Project 6 – Discussion*

***Edits in grey**

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? [relevant rubric items: "data exploration", "outlier investigation"]***

The intention of the project is to create a machine learning algorithm that can identify 'persons of interest' (POIs) from a list of people who worked at Enron. POIs are people that were involved with the fraud at Enron to some degree. The dataset provided for this project was labeled and contained 18 identified POIs. As a result a supervised learning algorithm was used to train the dataset.

The dataset contained financial and email information about each person. The original dataset contained 146 data points and referenced 21 features. This included features such as people's salary, exercised stock options, and how many emails they sent and received from POIs. The original dataset does not have values for all the features for each person. Missing values were represented as 'NaN's and were converted to 0.0 when the data was processed. A big outlier that was in the data was the sum of all the individuals' features. It appears that the 'Total' row from a pdf was read into the dataset and contained features that were understandably much much larger. To account for this, the data point was removed completely from the dataset.

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. [relevant rubric items: "create new features", "intelligently select features", "properly scale features"]***

I included 6 features ('salary', 'total_payments', 'exercised_stock_options', 'expenses', 'frac_to_poi', 'frac_from_poi') in feature_list, which were then passed through the SelectKBest feature selection function. Two of the features in the feature_list, ‘frac_to_poi’ and ‘frac_from_poi’, were generated from existing features. They represent the fraction of emails send to and from POIs to the person, respectively. The new features were a means of checking the relative number of emails that the person sent or received from POIs, instead of the absolute values presented.

No feature scaling was performed as the algorithms used (DecisionTreeClassifier and RandomForestClassifier) are invariant to scaling, unlike SVMs. The feature scores for the SelectKBest automatic future selection are - [10.28104113 9.21965362 26.43169945   0.78313337   7.40448556   1.07200143] respectively. The parameter, k, was selected using the GridSearchCV method, with options from 1 - 6 provided. The GridSearchCV method returned k = 5 as the best parameter for SelectKBest. As a result, the ‘expenses’ feature was not used.

3. ***What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]***

I used the Decision Tree Classifier after trying a couple of other algorithms such as Naive Bayes and Random Forest. The table below outlines their performance:

	Decision Tree	Random Forest	Gaussian NB
My Evaluation			
F1	0.4	0.4	0.136
Precision	0.4	0.4	0.6
Recall	0.4	0.4	0.077
Tester.py Evaluation			
F1	0.322	0.189	0.194
Precision	0.327	0.304	0.113

Recall	0.318	0.138	0.702
True Positives	635	275	1404
False Positives	1309	631	11036
False Negatives	1365	1725	596
True Negatives	11691	12369	1964

Based on the results, the Gaussian NB classifier doesn't meet the requirements of the tester.py script resulting in evaluation scores over 0.3. In addition to that, it seems to be classifying most events as positive, as seen by the exceedingly high number of false positives. The Random Forest performed better with a high number of true negatives, however, it did not meet the tester.py criteria.

The Decision Tree balanced both aspects, passing the tester.py criteria in addition to correctly classifying more true positives and a high number of true negatives as well. As compared to the Random Forest Classifier it leant more towards classifying false positives, however not to the degree of the Gaussian NB classifier. This meant it was more likely to flag a person as a POI, which I think is more important than leaning the other way, so a potential POI can be investigated and doesn't slip through the cracks.

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 -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: "discuss parameter tuning", "tune the algorithm"]***

The parameters of an algorithm are the factors specific to the algorithm that influence how the decision boundaries are formed. For example, with Decision Trees, one of the parameters that can be tweaked is 'min_samples_split'. This determines at what number of leaf (end nodes) to stop splitting the tree. Tuning the parameters of an algorithm refers to changing the numbers associated with the parameters to optimize the algorithms performance. Since tuning affects the decision boundary of the training dataset, over engineering the tuning parameters to the particulars of the dataset can lead to overfitting. The bias-variance tradeoff needs to be kept in mind.

I used GridSearchCV to automatically tune the parameters for the algorithm. For the Decision Tree I provided a list of parameters ranging from 2 – 5 to tune min_samples_split. The tuned value came to 2. For Random Forest, which was another classifier I was testing, I tested the parameter 'max_depth' between values of 2 – 4, and 4 was the optimized value.

5. ***What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: "discuss validation", "validation strategy"]***

Validation is the process of splitting the dataset or reserving some data to test against the algorithm after it has been trained. This allows for an objective check on how good the algorithm is with data it has not seen before and serves as a check against overfitting. A mistake that one can make is to train the algorithm on all the available data, leading the algorithm to be trained on the intricacies of the dataset and potentially be overfit.

The algorithm would have high variance and might not be able to generalize well to new data. Another mistake that is can be made in the machine learning workflow is to apply transforms to the data on both the training and testing set, or on the dataset as a whole before it is split. For example, with PCA, if both the training and testing data are transformed, then there is no objective way to test the PCs used in the training section.

I used the cross validation parameter (cv) in GridSearchCV. This inherently uses k-fold cross validation. The number of folds was set to 13.

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. [relevant rubric item: "usage of evaluation metrics"]***

I used the F1, precision, and recall scores as a means of testing the algorithm. Precision is the probability that the algorithm is correct about the choice it made (True Positives/(True Positives + False Positives)). Recall is the probability of classifying an object correctly given that the object is actually the class in question (True Positives/(True Positives + False Negatives)). The F1 score is the ~average of the precision and recall scores.

I split the dataset into a training and testing set. I used the training set to pass the pipeline to GridSearchCV. I then tested the fit classifier with the test dataset and computed the scores on those predictions and test labels. The results for the scores are listed in the table in question 3.

My F1 score, precision, and recall were all 0.4. High precision but poor recall indicates that the algorithm is good at identifying true positives (POIs) and has a higher false positive count. High recall and low precision mean there's a lower rate of identifying true positives (POIs) and has a higher false positive count. This equivalence for precision and recall in my results leads me to think that the algorithm is just as good at getting false positives as it is to get false negatives versus true positives (POIs). This is further elucidated in the tester.py evaluation where the number of false positives is 1309 and the number of false negatives in 1365.