Harsh Masher October 1, 2017

1. In 2000, Enron was generating an annual revenue of $100 billion, was ranked the seventh largest company in the fortune 500, and the sixth largest energy company in the world[1]. Enron reported it's first loss in 4 years, a whopping $618 million, in October 2001. This prompted the U.S. Securities and Exchange to launch an investigation which eventually led to Enron's downfall[1]. By 2002 Enron fell into bankruptcy due to corporate fraud.

   Due to the resulting federal fraud investigation, confidential email and financial information became public record. The dataset provided contains 146 records with 14 financial features, 6 email features, and one label/classifier (POI). Of the 146 records, 18 were labeled as POIs. All the features had "NaN" as the most frequent value except for the POI label. A machine learning algorithm can be trained using this dataset to accomplish the goal of this project and identify persons of interest who may have committed fraud an Enron.

   An important part of identifying persons of interest (POIs) was to identify and remove outliers. This can prove difficult as many of the outliers may be POIs who deviate from the average. I detected outliers by using matplotlib to create a "bonus versus salary" plot. It was clear from the visualization that one value was significantly larger than the rest. I iterated over the dictionary and printed out the key and value for this point and discovered it was labeled "TOTAL". I found this value in the PDF file that the data was pulled from and also discovered that "THE TRAVEL AGENCY IN THE PARK" was included as well as "EUGENE E LOCKHART" who had "NaN" values for each of the features. These three outliers were removed from the dataset using ".pop".

2. After removing outliers, I converted the dictionary into a dataframe and started performing exploratory data analysis (EDA). As part of EDA, I found the correlation between the different features and POI detection, created boxplots to visualize the data, and found the difference in mean value for each feature for POIs and non-POIs. The EDA performed led me to create several features such as "bonus_to_salary_ratio", "poi_email", "poi_to_total_emails", "now_versus_later", and "fraction_of_deferred_income_to_total_payments". "Poi_email" is the sum of all the emails a person sent, received, or shared with a POI, whereas "poi_to_total_emails" is the ratio of these emails to the total emails sent or received. I wanted to create these variables because I thought that POIs would communicate more with other pois than anyone else. "Now_versus_later" and "fraction_of_deferred_income_to_total_payments" are both ratios that indicate when a person is receiving payments. "Now_versus_later" is the sum of all the payments someone is set to receive immediately versus the sum of the payments someone is set to receive later. These variables are important because a POI may have incentive to get paid immediately if they know the future is bleak.

   To determine which features would be most useful in detecting POIs, I utilized SelectKBest to pick the 10 best features from the dataset. I chose 10 because it would provide enough data to make a sufficient prediction while also reducing the time to run the algorithm. In addition to using SelectKBest alone, I used GridSearchCV and Pipeline to pair SelectKBest with a Naïve Bayes classifier to select the "x" best features. This proved useful because it showed me that it was better to use 8 features instead of 10. The 8 features used in my analysis are "salary", "bonus", '"total_stock_value", "shared_receipt_with_poi", "bonus_to_salary_ratio", "exercised_stock_options", "poi_email", and "poi_to_total_emails". The SelectKBest scores and p-values for all the features are in the three tables below. The first table contains only the initial features, the second table contains the initial and new (bolded) features, and the last table contains only the final features selected for my analysis.

SelectKBest Scores and P-Values of Initial Features		
Feature Name	**SelectKBest Score**	**P-Value**
Bonus	0.04	0.838
Deferral Payments	0.14	0.712
Deferred Income	0.05	0.815
Director Fees	0.44	0.508
Exercised Stock Options	0.13	0.721
Expenses	0.04	0.845
From Messages	0.40	0.528
From Poi to this Person	4.47	0.037
From this Person to Poi	0.02	0.893
Loan Advances	2.72	0.102
Long Term Incentive	0.01	0.939
Other	0.06	0.814
Restricted Stock	0.02	0.890
Restricted Stock Deferred	0.31	0.578
Salary	0.01	0.931
Shared Receipt with POI	6.84	0.010
To Messages	1.92	0.169
Total Payments	0.38	0.538
Total Stock Value	0.10	0.758

SelectKBest Scores and P-Values of Initial and New Features		
Bonus	30.73	0.000
Deferral Payments	0.01	0.921
Deferred Income	8.79	0.004
Exercised Stock Options	9.68	0.002
Expenses	4.18	0.044
From Messages	0.44	0.511
From Poi to this Person	4.96	0.028
From this Person to POI	0.11	0.739
Loan Advances	7.04	0.009
Long Term Incentive	7.56	0.007
Other	3.20	0.077
Salary	15.86	0.000
Shared Receipt with POI	10.72	0.001
To Messages	2.62	0.109
Total Payments	8.96	0.003
Total Stock Value	10.63	0.002
Bonus to Salary Ratio	**21.12**	**0.000**
POI Email	**10.20**	**0.002**
Total Messages	**0.64**	**0.421**
Poi to Total Emails	**11.34**	**0.001**

Now	**0.01**	**0.911**
Later	**7.10**	**0.009**
New Versus Later	**0.15**	**0.701**
Fraction of Deferred Income to Total Payments	**0.15**	**0.695**

**** bolded features are new, created features ****

SelectKBest Scores and P-Values of Final Selected Features		
Feature Name	**SelectKBest Score**	**P-Value**
Salary	5.99	0.016
Bonus	6.30	0.014
Total Stock Value	2.35	0.129
Shared Receipt with Poi	4.35	0.040
Bonus to Salary Ratio	**6.90**	**0.010**
Exercised Stock Options	1.55	0.216
Poi Email	**4.30**	**0.041**
Poi to Total Emails	**4.63**	**0.034**

**** bolded features are new, created features ****

3. After selecting the different features, I decided to try out a variety of different algorithms and check which provided the best results. The algorithms tested where Naïve-Bayes, Decision Tree, K-Means Clustering, Random Forest, Adaboost, K-Nearest Neighbors and support vector machine (linear and rbf kernel). The support vector machines took a long time to run so were immediately ruled out for the final analysis. I compared the precision and recall scores for the remaining classifiers and found that the three best classifiers were K-Nearest Neighbors, Naïve-Bayes, and Decision Tree. These algorithms had precision scores of 0.61694, 0.33659, 0.28358 and recall scores of 0.39700, 0.25800, 0.33150, respectively.

4. Most algorithms have many parameters that can be tuned to improve performance. If parameters are not tuned correctly the algorithm will be sub-optimal. To improve the performance of the algorithms, I utilized pipeline from the SKLearn library to combine the classifiers with SelectKBest to improve performance. For Naïve-Bayes I scaled the features using a MinMaxSclaer() to achieve a precision of 0.46073 and a recall of 0.30500. For the decision tree classifier, I found that adding more features resulted in lower precision and recall scores. Thus, I simply paired it with SelectKBest using pipeline and achieved a precision of 0.33092 and a recall of 0.36600.

Unlike the Naïve Bayes and Decision Tree classifiers, several parameters had to be tuned for K-Nearest Neighbors. After much trial an error, the following changes were made:

Parameter Name	**Default**	**Changed**
Leaf Size	30	2
Metric	Minkowski	Manhattan
Weights	Uniform	Distance

Combining these changes with SelectKBest via Pipeline yielded a precision of 0.69149 and a recall of 0.42250. As such, I initially chose KNearest Neighbors as my final algorithm**.** However, it was brought to my attention that K-Nearest Neighbors is an unsupervised learning algorithm that does not attend to the targets of your training points (i.e. POI label)[2]. As a result, I chose the **DecisionTree** Classifier as **my final algorithm**.

5. The dataset is split into two groups, testing and training, to validate the data. The training dataset is used to tune the algorithm, whereas the testing data is to validate it. A common mistake is to use the testing the data to tune the algorithm. Although this would lead to high accuracy, it would not translate well to real-world scenarios because the algorithm would be overfit. In this case, **StratifiedShuffleSplit** from the cross-validation package of the SKLearn library was used. It returns 1000 stratified randomized folds which each contain the same percentage of each class as the original dataset.
6. The two main evaluation metrics used to assess the performance of the algorithms are precision and recall. The goal is to get precision and recall scores both over 0.3. A precision of 0.3 would mean that 30% of the predicted POIs are correctly identified, whereas a recall of 0.3 means that the 30% of the time, POIs are correctly identified from the overall sample. The precision and recall for the three chosen classifiers are below:

Classifier	**Precision**	**Recall**
Naïve-Bayes	0.46073	0.30500
Decision Tree	0.33092	0.36600
K-Nearest Neighbor	0.69149	0.42250

** red means cannot use because is unsupervised learning algorithm**

<u>Resources</u>

1. http://www.cbc.ca/news/business/the-rise-and-fall-of-enron-a-brief-history-1.591559
2. https://github.com/scikit-learn/scikit-learn/blob/a95203b/sklearn/cluster/k_means_.py#L777