Response to Analysis Questions

Question #1:

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

Enron, a former corporate giant and at one point one of the largest companies in the United States, went bankrupt due to fraud committed by a select number of employees. The main objective of this project is to answer the following defined question. Through the utilization of machine learning*, can we develop a classifier that will effectively identify persons of interest linked to the Enron fraud scandal?*

Through techniques used in machine learning, the goal is to select certain meaningful features from our dataset and apply algorithms against those features to explore and analyze the data.

The dataset used for this project is a combination of the Enron email corpus and financial data. The dataset contains one-hundred and forty-six datapoints, each point being characterized by twenty-one distinct features. The data points represent an individual or person, while the features describe attributes or statistics concerning that person. The most import of these features is "poi" which divides the datapoints into two separate classes, "POI" and "non-POI". There are eighteen persons of interest in the dataset, and one-hundred and twenty-eight non-persons of interest. During the exploration of our features we will want to verify the cleanliness of the data. We found fourteen features that contained "NaN" as a value. These values were changed to zero to match the datatype for the feature. It was also important to remove any outliers or specific data points that reside far outside the range of other datapoints. The way individual outliers were identified varied, however each outlier removed was mostly done through visual inspection of the data. A combination of exporting the data to a .csv and print statements in python helped me find the following outliers. A useful function in python, "***.pop()"***, was used in our code to remove the following outliers from our dataset:

- "TOTAL" – this key value represented the sum of all the other keys values inside of the data dictionary. This would have grossly distorted the algorithms metrics when identifying a person of interest.
- "THE TRAVEL AGENCY IN THE PARK" – this key value seemed to represent an organization, perhaps on the payroll, and thus included in the financial data. However, to keep true to our data dictionary this key was removed as it did not represent an individual or person of interest.
- "LOCKHART EUGENE E" – this key contained "NaN" values for every single feature inside of the data dictionary. To avoid the similar problems the "TOTAL" column would have introduced, this key was removed from the dataset.

The final step I took was to clean up the values for the keys "BELFER ROBERT" and "BHATNAGAR SANJAY" due to them being entered incorrectly into the data dictionary. After cleaning the data, it was time to better understand the correlation between the "poi" feature and other features in the feature list

Question #2:

What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values.

Our final classification algorithm only utilized three features, which were “poi”, “total_stock_value”, and “exercised_stock_options”. During the process of coming to understand the data better as a result of the exploring the data, I wanted to combine or add together the two major categories of summed financial values (“total_stock_value” and “total_payments”) to assess the approximate net worth of the individual. Of course, assets belonging to each person was not reported in the financial dataset, and because of this I felt the name “total_financial_value” was an appropriate representation of the features data. My reasoning was I wanted to determine if the culmination of both categories correlated stronger to identifying a person of interest. This would allow me to measure this metric during my feature selection process.

Two feature selection methods were tested to compare performance on a “DecisionTree” classifier. The first feature select method was simply to utilize the “feature_importances_” property available to the “DecisionTree” classifier and use only the features with an importance greater than 0.2. Here is are the results for each feature and their associated computed importance (*features with an importance less than 0.2 were excluded in this write-up*). Important Features:

1. “to_messages” (0.220426513942)
2. “from_poi_to_this_person” (0.21197488041)

However, after fitting and predicting the classifier using the most important features, the results yielded zero for the recall and precision scores. Moving onto the “SelectKBest” feature selection process, the first thing I wanted to do was remove the any features that fell beneath the 85% variance threshold. “VarianceThreshold” was implemented to remove all these features based on the features training-set variance. This assisted the selection process to be more efficient as it removed having to evaluate the least impacting features. The selection process that was used in order to choose these three features was done through the implementation of the algorithm “SelectKBest”. Essentially, because no scoring parameter was passed in, “f_classif” scoring was used by the algorithm to compute the ANOVA-F value for each feature in the data dictionary. After, I sorted the features by highest to lowest scores I saved that value to a list called “scores_sorted”. I then used “matplotlib” to print a bar blot that visualized the features based on this score from highest to lowest. I used this visualization to notice the significant drop for features after the fifth feature. Here are the top five ranked features in that list. Highest Ranked SelectKBest Features:

1. “total_stock_value” (22.510549090242055)
2. “exercised_stock_options” (22.348975407306217)
3. “bonus” (20.792252047181535)
4. “salary” (18.289684043404513)
5. “total_financial_value” (16.989336421752615) (my created feature!)

I implemented this range of features (1-5) into my “param_grid” dictionary object, specifically the key, value pair shown here:

'feature_selection__k': [1, 2, 3, 4, 5]

This enabled “GridSearchCV” to try and use 1, 2, 3, 4, or 5 of the top features and determine how many features to utilize of the in order to return the most optimal results. I specified (1-5) as the range however, due to the information gained through my plot analysis. This is what makes “GridSearchCV” so powerful, feature selection can become more automated. For three of the classifiers (“SVC”, “K-Nearest Neighbors”, “AdaBoost”), principal component analysis(PCA), and a scaler (“MinMax”) were included into the steps of the pipeline to verify if any scaling or dimension reduction would be beneficial to the estimators. The reason behind exclusion for the first free was due to the coursework instructions where it mentioned that “DecisionTrees” and “LinearRegressions” do not benefit from feature scaling. However, the reason ultimately behind “GridSearchCV” returning an optimal estimator that did not utilize either scaling or principal component analysis was because these types of modifications to the features only benefit algorithms in which two dimensions affecting the outcome will be affected by rescaling. However, because my “AdaBoost” algorithm was using “DecisionTree” as its base estimator then “GridSearchCV” didn’t use it.

Question #3:

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

The algorithm that was chosen in the end was a "AdaBoost" meta-estimator classification algorithm. The choice for this decision was made when comparing the performance between the other classification algorithm tested ("Naïve Bayes", "DecisionTree", "SVC", "K-Nearest Neighbors"). The following are results printed from the function "*evaluate_optimized_classifiers"* inside our python script which was used to validate each algorithms performance.

Naive Bayes Classifier Results:

- Accuracy Score: 0.883720930233
- Recall Score: 0.2
- Precision Score: 0.5

Decision Tree Classifier Results:

- Accuracy Score: 0.860465116279
- Recall Score: 0.0
- Precision Score: 0.0

SVC Classifier Results:

- Accuracy Score: 0.860465116279
- Recall Score: 0.0
- Precision Score: 0.0

K-Nearest Neighbors Classifier Results:

- Accuracy Score: 0.860465116279
- Recall Score: 0.0
- Precision Score: 0.0

As you can see the "Naïve Bayes" classifier generally had better accuracy than the other algorithms. However, because "Naïve Bayes" does not contain a wide variety of parameters to tune, I decided to choose a different algorithm. "AdaBoost" contains many parameters that can be tuned, such as the "base_estimator" which will allow a classifier to be passed in to be used for the estimation the algorithm performs. If none is provided, then "DecisionTree" is the classifier, so technically our algorithm is "AdaBoost" with a "DecisionTree" base estimator.

Question #4

What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune)

Based on the algorithm chosen, there are a great deal of parameters available for use. These parameters are important, and you need to think about them due to the possibility of overfitting. The odds of overfitting increase when you do not have a lot of training data to work with, like us. Luckily, there is a validation algorithm that makes parameter

tuning a breeze. A validation algorithm provides the ability to test a multitude of parameters available to an algorithm in order to find the optimal parameters to use for the classifier. For the parameter tuning process, "GridSearchCV" was used to systematically comb through the multiple parameters and decide which parameters provide the best performance. The best-tuned parameters will increase the likelihood for the desired results as you can pass a scoring parameter to "GridSearchCV" so it can tune the parameters according to the scoring strategies defining model evaluation rules. The scoring strategy used was "f1", which can be defined as a weighted average of the recall score and precision score. The optimal "f1" score is one and worst is zero. Below is the formula for "f1" score:

*f1 = 2 * (precision * recall) / (precision + recall)*

Another awesome thing about "GridSearchCV" is that it contains attributes much like the "feature_importances_" property for the "DecisionTree" classifier. These attributes allow you to access various information about the validation object. For example, I used a join of "best_estimator_" and "get_params()" to show the best-tuned parameters in order to achieve the highest "f1" score. Below are the best parameters used for the "AdaBoost" algorithm.

Best Parameters:

- algorithm="SAMME.R'
- base_estimator=None
- learning_rate=0.55
- n_estimators=100
- random_state=None

Question #5

What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

A validation strategy is incorporated in machine learning in order to give an estimate of performance based on evaluation metrics derived from an independent sample of the dataset. Typically, validation is achieved through cross-validation by splitting the data into training and testing subsets, with each subset contains a portion of the features and label pairs. A classic mistake that can occur if validation is not done correctly is overfitting. By feeding the algorithm a portion of the data as training data, it will be able to identify the patterns in the data. After the testing data is used to validate the steps taken. Validation is important because it allows you to control the Bias-Variance dilemma which can negatively impact the algorithms performance. A high-bias algorithm does not give to much attention to the data and it oversimplifies the patterns. Whereas, a high-variance algorithm produces overfitting, and gives way too much attention to the data. The validation strategy allows you to find the perfect balance between bias and variance in order to allow you algorithm to learn from the patterns in the training data in order to make predictions on the testing data, which it's never seen before. Due to the size of the dataset and the relatively minuscule percentage of persons of interest that existed inside that dataset, we had to use a validation strategy that would split the data into training and testing subsets but also ensure that could preserve a percentage of samples for each class ("POI" and "non-POI"). Here is the actual code to our cross-validation model selection:

model_selection.StratifiedShuffleSplit(n_splits=100, test_size = 0.3, random_state = 12)

Question #6

Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

For this project, we wanted to identify persons of interest in a fraud scandal and accuracy simply represents the percentage of individuals correctly labeled as persons of interest out of the amount of available people. The key word here is correctly labeled, being branded as a suspect of fraud is a terrible thing and it is a very serious accusation to make, not one you want to make incorrectly (especially if you are a machine). Therefore, the recall scores and precision scores are very important and weighed heavily on my choice of an algorithm. What the recall score tells us is the percentage of individuals that were correctly classified as persons of interest out of all the individuals predicted to be persons of interest, whether they end up being one or not. The higher the recall score, the more likely you are to falsely accuse an individual as a person of interest. The precision score on the other hand, represents the percentage of individuals predicted as persons of interest that end up being a real person of interest. A high precision score provides confidence that if the algorithm is labeling an individual as a person of interest, it is correct. However, the downside to this is there could be individuals that are right near the edge and too high of precision means that the algorithm doesn't take the risk of incorrectly labeling a person of interest, meaning someone could get off the hook. We aimed for a score at or above 0.3 for both the recall and precision scores.