# Project: Machine Learning - Identify Fraud from Enron Email

## Table of Contents



## Understand the Dataset and Question

**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”]**

The goal of the project to see if machine learning can be used to identify patterns in data, in our case if Persons Of Interest (POIs) can be predicted, by looking at their employee data. This data was made available following the collapse of Enron, following on from a fraud scandal, which caused the firm to go from one of the biggest in America, to nothing over a few weeks.

I wrote some code in **poi_id.py** (also below) to count the total number of people, the POI, which then left the number of non-POI to be calculated. I also counted the amount of features for each person.

In [1]:

```
import pickle

with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)

poi_num = 0

for person in data_dict:
    if data_dict[person]['poi'] == True:                                    #count pois
        poi_num += 1

print "\nThere are", len(data_dict), "persons"
print "\nThere are", poi_num, "poi"
print "\nThere are", len(data_dict) - poi_num, "non-poi"
print "\nThere are", len(data_dict.values()[0]), "features for each person"
```

```
There are 146 persons

There are 18 poi

There are 128 non-poi

There are 21 features for each person
```

I then wrote further code to print all of the features available, along with the amount of blank NaN data fields. I counted all of the NaN fields and divided this by the total fields, in order to calculate how much of the data was missing (NaN).

In [2]:

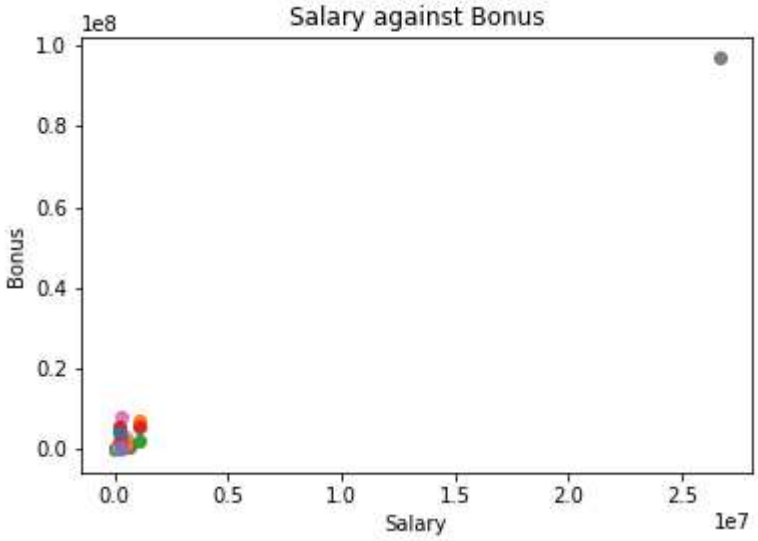
```
import numpy as np

data_sum = {}

for person in data_dict:
    for feature in data_dict[person]:                                    #set dat_di
            data_sum[feature] = 0

for person in data_dict:
    for feature in data_dict[person]:
        if data_dict[person][feature] == 'NaN':                          #count nans
            data_sum[feature] += 1

print "\nTable of features with NaN count:\n", data_sum                  #to create
print "\nTotal NaN sum:", sum(data_sum.values())
print "\nTotal number of values:", len(data_dict) * len(data_dict.values()[0])
print "\n", np.round(float(sum(data_sum.values())) /    \
                    float(len(data_dict) * len(data_dict.values()[0])) \
                         * 100, 2), '% of the data is NaN'               #divide nan
```

```
Table of features with NaN count:
{'salary': 51, 'to_messages': 60, 'deferral_payments': 107, 'total_payment
s': 21, 'loan_advances': 142, 'bonus': 64, 'email_address': 35, 'restricted_
stock_deferred': 128, 'total_stock_value': 20, 'shared_receipt_with_poi': 6
0, 'long_term_incentive': 80, 'exercised_stock_options': 44, 'from_message
s': 60, 'other': 53, 'from_poi_to_this_person': 60, 'from_this_person_to_po
i': 60, 'poi': 0, 'deferred_income': 97, 'expenses': 51, 'restricted_stock':
36, 'director_fees': 129}

Total NaN sum: 1358

Total number of values: 3066

44.29 % of the data is NaN
```

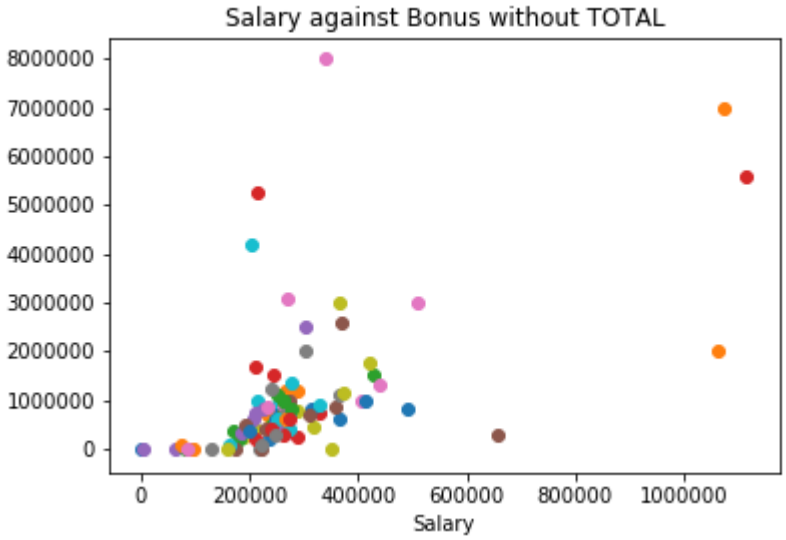
# Optimise Feature Selection / Engineering

**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”]**

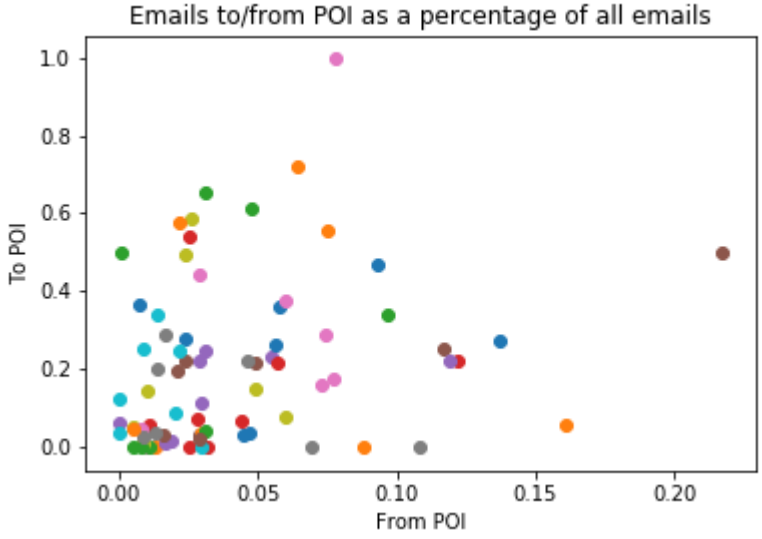
I then created a scatterplot of the bonus against the salary data, to see if there were any outliers in the data.




I then loaded the pickle into a pandas data frame and printed the list of names, which showed the name TOTAL was the oulier, so I removed this and output the scatterplot again. Note that this would mean that there were only **145 total people** and **127 non-POI**.




I then decided to calclate what percentage of emails sent or received were to and from POI, as this would give a better indication of how often someone was communicating with a POI compared to other employees. I then created a scatterplot of the emails to and from POI.



I also calculated when someone was shared on an email with a POI, which I had to divide by the total emails for that person. All 3 of these new features were left as a fraction between 0 and 1 so that they did not require scaling (instead of x100 to get %), before then adding to **my_dataset**.

I then used **SelectKBest** [1] to score my features and sort them in order, before appling **MinMaxScaler** to scale my top features along with POI. I felt the top 10 looked best(0:9), as these scored around 9 on SelecKBest, but I was also going to try 0:8 and 0:10 features to see if either side had better results.

# Pick and Tune an Algorithm

**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”]**

**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”]**

I then used **GridSearchCV** [2] to analyse my three agorithms - **GaussianNB** (Naive Bays), **SVM** (Support Vector Machine) and **KNearestNeighbors** and output the results to an excel table.

I provided a range of parameters for the SVM and KNearest Neighbors (KNN), so that they could be trialled and tuned by **GridSearchCV**, on my **SelectKBest** sorted features, in order to find the best results.

To the SVM I sent the parameters - [3] kernel - linear, rbf and sigmoid - to try all 3 kernels gamma - 1, 0.1, 0.001, 0.0001 - for how much each parameter can influence the model (high values = one point can have a greater influence on the model) C - 100, 100, 10, 0.1 - how much each value in the training set influences the model (larger C = use more training points)

To K Nearest Neighbors I sent the parameters - n_neighbors - 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 - how many points are taken to compare to (more points = more points checked for confidence, but also a greater area checked, so values further away from that being checked)

It is import to ensure that the learning algorithm is not overfit to the data, as this would show good accuracy on learning data, but poor accuracy when fed random new data. By using a grid search algorithm, each value is applied to the model to see how it effects the outcome. With every value being checked against the othres, this can increase the computational and learning time, which can be seen in my longer learning times for SVM and K nearest Neighbors. There is also the option to use a random search, which relies on randomly searching values before seeking out the best parameters. This would have the benefit of checking more parameters, but in a reduced time, at the risk of decreasing the accuracy as not all values are confirmed [4].

I output the efficiency of each algorithm measured by the accuracy, precission, recall and time taken to run.

I have read the excel tables back in below [5].

In [3]:

```
#Top 10 features (0:9)
import pandas as pd

compare = pd.read_excel("CompareAlgorithms09.xlsx", index_col=0) #print results - read back
print "\nCompare algorithms:\n", compare
```

```
Compare algorithms:
          1 Algorithm  2 Accuracy  3 Precision  4 Recall  5 Time(seconds)
0          GaussianNB    0.840703     0.425586  0.398857             0.33
1                 SVM    0.861678     0.303175  0.136224            22.69
2  K Nearest Neigbors    0.854875     0.063492  0.027438             4.51
```

In [4]:

```
#Top 11 features (0:10)
compare = pd.read_excel("CompareAlgorithms010.xlsx", index_col=0) #print results - read bac
print "\nCompare algorithms:\n", compare
```

```
Compare algorithms:
          1 Algorithm  2 Accuracy  3 Precision  4 Recall  5 Time(seconds)
0          GaussianNB    0.849026     0.386026  0.313605             0.34
1                 SVM    0.872294     0.231481  0.104403            22.75
2  K Nearest Neigbors    0.864719     0.010714  0.009354             4.44
```

In [5]:

```
#Top 9 features (0:8)
compare = pd.read_excel("CompareAlgorithms08.xlsx", index_col=0) #print results - read back
print "\nCompare algorithms:\n", compare
```

```
Compare algorithms:
          1 Algorithm  2 Accuracy  3 Precision  4 Recall  5 Time(seconds)
0          GaussianNB    0.843537     0.433116  0.401833             0.34
1                 SVM    0.856009     0.297619  0.120975            46.56
2  K Nearest Neigbors    0.859410     0.093254  0.052239            10.54
```

In [6]:

```
#Top 8 features (0:7)
compare = pd.read_excel("CompareAlgorithms07.xlsx", index_col=0) #print results - read back
print "\nCompare algorithms:\n", compare
```

```
Compare algorithms:
          1 Algorithm  2 Accuracy  3 Precision  4 Recall  5 Time(seconds)
0          GaussianNB    0.846372     0.434911  0.396967             0.32
1                 SVM    0.853741     0.343537  0.154904            21.68
2  K Nearest Neigbors    0.853175     0.162698  0.050123             4.50
```

Whilst GaussianNB has the lowest accuracy, this is only by 2 percentage points, with the precission and recall markedly higher than for the other 2 algorithms. With the longest time to process being 30 seconds, it is only an added bonus that GaussianNB is the fastest algorithm. I therefore decided to pick GaussianNB as my algorithm to be tested.

My guess that the top 10 (0:9) were the best features proved to be incorrect with my test results, whilst the top 11(0:10) performed even worse, the top 9(0:8) performed better, as a result I decided to go back and test the top 8(0:7) features aswell. Whilst the top 8 resulted in better accuracy and precision, the recall reduced, so I decided to change it back to the top 9(0:8), for later testing with tester.py.

# Validate and Evaluate

**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"]**

I used **train_test_split** to split my data into a training set and test set. I changed the **random_state** on each iteration of multiple checks, to ensure that I did not cut off all of the names at the start of the alphabet. [6]

I noticed **tester.py** uses **StratifiedShuffleSplit** to split the data, which is explained as being used due to the small size of the dataset.

By validating the algorithm against a separate dataset than that used to create the algorithm, you can verify that the algoritm is actually working out a decision based on the general patern of data points, and not the algorithm overfitting to the specific test data points. The more the algorithm is able to ignore errors in the training dataset, the better it will be able to ignore errors in the test dataset. [7]

**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 tester.py script gave the following results, with both precision and recall coming in at over 0.3 -

GaussianNB(priors=None) Accuracy: 0.85429 **Precision: 0.48716 Recall: 0.37950** F1: 0.42664 F2: 0.39705 Total predictions: 14000 True positives: 759 False positives: 799 False negatives: 1241 True negatives: 11201

**Precision** is calculated by dividing the True Positives by the sum of True Positives and False Positives. A score of 1.0 would have no False Positives, with a score of 0.5 having half True Positives and half False Positives, and shows the percentage when positively predicted results are correct.

My precision of 0.48716 means that **48.7%** of my POI predictions were correct.

**Recall** is calculated by dividing the True Positives by the sum of True Positives and False Negatives. A score of 1.0 would have no False Negatives, with a score of 0.5 having half True Positives and half False Negatives, and shows the percentage of POI who are correctly identified.

My recall of 0.37950 means that **37.9%** of POIs were correctly identified. [8]

# References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html)

[2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

[3] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html (https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html)

[4] https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models (https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models)

[5] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html)

[6] https://scikit-learn.org/0.15/modules/generated/sklearn.cross_validation.train_test_split.html (https://scikit-learn.org/0.15/modules/generated/sklearn.cross_validation.train_test_split.html)

[7] https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets (https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets)

[8] https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall)