Alex-Hyman\_HW7

Alex Hyman

11/24/2018

## Preprocessing

Hand-written digit images (0-9) were provided as grayscale pixel intensity values for each pixel in the image. The data preprocessing steps included making the pixel values binary, where if the grayscale value of the pixel was greater than 20, the pixel value became a 1, if the grayscale pixel intensity value was less than or equal to 20, the pixel value became 0. A threshold of 20 was utilized because pixels with a grayscale value of 20 or less could have occurred due to a smear or an accidental mark, not as an intentional mark. This binarization of pixels also standardized the variables so methods that utilize distance as a part of the classification would not be affected too much. The label values were also converted to be factors so the algorithm would predict the class, not the regress on the pixel values and estimate a continuous value.

Three different supervised learning algorithms were used to predict which digit was drawn based on the pixel values: KNN, SVM, and Random Forest. To evaluate performance of the different models, a training set that consisted of 80% of the labeled data, and a test set that contained the remaining 20% of the labeled data was created. The best models would be chosen as the models that achieved the best accuracy on the test data. This method was chosen as cross-validation would take too long to complete. After the models were tested, the best tuned model was used to predict the digits on the Kaggle test set.

#Reading training data  
trainset <- read.csv("train.csv")  
  
#Reading test data  
testset <- read.csv("test.csv")  
testset <- ifelse(testset > 20, 1, 0)  
  
#separating labels from features  
labels <- trainset$label %>% as.factor  
  
#Getting deleting the training labels from the training set  
features <- trainset %>% select(-"label")  
#Making the features either 0 or 1 if the pixel has an intensity value greater than 20  
features <- ifelse(features > 20, 1, 0)  
  
  
set.seed(9999)  
#making the training index index of the data  
train\_ix <- createDataPartition(labels, p = 0.8, list = F)  
#train labels  
train\_labels <- labels[train\_ix]  
#test labels  
test\_labels <- labels[-train\_ix]  
#train features  
train\_features <- features[train\_ix,]  
#test features  
test\_features <- features[-train\_ix,]

### KNN

The first supervised classification model that tested was the K-nearest neighbor algorithm. The algorithm was tested with k-values of 1, 3, and 5. Each of the models took a while to train. This is likely because all data that is used as the training data in the model is stored in the environment and the distance of each training point is measured against the distance of each of the test points.

Each of the three models performed well as they all had an accuracy of about 0.96, with the k=5 model scoring the highest in terms of test accuracy with a test accuracy of 0.962. I would be hesitant to use this model with larger datasets as it will take a long time to predict classes, but for this training set, the performance was pretty good! The reason this model works fairly well is that the data set is large enough that pixels important to certain numbers will be used multiple times to provide a decent idea of which pixels are important when predicting the class. However, if the image size was bigger or if the digits did not take up most of the area of the image, KNN would not perform well at all.

#KNN-1  
knn.1 <- knn(train = train\_features, test = test\_features, cl = train\_labels, k = 1)  
confusionMatrix(knn.1, test\_labels) #confusion matrix of the knn

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 818 0 3 2 0 2 1 0 2 5  
## 1 2 925 5 2 4 2 4 10 15 1  
## 2 0 4 798 7 0 0 1 3 3 1  
## 3 0 0 8 832 0 17 1 0 12 9  
## 4 0 1 0 1 774 0 1 1 2 11  
## 5 1 1 0 8 0 719 0 0 5 2  
## 6 4 0 1 0 3 10 819 0 6 0  
## 7 0 1 14 2 1 2 0 845 1 16  
## 8 1 2 2 11 1 2 0 0 752 4  
## 9 0 2 4 5 31 5 0 21 14 788  
##   
## Overall Statistics  
##   
## Accuracy : 0.9612

#KNN-3  
knn.3 <- knn(train = train\_features, test = test\_features, cl = train\_labels, k = 3)  
#confusion matrix of the knn  
#0.9618   
confusionMatrix(knn.3, test\_labels)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 817 0 3 3 0 2 2 0 1 5  
## 1 1 926 4 2 5 2 2 10 21 3  
## 2 0 4 802 5 0 0 1 2 2 1  
## 3 0 0 6 828 0 16 0 0 12 10  
## 4 0 1 0 0 767 0 1 2 2 7  
## 5 1 1 0 10 0 719 1 0 4 1  
## 6 5 0 0 0 6 11 820 0 4 0  
## 7 0 2 16 6 0 1 0 847 2 9  
## 8 0 1 1 10 0 3 0 0 750 2  
## 9 2 1 3 6 36 5 0 19 14 799  
##   
## Overall Statistics  
##   
## Accuracy : 0.9618

#KNN-5  
knn.5 <- knn(train = train\_features, test = test\_features, cl = train\_labels, k = 5)  
#confusion matrix of the knn  
#0.962  
confusionMatrix(knn.5, test\_labels)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 813 0 3 3 0 4 2 0 1 5  
## 1 2 928 6 2 7 3 2 13 22 2  
## 2 0 3 801 6 0 0 1 3 2 1  
## 3 0 0 4 833 0 15 0 0 10 11  
## 4 0 1 0 2 767 0 1 2 2 6  
## 5 2 1 0 7 0 717 0 0 4 0  
## 6 6 1 1 0 6 12 821 0 7 0  
## 7 1 1 16 6 1 2 0 845 2 8  
## 8 2 0 2 6 0 1 0 0 751 3  
## 9 0 1 2 5 33 5 0 17 11 801  
##   
## Overall Statistics   
## Accuracy : 0.962

### SVM

The SVM supervised learning algorithm was also used to classify which hand-drawn digit was displayed by the binarized pixel values. The SVM was tested with a cost of 1 and a cost of 0.5. The SVM with a cost of 0.5 performed slightly better with a testing accuracy of 0.924 vs. the testing accuracy of 0.919 of the model with a cost of 1. This meant that the model performed slightly better when misclassifications were penalized less than the default model. The SVM model performed quite well, which is somewhat expected with higher dimensional data. The model did take a decent amount of time to train the model, but the predictions were able to achieved much quicker than the KNN model; however performance in the SVM model was not as high as performance was within the KNN model.

#svm with linear kernel  
svm.linear <- svm(x = train\_features, y = train\_labels, kernel = "linear", cost = 1)

#Linear predictions  
preds.linear <- predict(svm.linear, test\_features)  
#confusion matrix  
#0.9189  
print(confusionMatrix(preds.linear, test\_labels))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 789 0 7 6 0 4 7 1 5 6  
## 1 0 914 10 3 2 3 7 6 17 5  
## 2 3 5 741 14 13 2 8 7 9 5  
## 3 2 3 20 785 0 37 1 3 34 16  
## 4 2 1 8 2 749 8 10 13 5 33  
## 5 9 4 3 27 3 658 9 1 5 2  
## 6 13 0 12 0 6 10 782 2 2 0  
## 7 0 3 12 8 6 6 0 823 2 17  
## 8 6 5 18 18 3 24 3 2 727 6  
## 9 2 1 4 7 32 7 0 22 6 747  
##   
## Overall Statistics  
##   
## Accuracy : 0.9189   
## 95% CI : (0.9128, 0.9246)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9098   
## Mcnemar's Test P-Value : NA

#svm with linear kernel  
svm.linear2 <- svm(x = train\_features, y = train\_labels, kernel = "linear", cost = 0.5)

#linear predictions  
preds2 <- predict(svm.linear2, test\_features)  
#confusion matrix  
#0.9245  
print(confusionMatrix(preds2, test\_labels))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 788 0 7 5 0 4 8 1 5 6  
## 1 0 912 10 2 2 3 6 6 16 5  
## 2 4 6 755 14 14 3 8 7 9 4  
## 3 2 5 14 794 0 38 0 2 29 15  
## 4 2 1 8 2 749 6 7 12 5 26  
## 5 10 3 3 20 3 663 9 1 4 3  
## 6 13 0 10 1 6 8 786 2 2 0  
## 7 1 3 11 9 5 5 0 826 2 18  
## 8 5 5 13 17 2 24 3 2 734 5  
## 9 1 1 4 6 33 5 0 21 6 755  
##   
## Overall Statistics  
##   
## Accuracy : 0.9245   
## 95% CI : (0.9186, 0.9301)  
## No Information Rate : 0.1115   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9161   
## Mcnemar's Test P-Value : NA

### Random Forest

The random forest model was also trained to classify the different hand-written digits in the dataset. Three different models were trained to classify the digits, one with 10 trees, a second with 100 trees, and a third with 500 trees. When evaluating the performance of the models, the random forest model with 500 trees achieved the highest testing accuracy with a test accuracy of 0.967. This model performed the best of all models trained with the testing data. Because of the high performance, I would expect this model to perform the best on the unlabeled test set provided by Kaggle. This intuitively makes sense as ensemble methods generally have higher performance than other models, even though a single decision tree could not perform nearly as well as a lot of trees on random subsets of data.

Additionally, the training and predicting on the data did not take too long for the models trained with 10 or 100 trees; however the model trained with 500 trees took a while to train, but not too long to predict. If I were to choose a model to use in production, I would likely use the random forest with 100 trees due to the short training time.

#random forest 10 trees  
random.forest.10 <- randomForest(x = train\_features, y = train\_labels, ntree = 10)  
#random forest predictions  
preds.forest.10 <- predict(random.forest.10, test\_features)  
#confusion matrix  
#0.9404  
print(confusionMatrix(preds.forest.10, test\_labels))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 802 0 5 4 1 4 1 0 6 7  
## 1 2 916 2 1 0 1 3 6 11 4  
## 2 1 3 794 15 2 2 0 8 10 2  
## 3 0 2 9 786 1 19 0 2 16 17  
## 4 2 2 3 1 769 2 1 10 6 15  
## 5 5 5 0 26 3 703 7 1 12 4  
## 6 8 2 3 0 5 7 813 0 2 1  
## 7 0 2 8 7 4 3 0 837 3 10  
## 8 4 2 8 19 4 9 2 0 735 7  
## 9 2 2 3 11 25 9 0 16 11 770  
##   
## Overall Statistics  
##   
## Accuracy : 0.9439

#random forest 100 trees  
random.forest.100 <- randomForest(x = train\_features, y = train\_labels, ntree = 100)  
#random forest predictions  
preds.forest.100 <- predict(random.forest.100, test\_features)  
#confusion matrix  
#0.9668   
print(confusionMatrix(preds.forest.100, test\_labels))

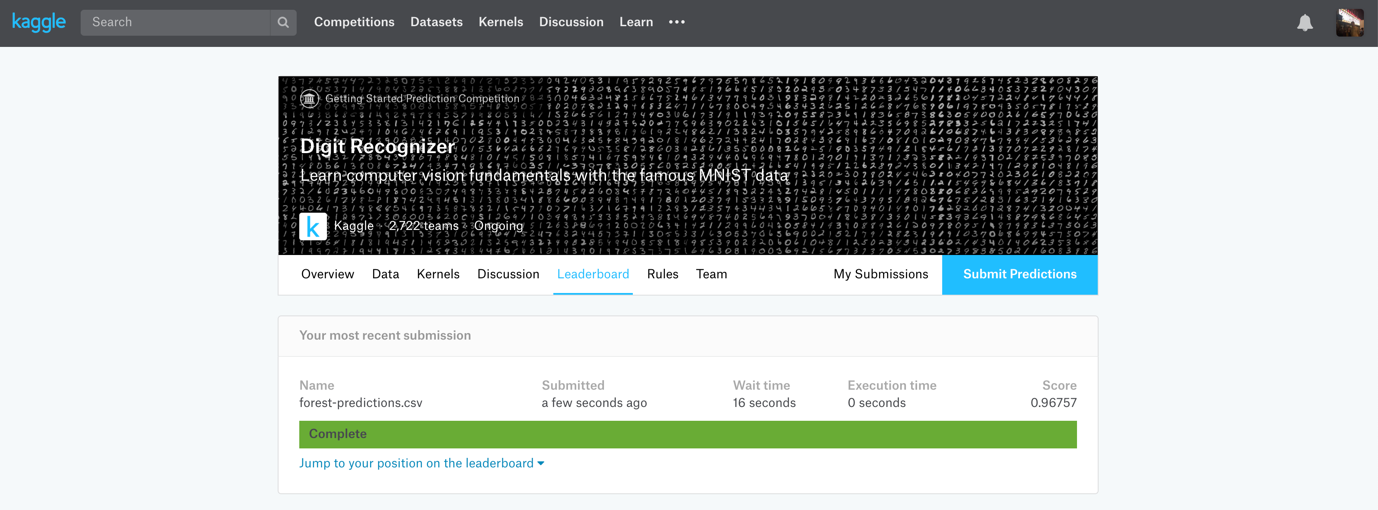
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 809 0 2 1 0 2 1 0 1 6  
## 1 0 916 1 0 1 2 3 4 6 1  
## 2 1 7 816 10 0 4 1 9 3 3  
## 3 0 4 3 830 0 12 0 0 10 17  
## 4 0 3 2 1 786 0 1 5 3 6  
## 5 3 1 0 7 0 721 2 0 3 0  
## 6 5 0 0 1 7 6 818 0 6 0  
## 7 1 2 5 4 1 1 0 850 1 6  
## 8 6 2 5 10 2 7 1 0 767 6  
## 9 1 1 1 6 17 4 0 12 12 792  
##   
## Overall Statistics  
##   
## Accuracy : 0.9653

#random forest 500 trees  
random.forest.500 <- randomForest(x = train\_features, y = train\_labels, ntree = 500)  
#random forest predictions  
preds.forest.500 <- predict(random.forest.500, test\_features)  
#confusion matrix  
#0.967  
print(confusionMatrix(preds.forest.500, test\_labels))

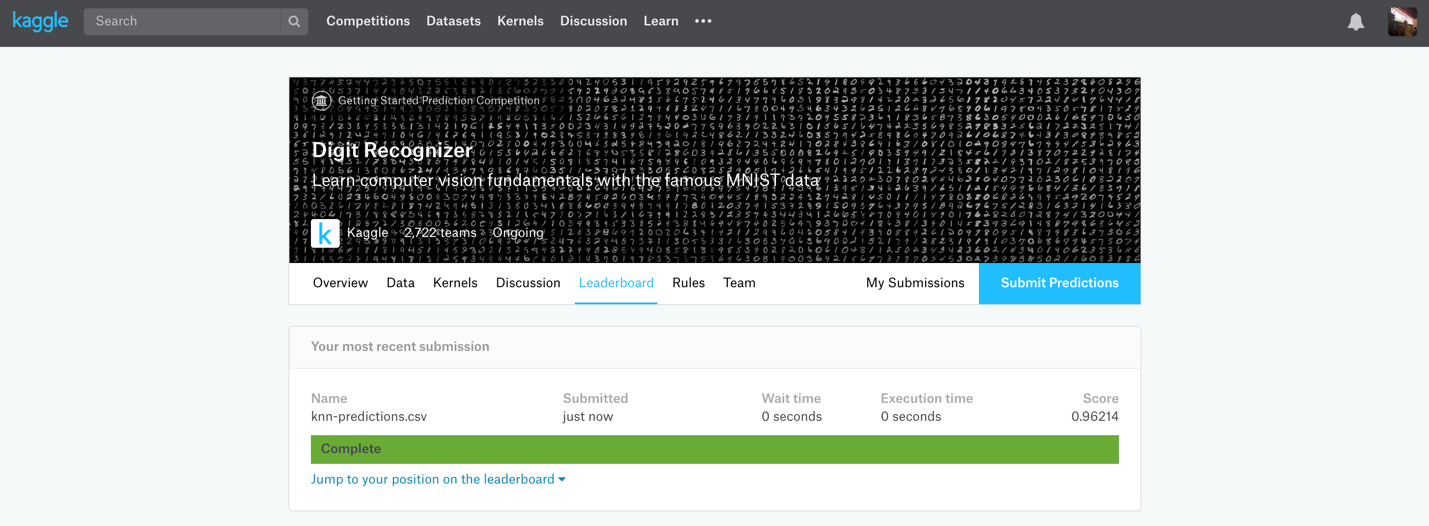
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 811 0 2 2 0 1 1 1 0 6  
## 1 0 919 1 0 1 1 3 5 7 1  
## 2 0 6 816 8 1 2 0 9 5 1  
## 3 0 5 4 830 0 10 0 0 10 17  
## 4 0 2 1 1 785 1 0 6 2 9  
## 5 1 1 0 8 0 727 0 0 3 1  
## 6 7 0 0 1 7 7 820 0 2 0  
## 7 1 2 5 4 2 1 0 847 1 6  
## 8 5 0 4 10 2 6 3 0 771 4  
## 9 1 1 2 6 16 3 0 12 11 792  
##   
## Overall Statistics  
##   
## Accuracy : 0.9669

## Kaggle Predictions

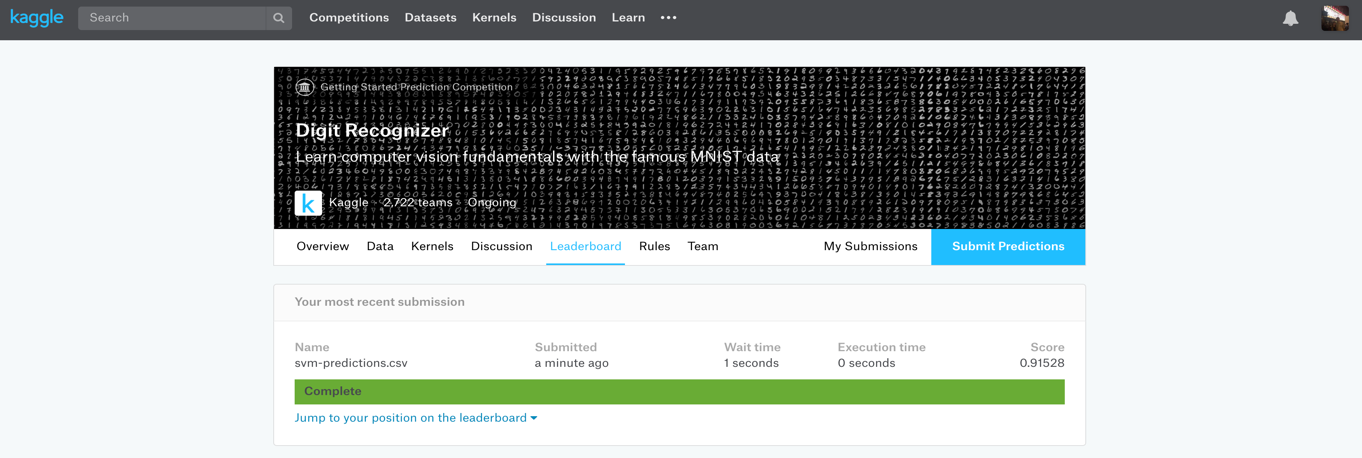
While I would have only used the 500 tree random forest model if I had to choose one model, I submitted the predictions for the best random forest, KNN, and SVM models to Kaggle. The first predictions sent to Kaggle were from the random forest model with 500 trees. The performance with the unlabeled Kaggle data scored an accuracy of 0.968, which was just about how well the model performed on the hold out group (0.967)!



The KNN model with the five nearest neighbors was expected to be the next best model as the hold out group had an accuracy of 0.962. When submitting the predictions on the unlabeled Kaggle test set, the model again had an accuracy of 0.962. This means that the model does not overfit the data and the test data chosen as our hold out group was an accurate representation of what real world data looks like.



The last model submitted to Kaggle was the SVM with a cost of 0.5. This was the algorithm that performed the lowest in terms of the holdout method, but still performed relatively well with an accuracy around 0.92. The SVM model scored about as expected with an accuracy of 0.915



As expected, the random forest algorithm performed the best on the unlabeled data submitted to Kaggle, with KNN scoring close behind. These two methods were somewhat expected to work the best because they are both either a true ensemble method, or a pseudo ensemble method in the case of KNN. The Random Forest algorithm would be the method of choice for this dataset primarily due to the significantly shorter prediction time required in Random Forest vs. KNN.

#Random forest prediction  
forest.prediction <- predict(random.forest.500, testset)  
#IDs for predictions  
test.ids <- seq\_along(forest.prediction)  
#forest data frame  
forest.df <- data.frame(ImageId = test.ids, Label = forest.prediction)  
#Saving prediction  
#accuracy: 0.96757  
write.csv(forest.df, "forest-predictions.csv", row.names = F)  
  
  
#SVM prediction  
svm.prediction <- predict(svm.linear2, testset)  
#IDs for predictions  
test.ids <- seq\_along(svm.prediction)  
#svm data frame  
svm.df <- data.frame(ImageId = test.ids, Label = svm.prediction)  
#Saving prediction  
#accuracy: 0.91528  
write.csv(svm.df, "svm-predictions.csv", row.names = F)  
  
#KNN prediction  
knn5.prediction <- knn(train = train\_features, test = testset, cl = train\_labels, k = 5)  
#IDs for predictions  
test.ids <- seq\_along(knn5.prediction)  
#knn data frame  
knn.df <- data.frame(ImageId = test.ids, Label = knn5.prediction)  
#Saving prediction  
#accuracy: 0.96214  
write.csv(knn.df, "knn-predictions.csv", row.names = F)