Machine Learning Project: The Enron Case

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

The Enron dataset contains emails and financial data from Enron employees. Enron Corporation, was an energy, commodities, and services company that went down in December 2001 as a result of massive accounting fraud. In the years from 2000–2002 more than 1.6 million emails sent and received by Enron executives were released.

The email + financial data contains the emails themselves, metadata about the emails such as number received by and sent from each individual, and financial information including salary and stock options.

Now machine learning models can be developed that are able to identify persons of interests (POIs), individuals who were involved in fraud or criminal activities during the Enron bankruptcy, based on features on the data.

The objective of this project is to train a machine learning model that can find out what features identify a POI.

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?

To get a feel for the data I printed out the data dict a person.

```
In [169]: ### get to know the data
          print (len(data_dict.keys()))
          print (data_dict['BUY RICHARD B'])
```

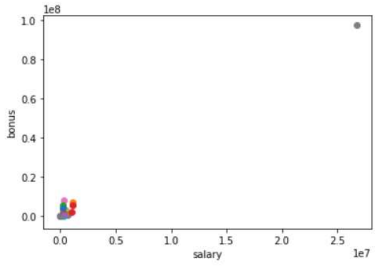
```
146
{'salary': 330546, 'to_messages': 3523, 'deferral_payments': 649584, 'total_payments': 2355702, 'loan_advances': 'NaN', 'bonus': 900000, 'email_address': 'rick.buy@enron.com', 'restricted_stock_deferred': 'NaN', 'deferred_income': -694862, 'total_stock_value': 3444470, 'expenses': 'NaN', 'from_poi_to_this_person': 156, 'exercised_stock_options': 2542813, 'from_messages': 1053, 'other': 400572, 'from_this_person_to_poi': 71, 'poi': False, 'long_term_incentive': 769862, 'shared_receipt_with_poi': 2333, 'restricted_stock': 901657, 'director_fees': 'NaN'}
```

Now that we got a feel for the dataset, lets visualize the data and check for outliers, especially when plotting salary and bonus.

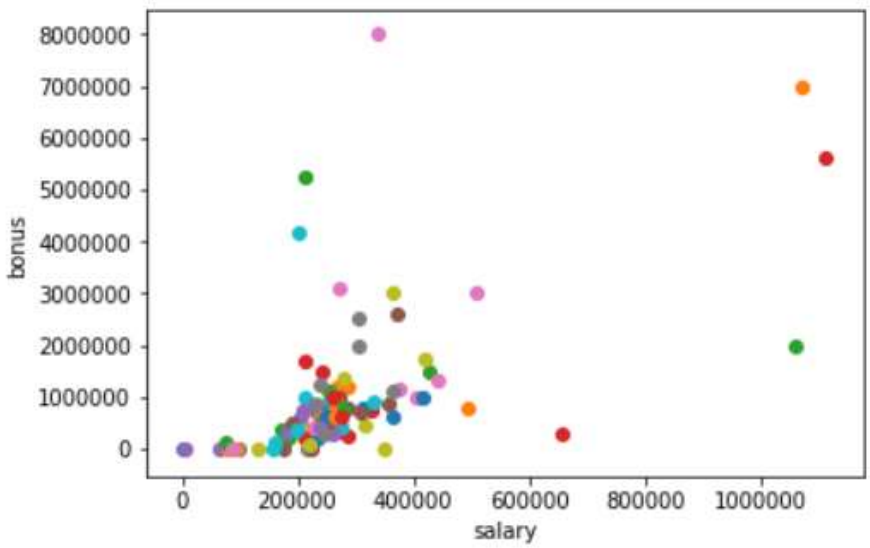
```
In [174]: features = ["salary", "bonus"]
          data = featureFormat(data_dict, features)

          for point in data:
              salary = point[0]
              bonus = point[1]
              plt.scatter( salary, bonus )

          plt.xlabel("salary")
          plt.ylabel("bonus")
          plt.show()
```



We clearly see an outlier, which seems to be the "total".




```
[('SKILLING JEFFREY K', 1111258), ('LAY KENNETH L', 1072321)]
```

After removing the outlier, the plot shows a better picture. We still have some data points with very high bonus and salary values, but this seems ok for C-level positions and possibly POIs.

Adding new features

We already have the features bonus and salary. It is quite possible that POI have relatively high bonus and salary numbers. However, lets create some new features. POIs possible wrote a lot of emails back and forth. Therefore, we define the features fraction_to_poi_email and fraction_from_poi_email.

Machine Learning starts here

In this section we will apply machine learning. Specifically, I will first split my data into training and testing data. In machine learning this is common practice to avoid overfitting.

I decided to use the k-fold cross validation method. When using the k-fold cross validation method, the training set is split into k smaller sets. A model is trained using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy). The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

Now that we have split the data, we can apply an algorithm. I first applied the decision tree classifier. Decision trees ask multiple linear questions, one after another. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

I first started with the default parameters and got an accuracy score of 83.8% after 0.003s. I ranked the feature importance. Bonus seems to be the feature of highest importance when identifying POIs (followed by salary). However, our self-build features also seem to have an influence.

```
In [202]: importances = clf.feature_importances_
          import numpy as np
          indices = np.argsort(importances)[::-1]

          for i in range(5):
              print ("{} feature {} ({})".format(i+1,features_list[i+1],importances[indices[i]]))
```

```
1 feature bonus (0.37837899543379006)
2 feature salary (0.2776255707762557)
3 feature fraction_from_poi_email (0.1804566210045662)
4 feature fraction_to_poi_email (0.16353881278538807)
5 feature shared_receipt_with_poi (0.0)
```

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

After that I tried a naive bayes classifier. This algorithm did not perform as good as the decision tree (slower). Therefore, I commented it out.

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?

Finally, I decided to do some manual tuning to my classifier to achieve better than .3 precision and recall. I chose a min samples split of 5, which is the minimum number of samples required to split an internal node. This can avoid overfitting.

5. **What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?**

Validation is basically testing your algorithm and its performance on an independent dataset that has not been used for training of the algorithm. A common mistake is to overfit your algorithm. If using PCA transformation only fit the PCA to the training features. PCA.transform however, has to be applied to training AND testing features. DO NOT refit the PCA to the test features.

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.**

My accuracy after tuning is with 86.5% slightly higher than before. Additionally, I calculated the precision and recall ratios.

```
done in 0.001s
Validating algorithm:
accuracy after tuning =  0.864864864865
precision =  0.5
recall =  0.4
```

Where precision measures the ratio of true positives (meaning a real POI is also predicted to be a POI) out of true positives plus false positives. A high precision states that nearly every time a POI shows up in my test set, I am able to identify him or her.

And recall measures the ratio of true positives in relation to true positives and false negatives. A high recall rate states that I am good at NOT falsely predicting POIs.With a precision of 0.5 and a recall of 0.4 my classifier beats the .3 both times. yay!