Project goal

The goal of this project was to use machine learning to predict whether a person is a person of interest based on their financial and email data/features. Machine learning is useful since it can find patterns in the data/features and use them to classify whether they are a person of interest. The data included 133 data points, 18 of which were POIs and 115 were not. My algorithm used 7 features.

There were features with missing data, denoted “NaN”, and were treated as 0 in the feature list. This actually proved useful to the algorithm since some features were missing mostly for one label, so if the algorithm saw that it was missing, it would be able to use that information for classification. There was one outlier with the data, which was the “TOTAL” data point. I removed that completely from the data. For my created feature (ratio between stock and payments), there were people who did not have a bonus or salary, so the ratio would be inflated. These were outliers for me so I set the ratio for those people to 0.

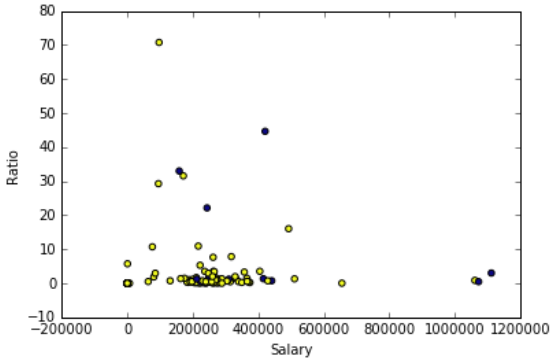
Feature selection

For my financial features, I wanted to use features that had a clear difference between POI and non-POI, and also features that were not missing many values. At first, I used salary and bonus and later added Exercised Stock Options since it was not missing many values, and Restricted Stock Deferred since it was a feature that mostly non-POI had. For email features, I used 'from_poi_to_this_person', 'from_this_person_to_poi', and 'shared_receipt_with_poi' because in the lesson, it was shown that these features were effective. I tried about 10 different combination of these before getting the performance I wanted. The combinations and performances are show below.

Features	F1	Recall	Precision
'poi','salary', 'bonus','from_poi_to_this_person', 'from_this_person_to_poi','shared_receipt_with_poi', 'ratio'	0.29094	0.29050	0.29137
'poi','salary', 'bonus','from_poi_to_this_person', 'from_this_person_to_poi','shared_receipt_with_poi'	0.26846	0.21000	0.37201
'poi','salary', 'bonus','ratio','from_poi_to_this_person', 'from_this_person_to_poi','shared_receipt_with_poi', 'exercised_stock_options','restricted_stock_deferred'	0.21551	0.18200	0.26415
'poi','salary', 'bonus','from_poi_to_this_person', 'from_this_person_to_poi','shared_receipt_with_poi', 'exercised_stock_options','restricted_stock_deferred'	0.30498	0.30650	0.30347

I did have to scale all these features using a minmax scaler to use SVM.

I also decided to create a feature which calculated the ratio between Total Stock Value and Total Payments. I noticed that for most POIs, this ratio was high and when combined with the salary, seemed to differentiate the two labels pretty well:



Unfortunately, it did not end up increasing the performance, so I left it out of the final feature set, but did append it to my_feature_list list.

Algorithm selection

I tried SVM, Gaussian NB and Decision Trees. SVM performed the best, followed by Decision Trees. I was not able to tune anything on NB, which is probably why it performed the worst.

Parameter tuning

Tuning the parameters of an algorithm is essential to its performance. It dictates how much each data point affects the algorithm. If not done well, it will cause overfitting on one extreme, and an untrained/poorly trained algorithm on the other.

I tuned the PCA components, gamma and C parameters of the SVM, and “min_samples_split”, “max_depth”, “min_samples_leaf”, and “max_leaf_nodes” parameters of the decision tree. I used GridSearchCV to do this. I used a range of values from the SKLearn Documentation, and when the optimal parameters was on the edge of that range, I extended the range. I printed the best parameters and fed it into my clf pipeline.

The Grid returned the following optimal PCA components:

Component	Explained Variance Ratio
1	0.55336179
2	0.16567931
3	0.08130132
4	0.06943843
5	0.04855277
Total	0.918333626212

The PCA reduced the number of components from 7 to 5, but still kept about 92% of the variance.

~~Note: I did want to combine KFold and GridSearchCV so that GridSearchCV would optimize to an average score of the K runs, but did not know how to do that. I am~~

~~guessing I would need to define a custom scoring function.~~ Previous reviewer explained how to do this. After doing further research, it looks like GridSearchCV uses StratifiedKFold by default anyway.

Validation

Validation is the process of assessing the performance of the algorithm on independent datasets. If done wrong, the data can be overfit and will not perform as well on real, separate data as it does on the training data. I validated my data by splitting it into independent training and testing sets, and also used KFold Cross Validation.

Evaluation

Since the majority of the data is non-POI, the accuracy score would be misleading for this case. I used Precision, Recall and F1 score to evaluate my performance. The precision was consistently high, with an average of about 0.4-0.5. I originally used F1 Score(around 0.4-0.5) as the objective of my algorithm, but found that it would give consistently high Performance score, but low Recall score (around 0.1-0.3), which meant that if classified as a POI, it is very likely a POI, but it would also fail to classify actual POIs correctly; there were many false negatives. This was most likely due to the lack of actual POIs in the original training set.

As a result, I used Recall as my objective and was able to achieve Recall and Precision above 0.3.