try a few other classification algorithms and compare and contrast them. We’ll determin which is the best and run the predictions against the test set and submit to kaggle.

##Loading the preparing the data

To start we will reimport the training and test datasets from the prior exercise. To refresh memory, the data set contains 42 thousand records of hand written images that have been scaned in and converted to 784 pixel which have a number ranging from 0 to 255 letting us know how shaded each pixel is. In addtiion, we have a column called labels where each has been classifed as the number that was written in the image.

Due to limited computer resources a smaller version the of the training set with 1400 samples has been provided. This was created using a set generation methodology of taking every 30th record from the training set. In a similar methodology the test set was paired down to a version with 999 records.

Firt let’s load packages needed for today’s task and set R to point to the working directory where the training and test data is sitting.

require(data.table)

## Loading required package: data.table

require(caret)

## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

#set working director  
setwd("R:/Graduate School/IST707 - Data Analysis/Week 8")

Next, we will read in the training and test sets

#Read in the data  
trainSet <- fread("Kaggle-digit-train-sample-small-1400.csv")  
testSet <- fread("Kaggle-digit-test-sample1000.csv")

## Warning in fread("Kaggle-digit-test-sample1000.csv"): Discarded single-line  
## footer: <<?,>>

Due to the number of variables, I am not going to print out the structure of the Data frame, but the label column is read in by R as an interger field. So the preprocessing step we’ll take at this point in time is to convert it to factors so that algorithms can use it for classification.

trainSet$label <- as.factor(trainSet$label)

In addition to converting the column, we’ll also setup training controls. Using the caret package allows us to setup a controls that can be applied to all the models. In this case, we’re going to use a cross fold validation method. What this means is when a training set is fed into the model, it will be broken up into smaller chunck, and held out of the training so the model can test the results of it output. The model with then repeat it with each chunck and center on the model’s accurancy.

For this case we’ll set the cross validation to three meaning the training data will be broken into three groups. Two of them will be used to train, one to test and algorithm with cycle through each test to come to the model preformance over the three runs.

controls <- trainControl(method="cv", number = 3)

## Previous Models

As mentioned in the previous section there were two models that have been tried previously. The first is the naive bayes and the second was the decsion tree. Naive bayes works by calculating 0 to 1 between two classifiers. It is much better suited for a problem where the classification has only two options such as yes or no, spam or not spam, etc. And we can see that born out in the model.

Some notes on features that we’ll see pop up again when evaluating the new models, when using the caret package we can apply some preprocessing the steps to the data.

First we request the model ignore columns where all the values are equal. There are number of columns that are all white space and have a value of zero for every entry and provide no additional data. In addition we are transaforming the values to their Z score using the center and scale methods in the preprocess argument. What this does is is subtracts the mean from the all values and divides them by their standard deviation. This will cause most of the values to be between -3 and 3 which means helps the algorithm since Naive Bayes works on a logistic function, the sigmotic curve between -3 and 3 helps it gain additional information.

As we can see from our model this preforms poorly after cross validation is applied. Accuracy results are 43% and recall when tested on the various version is only 36%.

#naive bays  
set.seed(52)  
nbModel <- train(label~., data = trainSet, trControl = controls, method ="nb", preProcess = c("zv", "center","scale"))  
nbModel

## Naive Bayes   
##   
## 1400 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## Pre-processing: centered (622), scaled (622), remove (162)   
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 936, 931, 933   
## Resampling results across tuning parameters:  
##   
## usekernel Accuracy Kappa   
## FALSE NaN NaN  
## TRUE 0.4314776 0.3620097  
##   
## Tuning parameter 'fL' was held constant at a value of 0  
## Tuning  
## parameter 'adjust' was held constant at a value of 1  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were fL = 0, usekernel = TRUE and adjust  
## = 1.

The other model we tried with no major advantage was the decision tree. These are named for their tree like shape when drawn out and each branch is a decision based on the value of a particular variable.

For the training purposes we let the training data run down and build the entire tree, and then prune it so that decions of classification are being made generally about the data. This allows us to avoid over fitting the model in which outlyers in the training data are well classified but when tested against new data causing the model to get large amount of data wrong because of overly specific rules.

#Decision Tree  
set.seed(42)  
dtModel <- train(label~., data = trainSet, trControl = controls, method ="rpart")  
dtModel

## CART   
##   
## 1400 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 934, 934, 932   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa   
## 0.06650446 0.3693280 0.2947905  
## 0.07542579 0.3135340 0.2322320  
## 0.09448500 0.2442439 0.1535493  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.06650446.

As we can see from the outcome we gain a 37% accuracy with 29% reproducability by the model when testing in cross validation by using the decision tree. The ‘cp’ value tell the algorithm how agressively to generalize the data. This number ranges from 0 to 1. with 1 being the most agressive and 0 being the least. In this case by a medium agressiveness we achieve the maximum accuracy, but over all it doesn’t product results we can rely on.

## The new models

### Support Vector Machines

Support Vector Machines is a classification type that looks to maximize the distance between the two closets observations of different classes. It sets a classification boundary a mid point between oberservations. The distance between oberservation and the boundary is called the margin. By testing the observations on multiple dimesions it is able to calculate a boundary with high accuracy. This is a prime example of a supervised machine learning method as it use the data that has already been classifed as a test mechanism. By allow it to calculate the version of the model that produces the highest accuracy, we introduce cost, or recognition of point that are being misclassifed by the model.

By allowing some error we generally get better results with new data points into the mode. And as we can see with the output, but allowing error we get a classification accuracy across points of 92%. More importantly we get recall around 91% meaning as we introducted the model to various tests in cross validation, we we were able to maintain this high level of accuracy.

#SVM  
set.seed(23)  
supportModel <- train(label~., data = trainSet, trControl = controls, method = "svmRadial")

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.  
  
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

supportModel

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 1400 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 932, 937, 931   
## Resampling results across tuning parameters:  
##   
## C Accuracy Kappa   
## 0.25 0.8771303 0.8633383  
## 0.50 0.9021914 0.8912321  
## 1.00 0.9200471 0.9110997  
##   
## Tuning parameter 'sigma' was held constant at a value of 1.614032e-07  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 1.614032e-07 and C = 1.

### k nearest neighbor

This model works be classifying the various neighbors of a data point and those most commonly seen become the classification of the data point itself. It is a distance based formula, so similar to what we did with Naive Bays previously, we need to equalize the various data points.

Since the classifications are made based on their closets neighbors, the scale between various values should be on the same magnitude. To do this, we applied to remove the columns with only 1 value, on tranformed the values to their Z score as done above to the Naive Bayes. The other factor we tested was how many neighbors are within range to use. The tuneLength argument says test starting from 5 neighbors and increase by measure of 2. We always use odd numbers to avoid cases of ties.

#knn  
set.seed(147) #uses the same list of random numbers  
knnModel <- train(label~., data = trainSet, trControl = controls, method = "knn", preProcess = c("zv","center","scale"), tuneLength=10)  
knnModel

## k-Nearest Neighbors   
##   
## 1400 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## Pre-processing: centered (622), scaled (622), remove (162)   
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 934, 935, 931   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.8257018 0.8059990  
## 7 0.8249926 0.8051942  
## 9 0.8249667 0.8051340  
## 11 0.8128506 0.7915856  
## 13 0.8135613 0.7923530  
## 15 0.8021423 0.7795644  
## 17 0.7964290 0.7731722  
## 19 0.7864374 0.7619848  
## 21 0.7714464 0.7452265  
## 23 0.7643269 0.7372425  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

As we can see from the outcome the best result is 7 neighbors will decide the classification. This provides an 84% accuracy and 82% recall among the various tests.

### Random Forest

Our last model to review is Random Forest. Random forest works by taking randome samples of the data and building decision trees, then to test the data is ran down all of the different decision trees built and optimum answer amoung the trees is the classifier assigned. The challenge with decision tress is to figure out how many trees to build. Thankfully we can test that various models. Build too many trees the model can quickly become overfit and produce less accurate results with new data.

#random Forest  
set.seed(197)  
rfModel <- train(label~., data = trainSet, trControl = controls, method ="rf", tuneLengh=10)  
rfModel

## Random Forest   
##   
## 1400 samples  
## 784 predictor  
## 10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 934, 933, 933   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.7135691 0.6798611  
## 39 0.9049912 0.8943351  
## 783 0.8556947 0.8395077  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 39.

As we cane see from the output the random forest produces and outcome of 90% accuracy and nearly identical 89% recall in the ability to translate. As we see the optimal size of the forest 39 trees.

## Conclusion

So for this data set, we recommend utilize Support Vector Machines, although Random Forest is close second place behind it with nearly similiar stats and keeping in mind that these results come from a sampled data set. As for the k nearest neighbor model, there are features of hand writing depending upon the origin of the people sampled that could cause problems. A european 1 vs an Amerian 7. I personally have suffered 4s and 9s when pens are failing that look almost similar to people. 9 with a hood in handwritten form could also cause confusion with 3 or 8 and the model could easily have picked this up dropping it in accuracy.

That said here the prediction of the test set based on the most accurate model

summary(predict(supportModel,testSet))

## 0 1 2 3 4 5 6 7 8 9   
## 105 119 77 113 108 98 98 102 91 88