Data Analyst Nano Degree
Identify Fraud from Enron Emails

Enron Corporation is an American energy, commodities, and utilities based company, which is well known example of willful corporate fraud and corruption. It was the biggest accounting frauds in the history. Many professionals were questioned and accused for various fraud activities and some of them were sent to jail. This fraud led to the collapse of the company including bankruptcy in 2002. During its investigation the Federal Energy Regulation Commission released a large dataset of top executives and their emails and confidential financial data. As a Data Analyst with the help of Machine Learning lets figure out a best model which suites these data to detect whether a person is involved in this scandal or not.

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 for this project is ti build a predictive machine learning model to identify person of interest who is involved in this scandal, based on the financial and email data published by the Federal Energy Regulation Commission. Here I am planning to use 4 models and figure out the best of them. Regarding the dataset, it has 146 data points and 21 features. Out of which 18 of them are tagged as POI which majorly categorized as below

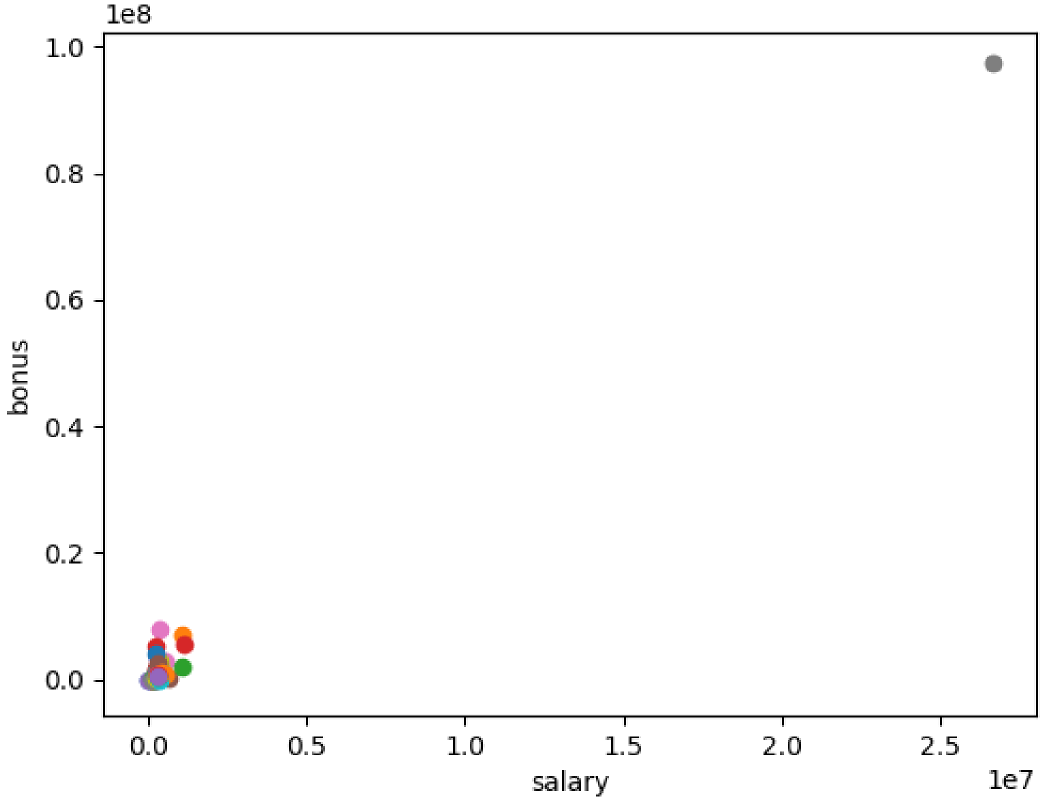
   Financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)

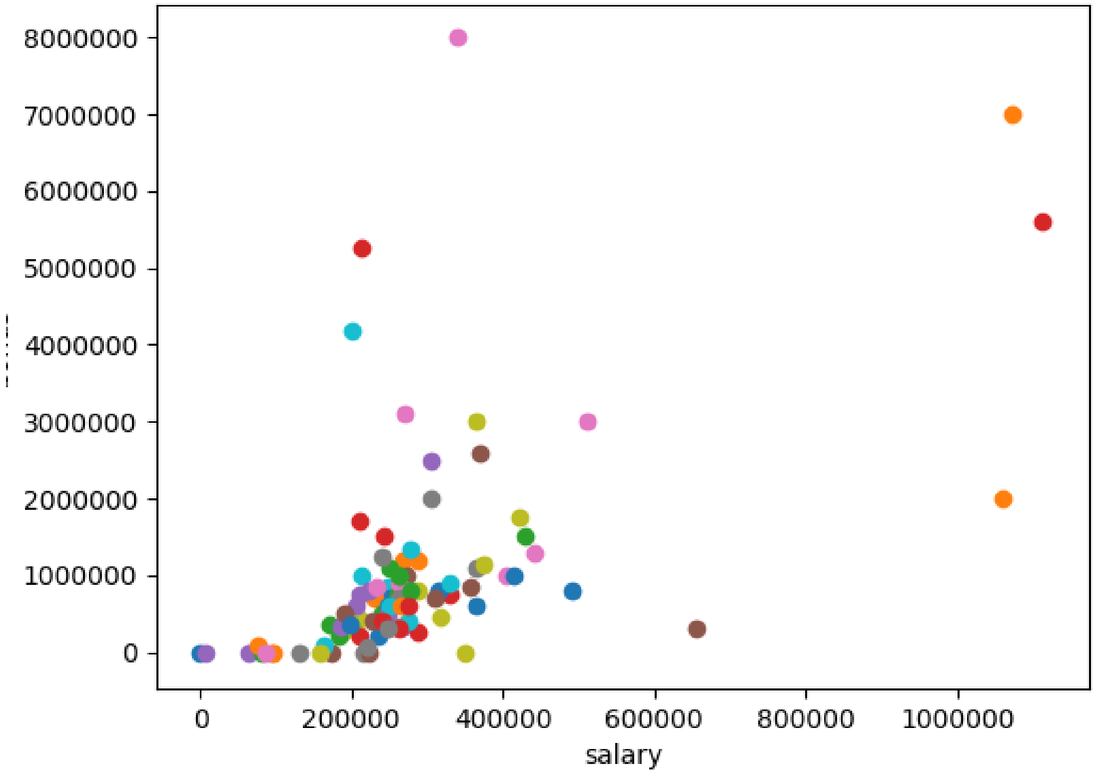
   Email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is ‘email_address’, which is a text string)

   POI label: [‘poi’] (boolean, represented as integer)

   On Analyzing the dataset in a scatter plot, I have found something interesting that there are 3 outliers present in the data. The first is “TOTAL” which was a dataset artifact, and the next is “THE TRAVEL AGENCY IN THE PARK”, which I guess is a mistake and finally “LOCKHART EUGENE E”, who has no values. By excluding them we end up having 143 data points. At last I counted all the NAN values and printed them feature wise in the output. (A sample output.txt file have been attached to visualize the command line output)

```
Exploring the Dataset
=====================
Number of features:  19
Number of datapoint: 146
Number of poi: 18
Number of non poi: 128
TOTAL

=========>Removing the outliers<=========
Number of datapoint excluding outliers: 143

NAN count
=========
poi : 0
salary : 49
deferral_payments : 105
total_payments : 20
loan_advances : 140
bonus : 62
restricted_stock_deferred : 126
deferred_income : 95
total_stock_value : 18
expenses : 49
exercised_stock_options : 42
long_term_incentive : 78
restricted_stock : 34
director_fees : 127
to_messages : 57
from_poi_to_this_person : 57
from_messages : 57
from_this_person_to_poi : 57
shared_receipt_with_poi : 57


Gaussian Naive Bayes
====================
GaussianNB Accuracy: 0.883720930233
             precision    recall  f1-score   support

        0.0       0.92      0.95      0.94        38
        1.0       0.50      0.40      0.44         5

avg / total       0.87      0.88      0.88        43
```


1e8
1.0
0.8
0.6
0.4
0.2
0.0
bonus
0.0
0.5
1.0
1.5
2.0
2.5
salary
1e7


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
200000
400000
600000
800000
1000000
salary

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”]**

Initially I tried using features except ‘email_address’ and ‘other’, because email address is a text string and other is not of any value to our analysis. In addition to that I have incorporated 3 additional features ‘Bonus-salary ratio’ which is helpful to find the ratio of Bonus and salary an employee gets, ‘from_this_persion_to_poi’ which measures how frequently a poi sends to this person and ‘from_poi0_to this person’ which measures how frequently this person sends email to poi. I also imputed some missing features of email to mean.

But The accuracy was not so good, probably due to overfitting so I tried to drop of some features and ended up using 8 features POI, Salary, bonus, deferred_income, Total_stock_value, excercised_stock_options, from_this_person_to_poi_percentage, bonus_salary_ratio which gave a pretty decent precision and recall value.

As a part of feature selection SelectKBest module from sklearn has been incorporated, and MinMaxScaler was utilized to make sure the features are weighed equally. During this process I was able to determine the best value of k for each algorithm.

```
6  features are selected and their importances and scores are displayed below:
No.  Feature                             Importance      Scores
1    bonus_salary_ratio                  0.612045834406  22.1067164085
2    exercised_stock_options             0.257817309955  16.9328653375
3    bonus                               0.130136855639  34.2129648303
4    total_stock_value                   0.0             16.8651432616
5    deferred_income                     0.0             16.3662859037
6    salary                              0.0             17.7678544529
```

Then I created a function which performs a pipeline process for classification. This function was designed to model, fit and predict the data. Later which I also designed a basic model of Gaussian Naïve Bayes to compare it as well.

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 initially planned on analysing the accuracy of the model on various algorithms such as Gaussian Naïve Bayes; Decision Tree; Support Vector Machine; K Nearest Neighbour. I have displayed the command line accuracy for all the algorithms used

```
Gaussian Naive Bayes
====================
GaussianNB Accuracy: 0.886363636364
             precision    recall  f1-score   support

        0.0       0.93      0.95      0.94        39
        1.0       0.50      0.40      0.44         5

avg / total       0.88      0.89      0.88        44
```

```
 K-Nearest Neighbor
==================

Best parameters are:  {'knn__leaf_size': 30, 'feature_selection__k': 2, 'knn__algorithm': 'auto', 'knn__n_neighbors': 1}

             precision    recall  f1-score   support

        0.0       0.87      0.89      0.88        38
        1.0       0.00      0.00      0.00         5

avg / total       0.77      0.79      0.78        43


Decision Tree Classifier
========================

Best parameters are:  {'dtc__max_depth': 2, 'dtc__criterion': 'gini', 'dtc__min_samples_split': 2, 'dtc__class_weight': 'balanced', 'feature_selection__k': 3, 'dtc__random_state': 42}

             precision    recall  f1-score   support

        0.0       0.94      0.79      0.86        38
        1.0       0.27      0.60      0.37         5

avg / total       0.86      0.77      0.80        43


Support Vector Machine Classifier
=================================

Best parameters are:  {'svc__gamma': 0.001, 'feature_selection__k': 3, 'svc__kernel': 'rbf', 'svc__C': 1000}

             precision    recall  f1-score   support

        0.0       0.88      0.97      0.93        38
        1.0       0.00      0.00      0.00         5

avg / total       0.78      0.86      0.82        43

(p2) C:\Users\Bharath Kumar\Desktop\DataAnalyst\Chapter 9\ud120-projects\final_project>
```

Out of all the algorithms the best accuracy is given by Gaussian Naïve Bayes with an accuracy of 88% with a Precision of 88% and recall of 89%. The best result for K nearest neighbour is an accuracy of 78% with a precision of 77% and a recall of 79%. The best result for Decision Tree is an accuracy of 80% with a precision of 86% and recall of 77%. Similarly Support Vector Machine has an accuracy of 82% with a precision of 78% and a recall of 86%.

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"]**

Parameter tuning is an essential process in machine learning to improve a models efficiency and accuracy. This helps the model to perform the algorithm at its best. For instance in K-means clustering algorithm it is important to specify the number of clusters in the dataset, which helps in reducing the over fitting and under fitting issues.

Since I ended up using Gaussian Naïve Bayes, which doesn't have parameters to tune. But to demonstrate the tuning process I used the GridSearchCV and with the help of pipelining process I manipulated the best parameters for all the other algorithms I used and listed in the above-mentioned figure.

Best Parameters: (This accuracy is calculated from Tester.py)

GaussianNB(priors=None)

Accuracy: 0.86787 Precision: 0.50577 Recall: 0.44326 F1: 0.44326 F2: 0.41266

Total predictions: 15000 True positives: 789 False positives: 771 False negatives: 1211 True negatives: 12229

K Nearest Neighbour :{'knn_leaf_size':30, 'feature_selection_k':2, 'knn_algorithm':'auto', 'knn__n_neighbors':1}

Accuracy: 0.77893 Precision: 0.17586 Recall: 0.17850 F1: 0.17717 F2: 0.17797

Total predictions: 15000 True positives: 357 False positives: 1673 False negatives: 1643 True negatives: 11327

Decision Tree Classifier: {'dtc_max_depth':2, 'dtc__criterion':'gini', 'dti__min_samples_split':2, 'dtc__class_witght':'balanced', 'feature_selection__k':3, 'dtc__random_state':42}

Accuracy: 0.76367 Precision: 0.29569 Recall: 0.55900 F1: 0.38678 F2: 0.47449

Total predictions: 15000 True positives: 1118 False positives: 2663 False negatives: 882 True negatives: 10337

Support Vector Machine: {‘svc__gamma’:0.001, ‘feature_selection__k’:3, ‘svc__kernel’:’rbf’, ‘svc__C’:1000}

Accuracy: 0.81680 Precision: 0.53571 Recall: 0.00750 F1: 0.01479 F2: 0.00934

Total predictions: 15000 True positives: 15 False positives: 13 False negatives: 1985 True negatives: 12987

It certainly is a tough call to choose the best model as all have its pros and cons but I prefer Gaussian Naïve Bayes as it has performed best score in all the categories.

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”]**

The validation is the most important part of building the model is where we check the accuracy of our model and tune it to analyse the performance. Cross validation is a statistical method to evaluate and compare the algorithm by splitting the data into training and testing section, so that it can be trained using the training data and can be tested using the testing data. One of the issued in machine learning is overfitting, when an algorithm is performed great on training set but performs poor on test set it is called as overfitting. It is necessary to split the data s training set and testing set, so that both have equal weight.

I have used StratifiedShuffleSplit from sklearns cross_validation module to randomly choose the test and train data and validate them in the analysis. In this project, we are dealing with a small and imbalanced dataset.

As seen throughout the lectures, working with smaller datasets is hard and in order to make validation models robust, we often go with k-fold cross-validation, which is what we do when we use a shuffle split.

But well, in this project, a stratified shuffle split is of choice. When dealing with small imbalanced datasets, it is possible that some folds contain almost none (or even none!) instances of the minority class. The idea behind stratification is to keep the percentage of the target class as close as possible to the one we have in the complete dataset.

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”]**

For this project I mainly preferred precision and recall as the main evaluation metrics as both Decision Tree and Gaussian Naïve Bayes were close enough which created a tie and I used accuracy as a secondary evaluation metrics which acted as a tie breaker to choose Gaussian Naïve Bayes to go for. Their performance were already discussed earlier in the document and the final results of Gaussian Naïve Bayes are Accuracy: 0.8897 Precision: 0.87   Recall: 0.88

Precision is the ability of the model to tag as positive sample as positive and vice versa, where as Recall is the ability of the model to find all the positive samples. This gives a conclusion that if my model predicts a person as POI it is 87% accurate that this person is actually a POI and my model can identify/recall a POI correctly about 88% with an accuracy of 88.97%