HW-5 Random Forest - Fonts Dataset

Jose Rodriguez, Johnny Nino Ladino, Dayana Sosa; University of Houston

This paper was prepared as partial completion of the course MATH 6350 (Statistical Learning & Data Mining) taught during the Fall 2020 semester at University of Houston.

Background

This report will cover the questions presented by Dr. Azencott over the fonts dataset from the University of California Irvine Machine Learning Repository. The fonts dataset is made up of 153 files that are in a csv format. Each file represents a font and within each file there are 412 columns with varying number of rows depending on the file chosen. Each row represents a 'case' and each case describes numerically a digitized image of some specific character (letter or digit) type in a certain font. Each image has a 20 by 20 size and therefore 400 pixels which are represented by 400 columns. The three font files named "Courier", "Calibri" and "Times" will be used for this report. The first 12/412 columns describe the font and the other 400 columns represent the gray levels of the 400 pixels of a specific image. For this dataset, font type will be our target variable and the 400 columns describing the pixel gray levels will be our features. The aim of this report will be to implement Random Forest Classifier to classify cases to a font type. This report will aim to answer all questions presented in Dr. Azencott's homework instructions in a systematic way while analyzing each question and result to have an in depth understanding of this dataset. The underlying code sustaining this report will be created using python and the IDE of Jupyter notebook.

Methodology

The following methodology was used for developing the code and answering the questions in the document

1. Code will be written in python and python libraries such as pandas, numpy, scikit learn, etc will be imported
2. Code and report will abide to the question numbers presented in Dr. Azencott's instructions
3. The report/code will use the Random Forest Classification algorithm to classify the cases into a class
4. The report/code will aim obtain the best Random Forest model and analyze the results

Preliminary Treatment of Data

Each of the three chosen font files has the same column structure. To begin the analysis, data cleaning and preparation needs to be done on the dataset. Out of the 12 non-related pixel columns, we choose to keep only three of them. The three columns that will be kept are font, strength, and italic. Table 1 below explains what each of these columns represent. Any row that contained any missing numerical data was discarded from each of the three files.

FEATURE	DESCRIPTION
FONT	Same as file name. Describes what font the dataset belongs to.
STRENGTH	Column lists values either equal to 0.4 or to 0.7: in each row, strength = 0.4 for normal character; strength = 0.7 for bold character
ITALIC	Column lists values either equal to 0 or to 1; in each row, italic = 0 for normal character; italic = 1 for italic character;

Table 1: 3 Column Descriptions

The three files (referred to as classes) were then filtered down to only include rows containing a strength value of 0.4 and italic value of 0. This filtered the dataset to only images of "normal" characters. The "Calibri", "Courier" and "Times" classes will be referred to as CL1, CL2 and CL3, respectively. Their respective sizes are shown in Table 2 below.

CLASS	ROWS	COLUMNS
CL1 (CALIBRI)	4,768	403
CL2 (COURIER)	4,262	403
CL3 (TIMES)	4,805	403
TOTAL	**13,835**	

Table 2: Class Size

A full data set denoted as DATA was created through the union of the three class CL1, CL2 and CL3 and has a size of 13,835 rows. DATA will contain only the 400 features related to the gray level of the pixels to create a feature matrix. The basic machine learning tool covered in Dr. Azencott's class known as Kmeans will be used to attempt an automatic clustering of the 3 classes CL1, CL2 and CL3.

Part 0

The means and standard deviations of DATA are computed to be used to standardize the dataset DATA. We standardize the dataset so values in certain features are not over weighed or under weighed when we are applying our classifier. We compute the mean and standard deviation of each column leaving us with $\hat{\mu}(mean) = \mu_{X1}, \dots, \mu_{X400}$ and $\hat{\sigma}(standard\ deviation) = \sigma_{X1}, \dots, \sigma_{X400}$. We then perform the following to rescale DATA:

$$\boldsymbol{SDATA(X_i, X_j) = \frac{DATA(X_i, X_j) - \mu_j}{\sigma_j}}$$

Equation 1: Standardized Matrix Equation

We store the standardized dataset DATA into a separate data frame named SDATA. From SDATA, we compute the correlation matrix, COR, of all 400 features. Table 3 below highlights the first 10 by 10 portion of the COR matrix.

	R0C0	R0C1	R0C2	R0C3	R0C4	R0C5	R0C6	R0C7	R0C8	R0C9
R0C0	1	0.92	0.79	0.63	0.45	0.28	0.17	0.08	0.02	-0.02
R0C1	0.92	1	0.89	0.70	0.51	0.31	0.18	0.08	0.01	-0.03
R0C2	0.79	0.89	1	0.87	0.64	0.42	0.26	0.15	0.06	-0.01
R0C3	0.63	0.70	0.87	1	0.84	0.60	0.40	0.25	0.12	0.03
R0C4	0.45	0.51	0.64	0.84	1	0.83	0.60	0.38	0.20	0.08
R0C5	0.28	0.31	0.42	0.60	0.83	1	0.83	0.58	0.35	0.19
R0C6	0.17	0.18	0.26	0.40	0.60	0.83	1	0.84	0.58	0.39
R0C7	0.08	0.08	0.15	0.25	0.38	0.58	0.84	1	0.84	0.61
R0C8	0.02	0.01	0.06	0.12	0.20	0.35	0.58	0.84	1	0.86
R0C9	-0.02	-0.03	-0.01	0.03	0.08	0.19	0.39	0.61	0.86	1

Table 3: 10 by 10 subset of Correlation Matrix

We implemented Principal Component Analysis (PCA) on SDATA to reduce the number of features to those that retain 95% of the variance explained by the data. We obtained p=127 features after implementing PCA on our SDATA. Hence, our SDATA matrix has a size of (13835 x 127).

Part 1

After implementing PCA, we randomly divided each class data set into its respective 80% train set and 20% test sets. The size of the train and test sets for each class are displayed below in Table 4. We can see that all classes are balanced with each other. Once the 80/20 split for each of the three classes was created, a union of each training and test set was made to have a singular training and test dataset that is equally divided by each class. We display the size for the TRAIN and TEST set below in Table 5.

CLASS	TRAIN	TEST
CL1 (CALIBRI)	(3418 x 127)	(954 x 127)
CL2 (COURIER)	(3409 x 127)	(853 x 127)
CL3 (TIMES)	(3844 x 127)	(961 x 127)

Table 4: Train & Test Set Sizes for Each Class

SET	ROWS	COLUMNS
TRAIN	11067	127
TEST	2768	127

Table 5: TRAIN & TEST Set Sizes (Combined)

Part 2

We will begin by outlining the basic principles of a Random Forest classifier. Random Forest is an ensemble of decision trees that is usually trained with the bagging method. The general idea of bagging is that by training several decision trees with different train sets, we reduce the variance of our model. The general overview is outlined in the steps below.

1. A node starts off by randomly choosing a feature from the most important features
2. A split decision is made based on a threshold for that feature for each case
3. The tree continues to make split decisions until the gini index of a subset in a node is fully pure or a set desired pureness
4. These decision trees are repeated for several important features
5. Finally, we bag all the decision tree results and "average" the predictions from the decision tree results

We use the Random Forest Classifier from sci-kit learn. The parameters that fed into this classifier are the number of trees, maximum number of features, and a random state. The number of trees indicates the number of decision trees that will be performed. The more trees we have, the less variance our model will have. Finding the right number of trees helps choosing the model with the lowest variance while considering computing time as well. The maximum number of features is a parameter indicating the size of the random subsets of features to consider when splitting a node. The random state is just a parameter to keep track of the recorded results and ensure repeatability.

The inputs to our random forest classifier will consist of the decorrelated numerical features that are obtained from the PCA reduction of our 400-pixel features. The 127 decorrelated features will be denoted as $U_1, \dots, U_{127}$. The other input to our Random Forest is the classes. The classes consist of the font class that each case belongs to (Calibri, Courier, or Times). The random forest outputs a prediction for the classes for a test set after being fitted on a train set.

Before the Random Forest function can begin predicting, we need to fit and train the model. Recalling, we have a total of 13,835 total cases. We perform an 80-20 train-test split, where our train set size has 11,067 cases and our test set size has 2,768 cases. Then we fit our train set on our classifier, the classifier then "learns" the data based on the classes that were fed into the model. Lastly, the predict function applies the random forest classifier and performs the same if-else statements that it learned from the fitting of the train set.

Performance outputs consists of global performances, confusion matrices and confidence intervals of both. The global performance gives insight as to how well the model did overall. The confusion matrix drills down on this performance and describes the number of classes that were predicted correctly and incorrectly. The confusion matrix helps understand whether there are classes that are much more difficult to classify. The confusion matrix also helps us determine how well the classifier is doing. The confidence interval helps us determine the margin of error for the global performance and confusion matrices diagonals to further understand the classifier's performance. If the confidence interval is large, then it might mean our results are not stable. If the confidence interval is small, it reassures that our results are stable. They allow us to understand if the accuracies are significant between one another as well.

Feature importance is a byproduct of our random forest classifier. The feature importance is based on the impurity of leaves in a random forest classifier. In Figure 1, we can observe that the first few set of $\boldsymbol{U}$ vectors have high importance. This means that U_1, U_2, U_4, U_5 have the highest total reduction of the criterion brought by that feature. The features that tend to be less important are the last features in the dataset, this is probably due to how PCA displays the most important features in the beginning.

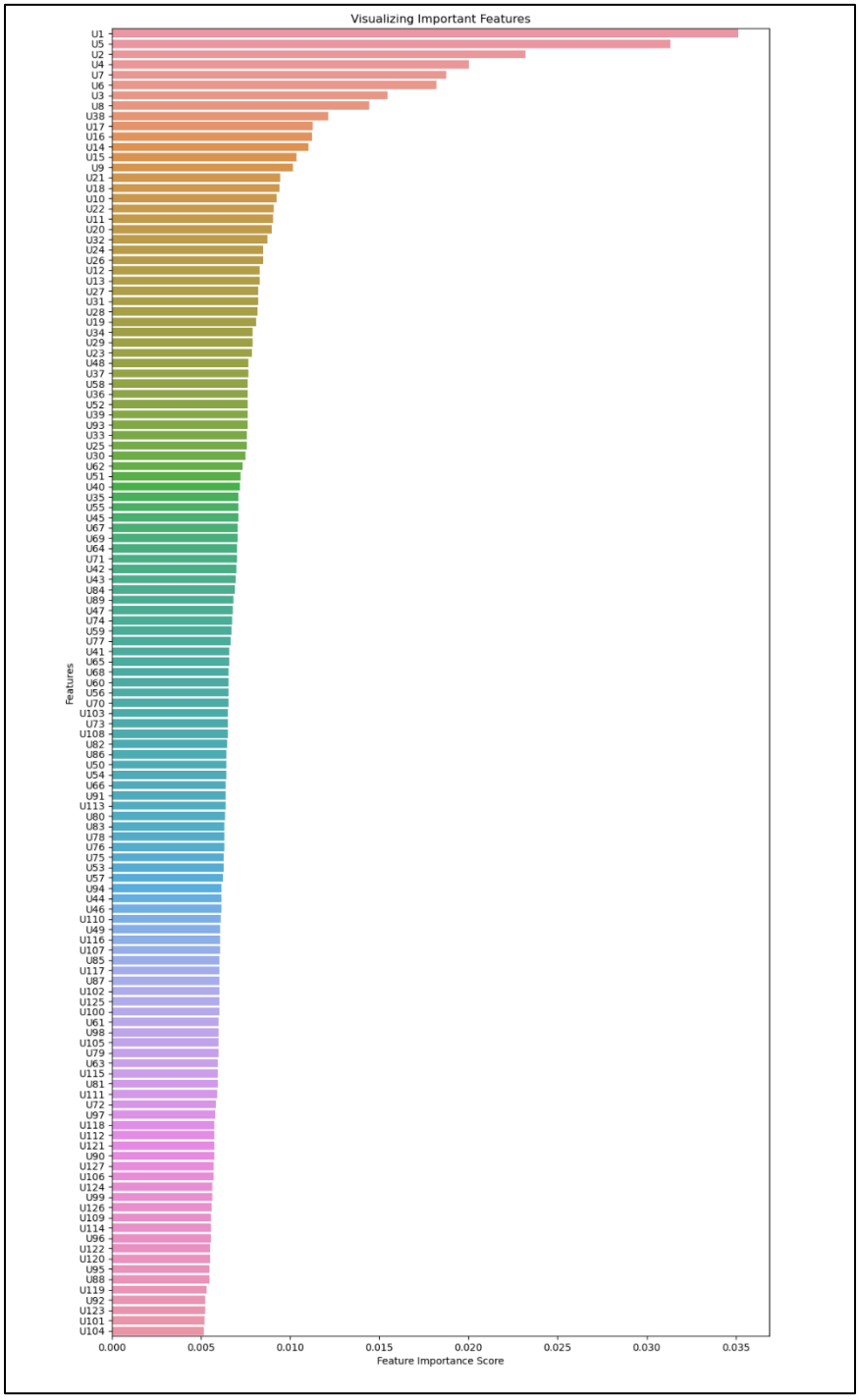

Figure 1: Feature Importance

Part 3

We will use the Random Forest Classifier using 100 trees and set the number of features to consider at $\sqrt{p} = \sqrt{127}$. We also specify a random state 0 to reproduce the results. We train our Random Forest Classifier on our train set described in Table 5. We then use the Random Forest to predict on both our train and test set. The accuracy results are shown in Table 6 below. We can see that the Random Forest Classifier performs exceptionally well on the Train set, yielding accuracy of about 99%. Meanwhile, the performance on the Test set is about 82%, which is not as good as the Train set and could indicate possible overfitting.

Set	Accuracy
TRAIN	**0.990**
TEST	**0.819**

Table 6: Random Forest with 100 Trees TRAIN & TEST Set Accuracies

The confusion matrices for the Train and Test sets after prediction using our Random Forest Classifier are shown below in Figure 2. We can see that in both Train and Test set, the Random Forest Classifier performs best when classifying cases as Calibri, Times, and Courier, in that order. We can also note that the highest misclassification seems to occur when a case is classified as Calibri but is actually Courier.

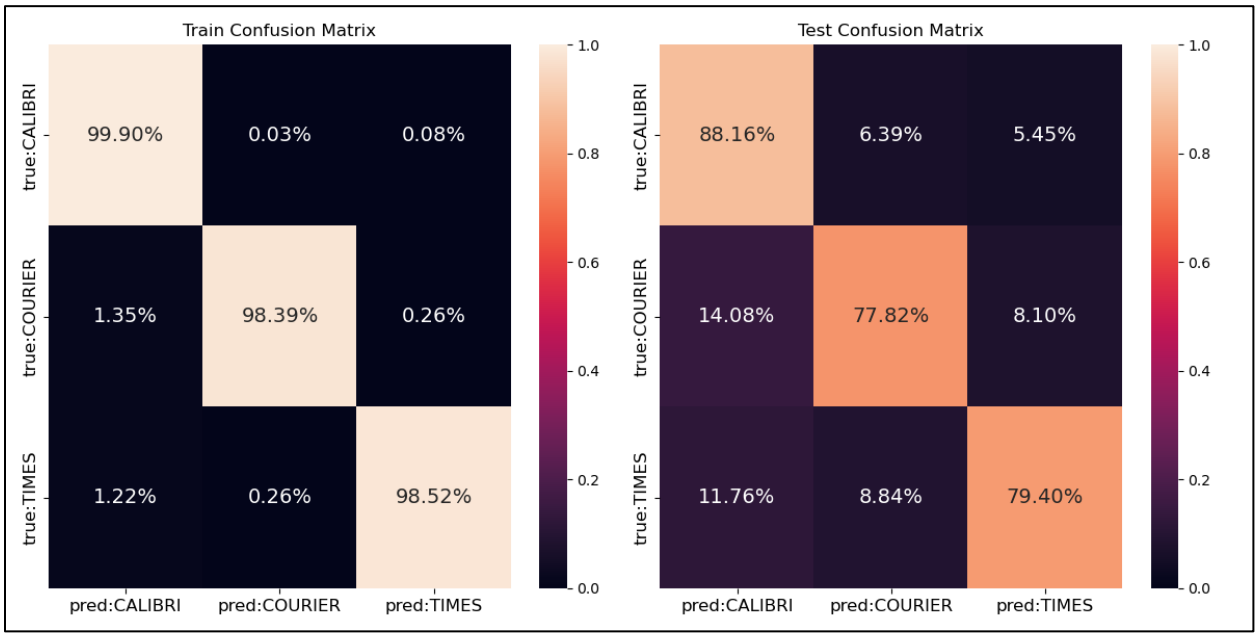

Figure 2: Random Forest with 100 Trees Confusion Matrix for Train and Test Set

We used the same parameters used in our previous Random Forest Classifier but tried different numbers of trees. Table 7 below displays the train and test accuracies obtained after implementing the Random Forest Classifier while utilizing different number of trees. We can see that as we increase the number of trees from 10 to 50, the Train accuracy increases from 98% to 99%. As we continue to increase the number of trees past 50, the Train accuracy plateaus at 99%. The Test accuracy increases as we increase the number of trees. When we use 200 trees, the Test accuracy reaches approximately 83% and only marginally, perhaps negligibly, increases.

ntrees	Train Accuracy	Test Accuracy
10	**0.984**	**0.737**
50	**0.990**	**0.812**
100	**0.990**	**0.819**
200	**0.990**	**0.834**
300	**0.990**	**0.835**
400	**0.990**	**0.838**

Table 7: Random Forests with Different Number of Trees Train & Test Set Performance

We obtained the 6 confusion matrices below in Figure 3 for each Random Forest Classifier using different number of trees on the Test set. The accuracy prediction for Calibri increases by about 11% as we increase the number of trees from 10 to 300, then slightly decreases from approximately 90.9% to 90.8% when we use 300 trees as oppose to 400 trees. For the Courier cases, the accuracy increases as we increase the number of trees from 10 to 200 by about 9%. When using 200, 300, and 400 trees, the random forest classifier accuracy for the Courier cases stays between 78-79%. For the Times cases, the accuracy increases about 10% when we increase the number of trees from 10 to 400. In all 6 models, the biggest misclassification occurs when a case is classified as Calibri but is actually Courier.

Figure 3: Six Test Set Confusion Matrices for Random Forests with Different Number of Trees

We created the plot in Figure 4 below to display the accuracy for the Test set after implementing each Random Forest Classifier with its respective number of trees. We can see that there is a large increase in accuracy when we increase the number of trees in our model from 10 to 50. Then there is a smaller increase when we increase the number of trees from 10 to 200. After 200 trees, the accuracy plateaus and stays around 83%-84%.

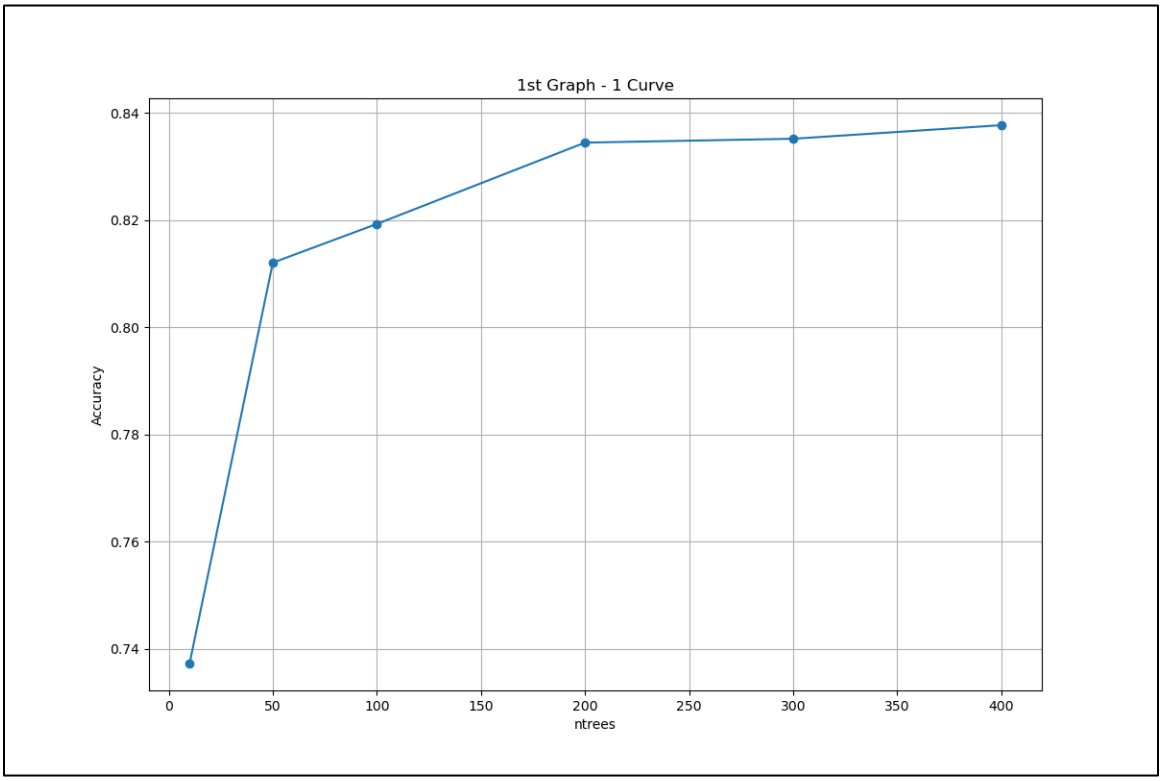

Figure 4: Accuracy on Test Set vs Number of Trees Curve

We utilized the six confusion matrices from Figure 3 to create three curves based on the three diagonal coefficients for each confusion matrix vs the number of trees used in the Random Forest model. Table 8 lists the diagonal coefficients for each confusion matrix. In Figure 5, we see that the accuracy for all three coefficients increases the most when we increased the number of trees from 10 to 200, then the accuracy plateaus for all three coefficients for the random forests that use more than 200 trees. The highest accuracy for Calibri occurs when we use 200 and 300 trees. Meanwhile, the accuracy for the second coefficient, corresponding to Courier, decreases when we increase from 200 to 300 trees and is equal when using 200 and 400 trees. The third coefficient, corresponding to Times, has slightly higher accuracy when using 400 trees as oppose to using 200 or 300 trees. Because of the closeness in accuracies, we calculated the computation time for the different models when using different number of trees and displayed the results in Table 9 below. We can see that the computation time significantly increases as we increase the number of trees in the model, essentially doubling when we increase the number of trees from 200 to 400. Taking into consideration the Test set accuracies and the computation times, we selected the best number of trees to use as 200. This “best” number of trees will be referred to as bntr.

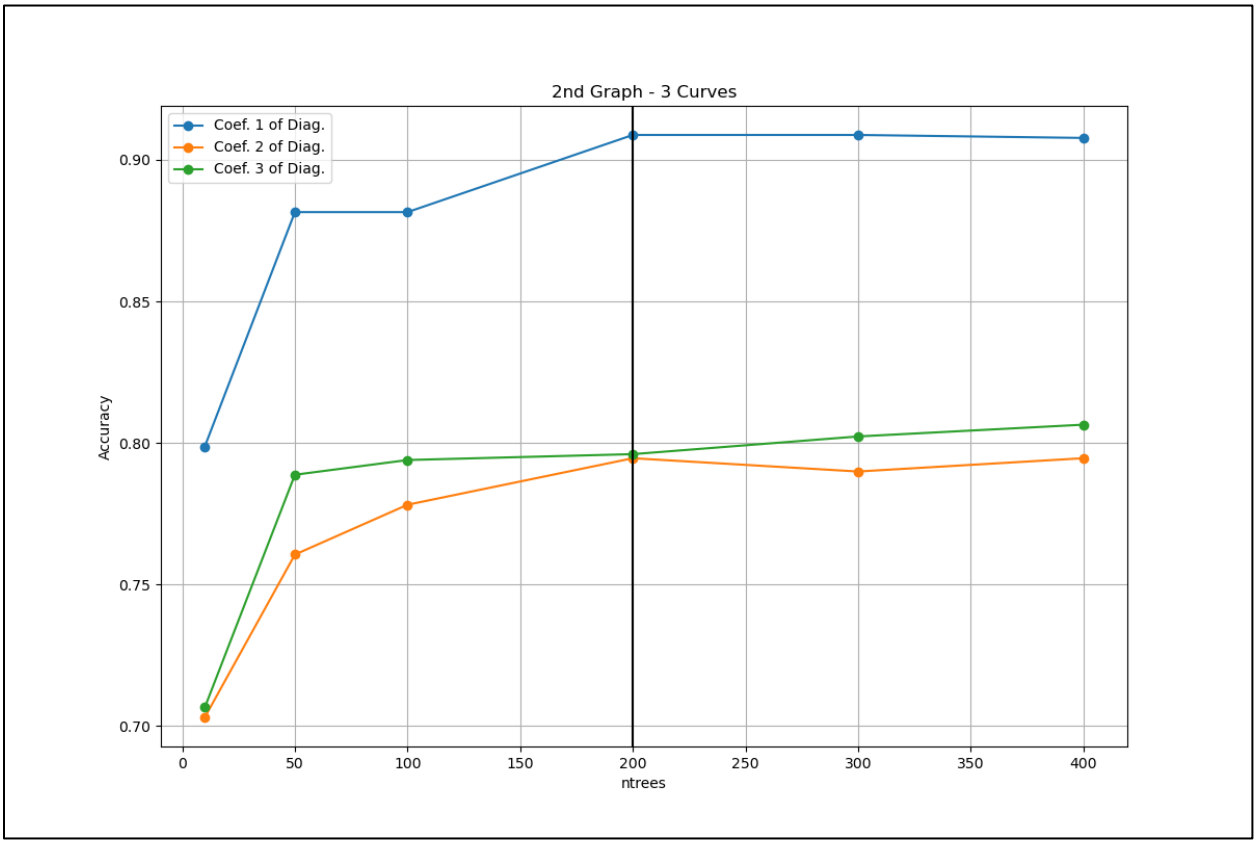

Figure 5: Plot of Diagonal Coefficients vs Number of Trees

ntrees	Coef. 1 of Diag.	Coef. 2 of Diag.	Coef. 3 of Diag.
10	0.799	0.703	0.707
50	0.882	0.761	0.789
100	0.882	0.778	0.794
200	0.909	0.795	0.796
300	0.909	0.790	0.802
400	0.908	0.795	0.806

Table 8: Confusion Matrix Diagonal for Test Set Using Different Number of Trees

ntrees	Computation Time (sec)
200	64.46
300	98.80
400	128.27

Table 9: Computation Time for Random Forest Classifier Using Different Number of Trees

Part 4

Gini impurity is a measure of how often a randomly chosen case from the data set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Determining feature importance can help us understand which features can be kept and which can be removed. Reducing the number of features can simplify the problem that is being modeled. In Table 10, we have a list of the top 10 eigen values with the top 10 feature importance scores.

Eigenvalue	Feature Importance Score
49.844	**0.037**
35.381	**0.023**
18.885	**0.016**
18.406	**0.020**
17.580	**0.032**
15.927	**0.018**
13.547	**0.019**
11.944	**0.014**
10.240	**0.010**
9.254	**0.009**

Table 10: Top 10 Feature IMP

In Figure 6, we see that the higher the eigen value, the higher the feature's importance is. This helps us understand PCA and how it is fundamental to choosing the best features to feed into our model. We measure the impurity by the following formula, where f_i is the frequency of label i at a node and C is the number of unique labels.

$$\sum_{i=1}^{C} f_i(1 - f_i)$$

Equation 2: Gini Impurity Calculation

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of cases that reach the node, divided by the total number of cases. The higher the value the more important the feature.

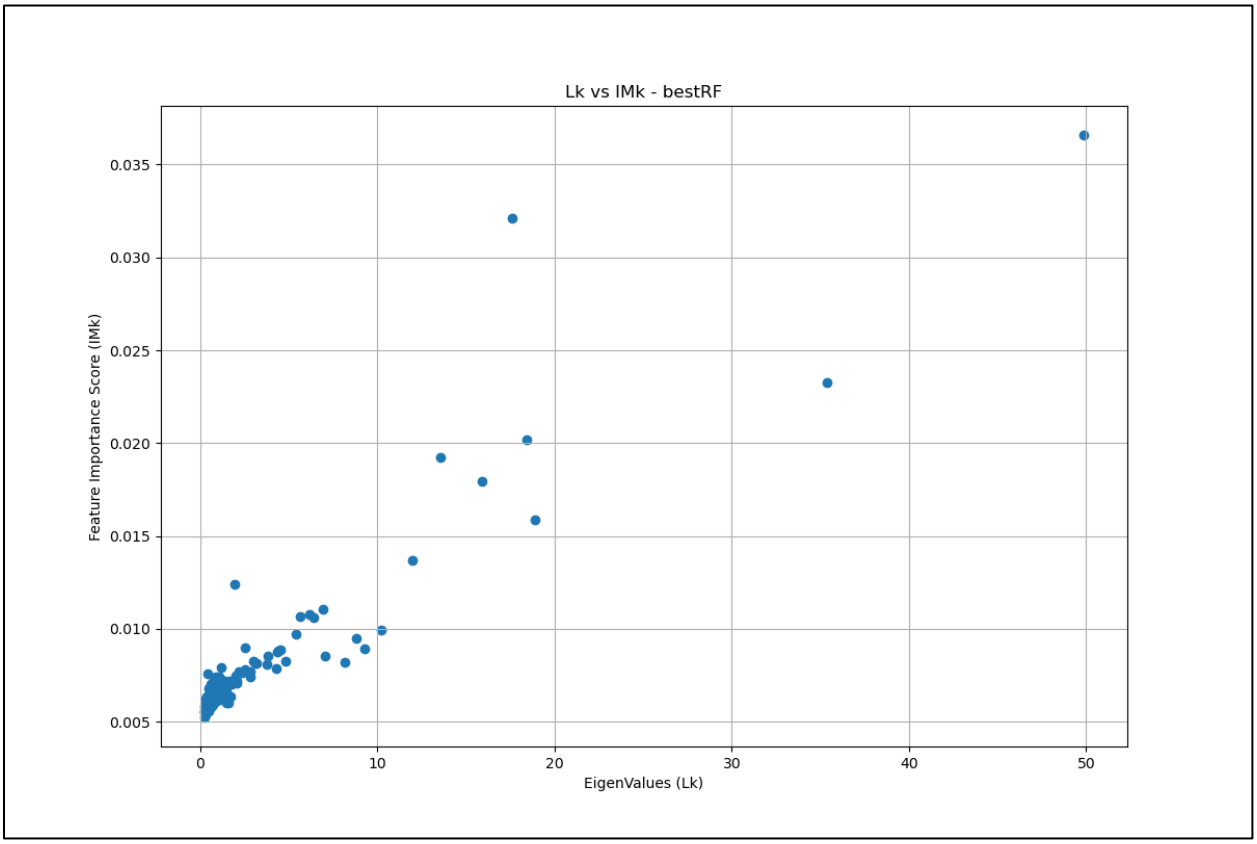

Figure 6: Lk vs IMk – bestRF

We create our best Random Forest Classifier using the parameters obtained in Part 3 and we will define it as bestRF. We use the best number of trees, bntr= 200. The max number of features we use to consider for the best split, ntry, will be ntry=$\sqrt{127} = 11.269$, which rounds to 11 since you cannot have a partial feature. The square root is used to average the variance of the model.

We perform the classifier with the same parameters in bestRF but using different train and test sets, denoting it as newRF. To achieve different random train and test sets, we use a different random state than the one used in Part 3. In Table 11, we see the train and test accuracies are very close for bestRF and newRF. This serves as reassurance that despite using different train and test sets, we can expect similar accuracy results. Bagging methods reduce variance, allowing us to have similar results when fitting our decision trees to different training sets. Table 12 displays the global accuracy confidence intervals for bestRF and newRF. The CI's do not seem to be significantly different and indicate both classifiers have similar performances.

Type	ntrees	Train Accuracy	Test Accuracy
bestRF (Original Sets)	**200**	**0.990**	**0.834**
newtRF (New Sets)	**200**	**0.991**	**0.840**

Table 11: Test & Train Perf - bestRF

Global Accuracy Type	Neg	Point	Plus
bestRF CI	**0.821**	**0.835**	**0.848**
newRF CI	**0.826**	**0.840**	**0.853**

Table 12: Global Accuracy – 95% Confidence Intervals

In Figure 7, we have the confusion matrices for the Test sets using bestRF and newRF, right and left respectively. Our scores along the main diagonals are relatively the same. We see that Calibri is the class that with the highest prediction accuracy in both bestRF and newRF. Using bestRF, the Times class does better in comparison to newRF. For the Courier class, both bestRF and newRF have essentially the same accuracy prediction. We see that the highest misclassification that occurs in both bestRF and newRF is when cases are classified as Calibri but are Courier.

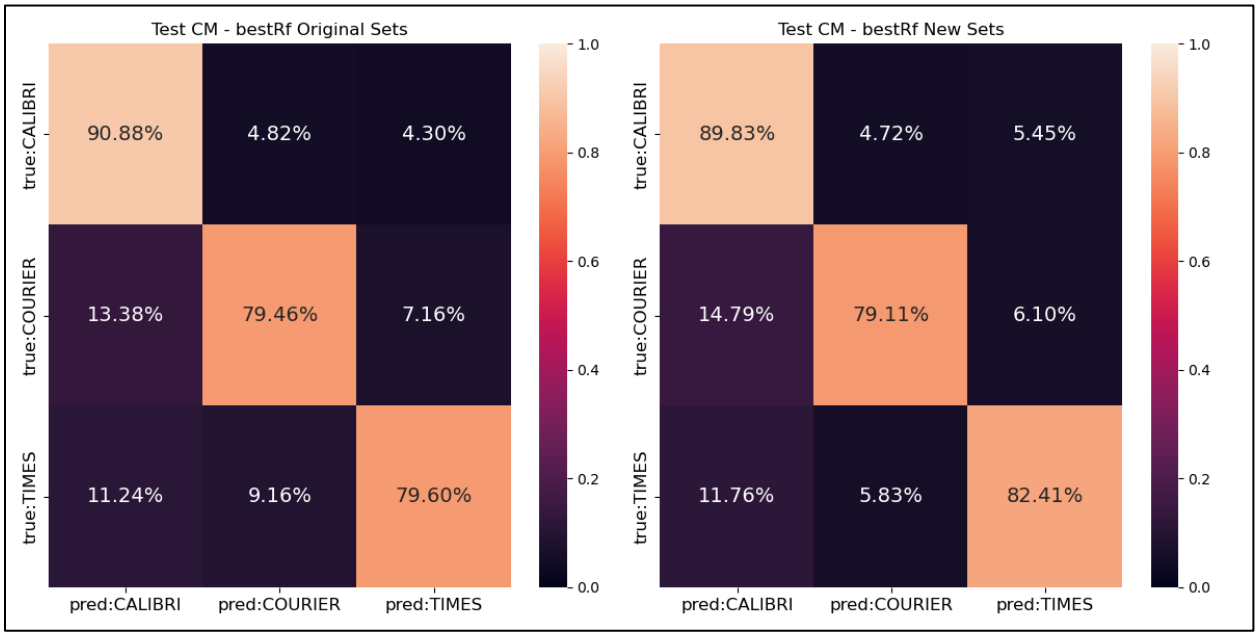

Figure 7: Test Confusion Matrices for bestRF (left) and newRF (right)

Below we have the confidence intervals for the diagonal in each confusion matrix. From our 2 test confusion matrices, we extract the diagonals to create 6 confidence intervals for the fonts that were classified correctly. Calculating a 95% interval allows us to get an idea of the margin of error for each. We can see from the confidence intervals in Table 8 that the range varies between 3% - 5%, thus indicating that the results are not too far away from the estimated accuracy score. From the diagonals and confidence intervals, we see that our classifier does best in classifying the font “CALIBRI” and the range for our 95% confidence interval is smaller compared to the other intervals.

Type & Class Accuracy	Neg	Point	Plus
bestRF CALIBRI	**89.05**	**90.88**	**92.71**
bestRF COURIER	**76.75**	**79.46**	**82.17**
bestRF TIMES	**77.06**	**79.60**	**82.15**
newRF CALIBRI	**87.91**	**89.83**	**91.75**
newRF COURIER	**76.38**	**79.11**	**81.84**
newRF TIMES	**80.01**	**82.41**	**84.82**

Table 13: Class Test Accuracy - 95 % CI's (Displayed as %)

Part 5

The random forest function was applied to 3 new different sets of problems using the previous ntry = $\sqrt{127}$=11 and ntrees = bntr = 200. The 3 new different train and test sets were composed of the same train and test dataset split from the bestRF problem in Part 4 except that the Y label for the classification was changed. The Y labels for the 3 new sets of problems were re-classified to convert the original 3 labels of CALIBRI, COURIER and TIMES to only 2 labels made up of one of the original labels and a new merged label composed of the remaining original labels. Table 14 below describes how the 3 new problem's Y labels were reclassified to obtain 2 class labels. The re-classification of the Y labels were applied to the train and test split separately for each problem.

PROBLEM	DESCRIPTION
PROBLEM 1	CL1 vs {CL2+C3} which can be shown as CALIBRI vs {COURIER+ TIMES}. The Class labels went from 3 to 2 labels.
PROBLEM 2	CL2 vs {CL1+C3} which can be shown as COURIER vs {CALIBRI+ TIMES}. The Class labels went from 3 to 2 labels.
PROBLEM 3	CL3 vs {CL1+C2} which can be shown as TIMES vs {CALIBRI+ COURIER}. The Class labels went from 3 to 2 labels.

Table 14: 3 New Problems

This new re-classification created an imbalance in both the training and test datasets, as the merged label had roughly twice as many cases as the original label within each of the 3 new problems. The training datasets were therefore separately cloned using randomoversampler from the sklearn library to correct these strong imbalances when fitting the random forest classifier. However, the test datasets were left alone since they need to be kept the same size with the original indexes to be used in a comparison later. Table 15 below displays the new training dataset sizes for problems 1, 2 and 3. As shown, the minority class (the original label) was duplicated to be the same size as the merged class label for each problem. The random forest function was applied to these 3 problems after re-classifying the Y label and applying oversampling to the training data sets. This generated the three classifiers RF1, RF2 and RF3 and their respective accuracies and test set confusion matrices were calculated. The accuracies are displayed as A1, A2 and A3 respectively and the confusion matrices are displayed as M1, M2 and M3 respectively.

	CL1	CL1 + CL2	CL1 + CL3	CL2	CL2 + CL3	CL3	Total
P1_TRAINSET Pre	**3814**				**7254**		**11068**
P2_TRAINSET Pre			**7658**	**3410**			**11068**
P3_TRAINSET Pre		**7224**				**3844**	**11068**
P1_TRAINSET Post	**7254**				**7254**		**14508**
P2_TRAINSET Post			**7658**	**7658**			**15316**
P3_TRAINSET Post		**7224**				**7224**	**14448**

Table 15: New Training Dataset Sizes

Table 16 below displays the global training and test accuracies for A1, A2 and A3. The training accuracy for all three are all relatively similar. The test accuracies seem to have more variability at first glance. However, the confidence intervals shown in Table 17 indicate a reasonable margin of error in estimating global test performance. The CI deltas between A1 and A2 seem to be significant between one another which indicates that all they have different performances while A2 and A3 have similar performances.

A	Train Accuracy	Test Accuracy
A1	**0.993**	**0.902**
A2	**0.991**	**0.876**
A3	**0.993**	**0.871**

Table 16: Global Test & Train Accuracy - A's

Global Accuracy for A	Neg	Point	Plus
A1	**0.891**	**0.902**	**0.913**
A2	**0.864**	**0.876**	**0.889**
A3	**0.859**	**0.871**	**0.884**

Table 17: Global Test Accuracy – 95% CI's for A's

The test confusion matrices for M1, M2 and M3 are displayed in Figure 8 below. The best class predictions for each of the three matrices seem to come from the merged class label. This could be due to having more cases in the merged label class. It is interesting to note that the original label in each of the matrices seem to have done worse than the merged label on all three matrices. The merged class label seems to be significant for all 3 problems in terms of class accuracy given the class confidence intervals displayed Table 18 and the visual in Figure 9. The confidence intervals sizes for the original labels are also relatively bigger than the merged class label intervals and indicate there is a better reasonable margin of errors in estimating performance for the merged class labels. The M1 original label test accuracy scores seem to be significantly higher than the M2 and M3 original class label, as well.

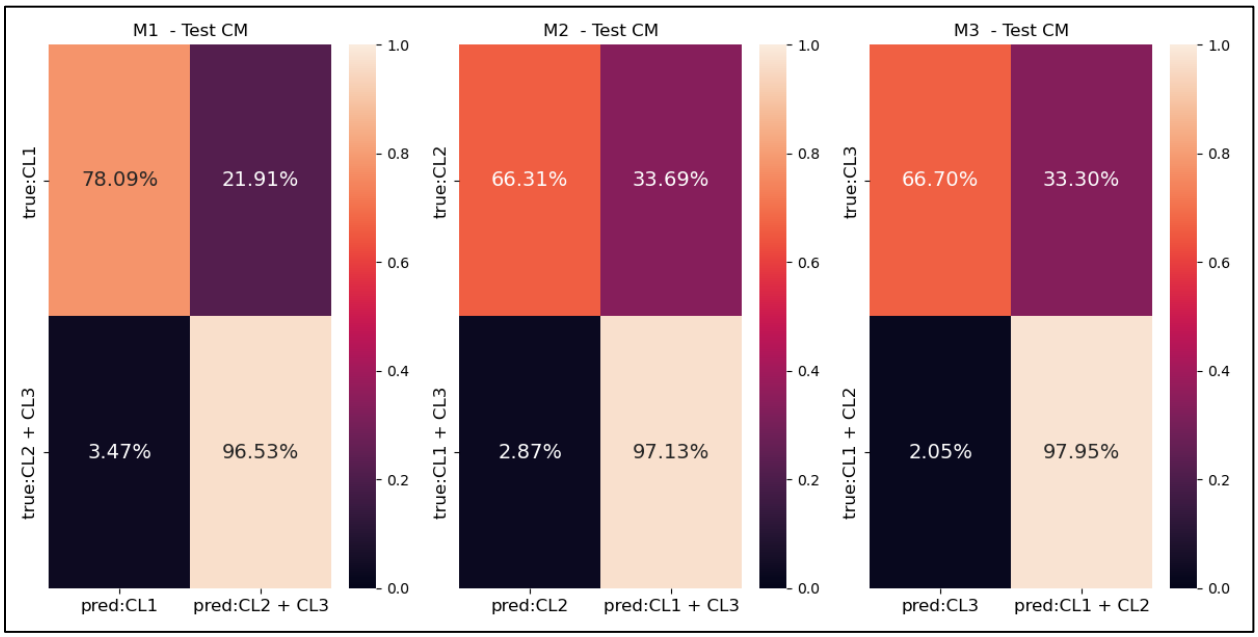

Figure 8: 3 M's Test CM

Problem & Class Accuracy	Neg	Point	Plus
M1 CL1	**75.47**	**78.09**	**80.72**
M1 {CL2 + CL3}	**95.68**	**96.53**	**97.37**
M2 CL2	**63.14**	**66.31**	**69.49**
M2 {CL1 + CL3}	**96.38**	**97.13**	**97.88**
M3 CL3	**63.72**	**66.70**	**69.68**
M3 {CL1 + CL2}	**97.30**	**97.95**	**98.60**

Table 18: Class Test Accuracy – 95 % CI's for M's (Displayed as %)

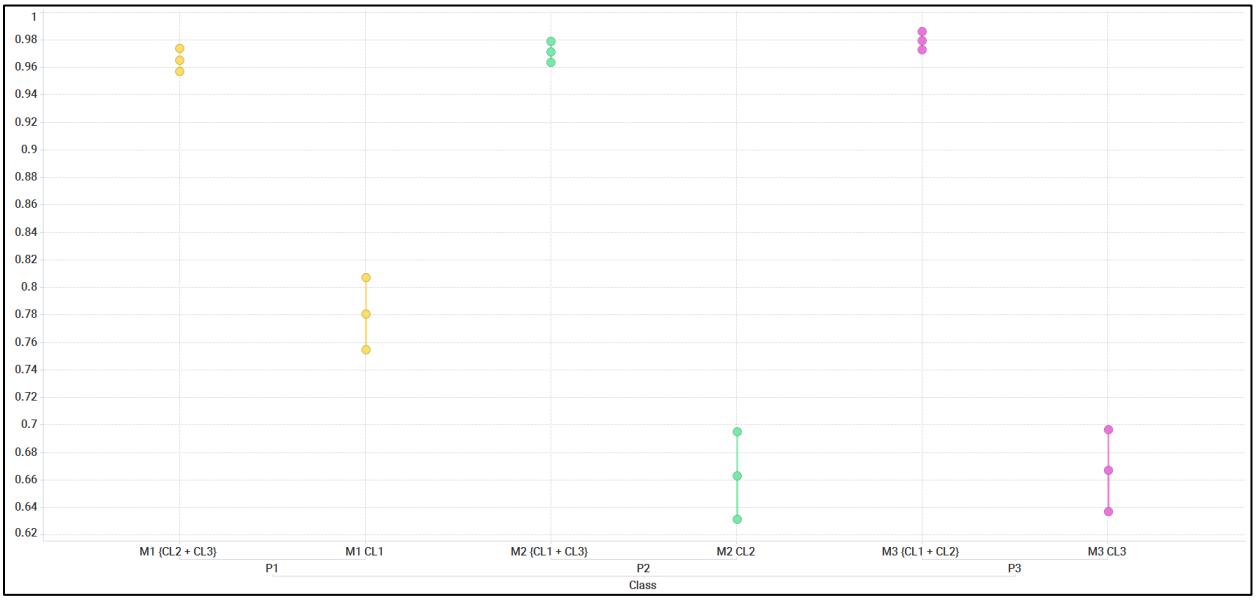

Figure 9: Class Test Accuracy CI Comparison Visual – M's

The bestRF predictor retrieved in Part 4 was applied separately to classify the problems from Table 14. This yielded the accuracies B1, B2 and B3 respectively and the test set confusion matrices BM1, BM2 and BM3 respectively. Applying the predictor bestRF on the 3 problems will aid in comparing the performances and confusion matrices found using the RF1, RF2 and RF3 that were calculated earlier.

Table 19 below displays the global test accuracy scores for B1, B2 and B3. The test accuracies seem to have limited variability at first glance. However, the confidence intervals shown in Table 20 indicate a reasonable margin of error in estimating global test performance. The deltas are not significant between one another which indicates that all three sets have similar performances between one another.

B	Test Accuracy
B1	**0.888**
B2	**0.888**
B3	**0.892**

Table 19: Global Test Accuracy - B's

Global Accuracy for B	Neg	Point	Plus
B1	**0.877**	**0.888**	**0.900**
B2	**0.877**	**0.888**	**0.900**
B3	**0.881**	**0.892**	**0.904**

Table 20: Global Test Accuracy – 95% CI's for B's

The test confusion matrices for BM1, BM2 and BM3 are displayed in Figure 10 below. The best class predictions for each of the three matrices come from the merged class label for BM2 and BM3. However, it is interesting to note that the original label in BM1 seems to have done better than the merged label which does not coincide with what we had seen earlier. Only BM1 CL1 vs BM2 {CL2 + CL3} seems to not be significant in terms of accuracy given the class confidence intervals displayed Table 21 and the visual in Figure 11. The confidence interval sizes for the original labels are also relatively bigger than the merged class label intervals except for Problem 1. The confidence interval range for the original and merged class label in problem 1 seem to be of similar sizes and not disjoint which indicate there is a better reasonable margin of error in estimating performance for problem 1. The trend of the merged class label's confidence intervals sizes being bigger than the original labels continue for problem 2 and 3, however. The M1 original label test accuracy score seems to be significant and largely better than original class label in M2 and M3.

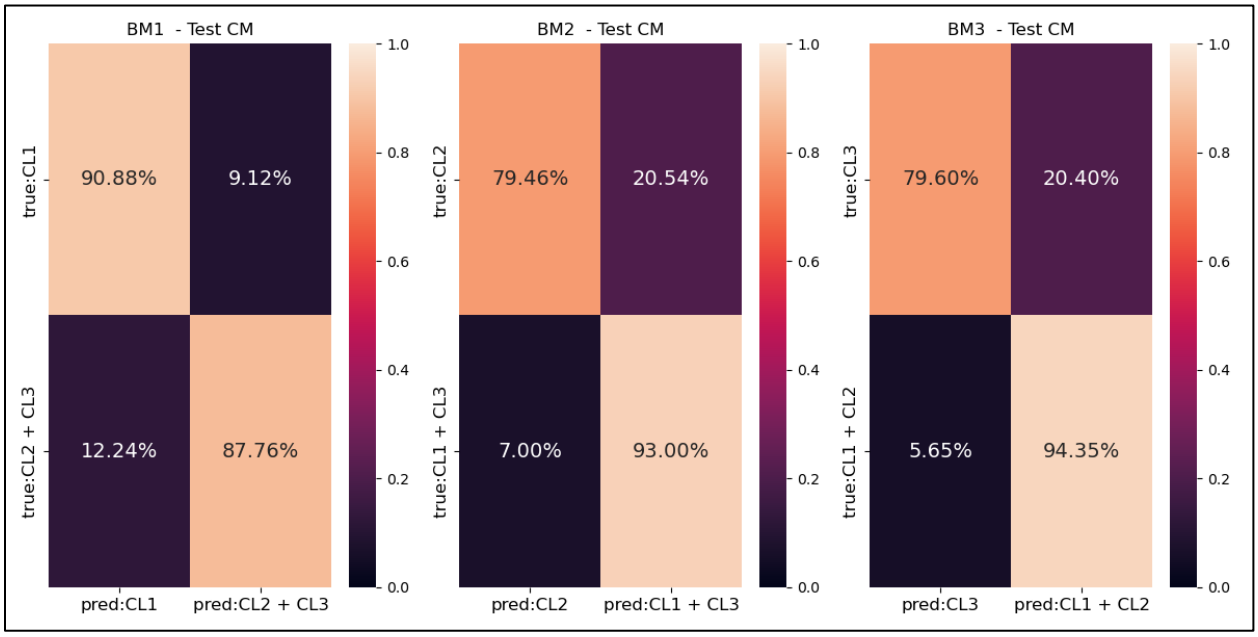

Figure 10: 3 BM's Tests CM

Problem & Class Accuracy	Neg	Point	Plus
BM1 CL1	**89.05**	**90.88**	**92.71**
BM1 {CL2 + CL3}	**86.25**	**87.76**	**89.26**
BM2 CL2	**76.75**	**79.46**	**82.17**
BM2 {CL1 + CL3}	**91.86**	**93.00**	**94.15**
BM3 CL3	**77.06**	**79.60**	**82.15**
BM3 {CL1 + CL2}	**93.29**	**94.35**	**95.42**

Table 21: Class Test Accuracy – 95% CI's for BM's (Displayed as %)

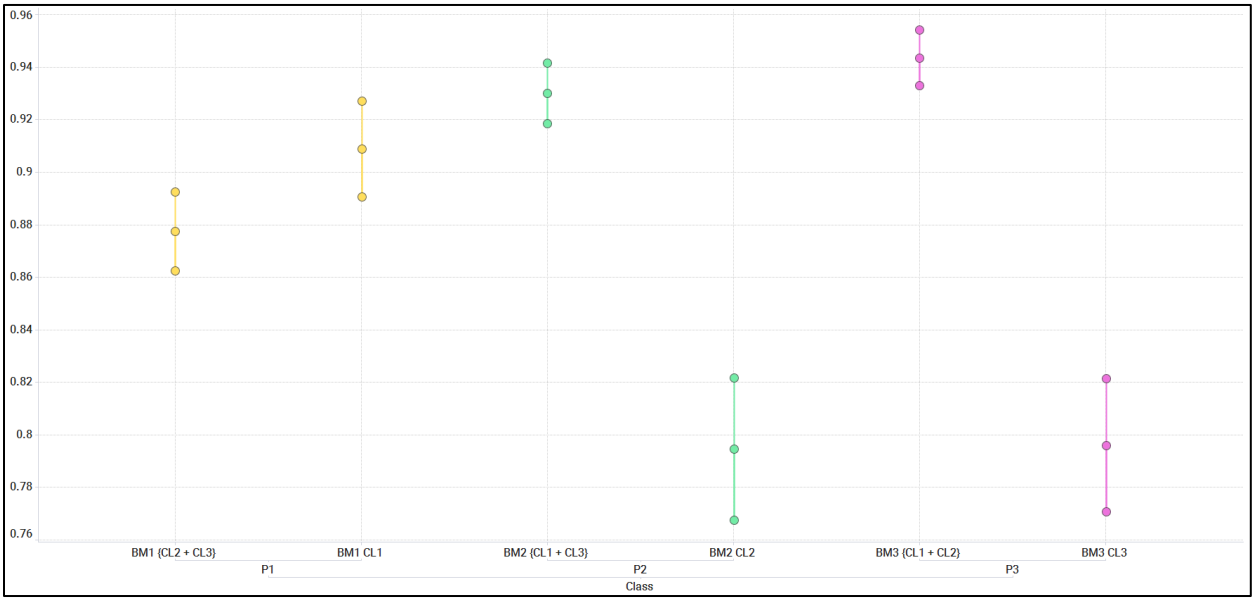

Figure 11: Class Test Accuracy CI Comparison Visual – BM's

The global test scores can now be compared between the set of A's (A1, A2 and A3) vs the set of B's (B1, B2 and B3). Likewise, the class test scores can now be compared between the set of M's (M1, M2 and M3) vs the set of BM's (BM1, BM2 and BM3). The set of global test scores A's and class test scores M's belong to the classifiers RF1, RF2 and RF3 that were computed separately based on each respective problem. The set of global test scores B's and class test scores BM's belong to the predictor bestRF that were computed in Part 4 and applied separately to each respective problem.

At first glance, there seems to be better global test accuracy scores between the A's and B's depending on the problem as seen in Figure 12 below. However, none of the global test accuracy scores and global confidence intervals between A1, A2 and A3 vs B1, B2 and B3 seem to be significant between each problem set (Problem 1, Problem 2 and Problem 3) or between each global set (A's vs B's). None of the confidence intervals indicate a disjoint between the problem sets or global sets, which indicate RF1, RF2 and RF3 vs bestRF each have a reasonable and similar margin of error in estimating global test performance.

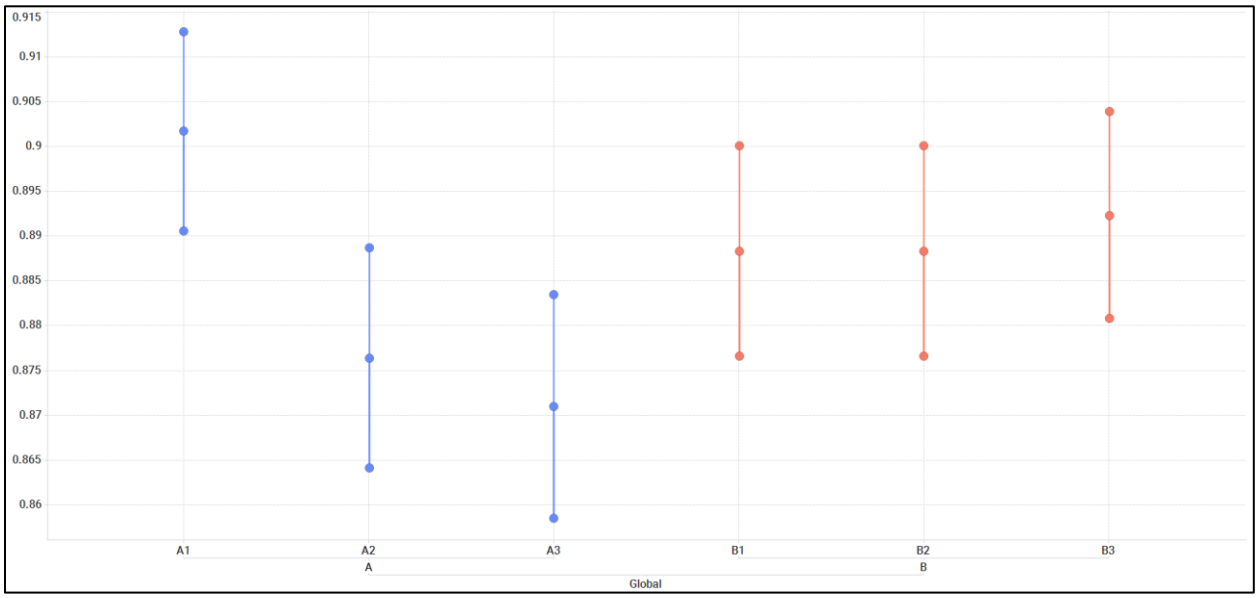

Figure 12: Global Test Accuracy CI Comparison Visual

The class test CI accuracy scores between problem sets is displayed in Figure 13 below. Within each problem set, the original and merged class label for all three problems seems to be significant. Specifically, the BM original class label seems to have done better than the M original class label for each problem set. On the other hand, the BM merged class label seems to have done worse than the M merged class label. This indicates that the original labels are better classified by bestRF where there are three labels and the merged class labels are better classified by RF1, RF2 and RF3, respectively, where there are two labels. It is also interesting to note that the best and worst scores and CI's came from the M set. There is a larger discrepancy in accuracy between the M set within each problem than the BM set for each problem.

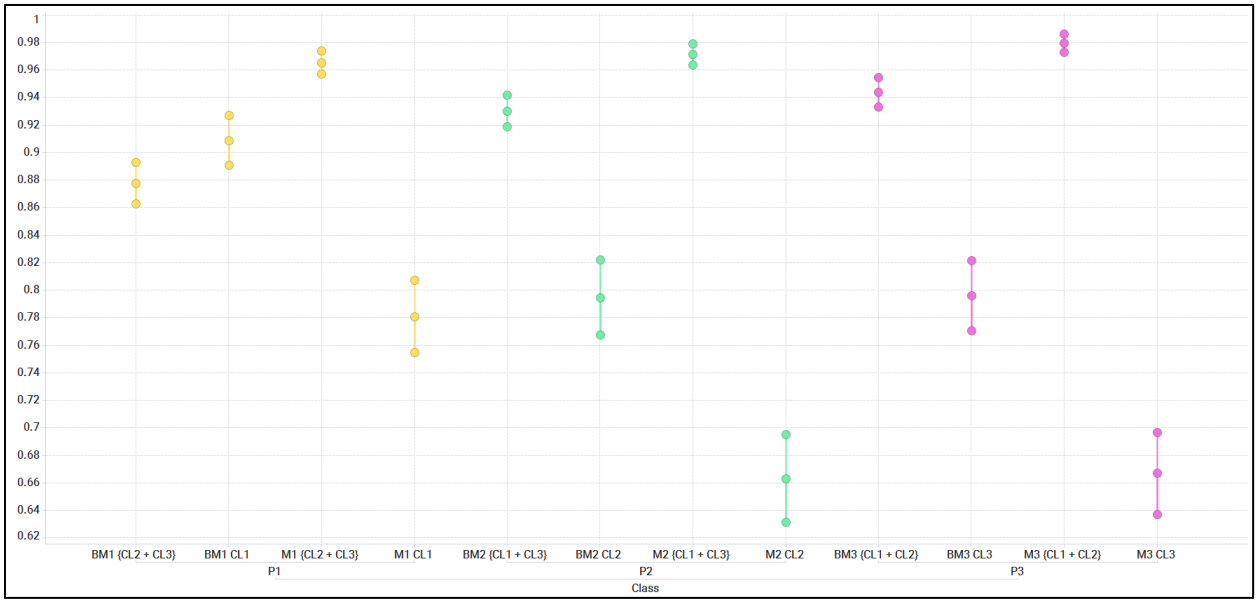

Figure 13: Class Test Accuracy CI Comparison Visual

Summary

We performed pre-processing and filtering of the data to properly setup the dataset. Next, we standardized the features to be used when performing PCA to reduce the number of features in our data. After splitting our data in a balanced Train and Test set, we were able to utilize the Random Forest Classifier. After running random forests with different number of trees, we found an optimal model that utilized 200 trees and 11 features to split on. We ran our best classifier on a different partition of train and test data to compare results. We also utilized data visualizations to investigate feature importance for different random forest models. The random forest function was applied on 3 different sets of problems to determine if there are font types that are easier to distinguish. We used the 3 binary classification tasks while applying cloning and compared results with our original bestRF results. Depending on the goal in mind, either the RF1, RF2, RF3 or bestRF classifier might be better for a specific class(es) as the class test accuracies have different ranges that could lead to better classification for two labels or for one label. In terms of classification and from our results, we conclude that Courier and Times seem to be more similar in style compared to Calibri. In the table below, we see the distinction of the vowels for each font.

CALIBRI	A E I O U
COURIER	A E I O U
TIMES	A E I O U

Acknowledgements

The authors gratefully acknowledge the guidance of Professor Robert Azencott.

Fonts Dataset Source

This dataset was taken from the University of California Irvine repository of machine learning datasets. The dataset can be found at following link: https://archive.ics.uci.edu/ml/machine-learning-databases/00417/

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013) *An Introduction to Statistical Learning with applications in R*, www.StatLearning.com, Springer-Verlag, New York

Work Distribution

Jose Rodriguez: 33%
Dayana Sosa: 33%
Johnny Nino Ladino: 33%

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from scipy import stats
import math
from matplotlib.pyplot import cm
from collections import Counter, defaultdict


fsz=(12,8)

# Import text files of three specific fonts
Calibri = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\HW-2\fonts\CALIBRI.csv")
Courier = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\HW-2\fonts\COURIER.csv")
Times = pd.read_csv(r"C:\Users\Rodri\Desktop\Fall 2020\MATH 6350 - Statistical Learning & Data Mining\2. Homework\HW-2\fonts\TIMES.csv")

################################################
#  ** Preliminary Treatment of the dataset **  #
################################################

# Discard the 9 columns below
# fontVariant, m_label, orientation, m_top, m_left, originalH, originalW, h, w
Calibri.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True)
Courier.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True)
Times.drop(columns = ['fontVariant', 'm_label', 'orientation', 'm_top', 'm_left', 'originalH', 'originalW', 'h', 'w'], inplace=True)

# any row containing missing numerical data should be be discarded
Calibri.dropna(how='any', inplace = True)
Courier.dropna(how='any', inplace = True)
Times.dropna(how='any', inplace = True)
```

```
# define then three CLASSES of images of "normal" characters as follows
# strength =0.4 and italic = 0
CL1 =(Calibri[(Calibri['strength']==0.4) & (Calibri['italic']==0)])
CL2 =(Courier[(Courier['strength']==0.4) & (Courier['italic']==0)])
CL3 =(Times[(Times['strength']==0.4) & (Times['italic']==0)])

# union of the three classes CL1 , CL2, CL3 and hence regroups N= n1 +n2 +n3 cases
DATA = pd.concat([CL1, CL2, CL3])
DATA.reset_index(inplace=True, drop=True)

######################
#  **** PART 0 ****  #
######################

# mean and standard deviation
m = DATA.iloc[:,3:].mean()
sd = DATA.iloc[:,3:].std()
# standardizing the features - rescaled data matrix
SDATA = (DATA.iloc[:,3:] - m)/sd

# Adding the font column to the standardized matrix
SDATA.reset_index(inplace=True,drop=True)

#ID
ID = DATA.iloc[:,0]


####################
#  ** PART PCA **  #
####################

pca_model = PCA(.95) #  gives p of 127
pca_model.fit(SDATA)
SDATA_PCA = pca_model.transform(SDATA)
SDATA_PCA = pd.DataFrame(SDATA_PCA, columns=['U%i' % i for i in range(1,128,1)])

# Adding the font column to the standardized PCA DF
SDATA_PCA_ID = pd.concat([DATA.iloc[:,0], SDATA_PCA], axis=1)

# Creating CL1, CL2 and CL3 out of standardized PCA DF
SCL1 = SDATA_PCA_ID[SDATA_PCA_ID['font'] == 'CALIBRI']
SCL2 = SDATA_PCA_ID[SDATA_PCA_ID['font'] == 'COURIER']
SCL3 = SDATA_PCA_ID[SDATA_PCA_ID['font'] == 'TIMES']


######################
#  ** Question 1 **  #
######################
```

```
## Creating Train/Test Sets from 3 Classes
testCL1 = SCL1.sample(frac = 0.2, random_state = 0)
trainCL1 = SCL1[~SCL1.index.isin(testCL1.index)]

testCL2 = SCL2.sample(frac = 0.2, random_state = 0)
trainCL2 = SCL2[~SCL2.index.isin(testCL2.index)]

testCL3 = SCL3.sample(frac = 0.2, random_state = 0)
trainCL3 = SCL3[~SCL3.index.isin(testCL3.index)]

TESTSET  = pd.concat([testCL1, testCL2, testCL3])
TRAINSET = pd.concat([trainCL1, trainCL2, trainCL3])

# Creating X and Y values of from Training and Test Datasets
X_Train = TRAINSET.iloc[:,1:]
Y_Train = TRAINSET.iloc[:,0]

X_Test = TESTSET.iloc[:,1:]
Y_Test = TESTSET.iloc[:,0]

#X_Train, X_Test, Y_Train, Y_Test = train_test_split(SDATA_PCA.iloc[:,1:], SDATA_PCA.iloc[:,0], test_size=0.2, ran
dom_state=0)

######################
#  ** Question 2 **  #
######################


clf = RandomForestClassifier(random_state=0)
clf.fit(X_Train, Y_Train)

# Finding the important features
feature_imp_dict = {'Feature Importance Score':pd.Series(clf.feature_importances_), 'Feature': pd.Series(SDATA_PCA
.columns)}
feature_imp = pd.DataFrame(feature_imp_dict )
feature_imp.sort_values(by=['Feature Importance Score'], ascending= False, ignore_index=True, inplace=True)

# Visualizing the important features
plt.figure(figsize=(12,24))
# Creating a bar plot
sns.barplot(x=feature_imp.iloc[:,0], y=feature_imp.iloc[:,1])
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.savefig('Feature Importance (Q2).png')
```

```
########################
#  ** Question 3.1 **  #
########################

clf = RandomForestClassifier(n_estimators=100, max_features="sqrt", random_state=0)
clf.fit(X_Train, Y_Train)

Y_Pred_Train = clf.predict(X_Train)
Y_Pred = clf.predict(X_Test)

# Making the Confusion Matrix for Train & Test data
trainconf = confusion_matrix(Y_Train, Y_Pred_Train, labels = ['CALIBRI', 'COURIER', 'TIMES'], normalize = 'true')
testconf = confusion_matrix(Y_Test, Y_Pred, labels = ['CALIBRI', 'COURIER', 'TIMES'], normalize = 'true')
# Performance for both sets
trainperf = metrics.accuracy_score(Y_Train, Y_Pred_Train)
testperf = metrics.accuracy_score(Y_Test, Y_Pred)

test_train_perf_dict = {'Accuracy': [trainperf,testperf] ,'Kind': ['Train','Test']}
test_train_perf_df_3_1 = pd.DataFrame(test_train_perf_dict)

# Visuals for Report
cmtx_train = pd.DataFrame(trainconf,
    index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
    columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])
cmtx_test = pd.DataFrame(testconf,
    index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
    columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])

# Confusion Matrix Plots For Train & Tests
f, axes = plt.subplots(1, 2, figsize = (12,6))
a = sns.heatmap(cmtx_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[0])
a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
a.set_yticklabels(a.get_yticklabels(), fontsize = 12, verticalalignment = 'center')
b = sns.heatmap(cmtx_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[1])
b.set_xticklabels(b.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
b.set_yticklabels(a.get_yticklabels(), fontsize = 12,  verticalalignment = 'center')
axes[0].set_title("Train Confusion Matrix")
axes[1].set_title("Test Confusion Matrix")
f.tight_layout()
plt.savefig("Confusion Matrices (Q3.1).png")



########################
#  ** Question 3.2 **  #
########################
```

```
ntrees = [10, 50, 100, 200, 300, 400]
trainperf = []
testperf = []
testconf = {}

# RF For Loop
for i in range(len(ntrees)):
    # Fitting RF to the Training set
    clf = RandomForestClassifier(n_estimators=ntrees[i], max_features="sqrt", random_state=0)
    clf.fit(X_Train, Y_Train)
    # Predicting the Train/Test Set results
    Y_Pred_Train = clf.predict(X_Train)
    Y_Pred = clf.predict(X_Test)
    # Appending Performance for both sets
    trainperf.append(metrics.accuracy_score(Y_Train, Y_Pred_Train))
    testperf.append(metrics.accuracy_score(Y_Test, Y_Pred))
    testconf[i] = confusion_matrix(Y_Test, Y_Pred, labels = ['CALIBRI', 'COURIER', 'TIMES'], normalize = 'true')


# Test and Train Accuracy Performances
test_train_perf_dict = {'ntrees': ntrees, 'Train Accuracy': trainperf ,'Test Accuracy': testperf}
test_train_perf_df = pd.DataFrame(test_train_perf_dict)


# Confusion Matrix Plots
f, axes = plt.subplots(3, 2, figsize = (12,6))

# Confusion Matrix RF For Loop
for i in range(len(testconf)):
    # Visuals for Report
    cmtx_test = pd.DataFrame(testconf[i],
        index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
        columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])
    #Plotting itself
    a = sns.heatmap(cmtx_test, annot=True, fmt = '.02%', annot_kws={"size": 10}, vmin = 0, vmax = 1, ax = axes.fla
t[i])
    a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 8, horizontalalignment = "center")
    a.set_yticklabels(a.get_yticklabels(), fontsize = 8, verticalalignment = 'center')
    #Title
    axes.flat[i].set_title(str(ntrees[i]) + " Trees - Test CM")

f.tight_layout()
plt.savefig("6 Test Confusion Matrices.png")

# 1st Graph
plt.figure(figsize=fsz)
plt.grid()
```

```
plt.plot(test_train_perf_df.iloc[:,0],  test_train_perf_df.iloc[:,2], marker='o')
plt.xlabel('ntrees')
plt.ylabel('Accuracy')
plt.title('1st Graph - 1 Curve')
#plt.axvline(x=k, label='"best" value k* at x = ' + str(k), c='k')
plt.savefig('1st Graph (Q3.2).png')


# DataFrame for 2nd Graph
testconf_diag = [list(testconf[i].diagonal()) for i in range(len(testconf))]
testconf_diag_df = pd.DataFrame(testconf_diag, columns=['Coef. 1 of Diag.','Coef. 2 of Diag.','Coef. 3 of Diag.'])
testconf_diag_df = pd.concat([test_train_perf_df.iloc[:,0], testconf_diag_df],axis=1)


# 2nd Graph
bntr = 200

plt.figure(figsize=fsz)
plt.grid()
plt.plot(testconf_diag_df.iloc[:,0],  testconf_diag_df.iloc[:,1], marker='o')
plt.plot(testconf_diag_df.iloc[:,0],  testconf_diag_df.iloc[:,2], marker='o')
plt.plot(testconf_diag_df.iloc[:,0],  testconf_diag_df.iloc[:,3], marker='o')
plt.xlabel('ntrees')
plt.ylabel('Accuracy')
plt.title('2nd Graph - 3 Curves')
plt.axvline(x=bntr, label='"best" ntrees bntr at x = ' + str(bntr), c='k')
plt.legend(labels=['Coef. 1 of Diag.','Coef. 2 of Diag.','Coef. 3 of Diag.'])
plt.savefig('2nd Graph (Q3.2).png')


########################
#  ** Question 4.1 **  #
########################

# bestRF
clf_bestRF = RandomForestClassifier(n_estimators=bntr, max_features="sqrt", random_state=0)
clf_bestRF.fit(X_Train, Y_Train)
# Predicting the Train/Test Set results
Y_Pred_Train = clf_bestRF.predict(X_Train)
Y_Pred = clf_bestRF.predict(X_Test)

# For later 5.2
Y_Pred_bestRF = Y_Pred

# For comparison in 4.2
trainperf_bestRF = []
testperf_bestRF = []
testconf_bestRF = {}
```

```
trainperf_bestRF.append(metrics.accuracy_score(Y_Train, Y_Pred_Train))
testperf_bestRF.append(metrics.accuracy_score(Y_Test, Y_Pred))
testconf_bestRF['bestRF Original Sets'] = confusion_matrix(Y_Test, Y_Pred, labels = ['CALIBRI', 'COURIER', 'TIMES'
], normalize = 'true')


# Plotting Lk vs IMk
Lk = pca_model.explained_variance_ #eigenvalues from PCA model
IMk = clf.feature_importances_

# Finding the important features
feature_imp_dict_bestRF = {'EigenValues': pd.Series(Lk), 'Feature Importance Score':pd.Series(IMk)}
feature_imp_bestRF = pd.DataFrame(feature_imp_dict_bestRF)

# Visualizing the important features
plt.figure(figsize=fsz)
plt.grid()
# Creating a scatter plot
plt.scatter(x=feature_imp_bestRF.iloc[:,0], y=feature_imp_bestRF.iloc[:,1])
# Add labels to your graph
plt.ylabel('Feature Importance Score (IMk)')
plt.xlabel('EigenValues (Lk)')
plt.title("Lk vs IMk - bestRF")
plt.savefig('Feature Importance (Q4.1).png')


########################
#  ** Question 4.2 **  #
########################

## Creating Train/Test Sets from 3 Classes
testCL1_n = SCL1.sample(frac = 0.2, random_state = 1)
trainCL1_n = SCL1[~SCL1.index.isin(testCL1_n.index)]

testCL2_n = SCL2.sample(frac = 0.2, random_state = 1)
trainCL2_n = SCL2[~SCL2.index.isin(testCL2_n.index)]

testCL3_n = SCL3.sample(frac = 0.2, random_state = 1)
trainCL3_n = SCL3[~SCL3.index.isin(testCL3_n.index)]

TESTSET_n  = pd.concat([testCL1_n, testCL2_n, testCL3_n])
TRAINSET_n = pd.concat([trainCL1_n, trainCL2_n, trainCL3_n])

# Creating X and Y values of from Training and Test Datasets
X_Train_n = TRAINSET_n.iloc[:,1:]
Y_Train_n = TRAINSET_n.iloc[:,0]

X_Test_n = TESTSET_n.iloc[:,1:]
```

```
Y_Test_n = TESTSET_n.iloc[:,0]

# RF
newRF = RandomForestClassifier(n_estimators=bntr, max_features="sqrt", random_state=1)
newRF.fit(X_Train_n, Y_Train_n)

Y_Pred_Train_n = newRF.predict(X_Train_n)
Y_Pred_n = newRF.predict(X_Test_n)

# Appending
trainperf_bestRF.append(metrics.accuracy_score(Y_Train_n, Y_Pred_Train_n))
testperf_bestRF.append(metrics.accuracy_score(Y_Test_n, Y_Pred_n))
testconf_bestRF['bestRF New Sets'] = confusion_matrix(Y_Test_n, Y_Pred_n, labels = ['CALIBRI', 'COURIER', 'TIMES']
, normalize = 'true')

# Test and Train Accuracy Performances
test_train_perf_dict_bestRF = {'Type': ['bestRF Original Sets', 'bestRf New Sets'],'ntrees': [bntr, bntr], 'Train
Accuracy': trainperf_bestRF ,'Test Accuracy': testperf_bestRF}
test_train_perf_df_bestRF = pd.DataFrame(test_train_perf_dict_bestRF)


# Visuals for Report
cmtx_test_OG = pd.DataFrame(testconf_bestRF['bestRF Original Sets'],
    index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
    columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])
cmtx_test_New = pd.DataFrame(testconf_bestRF['bestRF New Sets'],
    index=['true:CALIBRI', 'true:COURIER', 'true:TIMES'],
    columns=['pred:CALIBRI', 'pred:COURIER', 'pred:TIMES'])

# Confusion Matrix Plots For Train & Tests
f, axes = plt.subplots(1, 2, figsize = (12,6))
a = sns.heatmap(cmtx_test_OG, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[0])
a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
a.set_yticklabels(a.get_yticklabels(), fontsize = 12, verticalalignment = 'center')
b = sns.heatmap(cmtx_test_New, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[1])
b.set_xticklabels(b.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
b.set_yticklabels(a.get_yticklabels(), fontsize = 12,  verticalalignment = 'center')
axes[0].set_title("Test CM - bestRf Original Sets")
axes[1].set_title("Test CM - bestRf New Sets")
f.tight_layout()
plt.savefig("Confusion Matrices - New vs Old (Q4.2).png")


# Confidence Intervals
# global 95% confidence interval
z=1.96
type = ['bestRF Original Sets', 'bestRF New Sets']
```

```
# Confidence Intervals for all classes in CM
# global 95% confidence interval
z=1.96
CM_Labels = ['CALIBRI', 'COURIER', 'TIMES']
point = []
plus = []
neg = []

for i in range(len(type)):
    for j in range(testconf_bestRF[type[0]].shape[0]):
        N = TESTSET.groupby('font').size()[CM_Labels[j]]
        p_N = np.diagonal(testconf_bestRF[type[i]])[j]
        sigma = math.sqrt(((p_N * (1 - p_N)) / N))

        interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
        testconfint_point = round(np.diagonal(testconf_bestRF[type[i]])[j]*100, 2)
        testconfint_neg = round(interval[0]*100, 2)
        testconfint_plus = round(interval[1]*100, 2)

        point.append(testconfint_point)
        neg.append(testconfint_neg)
        plus.append(testconfint_plus)

#DataFrame of results
label = ['bestRF CALIBRI', 'bestRF COURIER', 'bestRF TIMES', 'newRF CALIBRI', 'newRF COURIER', 'newRF TIMES']
conf_int_dict = {'Type & Class Accuracy': label, 'Neg': neg, 'Point': point, "Plus": plus}
conf_int_df = pd.DataFrame(conf_int_dict)


# Global Confidence Intervals
# global 95% confidence interval
z=1.96
g_point = []
g_plus = []
g_neg = []

for i in range(len(type)):
    N = sum(TESTSET.groupby('font').size())
    p_N = testperf_bestRF[i]
    sigma = math.sqrt(((p_N * (1 - p_N)) / N))

    interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
    g_testconfint_point = round(testperf_bestRF[i] *100, 2)
    g_testconfint_neg = round(interval[0]*100, 2)
    g_testconfint_plus = round(interval[1]*100, 2)

    g_point.append(g_testconfint_point)
    g_neg.append(g_testconfint_neg)
```

```
    g_plus.append(g_testconfint_plus)

#DataFrame of results
label_g = ['bestRF CI', 'newRF CI']
g_conf_int_dict = {'Type Global CI': label_g, 'Neg': g_neg, 'Point': g_point, "Plus": g_plus}
g_conf_int_df = pd.DataFrame(g_conf_int_dict)



########################
#  ** Question 5.1 **  #
########################

P1_Y_Train = Y_Train.replace({'CALIBRI': 'CL1', 'COURIER': 'CL2 + CL3', 'TIMES': 'CL2 + CL3' })
P1_Y_Test = Y_Test.replace({'CALIBRI': 'CL1', 'COURIER': 'CL2 + CL3', 'TIMES': 'CL2 + CL3' })

P2_Y_Train = Y_Train.replace({'CALIBRI': 'CL1 + CL3', 'COURIER': 'CL2', 'TIMES': 'CL1 + CL3' })
P2_Y_Test = Y_Test.replace({'CALIBRI': 'CL1 + CL3', 'COURIER': 'CL2', 'TIMES': 'CL1 + CL3' })

P3_Y_Train = Y_Train.replace({'CALIBRI': 'CL1 + CL2', 'COURIER': 'CL1 + CL2', 'TIMES': 'CL3' })
P3_Y_Test = Y_Test.replace({'CALIBRI': 'CL1 + CL2', 'COURIER': 'CL1 + CL2', 'TIMES': 'CL3' })


# Train and Test Sets for P1, P2 , P3

P1_TESTSET  = pd.concat([P1_Y_Test, X_Test], axis=1)
P1_TRAINSET = pd.concat([P1_Y_Train, X_Train], axis=1)

P2_TESTSET  = pd.concat([P2_Y_Test, X_Test], axis=1)
P2_TRAINSET = pd.concat([P2_Y_Train, X_Train], axis=1)

P3_TESTSET  = pd.concat([P3_Y_Test, X_Test], axis=1)
P3_TRAINSET = pd.concat([P3_Y_Train, X_Train], axis=1)

# Creating X and Y values of from Training and Test Datasets for P's

# P1
X_Train_P1 = P1_TRAINSET.iloc[:,1:]
Y_Train_P1 = P1_TRAINSET.iloc[:,0]

X_Test_P1 = P1_TESTSET.iloc[:,1:]
Y_Test_P1 = P1_TESTSET.iloc[:,0]

# P2
X_Train_P2 = P2_TRAINSET.iloc[:,1:]
Y_Train_P2 = P2_TRAINSET.iloc[:,0]

X_Test_P2 = P2_TESTSET.iloc[:,1:]
```

```
Y_Test_P2 = P2_TESTSET.iloc[:,0]

# P3
X_Train_P3 = P3_TRAINSET.iloc[:,1:]
Y_Train_P3 = P3_TRAINSET.iloc[:,0]

X_Test_P3 = P3_TESTSET.iloc[:,1:]
Y_Test_P3 = P3_TESTSET.iloc[:,0]


# For Refernce in Balancing
P_Train = [P1_TRAINSET, P2_TRAINSET, P3_TRAINSET]
P_Test = [P1_TESTSET, P2_TESTSET, P3_TESTSET]

P_Train_Name = ['P1_TRAINSET Pre', 'P2_TRAINSET Pre', 'P3_TRAINSET Pre']
P_Test_Name = ['P1_TESTSET Pre', 'P2_TESTSET Pre', 'P3_TESTSET Pre']

P_Train_Size_Pre = {P_Train_Name[0]:P_Train[0].groupby('font').size(), P_Train_Name[1]:P_Train[1].groupby('font').
size(), P_Train_Name[2]:P_Train[2].groupby('font').size()}
P_Test_Size_Pre = {P_Test_Name[0]:P_Test[0].groupby('font').size(), P_Test_Name[1]:P_Test[1].groupby('font').size(
), P_Test_Name[2]:P_Test[2].groupby('font').size()}


#########################
# Balancing the Classes #
#########################

from imblearn.over_sampling import RandomOverSampler
from sklearn.utils import check_X_y, check_random_state, safe_indexing

# Over Sampling of Minority Classes
# define oversampling strategy
oversample = RandomOverSampler(sampling_strategy='not majority', random_state = 0)

# fit and apply the transform
#P1
X_Train_P1, Y_Train_P1 = oversample.fit_resample(X_Train_P1, Y_Train_P1)
#X_Test_P1, Y_Test_P1 = oversample.fit_resample(X_Test_P1, Y_Test_P1)
#P2
X_Train_P2, Y_Train_P2 = oversample.fit_resample(X_Train_P2, Y_Train_P2)
#X_Test_P2, Y_Test_P2 = oversample.fit_resample(X_Test_P2, Y_Test_P2)
#P3
X_Train_P3, Y_Train_P3 = oversample.fit_resample(X_Train_P3, Y_Train_P3)
#X_Test_P3, Y_Test_P3 = oversample.fit_resample(X_Test_P3, Y_Test_P3)


# Merging the new a balanced matrix
# P1
P1_TRAINSET = pd.concat([Y_Train_P1, X_Train_P1], axis=1)
```

```
P1_TESTSET = pd.concat([Y_Test_P1, X_Test_P1], axis=1)
# P2
P2_TRAINSET = pd.concat([Y_Train_P2, X_Train_P2], axis=1)
P2_TESTSET = pd.concat([Y_Test_P2, X_Test_P2], axis=1)
# P3
P3_TRAINSET = pd.concat([Y_Train_P3, X_Train_P3], axis=1)
P3_TESTSET = pd.concat([Y_Test_P3, X_Test_P3], axis=1)


# Class Dist. After Balancing
P_Train = [P1_TRAINSET, P2_TRAINSET, P3_TRAINSET]
P_Test = [P1_TESTSET, P2_TESTSET, P3_TESTSET]

P_Train_Name = ['P1_TRAINSET Post', 'P2_TRAINSET Post', 'P3_TRAINSET Post']
P_Test_Name = ['P1_TESTSET Post', 'P2_TESTSET Post', 'P3_TESTSET Post']

P_Train_Size_Post = {P_Train_Name[0]:P_Train[0].groupby('font').size(), P_Train_Name[1]:P_Train[1].groupby('font')
.size(),P_Train_Name[2]:P_Train[2].groupby('font').size()}
P_Test_Size_Post = {P_Test_Name[0]:P_Test[0].groupby('font').size(), P_Test_Name[1]:P_Test[1].groupby('font').size
(), P_Test_Name[2]:P_Test[2].groupby('font').size()}

# Creating DataFrames
P_Train_Size_Pre_df = pd.DataFrame(P_Train_Size_Pre)
P_Train_Size_Post_df = pd.DataFrame(P_Train_Size_Post)

P_Test_Size_Pre_df = pd.DataFrame(P_Test_Size_Pre)
P_Test_Size_Post_df = pd.DataFrame(P_Test_Size_Post)

# Compiling Pre and Post Train and Test into DF
P_Train_Size_df = pd.concat([P_Train_Size_Pre_df,P_Train_Size_Post_df], axis=1).T
P_Test_Size_df = pd.concat([P_Test_Size_Pre_df,P_Test_Size_Post_df], axis=1).T


#########################
#     Random Forest     #
#########################

# Setting up for RF Loop

X_Train_P = [X_Train_P1, X_Train_P2, X_Train_P3]
Y_Train_P = [Y_Train_P1, Y_Train_P2, Y_Train_P3]

X_Test_P = [X_Test_P1, X_Test_P2, X_Test_P3]
Y_Test_P = [Y_Test_P1, Y_Test_P2, Y_Test_P3]

trainperf_P = []
testperf_P = []
testconf_P = {}
```

```
CM_Labels_P = [['CL1', 'CL2 + CL3'],['CL2', 'CL1 + CL3'],['CL3', 'CL1 + CL2']]


# RF For Loop
for i in range(len(X_Train_P)):
    # Fitting RF to the Training set
    clf = RandomForestClassifier(n_estimators=bntr, max_features="sqrt", random_state=0)
    clf.fit(X_Train_P[i], Y_Train_P[i])
    # Predicting the Train/Test Set results
    Y_Pred_Train = clf.predict(X_Train_P[i])
    Y_Pred = clf.predict(X_Test_P[i])
    # Appending Performance for both sets
    trainperf_P.append(metrics.accuracy_score(Y_Train_P[i], Y_Pred_Train))
    testperf_P.append(metrics.accuracy_score(Y_Test_P[i], Y_Pred))
    testconf_P[i] = confusion_matrix(Y_Test_P[i], Y_Pred, labels = CM_Labels_P[i], normalize = 'true')


A =['A1','A2','A3']

# Test and Train Accuracy Performances
test_train_perf_dict_A = {'A': A, 'Train Accuracy': trainperf_P ,'Test Accuracy': testperf_P}
test_train_perf_A_df = pd.DataFrame(test_train_perf_dict_A)


# Confusion Matrix Plots
f, axes = plt.subplots(1, 3, figsize = (12,6))

#CM Labels
CM_Labels_P_true = [['true:CL1', 'true:CL2 + CL3'],['true:CL2', 'true:CL1 + CL3'],['true:CL3', 'true:CL1 + CL2']]
CM_Labels_P_pred = [['pred:CL1', 'pred:CL2 + CL3'],['pred:CL2', 'pred:CL1 + CL3'],['pred:CL3', 'pred:CL1 + CL2']]
M =['M1','M2','M3']

# Confusion Matrix RF For Loop
for i in range(len(X_Train_P)):
    # Visuals for Report
    cmtx_test = pd.DataFrame(testconf_P[i],
        index=CM_Labels_P_true[i],
        columns=CM_Labels_P_pred[i])
    #Plotting itself
    a = sns.heatmap(cmtx_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes.fla
t[i])
    a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
    a.set_yticklabels(a.get_yticklabels(), fontsize = 12, verticalalignment = 'center')
    #Title
    axes.flat[i].set_title(str(M[i]) + "  - Test CM")

f.tight_layout()
plt.savefig("3 M's Tests CM.png")
```

```
# Confidence Intervals for all classes in CM
# global 95% confidence interval
z=1.96
CM_Labels_P = [['CL1', 'CL2 + CL3'],['CL2', 'CL1 + CL3'],['CL3', 'CL1 + CL2']]
point = []
plus = []
neg = []

for i in range(len(A)):
    for j in range(testconf_P[0].shape[0]):
        N = P_Test[i].groupby('font').size()[CM_Labels_P[i][j]]
        p_N = np.diagonal(testconf_P[i])[j]
        sigma = math.sqrt(((p_N * (1 - p_N)) / N))

        interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
        testconfint_point = round(np.diagonal(testconf_P[i])[j] *100, 2)
        testconfint_neg = round(interval[0]*100, 2)
        testconfint_plus = round(interval[1]*100, 2)

        point.append(testconfint_point)
        neg.append(testconfint_neg)
        plus.append(testconfint_plus)

#DataFrame of results
label = ['M1 CL1', 'M1 {CL2 + CL3}', 'M2 CL2', 'M2 {CL1 + CL3}', 'M3 CL3', 'M3 {CL1 + CL2}' ]
conf_int_A_dict = {'Problem & Class Accuracy': label, 'Neg': neg, 'Point': point, "Plus": plus}
conf_int_A_df = pd.DataFrame(conf_int_A_dict)


# Global Confidence Intervals
# global 95% confidence interval
z=1.96
g_point = []
g_plus = []
g_neg = []

for i in range(len(A)):
    N = sum(P_Test[i].groupby('font').size())
    p_N = testperf_P[i]
    sigma = math.sqrt(((p_N * (1 - p_N)) / N))

    interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
    g_testconfint_point = round(testperf_P[i] *100, 2)
    g_testconfint_neg = round(interval[0]*100, 2)
    g_testconfint_plus = round(interval[1]*100, 2)
```

```
    g_point.append(g_testconfint_point)
    g_neg.append(g_testconfint_neg)
    g_plus.append(g_testconfint_plus)

#DataFrame of results
label_g = ['A1 CI', 'A2 CI', 'A3 CI']
g_conf_int_A_dict = {'A CI': label_g, 'Neg': g_neg, 'Point': g_point, "Plus": g_plus}
g_conf_int_A_df = pd.DataFrame(g_conf_int_A_dict)



########################
#  ** Question 5.2 **  #
########################
#C1 = CALIBRI, C2 = COURIER, C3= TIMES

# Setting up 5.2
#Y_Test_P_df = pd.concat([Y_Test_P1, Y_Test_P2, Y_Test_P3])
Y_Pred_bestRF_df = pd.DataFrame(Y_Pred_bestRF, columns = ['bestRF_pred'])

#Y_Pred_bestRF_df['Y_Test'] = Y_Test_P_df.reset_index(drop = True)
Y_Pred_bestRF_df['best_predCL1'] = np.where(Y_Pred_bestRF_df['bestRF_pred'] == 'CALIBRI', 'CL1', 'CL2 + CL3')
Y_Pred_bestRF_df['best_predCL2'] = np.where(Y_Pred_bestRF_df['bestRF_pred'] == 'COURIER', 'CL2', 'CL1 + CL3')
Y_Pred_bestRF_df['best_predCL3'] = np.where(Y_Pred_bestRF_df['bestRF_pred'] == 'TIMES', 'CL3', 'CL1 + CL2')
Y_Pred_bestRF_df['Y_TestCL1'] = Y_Test_P1.reset_index(drop = True)
Y_Pred_bestRF_df['Y_TestCL2'] = Y_Test_P2.reset_index(drop = True)
Y_Pred_bestRF_df['Y_TestCL3'] = Y_Test_P3.reset_index(drop = True)


#########################
#     Random Forest B    #
#########################

# Setting up for RF Loop
testperf_P = []
testconf_P = {}

CM_Labels_P = [['CL1', 'CL2 + CL3'],['CL2', 'CL1 + CL3'],['CL3', 'CL1 + CL2']]


B1 = metrics.accuracy_score(Y_Pred_bestRF_df['Y_TestCL1'], Y_Pred_bestRF_df['best_predCL1'])
B2 = metrics.accuracy_score(Y_Pred_bestRF_df['Y_TestCL2'], Y_Pred_bestRF_df['best_predCL2'])
B3 = metrics.accuracy_score(Y_Pred_bestRF_df['Y_TestCL3'], Y_Pred_bestRF_df['best_predCL3'])

BM1 = confusion_matrix(Y_Pred_bestRF_df['Y_TestCL1'], Y_Pred_bestRF_df['best_predCL1'], labels = CM_Labels_P[0], n
ormalize = 'true')
BM2 = confusion_matrix(Y_Pred_bestRF_df['Y_TestCL2'], Y_Pred_bestRF_df['best_predCL2'], labels = CM_Labels_P[1], n
ormalize = 'true')
```

```
BM3 = confusion_matrix(Y_Pred_bestRF_df['Y_TestCL3'], Y_Pred_bestRF_df['best_predCL3'], labels = CM_Labels_P[2], n
ormalize = 'true')

testperf_P = [B1,B2,B3]
testconf_P = {0 :BM1, 1: BM2, 2: BM3}

B =['B1','B2','B3']

# Test and Train Accuracy Performances
test_train_perf_dict_B = {'B': B,'Test Accuracy': testperf_P}
test_train_perf_B_df = pd.DataFrame(test_train_perf_dict_B)


# Confusion Matrix Plots
f, axes = plt.subplots(1, 3, figsize = (12,6))

#CM Labels
CM_Labels_P_true = [['true:CL1', 'true:CL2 + CL3'],['true:CL2', 'true:CL1 + CL3'],['true:CL3', 'true:CL1 + CL2']]
CM_Labels_P_pred = [['pred:CL1', 'pred:CL2 + CL3'],['pred:CL2', 'pred:CL1 + CL3'],['pred:CL3', 'pred:CL1 + CL2']]
BM =['BM1','BM2','BM3']

# Confusion Matrix RF For Loop
for i in range(len(X_Train_P)):
    # Visuals for Report
    cmtx_test = pd.DataFrame(testconf_P[i],
        index=CM_Labels_P_true[i],
        columns=CM_Labels_P_pred[i])
    #Plotting itself
    a = sns.heatmap(cmtx_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes.fla
t[i])
    a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
    a.set_yticklabels(a.get_yticklabels(), fontsize = 12, verticalalignment = 'center')
    #Title
    axes.flat[i].set_title(str(BM[i]) + "  - Test CM")

f.tight_layout()
plt.savefig("3 BM's Tests CM.png")


# Confidence Intervals for all classes in CM
# global 95% confidence interval
z=1.96
CM_Labels_P = [['CL1', 'CL2 + CL3'],['CL2', 'CL1 + CL3'],['CL3', 'CL1 + CL2']]
point = []
plus = []
neg = []

for i in range(len(A)):
```

```
    for j in range(testconf_P[0].shape[0]):
        N = P_Test[i].groupby('font').size()[CM_Labels_P[i][j]]
        p_N = np.diagonal(testconf_P[i])[j]
        sigma = math.sqrt(((p_N * (1 - p_N)) / N))

        interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
        testconfint_point = round(np.diagonal(testconf_P[i])[j] *100, 2)
        testconfint_neg = round(interval[0]*100, 2)
        testconfint_plus = round(interval[1]*100, 2)

        point.append(testconfint_point)
        neg.append(testconfint_neg)
        plus.append(testconfint_plus)

#DataFrame of results
label = ['BM1 CL1', 'BM1 {CL2 + CL3}', 'BM2 CL2', 'BM2 {CL1 + CL3}', 'BM3 CL3', 'BM3 {CL1 + CL2}' ]
conf_int_B_dict = {'Problem & Class Accuracy': label, 'Neg': neg, 'Point': point, "Plus": plus}
conf_int_B_df = pd.DataFrame(conf_int_B_dict)


# Global Confidence Intervals
# global 95% confidence interval
z=1.96
g_point = []
g_plus = []
g_neg = []

for i in range(len(A)):
    N = sum(P_Test[i].groupby('font').size())
    p_N = testperf_P[i]
    sigma = math.sqrt(((p_N * (1 - p_N)) / N))

    interval = stats.norm.interval(alpha = 0.95, loc=p_N, scale=sigma)
    g_testconfint_point = round(testperf_P[i] *100, 2)
    g_testconfint_neg = round(interval[0]*100, 2)
    g_testconfint_plus = round(interval[1]*100, 2)

    g_point.append(g_testconfint_point)
    g_neg.append(g_testconfint_neg)
    g_plus.append(g_testconfint_plus)

#DataFrame of results
label_g = ['B1 CI', 'B2 CI', 'B3 CI']
g_conf_int_B_dict = {'B CI': label_g, 'Neg': g_neg, 'Point': g_point, "Plus": g_plus}
g_conf_int_B_df = pd.DataFrame(g_conf_int_B_dict)
```

```
######################
#  ** DF To Excel **  #
######################

with pd.ExcelWriter('DataFrame Outputs.xlsx') as writer:
    SDATA_PCA_ID.to_excel(writer, sheet_name='SDATA_PCA_ID')
    test_train_perf_df_3_1.to_excel(writer, sheet_name='Test & Train Perf (Q3.1)')
    test_train_perf_df.to_excel(writer, sheet_name='Test & Train Perf')
    testconf_diag_df.to_excel(writer, sheet_name='Test Conf Diag')
    feature_imp_bestRF.to_excel(writer, sheet_name='Feature IMP - bestRF')
    test_train_perf_df_bestRF.to_excel(writer, sheet_name='Test Train Perf - bestRF')
    conf_int_df.to_excel(writer, sheet_name='Class Accuracy - CI')
    g_conf_int_df.to_excel(writer, sheet_name='Global Accuracy - CI')
    P_Train_Size_df .to_excel(writer, sheet_name='Ps Train Size')
    P_Test_Size_df.to_excel(writer, sheet_name='Ps Test Size')
    test_train_perf_A_df.to_excel(writer, sheet_name='Test Train Perf - As')
    conf_int_A_df.to_excel(writer, sheet_name='Class Accuracy - CI for As.')
    g_conf_int_A_df.to_excel(writer, sheet_name='Global Accuracy - CI for As.')
    test_train_perf_B_df.to_excel(writer, sheet_name='Test Train Perf - Bs')
    conf_int_B_df.to_excel(writer, sheet_name='Class Accuracy - CI for Bs.')
    g_conf_int_B_df.to_excel(writer, sheet_name='Global Accuracy - CI for Bs.')
```