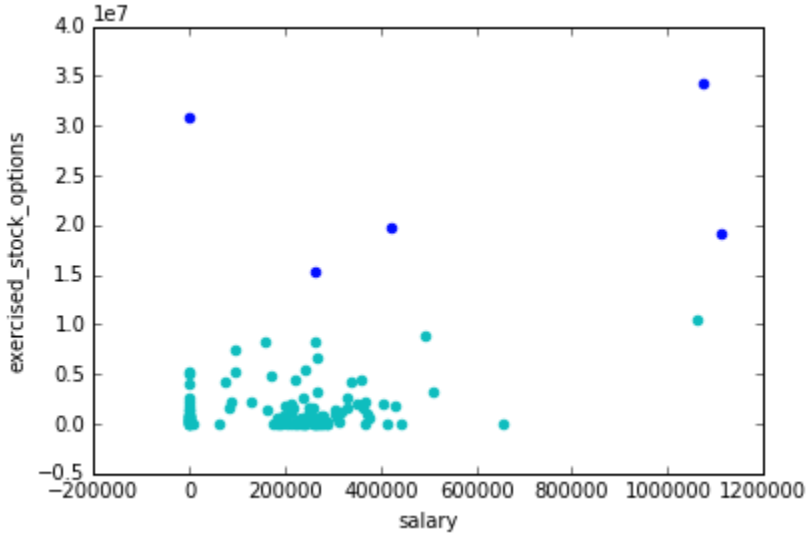
Enron Submission Free-Response Questions



1: Summarize for us the goal of this project…

The goal of this project was to optimize a machine learning algorithm to correctly identify fraud from Enron financial data and emails. In 2002, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. In this project, we will use machine learning algorithms to build a person or persons of interest identifier based on the publicly available financial and email date which resulted from the Enron scandal.

Employees
There are 146 Enron employees in the dataset. 18 of them are persons of interest (POIs).

Features
There are 14 financial features in the dataset They are: exercised_stock_options, total_stock_value, bonus, salary, deferred_income, long_term_incentive, restricted_stock, total_payments, loan_advances, expenses, director_fees, deferral_payments, restricted_stock_deferred, and other.

There are 6 email features. They are: to_messages, from_messages, from_poi_to_this_person, from_this_person_to_poi, shared_receipt_with_poi, and email_address.

And there is 1 feature which indicates whether or not the Enron employee is a person of interest and that is the poi label.

Missing Features
All of the features, except poi, have missing values which are represented as ‘NaN’. These ‘NaNs’ are converted to zeros when featureformat is called.

Outliers

There were two outliers which I noticed from the FindLaw PDF file that we were given as part of the project. The outliers were ‘TOTAL’, and ‘THE TRAVEL AGENCY IN THE PARK.’ I removed them from the from the dataset.

The goals for this project were exploring the data for learning, cleaning, and preparing the data, selecting the features which optimize the algorithm, creating new features, and picking the best algorithm from several supervised machine learning algorithms, then validating the model to accurately identify the person of interest.

2: What features did you end up using in your POI identifier…

I end up using these features:

```
features_list = ['poi',
                 'bonus',
                 #'deferral_payments',
                 'deferred_income',
                 #'director_fees',
                 'exercised_stock_options',
                 #'expenses',
                 #'loan_advances',
                 #'long_term_incentive',
                 #'restricted_stock',
                 #'restricted_stock_deferred',
                 'salary',
                 #'total_payments',
                 'total_stock_value',
                 #'from_messages',
                 #'from_poi_to_this_person',
                 #'from_this_person_to_poi',
                 #'shared_receipt_with_poi',
                 #'to_messages',
                 'bonus_salary_ratio']
                 #'total_exercised_ratio'] ### Added 2 new features
```

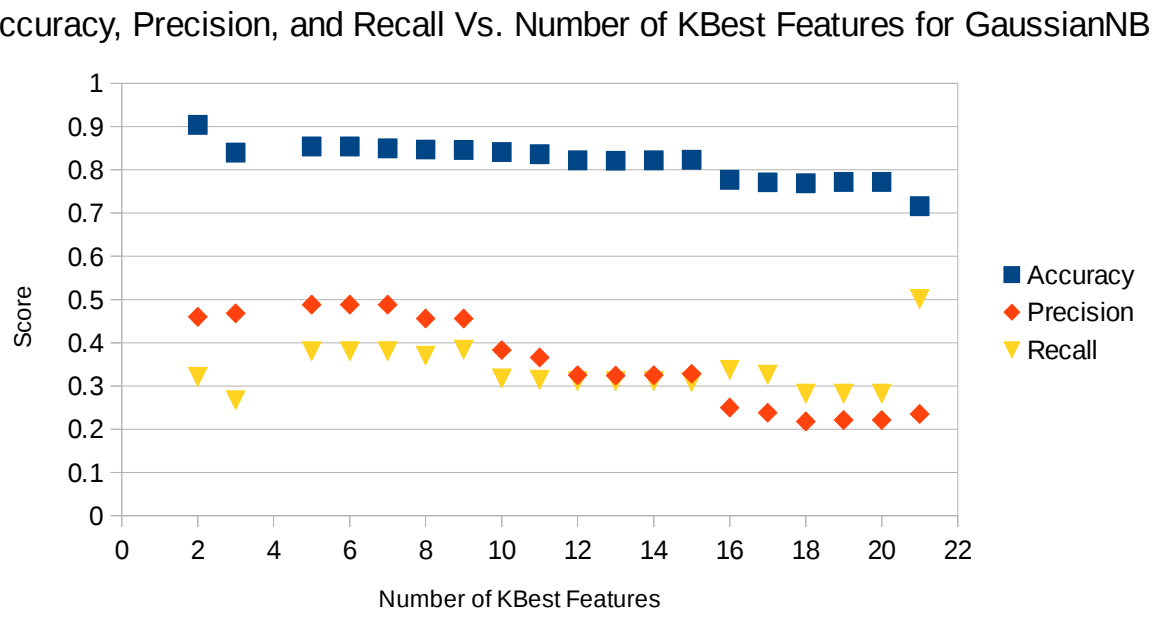
I excluded ‘email_address’, and ‘other’ from the features list. I excluded email because it wouldn’t have helped, and ‘other’ because it wasn’t clear what ‘other’ was. I created two additional features. They were bonus_salary_ratio and total_exercised_ratio. The first feature I created, bonus_salary_ratio, is a reflection on how large a POIs bonus would be relative to their salary. I created it by dividing a POIs bonus by their salary. A very large ratio might indicate a potential POI. The second feature I created, total_exercised_ratio, is a reflection on what portion of their total stock value is from their exercised stock options. I thought is would be a good indicator as it is generally an indicator of where someone buys low and sells high which might indicate criminal activity by inside trading. I thought these ratios would be good indicators of POIs because this ratio might indicate unreasonable income, and I thought following the money was a good measure for POIs. As it turned out, based on the KBest rankings, bonus_salary_ratio turned out to be a good indicator for POI with a score of 10.78, and

(despite being my favorite feature) total_exercised_ratio turned out to be a poor indicator for POI with a score of 0.14.

After removing the ‘poi’ feature from the list, I used SelectKBest to rank the features. Here are how the features scored:

exercised_stock_options	24.81
total_stock_value	24.18
bonus	20.79
salary	18.28
deferred_income	11.45
bonus_salary_ratio	10.78
long_term_incentive	9.92
restricted_stock	9.21
total_payments	8.77
shared_receipt_with_poi	8.58
loan_advances	7.18
expenses	6.09
from_poi_to_this_person	5.24
from_this_person_to_poi	2.38
director_fees	2.12
to_messages	1.64
deferral_payments	0.22
from_messages	0.16
total_exercised_ratio	0.14
restricted_stock	0.06

After knowing how the features scored, I tried 17 combinations of features for each of my classifiers. To achieve this, I removed the lowest KBest scored feature from my list of KBest features until I had 4 of the strongest features left. For GaussianNB, the features plotted out like this:



And for the other classifiers, here are the best features for them:

KNeighborsClassifier()-4 features

feature_list = ['poi', 'bonus', 'excercised_stock_options, 'total_stock_value']

DecisionTreeClassifer()-11 features:

features_list = ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'long_term_incentive', 'restricted_stock', 'salary', 'total_payments', 'total_stock_value', 'shared_receipt_with_poi', 'bonus_salary_ratio']

DecisionTreeClassifer()-9 features

features_list = ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'long_term_incentive', 'restricted_stock', 'salary', 'total_stock_value', 'bonus_salary_ratio']

AdaBoostClassifier()-14

features_list = ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'expenses', 'loan_advances', 'long_term_incentive', 'restricted_stock', 'salary', 'total_payments', 'total_stock_value', 'from_poi_to_this_person', 'shared_receipt_with_poi', 'bonus_salary_ratio']

However, these ones didn't score (from the tester.py program) above the 0.3 requirement. Additionally, the decisiontree and adaboost algorithms ran all night, and the adaboost was still running in the

morning after 12 hours so I wasn't able to get that score from the tester.py program for this one. I did include screen-shots of the other classifier's outcomes in the files I submitted.

As for the features, while I tried a combination all of the features, the best scores I got were the ones related to the money, and the all of the features were high score features. Additionally, I used standardscaling to bring all the features to the same magnitute. This does it by ensuring that for each feature the mean is 0 and the variance is 1.

3: What algorithm did you end up using? …

The best algorithm was GaussianNB. It seemed like one of the best classifiers from the beginning, and as I continued to optimize the algorithms, it turned out to be the best one. Besides GaussianNB, I tried KNeighborsClassifier, DecisionTreeClassifier, AdaboostClassifier.

Kneighbors, Adaboost, and DecisionTree remained unchanged the most as I optimized the parameters, but with the right features, their scored did change. LinearSVC. LinearSVC() didn't seem very fit as a classifier for this dataset.

4: What does it mean to tune the parameters of an algorithm…

Tuning the parameters of an algorithm involves changing the algorithm input parameters and measuring the performance for each change. When there are many parameters, you enter the parameters as a range in a parameter_grid function that you pass to GridSearchCV, and measure the performance of each combination in order to determine the best set of parameters. The parameters influence the performance for a given metric. An incorrectly tuned algorithm can give poor performance on the training set (underfitting) or poor generalization (overfitting). Parameter tuning improves the performance of the algorithm, and one that isn't tuned well won't perform as a reliable model. Adaboost and LinearSVC had the lowest scores. LinearSVC was better than SVC.

Except for GaussianNB, I optimized the classifiers by using GridSearchCV. To tune my models with GridSearchCV, I used the SkLearn page for each classifier and put the parameters into a parma_grid function, and then I added para_grid to GridSearchCV. For GaussianNB, which didn't have enough parameters to tune with GridSearchCV, I used feature selection to optimize that algorithm.

Here are the parameters that I tuned:

Kneighbors:

```
kngs_param_grid = {'n_neighbors': [3,4,5,6,7,8,9,10],
                   'weights': ['uniform', 'distance'],
                   'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
                  }
```

DecisionTree:

```
dtgs_param_grid = {'criterion': ['gini', 'entropy'],
                   'splitter': ['best', 'random'],
                   'max_depth': [1, 2, 3, 4, 5, 6, 7,8,9,10,15,20],
                   'min_samples_split': [2, 3, 4, 5, 6,7,8],
                   'min_samples_leaf': [1,2,3,4,5,6,7,8]
```

```
            }
```

AdaBoost:

```
ada_param_grid = {'algorithm': ['SAMME.R'],
            'n_estimators': [25, 50, 100, 150, 175, 200, 225, 250],
            'learning_rate': [1, 2, 3, 4],
            'random_state': [1, 2, 3, None]
           }
```

LinearSVC:

```
lsvc_param_grid = {'penalty': ['l2'],
   'loss': ['squared_hinge', 'hinge'],
   'C': [1, 2, 2],
   'random_state': [1, 2, 3, 4, None]
   }
```

5: What is validation…

Validation is the testing of a machine learning algorithm by applying it to data that was not seen during training. The purpose of this is to get an estimate of the algorithms generalization performance. A classic mistake in validation is not to have separation between the data sets used for parameter tuning and those used for obtaining a validation score. This can lead to an over-estimate of the algorithms ability to generalize. The poi_id.py program separated the dataset into training and testing sets. The training set was 70% of the data while the testing set was 30%. The analysis was validated by using a stratified shuffle split cross-validation approach. The tester.py program randomly split the dataset into 1000 folds whereby a fold is used to test the data, and others are used for training. This process is repeated multiple times, and the scores are averaged. Because our dataset was small, this was necessary to adequately validate our model. And finally, while the main advantage for using this validation scheme is that we can achieve more accurate models, the disadvantage can be increased computational times as I saw while running it.

6: Give at least 2 evaluation metrics…

The metrics I choose to use were Accuracy, Precision, and Recall. The accuracy score tells how often the test samples were classified correctly. The precision score is an estimate of the probability that the label of an example is true given that the algorithm predicted it is true. It's the probability that a person is actually a POI given our algorithm predicts they are. The recall score is an estimate of the probability that the algorithm will predict true for an example given the actual label is true. In our context this is the probability that the algorithm will identify someone as a POI if they in fact are.

My best score was for the GaussianNB. Here are the values:

From poi_id.py:

```
g_accuracy:  0.904761904762          g_precision_score:  0.333333333333
g_recall_score:  0.333333333333      g_f1_score:  0.333333333333
```

And from tester.py:

Accuracy: 0.85464 Precision: 0.48876 Recall: 0.38050 F1: 0.42789

The accuracy score is the fraction of correct predictions (TP+TN) from all samples (TP+TN+FP+FN). The precision score is the ratio of true positives (TP) out of all positives (TP+FP). The recall score is the fraction of true predictions from from positive predictions (TP+FN), and the f1 score is the weighted average of the precision and recall scores.

So regarding my scores, A precision score of 0.488 means that of the individuals labeled by my model as persons of interest, 48.8% of them were indeed persons of interest. A recall score of 0.380 means that my model identified 38.0% of persons of interest present in the entire dataset.

And finally, while my algorithm isn't very accurate, it does illustrate the importance of machine learning. With a properly tuned algorithm, machine learning can provide insights from data that can inform better decisions, and enable companies to create more efficient systems, and even greater income.