Identifying Fraud from Enron Data

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"]**

Project Goals:

- Understanding the dataset.
- Analyzing the problem.
- Selecting the perfect tools that help us analyze and visualize the dataset.
- Performing data wrangling for data cleaning like outliers.
- Feature selection and scaling.
- Choosing an algorithm.
- Coding to find the accuracy and prediction.
- Writing a report on my findings.

Background Information:

In 2000, Enron was one of the largest companies in united states but collapsed into bankruptcy due to corporate fraud. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data to executives.

In this project, using the data available and with the help of scikit-learn and machine learning methodologies I will investigate the person of interest who may have committed the fraud.

Number of features: 8
Number of people in the dataset: 146
Number of poi in the dataset: 18
Number of person who is not poi: 128

As you can see above, the number of features I have selected is 8. There are total of 146 people in the dataset out of which 18 are claimed to be the person of interest.

Outliers:

I have used **matplotlib's pyplot** for plotting the outliers. Fig 1 shows the scatter plot with outlier for salary and bonus. In fig 2, the outlier is removed.

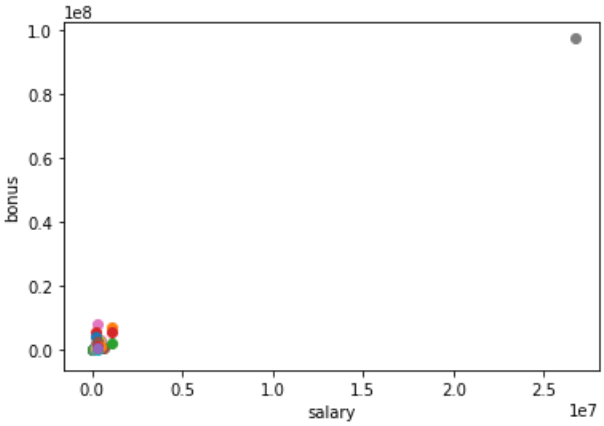

Fig 1

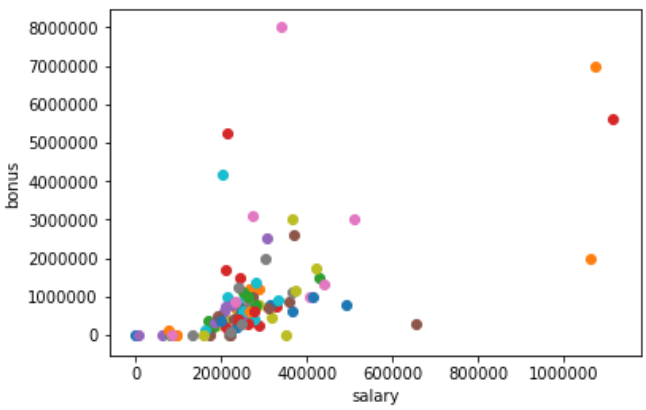

Fig 2

I have removed the outlier by removing the 0^{th} index from the
'TOTAL' -> A spreadsheet artifact,
'LOCKHART EUGENE E' -> It has non NaN values,
'THE TRAVEL AGENCY IN THE PARK'-> does not represent any individual of interest.

After removing the outliers,
TOTAL Number of data points after removing outliers is : 143

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"]**

Feature Engineering:

Initially I have used 'poi', 'salary','bonus', 'total_stock_value', 'exercised_stock_options', 'shared_receipt_with_poi' features. I later added 'bonus_salary_ratio', 'from_poi_to_this_person' 'from_this_person_to_poi_percentage' to the features list.

I have added these new features inorder to find the strongest communication between poi and other person.

Selection Process:

I used a univariate feature selection process, select k-best, in a pipeline with grid search to select the features. Select k-best removes all but the k highest scoring features. The number of features, 'k', was chosen through an exhaustive grid search driven by the 'f1' scoring estimator, intending to maximize precision and recall.

Feature Selection:

To select the most influential features, the manual selection along with automated scikit learn's SelectKBest module. Then I have performed feature scaling using the MinMaxScalar module to make sure the features weighed equally before applying to the classifier.

After adding some new features these are features I am working with
['poi', 'salary','bonus', 'bonus_salary_ratio', 'total_stock_value', 'exercised_stock_options', 'from_this_person_to_poi_percentage', 'from_poi_to_this_person', 'shared_receipt_with_poi']
The first number is feature importance (from the decision tree classifier) and the second is feature score (from select k-best). The order of features is descending based on feature importance.

- Feature no. 1: bonus_salary_ratio (0.658595952706) (22.1067164085)
- Feature no. 2: shared_receipt_with_poi (0.180270198721) (6.1299573021)
- Feature no. 3: total_stock_value (0.161133848573) (16.8651432616)
- Feature no. 4: exercised_stock_options (0.0) (16.9328653375)
- Feature no. 5: bonus (0.0) (34.2129648303)
- Feature no. 6: salary (0.0) (17.7678544529)

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 focused on three algorithms, with parameter tuning incorporated into algorithm selection i.e. parameters tuned for more than one algorithm, and best algorithm-tune combination selected for final analysis. These algorithms were:

   - decision tree classifier
   - SVM
   - k-nearest neighbors

   Though I used the classfiication report for quick checks, I used tester.py's evaluation metrics to make sure I would get precision and recall above 0.3 for the Udacity grading system. Here is how each performed:

   - The decision tree classifier had a precision of 0.36472 and a recall of 0.62750, both above the 0.3 threshold.
   - The SVM classifier (with features scaled) had a gaudy precision of 0.83333, but a poor recall of 0.06500.
   - The k-nearest neighbors classifier had a precision of 0.32247 and a recall of 0.29200, both near but not both above the 0.3 threshold.

   The SVM classifier is not a good fit for the unbalanced classes in the Enron data set, i.e., the lack POIs vs. the abundance of non-POIs. The SVM overfit to the points in the training set, and poorly generalized to the test set resulting in a relatively small number of positive predictions of POIs.

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"]**

   Machine learning algorithms are parameterized just like functions and we can tweek these parameters to influence the outcome of the learning process. These parameters start off set to their default values but once we start modifying these parameters around it's known as 'tuning the algorithm'.

   The objective of algorithm tuning is to find the best point or points in the problem where performance can be optimum. The more tuned the parameters of an algorithm, the more biased the algorithm will be to the training data and test harness. This strategy can be effective, but it can also lead to more fragile models that overfit the test harness and don't perform as well in practice.

   There are a variety of certain ways e.g. a manual guess-and-check method or automatically with GridSearchCV and algorithm performance can be measured in a variety of ways (e.g. accuracy, precision, or recall). If you don't tune the algorithm well, performance may suffer.

The data won't be analyzed well and you won't be able to successfully make predictions on new data.

I used GridSearchCV for parameter tuning. It allows us to construct a grid of all the combinations of parameters, tries each combination, and then reports back the best combination/model.

For the chosen decision tree classifier for example, I tried out multiple different parameter values for each of the following parameters (with the optimal combination bolded). I used Stratified Shuffle Split cross validation to guard against bias introduced by the potential underrepresentation of classes (i.e. POIs).

- criterion=['**gini**', 'entropy']
- splitter=['**best**', 'random']
- max_depth=[None, 1, **2**, 3, 4]
- min_samples_split=[**1**, 2, 3, 4, 25]
- class_weight=[None, '**balanced**']

Note that a few of these chosen parameter values were different than their default values, which proves the importance of tuning.

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 a way to substantiate your machine learning algorithm's performance, i.e., to test how well your model has been trained. A classic validation mistake is testing your algorithm on the same data that is was trained on. Without separating the training set from the testing set, it is difficult to determine how well your algorithm generalizes to new data.
In my poi_id.py file, my data was separated into training and testing sets. The test size was 30% of the data, while the training set was 70%.
I also used tester.py's Stratified Shuffle Split cross validation as an alternate method to gauge my algorithm's performance. Because the Enron data set is so small, this type of cross validation was useful because it essentially created multiple datasets out of a single one to get more accurate results.

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"]**

The two notable evaluation metrics for this POI identifier are precision and recall. The average precision for my decision tree classifier was 0.36472 and the average recall was 0.62750. What do each of these mean?
- Precision is how often our class prediction (POI vs. non-POI) is right when we guess that class
- Recall is how often we guess the class (POI vs. non-POI) when the class actually occurred

In the context of our POI identifier, it is arguably more important to make sure we don't miss any POIs, so we don't care so much about precision. Imagine we are law enforcement using this algorithm as a screener to determine who to prosecute. When we guess POI, it is not the end of the world if they aren't a POI because we'll do due diligence. We won't put them in jail right away. For our case, we want high recall: when it is a POI, we need to make sure we catch them in our net. The decision tree algorithm performed best in recall (0.63) of the algorithms I tried, hence it being my choice for final analysis.

Reference:

1 https://classroom.udacity.com/nanodegrees
2 https://github.com/udacity/ud120-projects/blob/master/final_project
3 http://scikit-learn.org
4 https://stackoverflow.com/questions/22903267/what-is-tuning-in-machine-learning