Midterm

Johnny Nino Ladino; 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.

This paper is based on the assignment provided in the course. Contents of the paper have not been reviewed by the professor and are subject to correction by the author. The material in this paper was compiled, developed, and/or synthesized by the individual student and does not necessarily reflect any position of the Department of Mathematics; its students, staff, or faculty; or University of Houston.

Background and Class Creation

This report will cover the tasks for the Final Project presented by Dr. Azencott. The dataset that will be used for this project is named age_Gender.csv and was retrieved from the Kaggle website. Each row represents a 'case' and each case describes numerically a digitized image of an individual's face. Each image has a 48 by 48 size image and therefore, the image has 2,304 pixels. The pixels for each case are listed in within a single column named "pixels." Each one of these 2,304 pixels has a "gray level" which is an integer value between 0 and 255. There's a total of 23,705 images in this dataset, each row representing the face of a distinct individual. The dataset consists of 5 columns which are age, ethnicity, gender, image_name, and pixels. The age column describes the person's age in the image for each case. The ethnicity column describes the ethnicity of the person in the image for each case. The Gender column describes the Gender of the of the person in the image for each case. The image_name column is the specific name of the jpg file for each image. Lastly, the pixels column has the list of "gray level" integer values in the image for each case.

For this dataset, gender and age will be our target variable and the 2,304 values describing the gray levels will be our features. To keep the number of classes greater than 5 and less than 10, we will only look at ages from 1 to 75. This changes the number of cases from 23,705 images to 22,854 images. Bins will be created to categorize age. Ages 1 to 25 will be bin 1 and they will be labeled as the "Young" Age Group. Ages 26 to 50 will be bin 2 and they will be labeled as the "Middle" Age Group. Lastly, Ages 51 to 75 will be bin 3 and they will be labeled as the "Old" Age Group. Our target variable will have 6 classes. 3 classes will be the females within their respective Age Group. The other 3 classes will be the males within their 5 respective Age Groups. Table 1 below defines our classes. Our features are integer values ranging from 0 to 255. The aim of this report will be to classify the Gender and Age Group of the person displayed in each image. The underlying code sustaining this report will be created using python and the IDE of Jupyter notebook. Figure 1 below is a sample of images from our dataset.

The ability to classify gender and age has many practical applications. This could lead to creating devices that are able to do face recognition. Face recognition already exists today. There are features in smartphones that can unlock your phone using the camera. Being able to determine who's face it is, will help your phone decide whether to unlock or not. Being able to determine age and gender fits into those categories.

CLASS	Gender	Age Group
Class FY	Female	Young (Age:0-25)
Class FM	Female	Middle (Age:26-50)
Class FO	Female	Old (Age:51-75)
Class MY	Male	Young (Age:0-25)
Class MM	Male	Middle (Age:26-50)
Class MO	Male	Old (Age:51-75)

Table 1:Class Definition

Figure 1:Sample of Images from Dataset

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 use python libraries such as pandas, numpy, scikit learn etc
2. Code and report will abide to the requirements presented in Dr. Azencott's instructions
3. The report/code will start by applying PCA to our standardized data and will conduct analysis on results
4. The report/code will then use Kmeans clustering to measure features importance
5. The report/code will use Random Forest to classify our cases to our desired classes and optimize such classifier
6. Lastly, the report/code will apply Support Vector Machines to classify two classes from our dataset

Preliminary Treatment of Data

To begin the analysis, data cleaning and preparation needs to be done on the dataset. Out of the 5 columns in our dataset, we will keep only 3 of them; age, gender, and pixels. Table 2 below explains what each of these columns represent. Luckily, the dataset has no missing data.

COLUMN	DESCRIPTION
AGE	Column lists values of the person's age in the image, ranging from 1 to 116
GENDER	Column lists values either equal to 0 or to 1. 0 = Male, 1 = Female
PIXELS	Column lists array of values ranging from 0 to 255; each value representing gray level intensity

Table 2: Column Description

As mentioned in the "Background and Class Creation" section, to keep class size greater than 5 and less than 10, we drop the images of the people that are strictly older than 75. For our pixel column, we partition the pixel into each column for each image, thus we get 2,308 added columns to our dataset. This will make it easier to work with when standardizing, creating the test and train sets, and feeding our data into our model. We then create the classes that were labeled in Table 1. Table 3 below will give the size of each class along with a visual representation of the distribution of the classes.

CLASS	ROWS	COLUMNS
CLFY (FEMALE AND YOUNG)	4,464	2,308
CLFM (FEMALE AND MIDDLE)	5,166	2,308
CLFO (FEMALE AND OLD)	1,205	2,308
CLMY(MALE AND YOUNG)	3,173	2,308
CLMM(MALE AND MIDDLE)	6,296	2,308
CLMO(MALE AND OLD)	2,550	2,308
TOTAL	**22,854**	

Table 3:Class Size

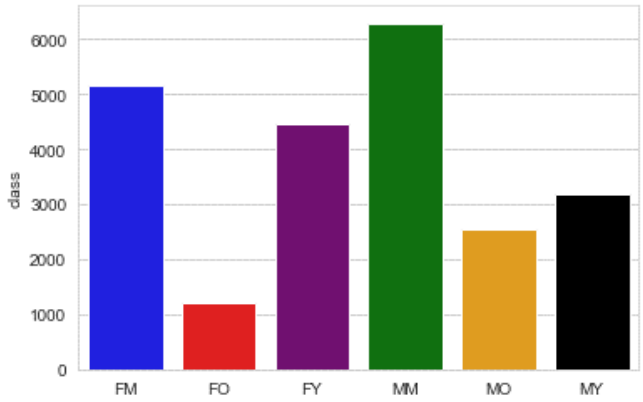

Figure 2: Class Size by Count

Standardizing the Data and Correlation Matrix

We will denote our dataset of pixels and classes as DATA. 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 $\hat{\mu}(mean) = \mu_{X1}, \dots, \mu_{X2,304}$ and $\hat{\sigma}(standard\ deviation) = \sigma_{X1}, \dots, \sigma_{X2,304}$. 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:Standard Matrix Equation

We store the standardized dataset DATA into a separate data frame named SDATA. From SDATA, we compute the correlation matrix, $CORR(X_i, X_j)$, and extract the 10 pairs X_i, X_j that have the highest absolute values in $|CORR(X_i, X_j)|$. Here, X_i and X_j, refer to the position of the pixel. Table 4 below highlights the 10 highest correlated pairs by pixel position. From our pixel positions X_i, X_j, we see that the highest correlated pairs tend to share the same row and column position.

$\boldsymbol{Corr(X_i, X_j)}$	$\boldsymbol{X_i}$	$\boldsymbol{X_j}$
0.9880	r18c23	r19c23
0.9876	r19c24	r18c24
0.9876	r23c6	r22c6
0.9876	r24c39	r25c39
0.9872	r22c41	r23c41
0.9871	r23c40	r22c40
0.9870	r24c39	r23c39
0.9869	r24c38	r25c38
0.9867	r24c7	r25c7
0.9866	r24c37	r25c37

Table 4: Highest Correlated Pairs by Pixel Position

Before moving forward, we would like to look at the features with the highest correlation within each class. This will allow us to see which pixel locations have higher correlations than others from different classes.

CLFY		**Corr**	**CLFM**		**Corr**	**CLFO**		**Corr**
r18c23	**r19c23**	0.99	**r19c24**	**r18c24**	0.99	**r20c33**	**r20c34**	0.99
r25c40	**r24c40**	0.99	**r20c24**	**r19c24**	0.99	**r21c13**	**r21c14**	0.99
r24c39	**r25c39**	0.99	**r19c23**	**r18c23**	0.99	**r24c38**	**r25c38**	0.99
r23c6	**r22c6**	0.99	**r22c6**	**r23c6**	0.99	**r19c33**	**r19c34**	0.99
CLMY			**CLMM**			**CLMO**		
r24c9	**r25c9**	0.99	**r2c22**	**r2c23**	0.99	**r0c21**	**r0c20**	0.99
r18c5	**r19c5**	0.99	**r1c24**	**r1c23**	0.99	**r1c26**	**r1c27**	0.99
r21c34	**r21c33**	0.99	**r1c23**	**r1c22**	0.99	**r0c29**	**r0c28**	0.99
r25c9	**r26c9**	0.99	**r1c25**	**r1c24**	0.99	**r0c26**	**r0c25**	0.99

Table 5:Top 4 Correlation

Table 5 is great because we see where the highest correlations appear for each class. For the young female class, we see that the highest correlation happens at pixel location (**r18c23**, **r19c23**) and for the old male class, we see that the highest correlation happens at pixel location (**r0c21**, **r0c20**). This is great for distinguishing our classes using our features!

Question 1 – PCA

We will explore a technique called Principal Component Analysis. This technique compresses the data by reducing the number of features. PCA decorrelates the features to avoid getting noise from our data. When we apply the PCA to our SDATA, we can determine that 95% of the data is explained with only 182 new features. This is great because it drastically lowers our feature size count from 2,304 to 182. For clarification, we feed the SDATA into our PCA and then transform it using the weights obtained through PCA to our features.

Approximate time taken to run PCA on SDATA: 1 minute. (Multiple runs were made.)

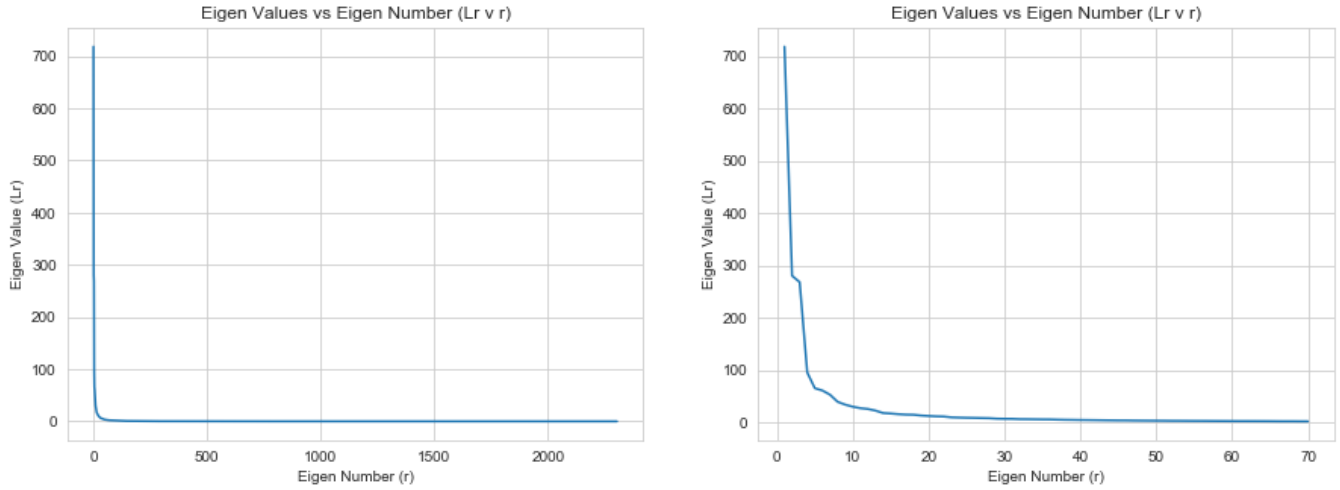

Figure 3:Eigen Values for SDATA (Left: All, Right: First 70)

In Figure 3, we have the plot of the Eigen values for SDATA for each feature. In the left, it's hard to see the importance of some of the features since we have so many of them. We choose to look at the first 70 in the right of Figure 3. Here we can see that the first 20 features have higher eigen values in comparison to the rest. Eigenvalues that are close to zero represent components that could possibly tell little information about the dataset.

Proportion of Variance Explained describes the variance explained by the r principal components. PVE(r) is calculated by the following equation:

$$\boldsymbol{PVE(r) = \frac{1}{p}\sum_{r=1}^{R} L_r.}$$

Equation 2: Percentage of Explained Variance (PVE)

In Equation 2, p is the total number of features in our original standardized data set, SDATA, which in our case is 2,304. In Figure 4, we have a Cumulative Scree Plot. For the Cumulative Scree Plot, we summed the first r eigenvalues to obtain cumulative PVE for the first r components. Hence, the Cumulative Scree Plot increases as r increases.

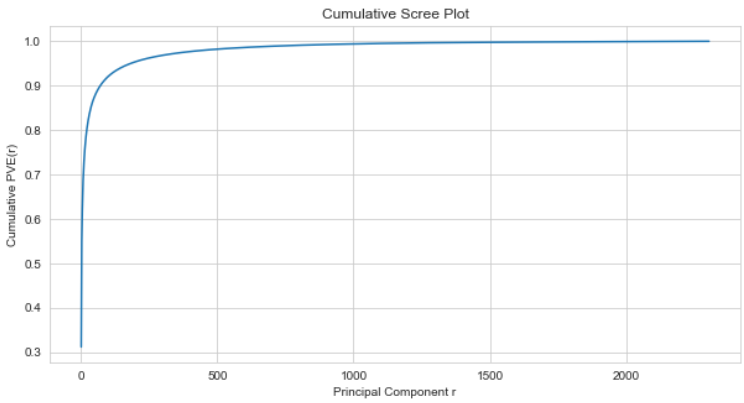

Figure 4: Cumulative Scree Plot

r	Eigen Values	Percentage of Variance Explained
1	718.39	31.18%
2	280.39	12.17%
3	268.27	11.64%

Table 6: First 3 Eigen Values and their PVE

In Figure 4 and Table 6, the r that gives us a PVE of 95% is r = 182. This allows us to reduce the number of features without losing a large amount of information from our data.

In Figure 5, we plot the first 3 Principal Components by each class, and we project these vectors onto a 3D plot. Here we can see if our classes are easily distinguishable by the application of PCA. Based on these 3D plots, we can make suggestions. There is a fair amount of separability between each class, but some are not as apparent. For example, MY and FY classes seem to have some partition between principal component 2, but it is not clear as well. Realistically, this makes sense since even humans have a hard time distinguishing the gender of a baby, but as the child grows to up to 25, gender becomes more distinguishable. The classification of age becomes ambiguous once we get closer to the partitions that were created. Classes such as FY and MO, have more apparent separability. This might indicate that it easier to distinguish between a younger aged female in comparison to an older aged male.

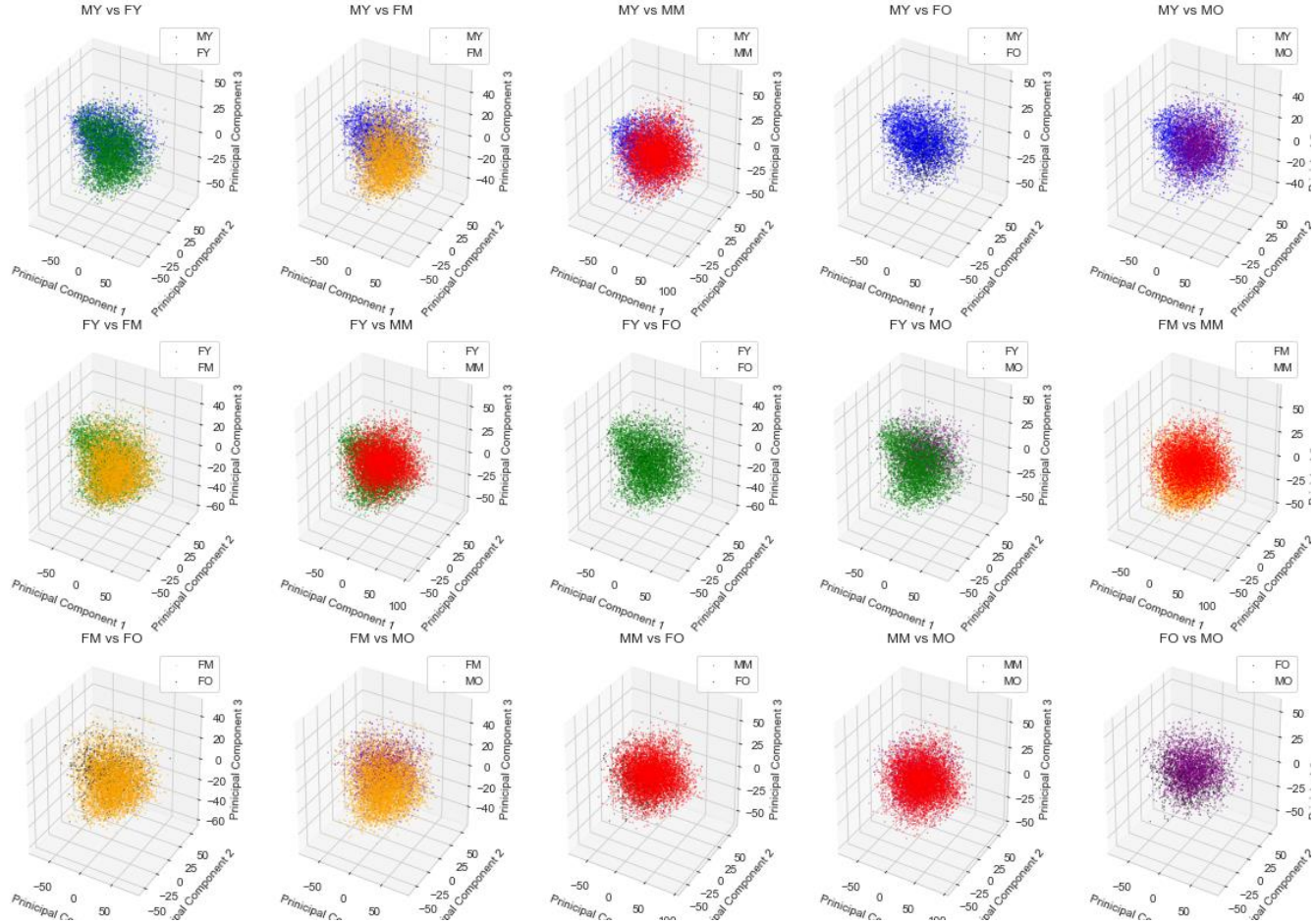

Figure 5: First 3 Principal Components plotted by Class

Question 2 – K-Means Clustering

We will use SDATA and apply the k-means algorithm for $K = 1, \ldots, 10$, using 50 runs with different initial centroids to partition our cases into K disjoint clusters.

Approximate time taken to run K-Means on SDATA: 40 minutes. (Multiple runs were made.)

$$Perf(k) = 1 - \frac{SSE(k)}{SSE(1)}$$

Equation 3: Calculating Perf(K)

For each K, we compute the reduction of variance, which we denote as Perf(k), using Equation 3. Where SSE is the sum of squares error for each case within a cluster. We then plot the Knee plot and Impurity plots to determine which is our best number of clusters that will partition our dataset. In the Knee plot, we see a bend at $K = 4$, but it is not as apparent where there is a significant bend, so we use the Impurity plot to help us as well.

The class frequency within each cluster was calculated using Equation 4 below. This helps analyze the percentage each class within each cluster to easily identify which class was predominately present in each cluster.

$$Fm(j) = \frac{A(m,j)}{Size(j)}$$

Equation 4: Class Frequency Within Each Cluster

The Gini index was calculated for each cluster using Equation 5 below. The Gini index represents the impurity of each cluster CLUj. The maximum Gini index for a set made of up 6 classes is 0.83. The higher the value and closer it is to 0.83, the more impure the cluster is. High purity for a cluster would be close to 0.

$$Gini(CLUj) = F_{(1,1)} * \left(1 - F_{(1,1)}\right) + \cdots + F_{(m,j)} * (1 - F_{(m,j)})$$

Equation 5: Gini by Cluster J

The impurity of the K-Means clustering for K clusters was calculated using Equation 6 below. The impurity was calculated by summing all the Gini indexes for each of the K clusters and gives a more holistic view of to the impurity of the clustering.

$$IMP(K) = gini(CLU1) + \ldots + gini(CLUj)$$

Equation 6: Impurity IMP(K)

From the Impurity plot, in Figure 6, it is hard to determine visually which K cluster has the lowest increase in impurity when increasing the number of clusters. This might indicate that our cases are difficult to cluster. Looking at our impurity at a granular view. We subtract the impurities for each step increase in K clusters. The step from K=3 to K=4 gives us a difference of 0.73, which is the smallest difference of all other steps for the impurity thus we choose K best to be 4.

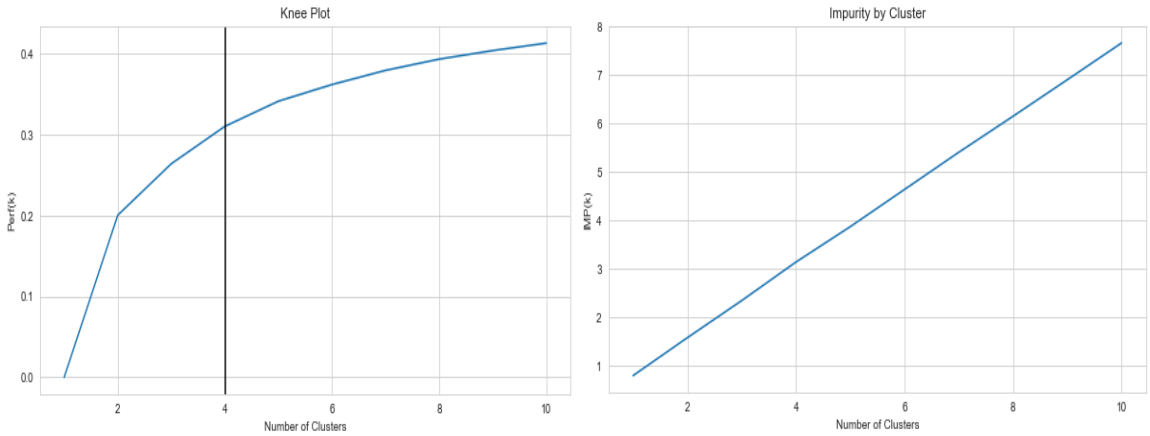

Figure 6: Knee by K Clusters Plot and Impurity by K Clusters Plot

Question 3 – Best K Cluster

In Table 7, we have all the details about the 4 Clusters in our K-Means. We have the center, size, dispersion, gini index, and frequencies of classes for each cluster CLUj. The first 3 coordinates for each cluster center is shown since we have 2,304 coordinates. Things that we point out are that the lowest gini index is of cluster 1. We can also see that in cluster 1, we have the highest frequency of MM classes compared to the MM frequencies for other clusters. We see a small frequency for FO and this is due to the size of this class being so small.

CLUj	Centerj1	Centerj2	Centerj3	Sizej	DISj	Gj	Fj	
0	0.51	0.54	0.57	5379.00	8402818.22	0.80	FM	0.15
							FO	0.06
							FY	0.25
							MM	0.19
							MO	0.09
							MY	0.25
1	-0.14	-0.17	-0.21	5638.00	8726895.46	0.75	FM	0.21
							FO	0.04
							FY	0.12
							MM	0.40
							MO	0.13
							MY	0.11
2	-0.07	-0.04	0.01	5783.00	9345696.66	0.79	FM	0.26
							FO	0.06
							FY	0.20
							MM	0.26
							MO	0.11
							MY	0.10
3	-0.25	-0.28	-0.32	6054.00	9830706.70	0.79	FM	0.28
							FO	0.06
							FY	0.21
							MM	0.25
							MO	0.10
							MY	0.11

Table 7:K Best Means Cluster Descriptive

In Figure 7, we have a 3D plot of each case plotted against each cluster label it was assigned to by the K-Means Clustering. If we pay close attention to cluster 1 plots in Figure 7, we can see that they tend to be partitioned by the first Principal Component. The first Principal Component may be able to distinguish a good amount of the MM cases based on the results of the plot and table. The same can be said about cluster 0, which remains to the left of the first principal component.

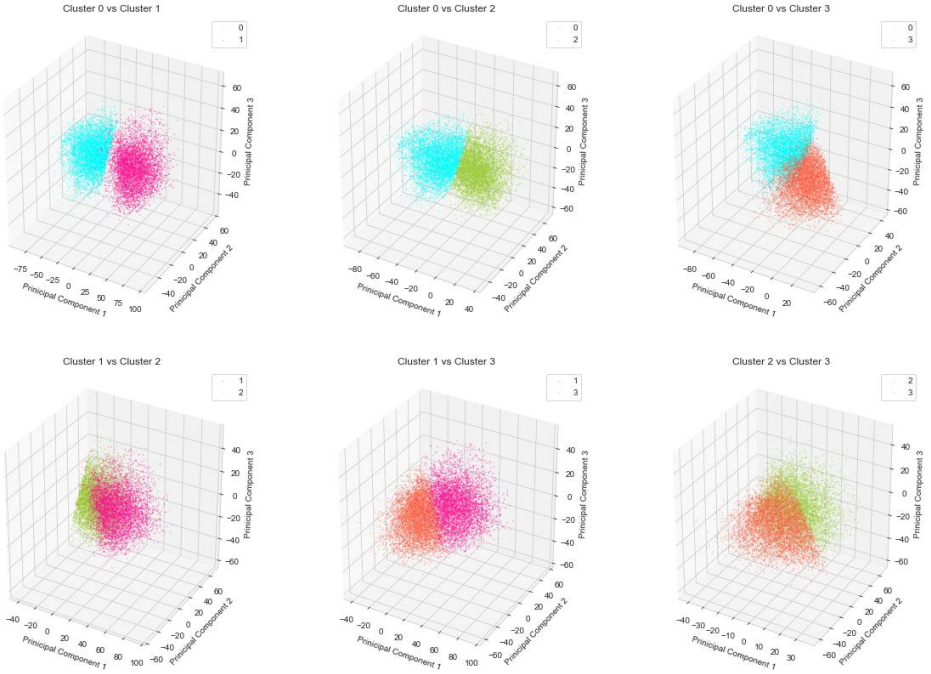

Figure 7: First 3 Principal Components by Cluster Assignment

Question 4 – Train and Test Splits/Rebalancing Classes

There are various ways to fix the imbalance in these classes. In this report we will explore which method gives us the best results. To makes things simple and quick starting off, we will use SMOTE and Random Under Sampler from the imblearn python library. SMOTE creates artificial cases using K nearest neighbors as a parameter. Random Under Sampler grabs a random subsample from the cases given. Figure 2 displays our size of classes. First, we will perform a train and test split then we will rebalance the classes from there.

The 80/20 approach of separating data will be taken for the standardized matrix, SDATA. CLFY, CLFM, CLFO, CLMY, CLMM, and CLMO were split into subsets of 80% and 20% at random. The 80% subset will be for training data and the other 20% subset will be for the test data. Once the 80/20 split for each of the 6 classes was created, a union of each training and test set was created to have a singular training and test dataset. The 80/20 approach helps prevent bias towards one specific class and helps ensure that all classes are considered for the training and test portion of the model evaluation. Table 8 below highlights how this approach splits the data evenly.

DATASET	CLFY ROW COUNT	CLFM ROW COUNT	CLFO ROW COUNT	CLMY ROW COUNT	CLMM ROW COUNT	CLMO ROW COUNT	TOTAL
TRAINSET (80%)	3571	4132	964	2538	5036	2040	18,281
TESTSET (20%)	893	1034	241	635	1260	510	4,573

Table 8:Class Size Before Rebalancing

We perform smote and use 30 as the number of nearest neighbors since this was the best number from our previous analysis in the midterm. We then resample our classes as follow.{'MM': 3300, 'FM': 3200, 'FY': 3100, 'MY': 3000, 'MO': 2900, 'FO': 2800}for the train test. {'MM': 820, 'FM': 800, 'FY': 780, 'MY': 760, 'MO': 740, 'FO': 720} for the test set. The reasoning behind this choice was that we wanted to average the size for each class while still maintaining size order. We can see the train rebalance in Figure 8.

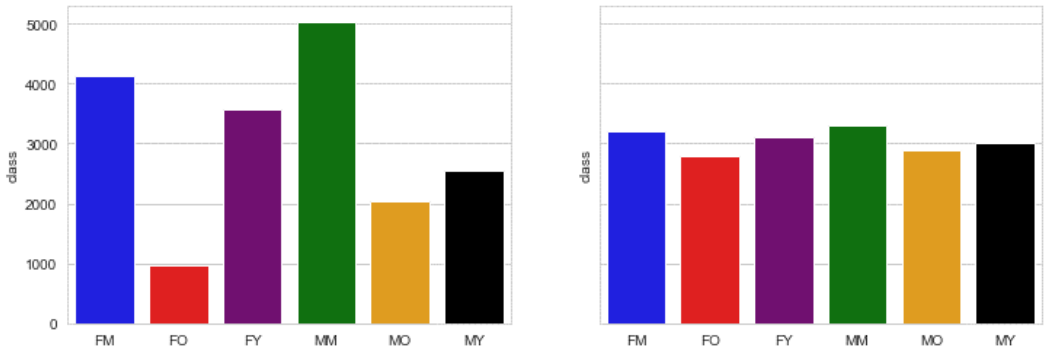

Figure 8: Train Set Rebalance

Question 5 – Random Forest

We will apply Random Forest on our rebalanced data to classify our SDATA. We will do iterations for 100, 200, 300, and 400 trees in our Random Forest. The maximum number of features we will consider will be the square root of 2,304, which rounded is 48.

Approximate time taken to run Random Forest on SDATA: 23 minutes. (Multiple runs were made.)

In Figure 9, we see that how well our Random Forest does. If just looking at our train and test accuracies, we see that there's a big gap and this might indicate overfitting in our model. Out of bag, OOB, is a score that will create a much more accurate score for training set. "The out-of-bag (OOB) error is the average error for each calculated using predictions from the trees that do not contain in their respective bootstrap sample." There is a big increase in accuracy for 100 trees to 200 trees.

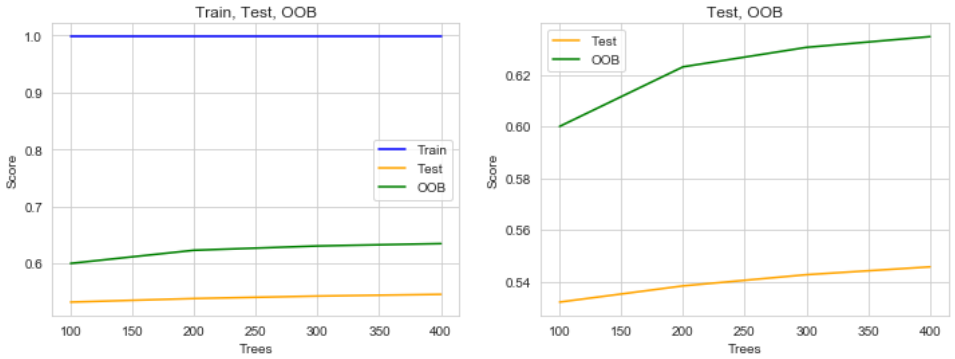

Figure 9: Train, Test and OOB Score

In Figure 10, we have the confusion matrix for each iteration of trees. We see that from 100 trees to 200 trees we have a 2 percent increase for correctly classifying MM. After, results kind of level off. We also see where there might be poor results. As mentioned earlier, we see that MY and FY get confused for each other. FM and FY are a big portion of the misclassification. It seems that it is more difficult to classify ages of young aged and middle-aged females.

Figure 10: Test Confusion Matrices

Question 6 – Best Random Forest

In Figure 11, we can see that how our diagonals in Figure 10 behave. We notice that our accuracy for FY decreases when going from 100 trees to 200 trees. We also notice how well the MM class does in comparison to other classes. We choose ntrees to be 300 since some accuracies decrease when going from 100 trees to 200 trees.

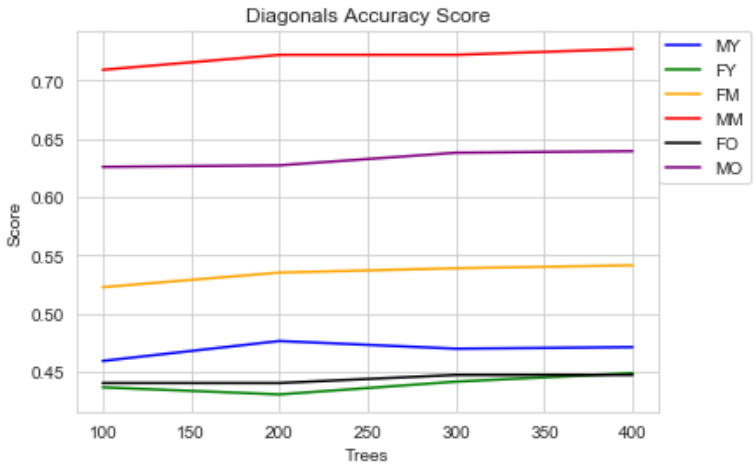

Figure 11: Diagonal Accuracy Scores

Question 7 – Feature Importance

We let our best Random Forest be of 300 trees, and then we output the most important features according to the best Random Forest. It is observed that all the features have small significance when it comes to predictive power. Feature importance is a byproduct of the random forest classifier. The feature importance is based on the impurity of leaves in a random forest classifier. The features that tend to be less important are the pixels the corners where was the face is usually located in the center of the image. 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 12 shows our feature importance in order of importance.

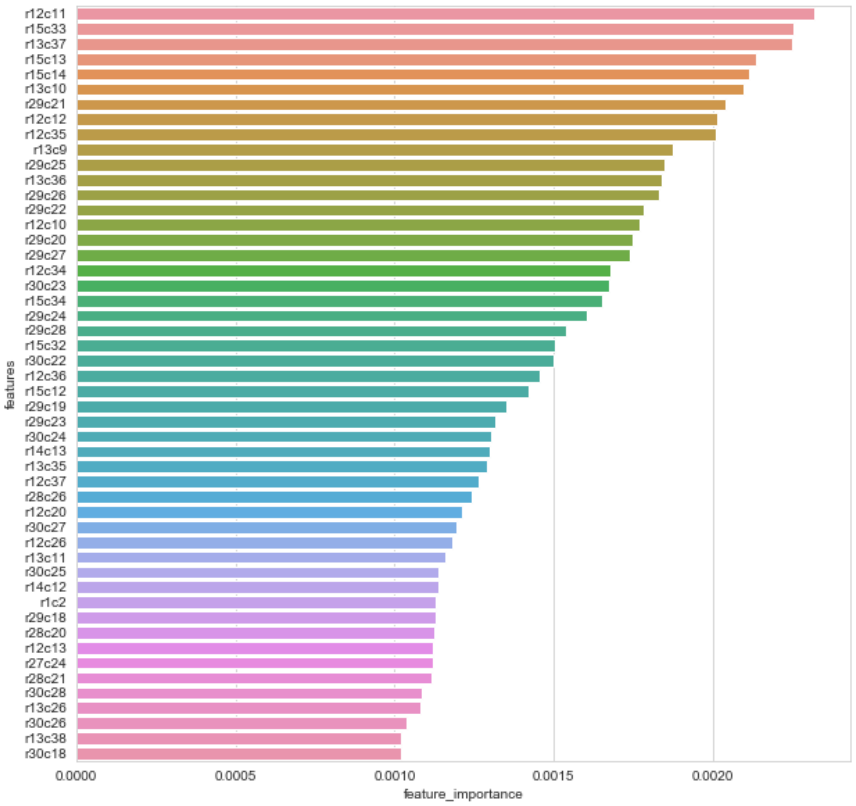

Figure 12: Feature Importance

Question 8 – Histogram and KS Test

We look at the at the histograms and discriminating power of some of the features. More specifically, we are looking at the most important feature, r12c11. Figure 13 below shows the histograms of r12c11 for each class.

From the histograms, we see that the FY and FM class have similar distributions. It's very difficult to determine anything from these histograms since there are 2,304 features and it seems that they all have either equal importance when looked individually. We will proceed to calculate discriminating power by using the Kolmogorov–Smirnov test, which is a test that looks at pairs of histograms and determines if they follow the same distribution. Formally, the hypothesis are as follows:

$$H_0: 2\ Independent\ Samples\ are\ drawn\ from\ the\ same\ continuous\ distribution$$
$$H_a: 2\ Independent\ Samples\ are\ \boldsymbol{not}\ drawn\ from\ the\ same\ continuous\ distribution$$

Equation 7: KS Test Hypothesis

We will compare features r12c11 and r34c6, the most important and least important feature from our random forest classifier. We have 6 classes, so we need to do the Kolmogorov–Smirnov test for each distinct pair. In Table 9 below, we see the discriminating power for each class with respect to the feature location. Discriminating power is calculated by doing 1 – *p-value* from the test.

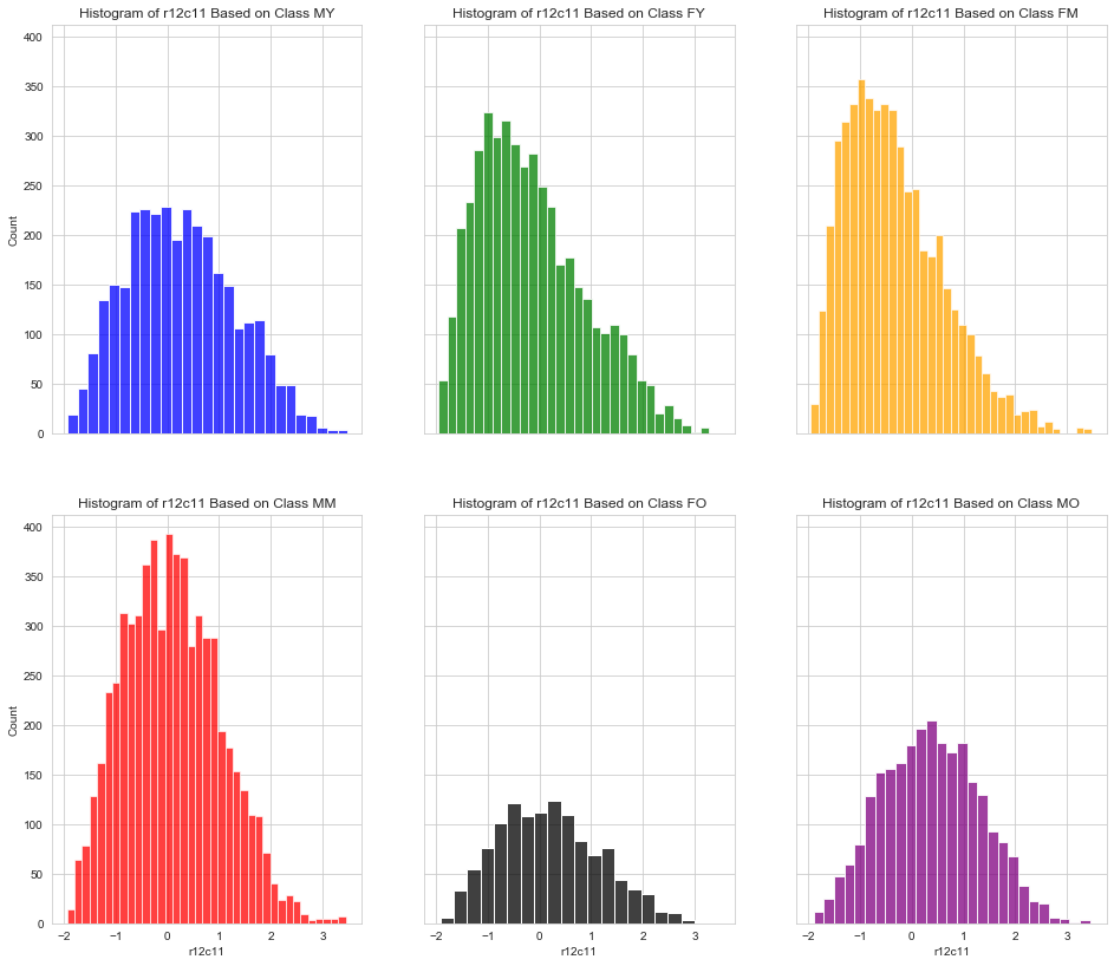

Figure 13:Histograms of Most Important Feature

	pvalue for Group FY & FM	pvalue for Group FY & FO	pvalue for Group FY & MY	pvalue for Group FY & MM	pvalue for Group FY & MO	pvalue for Group FM & FO	pvalue for Group FM & MY	pvalue for Group FM & MM	pvalue for Group FM & MO	pvalue for Group FO & MM	pvalue for Group FO & MO	pvalue for Group MY & MM	pvalue for Group MY & MO	pvalue for Group MM & MO
r12c11	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00	1.00
r34c6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.93	1.00	1.00	1.00

Table 9: KS Test between Most Important and Least Important Feature

Question 9 – 9 Clusters, Cluster 3 and Rebalancing

We grab the cluster with the lowest Gini from Question 3. We use the cases in this cluster to train our Random Forest Classifier to then be used to predict in Question 10. From our K Means Clustering, we see that within our 9 Cluster iteration, we have the lowest Gini index for the 4th cluster. This is displayed in Table 10.

cluster label=0	cluster label=1	cluster label=2	cluster label=3	cluster label=4	cluster label=5	cluster label=6	cluster label=7	cluster label=8
0.751	0.773	0.738	0.731	0.764	0.775	0.792	0.798	0.788

Table 10: Gini Index for Each Cluster in 9 Cluster K-Means

A train and test split is performed on this single cluster. Table 11 shows us the sizes of each class within this cluster. We perform smote again and random under sampling using the following. {'MM': 350, 'MO': 320, 'FM': 300, 'MY': 280, 'FY': 260, 'FO': 240} for the train set, and {'MM': 100, 'MO': 90, 'FM': 85, 'MY': 80, 'FY': 75, 'FO': 70} for the test set.

DATASET	CLFY ROW COUNT	CLFM ROW COUNT	CLFO ROW COUNT	CLMY ROW COUNT	CLMM ROW COUNT	CLMO ROW COUNT	TOTAL
TRAINSET (80%)	192	265	77	224	872	325	1,955
TESTSET (20%)	48	67	20	56	218	82	491

Table 11: Train and Test Sizes for Cluster 3 Before Rebalancing

The Best Random Forest Classifier is then applied on this resampled train and test set.

Question 10 – Best Random Forest on Cluster 3

Figure 14 demonstrates the confusion matrix for classification from Question 9. It is observed that our scores for the MY class increases, but the score for our MM class decreases from our results in Figure 10.

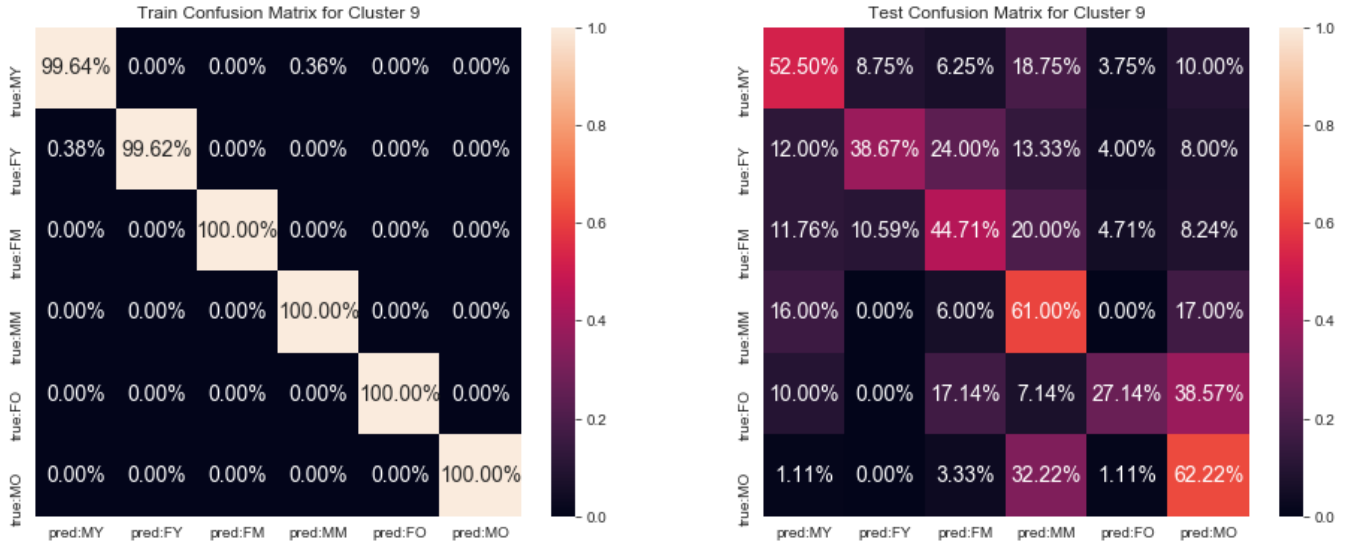

Figure 14: Test Confusion Matrix for Cluster 9

Question 11 – SVM

The last Machine Learning Technique we will use is the Supported Vector Machine. We will repeat the same train and test split on SDATA as we did for the Random Forest Classifier. The only difference is that we will look at two classes. We choose to look at classes MY and FM to see if SVM can distinguish between the two.

Approximate time taken to run SVM on Cluster 9: 8 minutes. (Multiple runs were made.)

In Table 12 we have the size of our classes after having applied smote with 30 nearest neighbors as one of the parameters.

DATASET	CLFM ROW COUNT	CLMY ROW COUNT	TOTAL
TRAINSET (80%)	4132	4132	8,264
TESTSET (20%)	1034	1034	2,068

Table 12: MY and FM Sizes after Rebalancing

After having applied SVM on this resampled 2 classification problem, we get the following train and accuracy scores. It is observed that when the problem is broken into smaller classification tasks the accuracy scores are better. This indicates that despite the difficulty of being able to distinguish between the 6 classes, we can still that there are notable differences between certain pairs of classes.

SVM	TRAIN SET SCORE	TEST SET SCORE
Linear	0.994	0.835

Table 13: Train and Test Accuracy Scores

The test confusion matrix is displayed in Figure 15. It is observed that the FM class has a higher accuracy score. This may be due to the size of FM being bigger than MY initially.

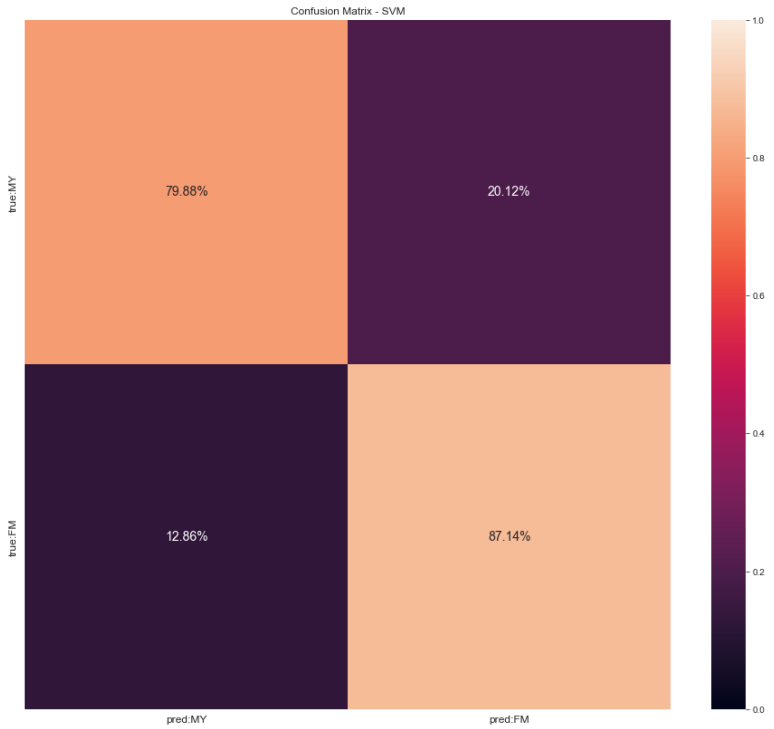

Figure 15: Test Confusion Matrix for SVM

Summary

We performed pre-processing and filtering of the data to properly setup the data. We performed data analysis to see which features had great predictive power. Next, we normalized the features to be used in the classification of the data and created a correlation matrix to ensure that the data was correlated. We applied PCA and plotted the PCA vectors to gather a better understanding on how each class behaved. We then applied K Means on SDATA to understand how well our data can be separated. We measured each cluster by its Gini indexes and impurity. The data was then split using the 80/20 approach to ensure a balanced sample was taken from each class. The Random Forest algorithm was applied in multiple iterations to find the optimal parameter of n trees. From the iterations from our Random Forest, we see that we chose the one that gave us the best results by observing the diagonals and accuracy scores. We investigated what were the most important and least important features from the Random Forest and understood that each individual pixel does not really tell much. We used the cluster wit the lowest Gini index to then train our Random Forest Classifier with the best parameters that we obtained previously. We measured how well the Random Forest did on this cluster and observed that results were poorer. Lastly, we used SVM to classify just two of our classes which had relatively higher accuracies. This allowed us to understand that, despite the difficulty of being able to classify our 6-class problem, there are distinctions between them. Further steps to possibly improve our

classification task is to use the PCA of SDATA. This could possibly get rid of noise from pixels that don't really tell much about our classes. To conclude, classifying age and gender is not an easy task and different techniques of Machine Learning should be considered.

Acknowledgements

The author gratefully acknowledges the guidance of Professor Robert Azencott.

Fonts Dataset Source

This dataset was taken from the Kaggle website. The dataset can be found at following link: https://www.kaggle.com/nipunarora8/age-gender-and-ethnicity-face-data-csv

Computer Specs

RAM: 4GB
STORAGE: Solid State Drive
PROCESSOR: Intel Core i3

Code:

```
#!/usr/bin/env python
# coding: utf-8

# In[285]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy import stats
import math
from numpy import genfromtxt
import png
from numpy import genfromtxt
from PIL import Image
from matplotlib.colors import ListedColormap
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import PCA
from scipy import stats
import scipy.linalg as la
from mpl_toolkits.mplot3d import Axes3D
from itertools import combinations
from sklearn.cluster import KMeans
from kneed import KneeLocator
from collections import Counter
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.svm import SVC
import time

## Import up sound alert dependencies
from IPython.display import Audio, display

def allDone():
  display(Audio(filename='Ventus.wav', autoplay=True))
## Insert whatever audio file you want above


# In[2]:


# Read in csv
df = pd.read_csv('age_gender.csv')

# Make a copy
df_copy = df.copy()
df.head()


# In[3]:
```

```
df.shape


# # Question 1

# ![image.png](attachment:image.png)

# In[4]:


# Split the the pixel column so that each pixel can have it's own column
df['pixels'] = df['pixels'].map(lambda x: np.array(x.split(' '), dtype=np.float32))


# In[5]:


# Filter ages above 75
df = df[df.age <= 75]
df = df.reset_index(drop=True)


# In[6]:


# Create bins for age groups
df['age_group_i'] = pd.cut(df.age, [0, 25, 50, 75], labels= [1, 2, 3])


# In[7]:


# Create classes for age and gender
conditions = [
    (df["age_group_i"] == 1) & (df['gender'] == 1),
    (df["age_group_i"] == 2) & (df['gender'] == 1),
    (df["age_group_i"] == 3) & (df['gender'] == 1),
    (df["age_group_i"] == 1) & (df['gender'] == 0),
    (df["age_group_i"] == 2) & (df['gender'] == 0),
    (df["age_group_i"] == 3) & (df['gender'] == 0)
]

choices = ['FY', 'FM', 'FO', 'MY', 'MM', 'MO']

df['class'] = np.select(conditions, choices)


# In[8]:


# Drop Columns that won't be used
df = df.drop(columns = ['ethnicity', 'img_name'])


# In[9]:


df.isna().sum()


# In[10]:
```

```
# Aesthetics
sns.set_style("whitegrid")


# In[11]:


# Color Palette as a Dictionary
color_dic = {'MY':'blue', 'MM':'red', 'MO':'purple', 'FY':'green', 'FM':'orange', 'FO':'black'}


# In[12]:


# Distribution of classes
sns.barplot(x=df['class'].value_counts().sort_index().index,                    y=df['class'].value_counts().sort_index(),
palette=color_dic.values())


# In[13]:


image = Image.fromarray((df.pixels[22853].reshape(48,48)).astype(np.uint8))

image.convert('RGB')


# In[14]:


# Create appropiate column names for each pixel
col_names = []
for i in range(0,48):
    for j in range(0,48):
        col_names.append('r'+str(i)+'c'+str(j))

df[col_names] = pd.DataFrame(df.pixels.tolist(), index= df.index, columns = col_names)


# In[15]:


df.head()


# In[181]:


# Only want pixels and class
df = df.loc[:,'class':]


# In[182]:


# Scaling
scaler = StandardScaler()
scaler.fit(df.iloc[:,1:])
sX = scaler.transform(df.iloc[:,1:])
sX = pd.DataFrame(sX, columns = col_names)
```

In[19]:

```
# PCA
start = time.time()

pca = PCA(.95)
pca.fit(sX)
print(pca.n_components_)
pcaX = pca.transform(sX)
pcaX = pd.DataFrame(pcaX)

end = time.time()
print(end - start)
```

In[20]:

```
# Correlation Matrix
start = time.time()
corr_sX = sX.corr()

end = time.time()
print(end - start)
```

In[21]:

```
# Getting eigen values and eigen vectors of correlation Matrix
eigen_vals, eigen_vecs = la.eig(corr_sX)
eigen_vals = eigen_vals.real
```

In[22]:

```
# Plotting eigen values for each eigen number
fig, axs = plt.subplots(1, 2, figsize = (15,5))

sns.lineplot(x = [number for number in range(1,2305)], y = eigen_vals, ax=axs[0])

axs[0].set_ylabel('Eigen Value (Lr)')
axs[0].set_xlabel('Eigen Number (r)')
axs[0].title.set_text('Eigen Values vs Eigen Number (Lr v r)')

# Zoomed in version of above
sns.lineplot(x = [number for number in range(1,71)], y = eigen_vals[0:70], ax=axs[1])

axs[1].set_ylabel('Eigen Value (Lr)')
axs[1].set_xlabel('Eigen Number (r)')
axs[1].title.set_text('Eigen Values vs Eigen Number (Lr v r)')
```

In[23]:

```
# Proportion of Variance Explained after each r
W = eigen_vecs.transpose()
```

```
PVE_r = np.cumsum(eigen_vals / len(W))


# In[24]:


# Plotting PVE
fig, axs = plt.subplots(1, 1, figsize = (10,5))

sns.lineplot(x = [number for number in range(1,2305)], y = PVE_r)

plt.ylabel('Cumulative PVE(r)')
plt.xlabel('Principal Component r')
plt.title('Cumulative Scree Plot')


# In[25]:


# Getting the first 3 eigen numbers had their eigenvalues, and PVE
pca_coor = pd.DataFrame({'r':[number for number in range(1,4)],
                         'eigen values' : eigen_vals[:3],
                         'percentage of variance explained': eigen_vals[:3] / len(W)})

pca_coor


# In[26]:


# Creating DataFrame to plot the first 3 PCA vectors with respective class
threeD = pcaX.iloc[:,0:3].join(df.loc[:,'class'])


# In[27]:


threeD.head()


# In[28]:


# Getting classes names in a list
classes = threeD['class'].unique()


# In[165]:


# Getting classes names in a list
classes = threeD['class'].unique()

# Creating combinations for each class
iteration_obj = combinations(classes, 2)

pairs = [' '.join(i) for i in iteration_obj]
pairs = [pair.split() for pair in pairs]
pairs


# In[30]:
```

```
# axes instance
fig = plt.figure(figsize = (20,14))
ax = []

for i in range(0, len(pairs)):
    ax = fig.add_subplot(3, 5, i+1, projection='3d')
    # plot
    ax.scatter(threeD[threeD['class'] == pairs[i][0]].iloc[:,0], threeD[threeD['class'] == pairs[i][0]].iloc[:,1],
threeD[threeD['class'] == pairs[i][0]].iloc[:,2], alpha = 1, s = .05, color = color_dic[pairs[i][0]])
    ax.scatter(threeD[threeD['class'] == pairs[i][1]].iloc[:,0], threeD[threeD['class'] == pairs[i][1]].iloc[:,1],
threeD[threeD['class'] == pairs[i][1]].iloc[:,2], alpha = 1, s = .05, color = color_dic[pairs[i][1]])

    ax.set_xlabel('Prinicipal Component 1')
    ax.set_ylabel('Prinicipal Component 2')
    ax.set_zlabel('Prinicipal Component 3')
    ax.title.set_text('%s vs %s' % (pairs[i][0], pairs[i][1]))

    # legend
    ax.legend(labels=[pairs[i][0],pairs[i][1]])


# # Question 2

# ![image.png](attachment:image.png)

# In[31]:


# Running Kmeans
k = list(range(1,11))
sum_disp_k = []
centers_k = []
labels_k = []

start = time.time()

for i in k:
    kmeans = KMeans(n_clusters=i, n_init=50, random_state=0)
    kmeans.fit(sX)
    sum_disp_k.append(kmeans.inertia_)
    centers_k.append(kmeans.cluster_centers_)
    labels_k.append(kmeans.labels_)

end = time.time()
print(end - start)


# In[95]:


kl = KneeLocator(k, sum_disp_k, curve = 'convex', direction = 'decreasing')
kl.elbow


# In[96]:


sum_disp_k
```

In[97]:

```
fig, axs = plt.subplots(1, 1, figsize = (10,5))

sns.lineplot(x=k, y=sum_disp_k)

plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.title('Elbow Plot')
plt.axvline(x=kl.elbow, color = 'Black')
```

In[98]:

```
perf_k = []
for i in range(0,len(sum_disp_k)):
    perf_k.append(1 - (sum_disp_k[i]/sum_disp_k[0]))

perf_k
```

In[293]:

```
fig, axs = plt.subplots(1, 1, figsize = (10,5))

sns.lineplot(x=k, y=perf_k)

plt.xlabel('Number of Clusters')
plt.ylabel('Perf(k)')
plt.title('Knee Plot')
plt.axvline(x=kl.elbow, color = 'Black')
```

In[100]:

```
clusterX = sX.copy()
```

In[101]:

```
for i in range(1,len(labels_k)+1):
    clusterX['clusterk'+str(i)]=labels_k[i-1]
```

In[116]:

```
print(clusterX.shape)
clusterX.head()
```

In[103]:

```
clusterX = clusterX.join(df.loc[:,'class'])
```

In[104]:

```
FJS = []

for i in range(1, len(labels_k)+1):
    FJS_temp = []
    for j in range(0,i):
        FJS_temp.append(clusterX[clusterX['clusterk'+str(i)] == j]['class'].value_counts().sort_index() / len(clusterX[clusterX['clusterk'+str(i)] == j]))
    print('This is fj for clusterk' + str(i), FJS_temp)
    FJS.append(FJS_temp)
```

In[105]:

```
GINI = []

for i in range(0,len(FJS)):
    gini_temp = []
    for j in range(0, i+1):
        gini_sum = None
        f1s = []
        for a in range(0,6):
            f1s.append(FJS[i][j][a] * (1-FJS[i][j][a]))
        gini_sum = sum(f1s)
        gini_temp.append(gini_sum)
    print('This is gini for clusterk' + str(i), gini_temp)
    GINI.append(gini_temp)
```

In[106]:

```
IMP = []

for i in range(0,len(GINI)):
    IMP.append(sum(GINI[i]))
```

In[107]:

```
[j-i for i, j in zip(IMP[:-1], IMP[1:])]
```

In[108]:

```
fig, axs = plt.subplots(1, 1, figsize = (10,5))

sns.lineplot(x=k, y=IMP)

plt.xlabel('Number of Clusters')
plt.ylabel('IMP(k)')
plt.title('Impurity by Cluster')
```

Question 3

![image.png](attachment:image.png)

In[294]:

```
centers_k[3]
```

In[119]:

```
#center
print(centers_k[3])

#size
print(pd.Series(labels_k[3]).value_counts())

#ginis
print(GINI[3])

#frequencies
print(FJS[3])


# Calculate Disp(j) for each Cluster j #align SDATA with cluster assignment from K*
SDATA_CLUS1 = clusterX[clusterX['clusterk4'] == 0]
SDATA_CLUS2 = clusterX[clusterX['clusterk4'] == 1]
SDATA_CLUS3 = clusterX[clusterX['clusterk4'] == 2]
SDATA_CLUS4 = clusterX[clusterX['clusterk4'] == 3]

# only want the data for each observation in each cluster, so we drop the "cluster" column
CL1_SDATA = SDATA_CLUS1.iloc[:,:2304]
CL2_SDATA = SDATA_CLUS2.iloc[:,:2304]
CL3_SDATA = SDATA_CLUS3.iloc[:,:2304]
CL4_SDATA = SDATA_CLUS4.iloc[:,:2304]


# get each cluster center
CLUS1_cntr = centers_k[3][0]
CLUS2_cntr = centers_k[3][1]
CLUS3_cntr = centers_k[3][2]
CLUS4_cntr = centers_k[3][3]

#calculte the difference between the vector of each observation in cluster j and the center of cluster j
norms1 = []
for i in range(len(CL1_SDATA)):
    norms1.append(np.linalg.norm(CL1_SDATA.iloc[i] - CLUS1_cntr))

norms2 = []
for i in range(len(CL2_SDATA)):
    norms2.append(np.linalg.norm(CL2_SDATA.iloc[i] - CLUS2_cntr))

norms3 = []
for i in range(len(CL3_SDATA)):
    norms3.append(np.linalg.norm(CL3_SDATA.iloc[i] - CLUS3_cntr))

norms4 = []
for i in range(len(CL4_SDATA)):
    norms4.append(np.linalg.norm(CL4_SDATA.iloc[i] - CLUS4_cntr))

# Square each norm for each cluster j
sq_norms1 = np.square(np.array(norms1)) #can only square an array, not a list
sq_norms2 = np.square(np.array(norms2))
```

```
sq_norms3 = np.square(np.array(norms3))
sq_norms4 = np.square(np.array(norms4))

# sum up all of the above values for each cluster j
DIS_CL1 = sum(sq_norms1)
DIS_CL2 = sum(sq_norms2)
DIS_CL3 = sum(sq_norms3)
DIS_CL4 = sum(sq_norms4)


# In[295]:


print(DIS_CL1, DIS_CL2,DIS_CL3,DIS_CL4)


# In[122]:


W = pd.DataFrame(W)


# In[184]:


cluster4 = pcaX.iloc[:,0:3].join(clusterX['clusterk4'])


# In[185]:


cluster4.head()


# In[186]:


# Getting classes names in a list
clusters = ['0','1','2','3']

# Creating combinations for each class
iteration_obj = combinations(clusters, 2)

pairs_num = list(iteration_obj)


# In[299]:




# In[296]:


cluster_color = {0:'cyan', 1:'deeppink', 2:'yellowgreen', 3:'tomato'}


# In[303]:


# axes instance
```

```
# axes instance
fig = plt.figure(figsize = (20,14))
ax = []

for i in range(0, len(pairs_num)):
    ax = fig.add_subplot(2, 3, i+1, projection='3d')
    # plot
    ax.scatter(cluster4[cluster4['clusterk4'] == int(pairs_num[i][0])].iloc[:,0], cluster4[cluster4['clusterk4'] == int(pairs_num[i][0])].iloc[:,1], cluster4[cluster4['clusterk4'] == int(pairs_num[i][0])].iloc[:,2], alpha = 1, s = .05, color = cluster_color[int(pairs_num[i][0])])
    ax.scatter(cluster4[cluster4['clusterk4'] == int(pairs_num[i][1])].iloc[:,0], cluster4[cluster4['clusterk4'] == int(pairs_num[i][1])].iloc[:,1], cluster4[cluster4['clusterk4'] == int(pairs_num[i][1])].iloc[:,2], alpha = 1, s = .05, color = cluster_color[int(pairs_num[i][1])])

    ax.set_xlabel('Prinicipal Component 1')
    ax.set_ylabel('Prinicipal Component 2')
    ax.set_zlabel('Prinicipal Component 3')
    ax.title.set_text('Cluster %s vs Cluster %s' % (pairs_num[i][0], pairs_num[i][1]))

    # legend
    ax.legend(labels=[pairs_num[i][0],pairs_num[i][1]])


# # Question 4

# ![image.png](attachment:image.png)

# In[189]:


sdf = sX.join(df.loc[:,'class'])


# In[190]:


# train and test split
sdfFY_train, sdfFY_test, clFY_train, clFY_test = train_test_split(sdf[sdf['class'] == 'FY'].iloc[:,0:2304], sdf[sdf['class'] == 'FY'].iloc[:,2304], test_size=0.2, random_state=0)
sdfFM_train, sdfFM_test, clFM_train, clFM_test = train_test_split(sdf[sdf['class'] == 'FM'].iloc[:,0:2304], sdf[sdf['class'] == 'FM'].iloc[:,2304], test_size=0.2, random_state=0)
sdfFO_train, sdfFO_test, clFO_train, clFO_test = train_test_split(sdf[sdf['class'] == 'FO'].iloc[:,0:2304], sdf[sdf['class'] == 'FO'].iloc[:,2304], test_size=0.2, random_state=0)
sdfMY_train, sdfMY_test, clMY_train, clMY_test = train_test_split(sdf[sdf['class'] == 'MY'].iloc[:,0:2304], sdf[sdf['class'] == 'MY'].iloc[:,2304], test_size=0.2, random_state=0)
sdfMM_train, sdfMM_test, clMM_train, clMM_test = train_test_split(sdf[sdf['class'] == 'MM'].iloc[:,0:2304], sdf[sdf['class'] == 'MM'].iloc[:,2304], test_size=0.2, random_state=0)
sdfMO_train, sdfMO_test, clMO_train, clMO_test = train_test_split(sdf[sdf['class'] == 'MO'].iloc[:,0:2304], sdf[sdf['class'] == 'MO'].iloc[:,2304], test_size=0.2, random_state=0)


# In[191]:


X_train = pd.concat([sdfFY_train, sdfFM_train, sdfFO_train, sdfMY_train, sdfMM_train, sdfMO_train])
X_test = pd.concat([sdfFY_test, sdfFM_test, sdfFO_test, sdfMY_test, sdfMM_test, sdfMO_test])
y_train = pd.concat([clFY_train, clFM_train, clFO_train, clMY_train, clMM_train, clMO_train])
y_test = pd.concat([clFY_test, clFM_test, clFO_test, clMY_test, clMM_test, clMO_test])


# In[192]:
```

```
print(len(sdfFO_train))
print(len(sdfMO_train))
print(len(sdfMY_train))
print(len(sdfFY_train))
print(len(sdfFM_train))
print(len(sdfMM_train))
```

In[305]:

```
sns.barplot(x=y_train.value_counts().sort_index().index, y=y_train.value_counts().sort_index(), palette=color_dic.values())
```

In[194]:

```
upper = {'FO':2800, 'MO':2900, 'MY': 3000}
```

In[195]:

```
sm = SMOTE(random_state=0, sampling_strategy=upper, k_neighbors=30)
X_res_train, y_res_train = sm.fit_resample(X_train, y_train)
print('Resampled dataset shape %s' % Counter(y_res_train))
```

In[196]:

```
under = {'MM': 3300, 'FM': 3200, 'FY': 3100}
```

In[197]:

```
rus = RandomUnderSampler(random_state=0, sampling_strategy=under)
X_res_train, y_res_train = rus.fit_resample(X_res_train, y_res_train)
print('Resampled dataset shape %s' % Counter(y_res_train))
```

In[198]:

```
sns.barplot(x=y_res_train.value_counts().sort_index().index, y=y_res_train.value_counts().sort_index())
```

In[199]:

```
print(len(sdfFO_test))
print(len(sdfMO_test))
print(len(sdfMY_test))
print(len(sdfFY_test))
print(len(sdfFM_test))
print(len(sdfMM_test))
```

In[309]:

```
upper = {'FO':720, 'MO':740, 'MY':760}
under = {'FY':780, 'FM':800, 'MM':820}

sm = SMOTE(random_state=0, sampling_strategy=upper, k_neighbors = 30)
X_res_test, y_res_test = sm.fit_resample(X_test, y_test)
print('Resampled dataset shape %s' % Counter(y_res_test))

rus = RandomUnderSampler(random_state=0, sampling_strategy=under)
X_res_test, y_res_test = rus.fit_resample(X_res_test, y_res_test)
print('Resampled dataset shape %s' % Counter(y_res_test))


# In[307]:


# axes instance
fig, ax = plt.subplots(1,2, figsize = (12,4), sharey = True)

sns.barplot(x=y_train.value_counts().sort_index().index, y=y_train.value_counts().sort_index(), ax = ax[0], palette = color_dic.values())
sns.barplot(x=y_res_train.value_counts().sort_index().index, y=y_res_train.value_counts().sort_index(), ax = ax[1], palette = color_dic.values())


# In[310]:


# axes instance
fig, ax = plt.subplots(1,2, figsize = (12,4), sharey=True)

sns.barplot(x=y_test.value_counts().sort_index().index, y=y_test.value_counts().sort_index(), ax = ax[0], palette = color_dic.values())
sns.barplot(x=y_res_test.value_counts().sort_index().index, y=y_res_test.value_counts().sort_index(), ax = ax[1], palette = color_dic.values())


# # Question 5

# ![image.png](attachment:image.png)

# In[217]:


ntrees = [100,200,300,400]
f = len(X_train.columns)
y_pred_train = []
y_pred_test = []
oob_score = []

start = time.time()

for tree in ntrees:
    clf = RandomForestClassifier(n_estimators=tree, max_features= round(math.sqrt(f)), random_state=0, oob_score=True)
    clf.fit(X_res_train, y_res_train)
    y_pred_train.append(clf.predict(X_res_train))
    y_pred_test.append(clf.predict(X_res_test))
    oob_score.append(clf.oob_score_)

end = time.time()
print(end - start)
```

In[218]:

```
y_pred_train
```

In[219]:

```
y_pred_test
```

In[220]:

```
accuracies_train = []
accuracies_test = []
for i in range(0, len(ntrees)):
    accuracies_train.append(accuracy_score(y_true=y_res_train, y_pred = y_pred_train[i]))
    accuracies_test.append(accuracy_score(y_true=y_res_test, y_pred = y_pred_test[i]))
```

In[221]:

```
print(accuracies_test)
print(accuracies_train)
print(oob_score)
```

In[317]:

```
train_test_oob = {'Train':'blue','Test':'orange', 'OOB':'green'}
```

In[318]:

```
fig, ax = plt.subplots(1,2, figsize = (12,4))
sns.lineplot(x = ntrees, y = accuracies_train, ax=ax[0], color = 'blue')
sns.lineplot(x = ntrees, y = accuracies_test, ax=ax[0], color = 'orange')
sns.lineplot(x = ntrees, y = oob_score, ax=ax[0], color = 'green')
ax[0].set_ylabel('Score')
ax[0].set_xlabel('Trees')
ax[0].title.set_text('Train, Test, OOB')
ax[0].legend(labels=['Train','Test','OOB'])

sns.lineplot(x = ntrees, y = accuracies_test, ax=ax[1], color = 'orange')
sns.lineplot(x = ntrees, y = oob_score, ax=ax[1], color = 'green')
ax[1].set_ylabel('Score')
ax[1].set_xlabel('Trees')
ax[1].title.set_text('Test, OOB')
ax[1].legend(labels=['Test','OOB'])
```

In[311]:

```
sns.lineplot(x = ntrees, y = accuracies_test)
sns.lineplot(x = ntrees, y = oob_score)
```

In[223]:

```
sns.lineplot(x = ntrees, y = accuracies_test)
```

In[224]:

```
classes
classes_index =[]
classes_columns = []
for i in range(0,len(classes)):
    classes_index.append('true:%s'%classes[i])
    classes_columns.append('pred:%s'%classes[i])
```

In[225]:

```
fig = plt.figure(figsize = (16,14))
axes = []

for i in range(0,len(ntrees)):
    axes.append(fig.add_subplot(2, 2, i+1))

    cmtx_a_test = pd.DataFrame(
    confusion_matrix(y_true=y_res_test,y_pred = y_pred_test[i], labels = classes, normalize = 'true'),
    index=classes_index,
    columns=classes_columns)

    sns.heatmap(cmtx_a_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax=axes[i])
    axes[i].title.set_text('Test Confusion Matrix for '+str(ntrees[i])+' Trees.')
```

Question 6

![image.png](attachment:image.png)

In[227]:

```
diag = []
for i in range(0,len(ntrees)):
    cmtx_a_test = pd.DataFrame(
    confusion_matrix(y_true=y_res_test,y_pred = y_pred_test[i], labels = classes, normalize = 'true'),
    index=classes_index,
    columns=classes_columns)
    diag.append(np.diagonal(cmtx_a_test))

diag
```

In[228]:

```
diag = np.transpose(diag)
```

In[330]:

diag

In[333]:

pd.Series(classes).sort_values()

In[329]:

```
for i in range(0, len(diag)):
    sns.lineplot(x=ntrees,y=diag[i], color=color_dic[classes[i]])

plt.ylabel('Score')
plt.xlabel('Trees')
plt.title('Diagonals Accuracy Score')
plt.legend(labels = classes, loc=(1,.60))
```

Question 7

![image.png](attachment:image.png)

In[230]:

```
y_pred_train_best = None
y_pred_test_best = None
y_pred_train_oob = None

bestRF = RandomForestClassifier(n_estimators=300, max_features= round(math.sqrt(f)), random_state=0, oob_score=True)
bestRF.fit(X_res_train, y_res_train)
y_pred_train_best = bestRF.predict(X_res_train)
y_pred_test_best = bestRF.predict(X_res_test)
y_pred_train_oob = bestRF.oob_score_
```

In[231]:

```
feature_impdf = pd.DataFrame({'feature_importance': bestRF.feature_importances_, 'features':sX.columns})
feature_impdf = feature_impdf.set_index('features')
```

In[233]:

feature_impdf['feature_importance'].sort_values(ascending = False)[0:50]

In[234]:

bestRF.feature_importances_

In[235]:

```
fig, axs = plt.subplots(1, 1, figsize = (10,10))

sns.barplot(x = feature_impdf['feature_importance'].sort_values(ascending = False)[0:50],
        y = feature_impdf['feature_importance'].sort_values(ascending = False)[0:50].index)
#plt.ylim(0,.0025)


# # Question 8

# ![image.png](attachment:image.png)

# In[237]:


top = feature_impdf['feature_importance'].sort_values(ascending = False).index[0]
bottom = feature_impdf['feature_importance'].sort_values().index[0]

fig, axes = plt.subplots(2, 3, figsize = (16,14), sharex=True, sharey=True)

for i in range(0,len(classes)):
    sns.histplot(sdf[sdf['class'] == classes[i]][top], color = color_dic[classes[i]], ax=axes[i//3,i%3])
    axes[i//3,i%3].title.set_text('Histogram of %s Based on Class %s' % (top, classes[i]))


# In[238]:


select_feat = [top, bottom,'class']
select_df = sdf[select_feat]

# KS-test to Compare Histograms Between Pairs of Classes
 #   Null hypotheses: Two group histograms are equal
 #   Alternative hypotheses: Two group histograms are different (two-tailed)

KS_test_results_1_2 = {}
KS_test_results_1_3 = {}
KS_test_results_1_4 = {}
KS_test_results_1_5 = {}
KS_test_results_1_6 = {}
KS_test_results_2_3 = {}
KS_test_results_2_4 = {}
KS_test_results_2_5 = {}
KS_test_results_2_6 = {}
KS_test_results_3_4 = {}
KS_test_results_3_5 = {}
KS_test_results_3_6 = {}
KS_test_results_4_5 = {}
KS_test_results_4_6 = {}
KS_test_results_5_6 = {}

# loop over column_list and execute code explained above
for column in select_df.columns[:2]:
    groupFY = select_df[select_df['class'] == 'FY'][column]
    groupFM = select_df[select_df['class'] == 'FM'][column]
    groupFO = select_df[select_df['class'] == 'FO'][column]
    groupMY = select_df[select_df['class'] == 'MY'][column]
    groupMM = select_df[select_df['class'] == 'MM'][column]
    groupMO = select_df[select_df['class'] == 'MO'][column]
    # add the output to the dictionary
    KS_test_results_1_2[column] = stats.ks_2samp(groupFY,groupFM)
    KS_test_results_1_3[column] = stats.ks_2samp(groupFY,groupFO)
```

```
    KS_test_results_1_4[column] = stats.ks_2samp(groupFY,groupMY)
    KS_test_results_1_5[column] = stats.ks_2samp(groupFY,groupMM)
    KS_test_results_1_6[column] = stats.ks_2samp(groupFY,groupMO)
    KS_test_results_2_3[column] = stats.ks_2samp(groupFM,groupFO)
    KS_test_results_2_4[column] = stats.ks_2samp(groupFM,groupMY)
    KS_test_results_2_5[column] = stats.ks_2samp(groupFM,groupMM)
    KS_test_results_2_6[column] = stats.ks_2samp(groupFM,groupMO)
    KS_test_results_3_4[column] = stats.ks_2samp(groupFO,groupMY)
    KS_test_results_3_5[column] = stats.ks_2samp(groupFO,groupMM)
    KS_test_results_3_6[column] = stats.ks_2samp(groupFO,groupMO)
    KS_test_results_4_5[column] = stats.ks_2samp(groupMY,groupMM)
    KS_test_results_4_6[column] = stats.ks_2samp(groupMY,groupMO)
    KS_test_results_5_6[column] = stats.ks_2samp(groupMM,groupMO)


    # KS Test
KS_results_df_1_2 = pd.DataFrame.from_dict(KS_test_results_1_2,orient='Index')
KS_results_df_1_2.columns = ['KS statistic for Group FY & FM','pvalue for Group FY & FM']
KS_results_df_1_3 = pd.DataFrame.from_dict(KS_test_results_1_3,orient='Index')
KS_results_df_1_3.columns = ['KS statistic for Group FY & FO','pvalue for Group FY & FO']
KS_results_df_1_4 = pd.DataFrame.from_dict(KS_test_results_1_4,orient='Index')
KS_results_df_1_4.columns = ['KS statistic for Group FY & MY','pvalue for Group FY & MY']
KS_results_df_1_5 = pd.DataFrame.from_dict(KS_test_results_1_5,orient='Index')
KS_results_df_1_5.columns = ['KS statistic for Group FY & MM','pvalue for Group FY & MM']
KS_results_df_1_6 = pd.DataFrame.from_dict(KS_test_results_1_6,orient='Index')
KS_results_df_1_6.columns = ['KS statistic for Group FY & MO','pvalue for Group FY & MO']
KS_results_df_2_3 = pd.DataFrame.from_dict(KS_test_results_2_3,orient='Index')
KS_results_df_2_3.columns = ['KS statistic for Group FM & FO','pvalue for Group FM & FO']
KS_results_df_2_4 = pd.DataFrame.from_dict(KS_test_results_2_4,orient='Index')
KS_results_df_2_4.columns = ['KS statistic for Group FM & MY','pvalue for Group FM & MY']
KS_results_df_2_5 = pd.DataFrame.from_dict(KS_test_results_2_5,orient='Index')
KS_results_df_2_5.columns = ['KS statistic for Group FM & MM','pvalue for Group FM & MM']
KS_results_df_2_6 = pd.DataFrame.from_dict(KS_test_results_2_6,orient='Index')
KS_results_df_2_6.columns = ['KS statistic for Group FM & MO','pvalue for Group FM & MO']
KS_results_df_3_4 = pd.DataFrame.from_dict(KS_test_results_3_4,orient='Index')
KS_results_df_3_4.columns = ['KS statistic for Group FO & MY','pvalue for Group FO & MY']
KS_results_df_3_5 = pd.DataFrame.from_dict(KS_test_results_3_5,orient='Index')
KS_results_df_3_5.columns = ['KS statistic for Group FO & MM','pvalue for Group FO & MM']
KS_results_df_3_6 = pd.DataFrame.from_dict(KS_test_results_3_6,orient='Index')
KS_results_df_3_6.columns = ['KS statistic for Group FO & MO','pvalue for Group FO & MO']
KS_results_df_4_5 = pd.DataFrame.from_dict(KS_test_results_4_5,orient='Index')
KS_results_df_4_5.columns = ['KS statistic for Group MY & MM','pvalue for Group MY & MM']
KS_results_df_4_6 = pd.DataFrame.from_dict(KS_test_results_4_6,orient='Index')
KS_results_df_4_6.columns = ['KS statistic for Group MY & MO','pvalue for Group MY & MO']
KS_results_df_5_6 = pd.DataFrame.from_dict(KS_test_results_5_6,orient='Index')
KS_results_df_5_6.columns = ['KS statistic for Group MM & MO','pvalue for Group MM & MO']


KS_test_Results_df = pd.concat([KS_results_df_1_2,
                                KS_results_df_1_3,
                                KS_results_df_1_4,
                                KS_results_df_1_5,
                                KS_results_df_1_6,
                                KS_results_df_2_3,
                                KS_results_df_2_4,
                                KS_results_df_2_5,
                                KS_results_df_2_6,
                                KS_results_df_3_4,
                                KS_results_df_3_5,
                                KS_results_df_3_6,
                                KS_results_df_4_5,
                                KS_results_df_4_6,
```

```
                        KS_results_df_5_6], axis=1)
KS_test_Results_df

# Discrimnation Power based of KS Results
pow12 = 1 - KS_test_Results_df.iloc[:,1]
pow13 = 1 - KS_test_Results_df.iloc[:,3]
pow14 = 1 - KS_test_Results_df.iloc[:,5]
pow15 = 1 - KS_test_Results_df.iloc[:,7]
pow16 = 1 - KS_test_Results_df.iloc[:,9]
pow23 = 1 - KS_test_Results_df.iloc[:,11]
pow24 = 1 - KS_test_Results_df.iloc[:,13]
pow25 = 1 - KS_test_Results_df.iloc[:,15]
pow26 = 1 - KS_test_Results_df.iloc[:,17]
pow34 = 1 - KS_test_Results_df.iloc[:,19]
pow35 = 1 - KS_test_Results_df.iloc[:,21]
pow36 = 1 - KS_test_Results_df.iloc[:,23]
pow45 = 1 - KS_test_Results_df.iloc[:,25]
pow46 = 1 - KS_test_Results_df.iloc[:,27]
pow56 = 1 - KS_test_Results_df.iloc[:,29]

Discrim_Power = pd.concat([pow12,
                        pow13,
                        pow14,
                        pow15,
                        pow16,
                        pow23,
                        pow24,
                        pow25,
                        pow26,
                        pow35,
                        pow36,
                        pow45,
                        pow46,
                        pow56], axis=1)
# Discrim_Power.columns = ['KS - Discriminating Power of Group FY & FM', 'KS - Discriminating Power of Group FY &
FO', 'KS - Discriminating Power of Group FM & FO']


# In[239]:


KS_test_Results_df


# In[240]:


Discrim_Power


# # Question 9

# ![image.png](attachment:image.png)

# In[253]:


for i in range(0,len(GINI)):
    print('when using k=' + str(i+1), min(GINI[i]))


# In[242]:
```

```
GINI[8]
```

```
# In[243]:
```

```
for i in range(0,len(GINI[8])):
    print('cluster label='+str(i), GINI[8][i])
```

```
# In[257]:
```

```
cluster9 = sX.copy()
cluster9['cluster9'] = labels_k[8]
cluster9 = cluster9.join(df.loc[:,'class'])

cluster9.head()
```

```
# In[259]:
```

```
clu9 = cluster9[cluster9['cluster9'] == 3]
```

```
# In[260]:
```

```
clu9FY_train, clu9FY_test, clFY_train, clFY_test = train_test_split(clu9[clu9['class'] == 'FY'].iloc[:,0:2304], clu9[clu9['class'] == 'FY'].loc[:,'class'], test_size=0.2, random_state=0)
clu9FM_train, clu9FM_test, clFM_train, clFM_test = train_test_split(clu9[clu9['class'] == 'FM'].iloc[:,0:2304], clu9[clu9['class'] == 'FM'].loc[:,'class'], test_size=0.2, random_state=0)
clu9FO_train, clu9FO_test, clFO_train, clFO_test = train_test_split(clu9[clu9['class'] == 'FO'].iloc[:,0:2304], clu9[clu9['class'] == 'FO'].loc[:,'class'], test_size=0.2, random_state=0)
clu9MY_train, clu9MY_test, clMY_train, clMY_test = train_test_split(clu9[clu9['class'] == 'MY'].iloc[:,0:2304], clu9[clu9['class'] == 'MY'].loc[:,'class'], test_size=0.2, random_state=0)
clu9MM_train, clu9MM_test, clMM_train, clMM_test = train_test_split(clu9[clu9['class'] == 'MM'].iloc[:,0:2304], clu9[clu9['class'] == 'MM'].loc[:,'class'], test_size=0.2, random_state=0)
clu9MO_train, clu9MO_test, clMO_train, clMO_test = train_test_split(clu9[clu9['class'] == 'MO'].iloc[:,0:2304], clu9[clu9['class'] == 'MO'].loc[:,'class'], test_size=0.2, random_state=0)
```

```
X_train_clu9 = pd.concat([clu9FY_train, clu9FM_train, clu9FO_train, clu9MY_train, clu9MM_train, clu9MO_train])
X_test_clu9 = pd.concat([clu9FY_test, clu9FM_test, clu9FO_test, clu9MY_test, clu9MM_test, clu9MO_test])
y_train_clu9 = pd.concat([clFY_train, clFM_train, clFO_train, clMY_train, clMM_train, clMO_train])
y_test_clu9 = pd.concat([clFY_test, clFM_test, clFO_test, clMY_test, clMM_test, clMO_test])
```

```
# In[261]:
```

```
print(len(clu9FO_train))
print(len(clu9FY_train))
print(len(clu9MY_train))
print(len(clu9FM_train))
print(len(clu9MO_train))
print(len(clu9MM_train))
```

In[262]:

```
upper = {'FO':240, 'FY':260, 'MY':280, 'FM':300}
under = {'MO':320, 'MM':350}

sm = SMOTE(random_state=0, sampling_strategy=upper, k_neighbors=30)
X_res_train_clu9, y_res_train_clu9 = sm.fit_resample(X_train_clu9, y_train_clu9)
print('Resampled dataset shape %s' % Counter(y_res_train_clu9))

rus = RandomUnderSampler(random_state=0, sampling_strategy=under)
X_res_train_clu9, y_res_train_clu9 = rus.fit_resample(X_res_train_clu9, y_res_train_clu9)
print('Resampled dataset shape %s' % Counter(y_res_train_clu9))
```

In[263]:

```
print(len(clu9FO_test))
print(len(clu9FY_test))
print(len(clu9MY_test))
print(len(clu9FM_test))
print(len(clu9MO_test))
print(len(clu9MM_test))
```

In[264]:

```
upper = {'FO':70, 'FY':75, 'MY':80, 'FM':85, 'MO':90, }
under = {'MM':100}

sm = SMOTE(random_state=0, sampling_strategy=upper)
X_res_test_clu9, y_res_test_clu9 = sm.fit_resample(X_test_clu9, y_test_clu9)
print('Resampled dataset shape %s' % Counter(y_res_test_clu9))

rus = RandomUnderSampler(random_state=0, sampling_strategy=under)
X_res_test_clu9, y_res_test_clu9 = rus.fit_resample(X_res_test_clu9, y_res_test_clu9)
print('Resampled dataset shape %s' % Counter(y_res_test_clu9))
```

In[274]:

```
y_pred_train_clu9 = None
y_pred_test_clu9 = None

bestRF = RandomForestClassifier(n_estimators=300, max_features= 'sqrt', random_state=0, oob_score=True)
bestRF.fit(X_res_train_clu9, y_res_train_clu9)
y_pred_train_clu9 = bestRF.predict(X_res_train_clu9)
y_pred_test_clu9 = bestRF.predict(X_res_test_clu9)
```

In[275]:

```
bestRF.oob_score
```

Question 10

![image.png](attachment:image.png)

In[276]:

```
fig, axes = plt.subplots(1, 2, figsize = (16,6))

cmtx_a_train = pd.DataFrame(
confusion_matrix(y_true=y_res_train_clu9,y_pred = y_pred_train_clu9, labels = classes, normalize = 'true'),
index=classes_index,
columns=classes_columns)

sns.heatmap(cmtx_a_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[0])
axes[0].title.set_text('Train Confusion Matrix for Cluster 9')

cmtx_a_train = pd.DataFrame(
confusion_matrix(y_true=y_res_test_clu9,y_pred = y_pred_test_clu9, labels = classes, normalize = 'true'),
index=classes_index,
columns=classes_columns)

sns.heatmap(cmtx_a_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1, ax = axes[1])
axes[1].title.set_text('Test Confusion Matrix for Cluster 9')
```

Question 11

![image.png](attachment:image.png)

In[334]:

```
# Narrowing to just rows that fall under Cp & Cq
sdfFM_train, sdfFM_test, clFM_train, clFM_test = train_test_split(sdf[sdf['class'] == 'FM'].iloc[:,0:2304], sdf[sdf['class'] ==
'FM'].iloc[:,2304], test_size=0.2, random_state=0)
sdfMY_train, sdfMY_test, clMY_train, clMY_test = train_test_split(sdf[sdf['class'] == 'MY'].iloc[:,0:2304], sdf[sdf['class'] ==
'MY'].iloc[:,2304], test_size=0.2, random_state=0)

X_train_SVM = pd.concat([sdfFM_train, sdfMY_train])
X_test_SVM = pd.concat([sdfFM_test, sdfMY_test])
y_train_SVM = pd.concat([clFM_train, clMY_train])
y_test_SVM = pd.concat([clFM_test, clMY_test])
```

In[335]:

```
sm = SMOTE(random_state=0, k_neighbors=30)
X_res_train_SVM, y_res_train_SVM = sm.fit_resample(X_train_SVM, y_train_SVM)
print('Resampled dataset shape %s' % Counter(y_res_train_SVM))

sm = SMOTE(random_state=0)
X_res_test_SVM, y_res_test_SVM = sm.fit_resample(X_test_SVM, y_test_SVM)
print('Resampled dataset shape %s' % Counter(y_res_test_SVM))
```

In[337]:

```
# SVM
start = time.time()
svm = SVC(kernel='linear') # Linear Kernel
#svclassifier = SVC(kernel='poly', degree=8)
svm.fit(X_res_train_SVM, y_res_train_SVM)
```

```
# Predicting the train/test Set results
y_pred_train_SVM = svm.predict(X_res_train_SVM)
y_pred_test_SVM = svm.predict(X_res_test_SVM)

end = time.time()

print(end-start)


# In[338]:


# Performances
trainperf_SVM = accuracy_score(y_res_train_SVM, y_pred_train_SVM)
testperf_SVM = accuracy_score(y_res_test_SVM, y_pred_test_SVM)

print(trainperf_SVM)
print(testperf_SVM)


# In[339]:


Perf_SVM = pd.DataFrame({'train Performance': trainperf_SVM, 'test Performance': testperf_SVM }, index =[0]).T
testconf_SVM = confusion_matrix(y_res_test_SVM, y_pred_test_SVM, labels = ['MY', 'FM'], normalize = 'true')

# Visuals for Report
cmtx_SVM = pd.DataFrame(testconf_SVM ,
    index=['true:MY', 'true:FM'],
    columns=['pred:MY', 'pred:FM'])

# Confusion Matrix Plots
plt.figure(figsize=(16,14))
a = sns.heatmap(cmtx_SVM, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)
a.set_xticklabels(a.get_xticklabels(), rotation = 0, fontsize = 12, horizontalalignment = "center")
a.set_yticklabels(a.get_yticklabels(), fontsize = 12, verticalalignment = 'center')
a.set_title("Confusion Matrix - SVM")


# ![image.png](attachment:image.png)

# ![image.png](attachment:image.png)
```