Intro to Machine Learning Final Project

Enron Dataset

This dataset consists of 146 employees of Enron and 21 features consisting of both financial and email information. 18 of these 146 employees have been identified as “Person of Interest” or a “POI”. Two of these “employees” were determined to be outliers and were eliminated for this analysis. One was the key “TOTAL” which is an artifact of an earlier dataset where all the columns were being summed. The other is “THE TRAVEL AGENCY IN THE PARK” as it was not a valid employee. I investigated how many missing values there were in the dataset and came back with the follow results:

```
{'bonus': 63,
 'deferral_payments': 106,
 'deferred_income': 96,
 'director_fees': 128,
 'email_address': 33,
 'exercised_stock_options': 43,
 'expenses': 50,
 'from_messages': 58,
 'from_poi_to_this_person': 58,
 'from_this_person_to_poi': 58,
 'loan_advances': 141,
 'long_term_incentive': 79,
 'other': 53,
 'restricted_stock': 35,
 'restricted_stock_deferred': 127,
 'salary': 50,
 'shared_receipt_with_poi': 58,
 'to_messages': 58,
 'total_payments': 21,
 'total_stock_value': 19}
```

The features with the most amount of ‘NaN’ values were director fees, loan advances, and restricted sto ck deferred. This make sense as these values probably only apply to higher ranking employees.

Algorithm

The first algorithm tested was a Decision Tree Classifier:

Score	Precision	Recall
.81273	.2395	.2322

The next algorithm tested was the Gaussian Naïve Bayes.

Score	Precision	Recall
.33433	.1474	.8345

And last SVC:

Score	Precision	Recall
.866666	0.0	0.0

While the Naïve Bayes model has a significantly higher recall, it was identifying far too many false positives. I decided to use the Decision Tree model as it has a far more balanced score.

Features

I decided to add two new features to the dataset, each a fraction of emails sent to or from that employee and a person of interest.

These features gave a slight boost to the precision and recall of the Decision Tree model.

	Precision	Recall
Before new features	.2395	.2322
With new features	.2459	.2475

The fraction of emails sent to a POI proved to be an important feature in the final feature selection. I used the same decision tree classifier as before to weigh features, but this time trained it on the entire dataset, not just the training set I had used before. Because the limited amount of POI's, I wanted to make sure that the model was fit correctly and was properly weighing the features. After using the feature_importances_ attribute, I eliminated any feature that scored 0. I looped this process over the decision tree to keep eliminating features until only the most important remained: 'total_payments', 'shared_receipt_with_poi', 'fraction_to_poi', 'exercised_stock_options', 'expenses', 'restricted_stock'.

	Precision	Recall
All Features	.2459	.2475
Reduced Features	.3620	.3800

This had an extremely large impact on the model with a much higher precision and recall.

Algorithm Tuning

Each algorithm uses a set of parameters that can have drastic results on the final result. It's important to test each algorithm with a variety of parameters to ensure that you are getting the highest results possible. Thankfully, there are ways to automate this process to ensure you can test a large group of parameters programmatically. I tuned the algorithm using the Grid Search CV function, with weighting the scoring toward recall, since I wanted to model to favor identifying POIs. Using the Grid Search CV function is essential as it allows us to test many attributes to make sure that the model is properly tuned with the correct attributes. The parameter grid used for searching was determined by hand testing variables to get a good feel for an appropriate range. The parameters I used for testing were:

'max_features': ['auto', 'sqrt', 'log2']

Max features determines the amount of features to use when splitting. 'auto' and 'sqrt' will test by taking the max features as the square root of the features list of the data set. 'log2' will take the log2 of the list of features.

'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10]

Miminimum samples split will decide how many samples of the feature need to be present for the tree to split based on this feature.

'max_depth' : [None, 2, 4, 6, 8, 10]

Max depth is the depth of the decision tree. This will determine how many splits the tree will make.

After the algorithm tuning the model performed even better:

	Precision	**Recall**
Before Tuning	.3620	.3800
After Tuning	.3705	.3873

Validation

This new Decision Tree model was run through the provided tester.py script to properly test the new algorithm parameters. The tester.py program uses a k-folds cross validation program to split the data into training and testing sets many times to test the functionality of the model. Validation over many sets of data is important because if we were to test the model once over one training set, we may get inaccurate results, especially with the limited number of POIs. We want to test the model many, many times and take a look at the results to ensure they are accurate.

What we are looking for more than accuracy is a high precision and recall score. The precision identifies the number of true positives we've identified over the all positives. What this means is how many of the employees as a POI / non-POI correctly. Next we have the recall score is the number of true positives divided by true positives added to the false negatives. This is essentially how many employees were identified as a POI and actually are a POI. Because we would rather get all POIs while possibly incorrectly flagging some innocent people as POIs, we prefer a higher recall. While the accuracy went slightly down, the precision and recall of the model both rose to .392 and .366. The model is now much better at identifying POIs from the selected features.