Random Forests

Muhammad Banatwalla

November 30, 2020

1 Introduction

For this homework, we further explore the task of classification in regards to the font data we have been using. We again attempt to improve our results by trying a new method for classification. This time, we explore the random forest algorithm. Through this algorithm, we evaluate its success and look for ways to improve our initial results.

1.1 Description of Data

We used the same data set as we did in HW4. The 3 fonts we are working with are Century, Ebrima, and Gill, coming from the ”University of California Irvine Repository for Machine Learning Datasets.” For Class 1 we used the Century font with a size of 1649. We undersampled this class by 350 random cases after selection of the training and testing sets. For Class 2 we used the Ebrima font with a size of 1723, and for Class 3 we used the Gill font with a size of 1459. Class 1, Class 2, and Class 3 were unioned into a larger data set we call DATA with size 4831. Further, we then standardized DATA to get a new set called SDATA. Each case in the dataset describes numerically a digitized image of some specific character typed in the particular font.The image associated to this row has 400 features (or numerical descriptors). These 400 features represent the gray levels of the 400 pixels of the specific image. The last column is the fonts column that contains the labels of the fonts specifying what font each case belongs to.

1.2 PCA

Since our dataset contains 400 features, we performed dimension reduction on our dataset through PCA. PCA works by computing principal components which are new variables that are constructed as linear combinations of the initial variables. PCA tries to put maximum possible information in the first component then the maximum remaining in the second component and so on. We found 103 new features that had a proportion of variance explained of 95%. For all of our calculations, we will only be using these 103 new features.

2 Random Forest

Before one talks about random forest it is important to understand decision trees,

decision trees are a decision making algorithm that can be used for both regression and classification tasks. While they are easy to build, use and interpret, they are also unfortunately very inaccurate. They are very inflexible making them bad at interpreting new data that comes in.The random forest algorithm is in fact the decision tree algorithm applied again and again. Random forest makes 100s of decision trees for any particular case and then applies a voting like system to determine the class that the case will fall into. Thus random Forest uses the simplicity of decision trees but the variability of trees also adds flexibility for all new incoming data!

The randomForest() function in R can take in a various number of parameters depending on the structure of the data. This function implements Breiman's random forest algorithm. The function can be used for both regression and classification tasks. For our purposes, we will use the parameters formula, data, ntree, mtry and importance. There are many other parameter options such as sampsize, nodesize, maxnodes, etc. This function returns an object of class randomForest with a list of values. The following table shows the outputs with explanations:

Output Value	Description
call	the original call to randomForest
type	either regression, classification, or unsupervised
predicted	predicted values of the input data
importance	a matrix with mean decrease values
importanceSD	standard errors
localImp	a matrix containing casewise importance measures
ntree	number of trees grown
mtry	number of predictors sampled for spliting at each node
forest	a list that contains the entire forest
err.rate	vector error rates of the prediction
confusion	the confusion matrix of the prediction

votes	gives the fraction or number of (OOB) 'votes' from the random forest
oob.times	number of times cases are out of bag

Table 1: randomForest() Output Description

For our implementation of the random forest algorithm we will use a training data set that will amount to around 80% of our data. We will run PCA analysis and get the features that contribute to 95% of variation in our data and then run our random forest algorithm on those features. The number of features contributing to the 95% variance in our data was 103, so we set our mtry value to the square root of 103 which is approximately 10. We set the importance to be true as we wanted to find which features perform the best in discriminating between the classes.

rf = randomForest(fonts ~., data=trainset, ntree=300, mtry=10,importance=TRUE) The predict function takes in the arguments for the model function for which the predic tions are required and then takes in the argument of a new data that needs to be predicted. In our case, the model data was our random forest model on the training set and the new data was our test set. The output was the predictions made on the test set. predict(rf,newdata=testset1)

We will evaluate the confusion matrices for both the training set and the test set. The randomForest function automatically outputs the confusion matrix whereas the predict func tion outputs a vector that we table together to create the confusion matrix. We will visually display the accuracy values we get from these matrices on graphs to further analyze the results.

2.1 Initial Computations

We first ran the random forest algorithm with 100 trees. We compare the confusion matrices and global accuracies:

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	76%	18%	5%
Actual: EBRIMA	13%	80%	7%
Actual: GILL	9%	13%	78%

Table 2: Confusion Matrix for ntree=100 (Training Set)

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL

Actual: CENTURY	87%	11%	2%
Actual: EBRIMA	11%	80%	10%
Actual: GILL	9%	13%	77%

Table 3: Confusion Matrix for ntree=100 (Testing Set)

The confusion matrix of the random forest algorithm with a 100 trees on the training set yields an overall accuracy of 78%, It correctly classifies century 76% of the times, Ebrima 80% of the times and Gill 78% of the times. The confusion matrix of the random forest algorithm with 100 trees on the test set yields an overall accuracy of 83%, It correctly classifies century 87% of the times, Ebrima 80% of the times and Gill 77% of the times.

	Train	Test
Overall Accuracy	78%	83%
Confidence Interval	(77%, 79%)	(81%, 85%)

Table 4: RF Accuracy and C.I. (ntree = 100)

The overall accuracy of the computation improved from the training to testing set. The correct classification of the Century font improved from 76% to 87%, this accounts for the improvement in performance of the testing set. The accuracy for Ebrima stayed the same and only increased 1% for Gill. The confidence intervals of the train and test set do not overlap so this means that there is a significant difference in the performance on these sets. Although the intervals do not overlap, they are very close. Generally, we do not want to see a huge difference in intervals but a slight improvement.

After this initial computation, we ran the random forest algorithm using different values of ntree to find the value that yields the best performance. Below we display confusion matrices and graphs where we can analyze our results and choose a value for best ntree.

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	87%	10%	3%
Actual: EBRIMA	20%	69%	11%

Actual: GILL	13%	15%	72%

Table 5: Confusion Matrix for ntree=10 (Testing Set)

The above is the confusion matrix of the random forest algorithm with 10 trees on the test set. It yields an overall accuracy of 79%.It correctly classifies century 87% of the times, Ebrima 69% of the times and Gill 72% of the time.

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	88%	10%	2%
Actual: EBRIMA	14%	77%	9%
Actual: GILL	9%	14%	77%

Table 6: Confusion Matrix for ntree=50 (Testing Set)

The above is the confusion matrix of the random forest algorithm with 50 trees on the test set. It yields an overall accuracy of 83%. It correctly classifies century 88% of the times, Ebrima 77% of the times and Gill 77% of the time.

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	87%	11%	2%
Actual: EBRIMA	10%	82%	8%
Actual: GILL	9%	12%	79%

Table 7: Confusion Matrix for ntree=200 (Testing Set)

The above is the confusion matrix of the random forest algorithm with 200 trees on the test set. It yields an overall accuracy of 84%. It correctly classifies century 87% of the times, Ebrima 82% of the times and Gill 79% of the time.

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	87%	11%	2%
Actual: EBRIMA	11%	80%	9%
Actual: GILL	9%	13%	78%

Table 8: Confusion Matrix for ntree=300 (Testing Set)

The above is the confusion matrix of the random forest algorithm with 300 trees on the test set. It yields an overall accuracy of 83%. It correctly classifies century 87% of the times, Ebrima 80% of the times and Gill 78% of the times.

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	87%	11%	2%
Actual: EBRIMA	10%	80%	10%
Actual: GILL	9%	13%	78%

Table 9: Confusion Matrix for ntree=400 (Testing Set)

The above is the confusion matrix of the random forest algorithm with 400 trees on the test set. It yields an overall accuracy of 83%. It correctly classifies century 87% of the times, Ebrima 80% of the times and Gill 78% of the times.

We first plot the performance of each font versus the ntree values:

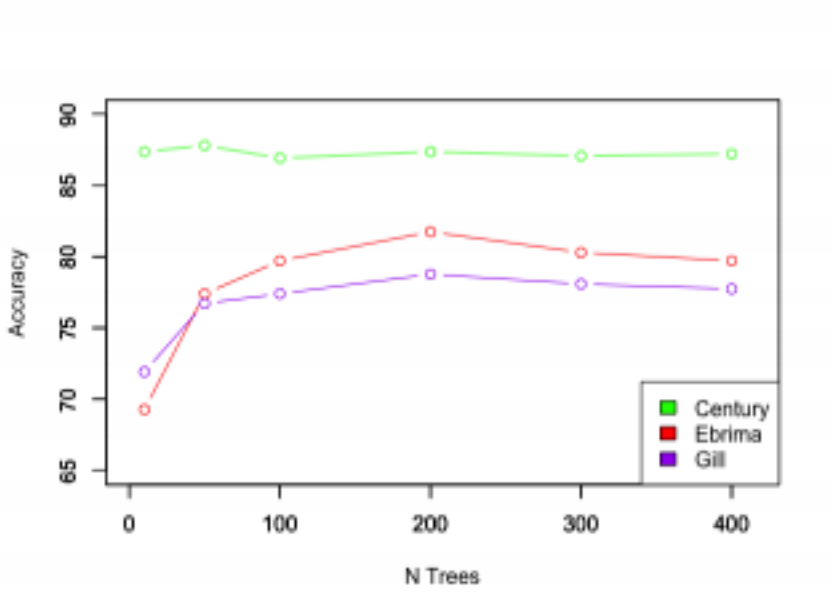

Figure 1: N Trees v. Fonts Accuracy

The above graph shows the accuracy of our random forest classification task on each of the fonts. The accuracy is presented on the vertical axis and the number of trees on the horizontal axis. We can tell that the Century accuracy peaks at 50 trees

where we achieve almost 80% accuracy, i.e our random forest classifier correctly classified the font “Century” in 80% of the cases in our test set. We can also tell that the accuracy peaks at 200 trees for Ebrima where we achieve around 82% accuracy, i.e our random forest classifier correctly classified the font “Ebrima” in 80% of the cases in our test set. We can tell that the accuracy also peaks at 200 trees for Gill where we achieve around 78% accuracy, i.e our random forest classifier correctly classified the font “Gill” in 78% of the cases in our test set.

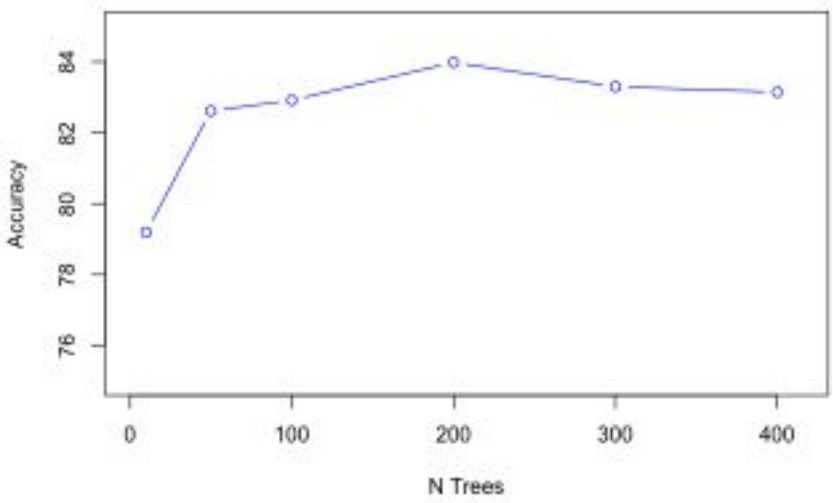

Figure 2: N Trees v. Accuracy

The above graph shows the overall accuracies of our random forest classification corre sponding to different numbers of trees. The accuracy is presented on the vertical axis and the number of trees on the horizontal axis. From this graph, as well as looking at the accuracy values, we were able to see that ntree=200 yielded the best results. This is what we chose as our best tree value.

We further examine the confidence intervals for the accuracies with ntree=200.

	Confidence Interval
Overall Accuracy	(82%, 86%)
Century Accuracy	(85%, 90%)
Ebrima Accuracy	(79%, 85%)
Gill Accuracy	(76%, 82%)

Table 10: 95% C.I. for Accuracy ntree=200

As per the confidence interval we can say with 95% confidence that when we take

into 6

account the randomness appearing due to differences in sample, we predict our model to be between 82% to 86% accurate overall. We can say with 95% confidence that when we take into account the randomness appearing due to differences in sample, we predict our model to be between 85% to 90% accurate for Century font. We can say with 95% confidence that when we take into account the randomness appearing due to differences in sample, we predict our model to be between 79% to 85% for Ebrima font. We can say with 95% confidence that when we take into account the randomness appearing due to differences in sample, we predict our model to be between 76% to 82% accurate for Gill font.

2.2 Computations with Best N Tree

Since the performance on the test set depends largely on the training set used to create the random forest model, we decided to check the stability of our results by trying a different training set. Here, we ran the random forest model using a new training set and test set with the best number of trees found before which was 200. We created a 95% confidence interval to compare the accuracies of the predictions for the test set from the two different models that used different training sets:

	Predicted: CENTURY	Predicted: EBRIMA	Predicted: GILL
Actual: CENTURY	87%	10%	3%
Actual: EBRIMA	10%	82%	8%
Actual: GILL	7%	11%	82%

Table 11: Confusion Matrix for ntree=200 (Model 2)

Above is the confusion matrix using ntree = 200 and a new random training and testing set. This yielded an overall accuracy of 85%. Additionally, 87% of the Century font were correctly predicted, 82% of the Ebrima font were correctly predicted and 82% of the Gill font were correctly predicted.

We created a 95% confidence interval to compare the accuracies of the predictions for the test set from the two different models that used different training sets:

C.I. Type	Model 1	Model 2
Overall Accuracy	(82%, 86%)	(83%, 87%)
Century Accuracy	(85%,	(85%,

	90%)	90%)
Ebrima Accuracy	(79%, 85%)	(78%, 86%)
Gill Accuracy	(76%, 82%)	(78%, 87%)

Table 12: C.I. Comparison (ntree = 200) Model 1 & Model 2

We can be 95% confident that true overall accuracy for the test set using the first model is between 82% and 86% and the second model is between 83% and 87%. The 95% confidence interval for the overall accuracy of both models has a wide overlap. That indicates that the unknown true accuracy for the test set if we change the training sets are similar. This is good news, we have low variance which is one of the goals of any machine learning model.

It means that different splits in the training set lead to similar results and high predictive accuracy. To further investigate, we checked the individual accuracies for each font from the two models. The accuracies for the predictions off the test set also had a wide overlap for the three different fonts. This further confirmed that we will have similar results if we change the training set. This is one of the benefits of bagging and random forest compared to single decision trees. Decision trees can have high variance where different splits in the training data can lead to very different trees. This can be very harmful because the model will give different results when using new data that it has never seen before.

2.2.1 Feature Importance

The randomForest function outputs an importance for each of the features used in the algorithm.Random Forest has two methods of calculating feature importance: mean decrease impurity and mean decrease accuracy. We plot them from our best randomForest model (ntree=200) to analyze:

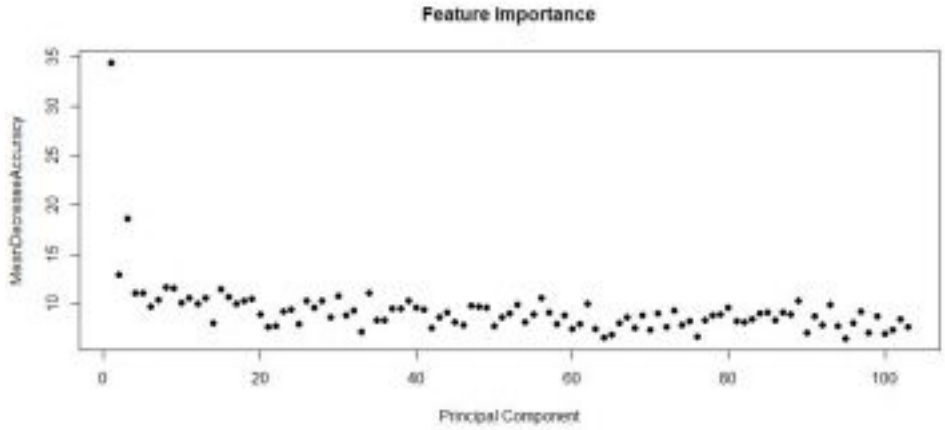

Figure 3: Feature v. Mean Decrease Accuracy

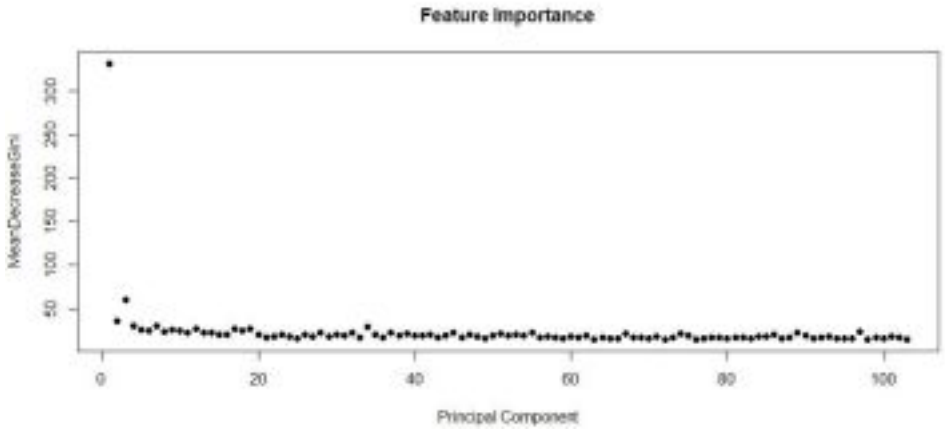

Figure 4: Feature v. Mean Decrease Gini

The first plot shows the importance of features by calculating how much the accuracy in predictions will decrease on the out of bag samples if that feature was dropped from the model. So, if principal component 1 was eliminated from the random forest model the accuracy would decrease by 45%. This makes sense because when we reduced the dimension of our dataset through principal component analysis, the majority of variation explained by the dataset is in pc1. The second most information is found in principal component 2, then 3, etc. It is interesting that off the random forest model the importance is similar but not in the exact order as from pca.

The second plot with MeanDecreaseGini vs Principal components plots the importance of the features by calculating the total decrease in node impurity that results from splits over that feature, averaged over all trees. Hence, the larger the mean decrease gini the more important the feature is by this method of calculating importance.

The results from both plots indicate that across all of the trees considered in the random forest, principal component 1, principal component 3, and principal component 2 are by far the three most important variables.

2.3 Random Forest with Binary Classifiers

In order to try to increase the accuracy of our classifier we decided to train multiple binary classifiers. Sometimes classifiers struggle when they need to classify more than two classes, training the model to classify one font versus the rest could make it easier for the classifier to learn and give better results. Therefore, we built three random forest models RF1 which classifies century fonts versus not century fonts. RF2 which classifies Ebrima fonts versus not Ebrima fonts and RF3 which classifies Gill fonts versus non Gill fonts. We split the classes by creating a target vector filled with zeros and ones, where 1 meant the case was the font we are distinguishing between and 0 meant it was not the font.

When we split our data for RF1 to classify between Century fonts and non Century fonts the training set had 1199 cases for Century and 2545 cases for not Century. Our dataset is unbalanced. This can make the classifier biased toward the class with more cases “not

Century” because the classifier may learn the classes with more samples better and remains weak on the smaller classes. So we balanced our dataset before training our three classifiers RF1, RF2, and RF3.

	Predicted: CENTURY	Predicted: NOT CENTURY
Actual: CENTURY	76%	24%
Actual: NOT CENTURY	5%	95%

Table 13: Confusion Matrix for ntree=200 (Century v. Not Century)

Overall Accuracy	85%
Confidence Interval	(83%, 87%)

Table 14: RF Accuracy and C.I. (Century v. Not Century)

After balancing the cases, century had 2545 cases and not century had 2545 cases. The overall accuracy of the model was 85%. The classifier did a really good job at accurately pre dicting cases that were not century 94%.The computing time for the algorithm was approx imately 47 seconds. Unfortunately, only 76% of the true class century cases were predicted by the model to be century fonts.

	Predicted: EBRIMA	Predicted: NOT EBRIMA
Actual: EBRIMA	67%	33%
Actual: NOT EBRIMA	5%	95%

Table 15: Confusion Matrix for ntree=200 (Ebrima v. Not Ebrima)

Overall Accuracy	88%
Confidence Interval	(86%, 90%)

Table 16: RF Accuracy and C.I. (Ebrima v. Not Ebrima)

Before cloning we had 2486 for not Ebrima and we had 1378 cases for Ebrima and after balancing both classes had 2486 cases.The classifier did a great job at predicting classes that were not Ebrima but unfortunately did not do well at predicting classes that were Ebrima.The computing time was approximately 49 seconds.

	Predicted: GILL	Predicted: NOT GILL
Actual: GILL	82%	18%
Actual: NOT GILL	5%	95%

Table 17: Confusion Matrix for ntree=200 (Gill v. Not Gill

Overall Accuracy	92%
Confidence Interval	(91%, 94%)

Table 18: RF Accuracy and C.I. (Gill v. Not Gill)

The training set had 1167 cases of font gill and 2577 of fonts that were not gill. After balancing, the training set had 2486 cases of font gill and cases of not font gill. We trained the random forest model and evaluated the accuracy of the model on the test set. The confusion matrix is shown above. RF3 was the best binary classifier thus far, it predicted 82% of cases of the test set that belong to font gill accurately.
The computing time for each of the three binary classifiers was less than one minute. Even with the larger datasets to balance out the two classes the random forest model ran pretty fast. The short computing time could be because it only has to classify two classes. All three binary classifiers consistently predict the not font on the test set with 95% accuracy. Although this is good news we would want our classifiers to have a better predictive accuracy for the fonts. It was concerning that only 76% of true class century were classified correctly by the RF1 model and that only 67% of true class ebrima fonts were classified correctly by the RF2 model. Therefore, we focused on trying to increase the results of how well the three binary classifiers predict the font since we know all three can predict well if it isn't the font. In particular we focused first on RF2 since it only accurately predicted 67% of true class ebrima. To try to increase results we tried different methods of balancing the data. We tried undersampling the not font instead of the original oversampling of the font cases. This did not change the accuracy of ebrima. Since RF1, RF2, AND RF3 was used with the best number of trees (200) that we got when we did our original model that classified all three fonts. We decided to check if a different number of trees would increase the accuracy. We tried 300,400,500,600, and 700 trees for ebrima versus not ebrima model. We got the best results with 700 trees but it only increased the accuracy of gill to 68%.
We now check how well best RF classified each font versus the rest.

	Predicted: CENTURY	Predicted: NOT CENTURY
Actual: CENTURY	87%	13%
Actual: NOT CENTURY	9%	91%

Table 19: Confusion Matrix for ntree=200 (Century v. Not Century) using best RF

	Predicted: EBRIMA	Predicted: NOT EBRIMA
Actual: EBRIMA	82%	18%
Actual: NOT EBRIMA	11%	89%

Table 20: Confusion Matrix for ntree=200 (Ebrima v. Not Ebrima) using best RF 11

	Predicted: GILL	Predicted: NOT GILL
Actual: GILL	82%	18%
Actual: NOT GILL	4%	96%

Table 21: Confusion Matrix for ntree=200 (Gill v. Not Gill) using best RF

Below are the 95% confidence intervals using the best RF and then using 3 separate models.

Font	best RF Model	binary RF Models
Century	(85%, 90%)	(73%, 80%)
Ebrima	(78%, 86%)	(62%, 72%)
Gill	(78%, 87%)	(78%, 86%)

Table 22: C.I. Comparison (ntree = 200) best RF & binary RF Models

RF1 is the model we used to distinguish between century fonts and not century fonts. When we compare the 95% confidence interval for the accuracies on the test set when we use a separate model RF1 versus when we classified century with the original best model classifying all three fonts they are completely disjoint. The same happened for the RF2 model that classified ebrima. This indicates that the random forest model that classified all three fonts yields the best accuracy for both century and ebrima. When comparing the confidence interval for the accuracies off the test set of the RF3 model versus the random forest model that classified all three fonts have a wide overlap. This

indicates that the accuracy for gill will be the same if we use a separate binary classifier to distinguish it versus using the original best model that classifies all three fonts.

3 Conclusion and Further Suggestions

In conclusion, the random forest model was able to classify the cases of the specific characters with its corresponding font really well. Random forest uses many methods that help increase results. Random Forest uses the simplicity of decision trees and adds flexibility by making many trees through bootstrapping the dataset. Then, the random forest model makes a decision by checking all the trees and seeing which result received the most votes. After finding which best number of trees yielded the best results, we verified that the results that we got on the test set was not just a matter of chance. We verified this by resplitting the data with a new training and test set. This yielded similar results. Even though we had good results with our best model there is always room for improvement. We then built three separate binary classifiers to classify each font individually but this didn't yield better results.

Since the performance of random forest often deteriorates when the number of features increases to further investigate we could try different methods of feature selection. In this project, we reduced the number of features through principal component analysis but we did notice that the order of importance wasn't exactly the order of the principal components. We could try creating a model that only uses only features of high importance calculated

from our best random forest model in section 2.2.1. Then evaluate the performance on the test set.

Due to limitations on time we were not able to implement adaptive boosting to try to improve our results. Adaptive boosting is an algorithm very similar to random forest that can help improve our classification performance. Instead of using multiple trees it uses stumps which is a tree composed of just one variable instead of all of them. Using just one feature makes stumps weak learners but that is something Adaptive boosting relies on. In Random Forest each tree's vote is the same but in ada boost in the forest of stumps each stump's vote had different weightage to it. The weightage of the stump is calculated by the accuracy of its classification. Features that are better discriminators result in better stumps thus improving the weights of the stump. Another important distinction is that in random forests each tree is made independently so it does not matter which tree is made first. In contrast, in Ada boost the order is important as the errors one stump makes influences how the second stump is made, and the errors made by the second stump influence the third stump and so on. . . The way that works is that each case is initially given an equal amount of weight, then after the first stump is run, which is the stump based on the feature that is the best discriminator, the cases that are correctly classified get a reduction in weight and the cases that are incorrectly classified get an increase. The amount of increase or decrease the weights get is dependent on how good of the classifier the first stump was. The new weights are normalized and we can then compute a weighted gini index to determine which feature should be used to create the next stump. The rest of the process continues and can be

summarized by the following steps:

1. Choose the best classifier based on the gini weighted indexes computed from the pre vious stump
2. Compute the accuracy of the stump
3. The cases that get correctly classified by this stump get a reduction in their weights and the ones incorrectly classified get an increase. The amount of increase or decrease is dependent on the accuracy of the stump
4. These new weights are normalized and we go back to step 1

The above steps are continued for a specified number of stumps, Generally the more stumps the better our classification but it also increases computing time so we have to make a decision for when it is no longer worth increasing the amount of stumps. At the end, for any particular case the weights of the stumps that put the case in different classes are added and the class that has the largest sum is predicted for that case. This the power of vote for a stump depends on its weight/Accuracy.