POI IDENTIFICATION FROM ENRON DATASET USING ML
SOHAN SAMANTA

INTRODUCTION

Enron corporation was an American Energy company based in Houston, Texas. It was formed in 1985 by Kenneth Lay after merging with Houston Natural gas and InterNorth. By the use of accounting loop holes and poor financial data, some members of the company hid millions of dollars in debt from failed deals and projects. The scandal eventually led to the bankruptcy of Enron Corporation. Many executives at Enron were indicted with a variety of charges and some were even later imprisoned.

In this project, we will try to play detective and build a program that will be able to identify persons of interest from email and financial data made public by the authorities. By person of interest we refer to any employee or separate individual who was either involved directly or indirectly in the Enron fraud or had some knowledge about it but chose to hide it and were later indicted for it by the government.

This project flow can be divided into 4 major parts:

1. Data Set
2. Feature Selection
3. Picking and tuning an algorithm
4. Validation and Evaluation

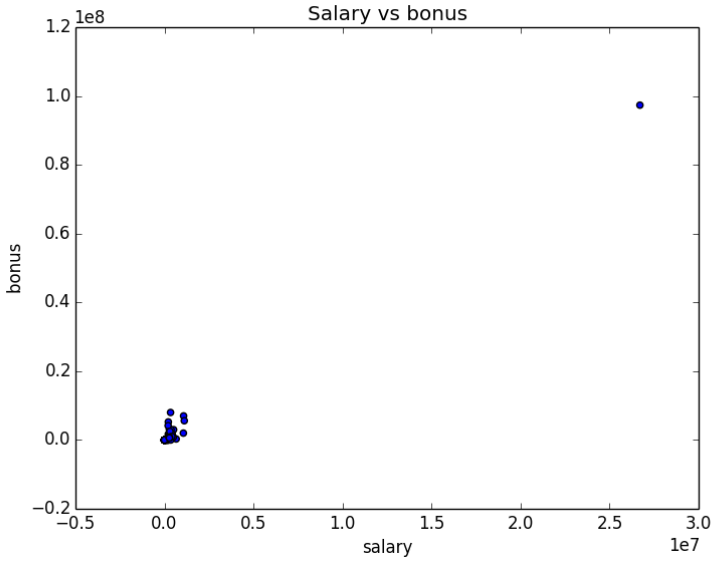
THE ENRON DATA SET

The data set is comprised of two major subdivisions of information – financial information and email data.

The data is stored in a pickle file in a dictionary of dictionary format. The first thing I am interested in after downloading is the structure and length of the data. Since each item corresponds to one person who could be a potential POI (person of interest), the length essentially gives us a number of potential suspects.
An example of the way the data is stored is provided below:

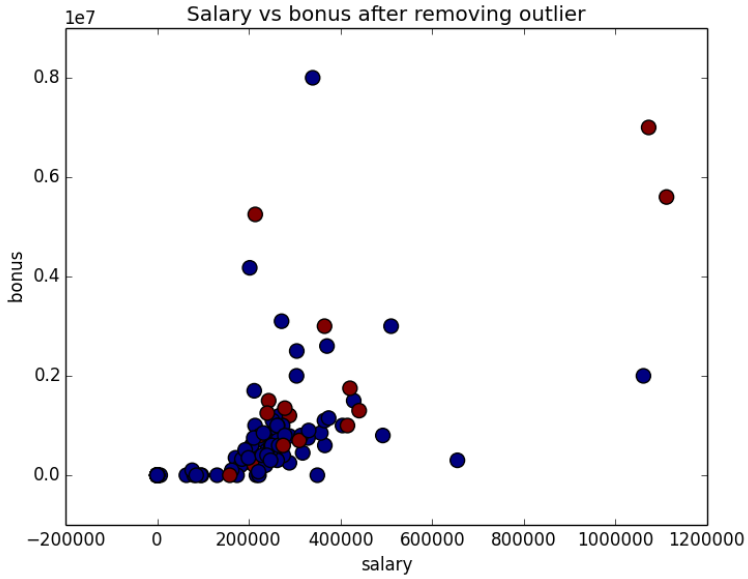
{'METTS MARK': {'salary': 365788, 'to_messages': 807, 'deferral_payments': 'NaN', …….},
'BAXTER JOHN C': {'salary': 267102, 'to_messages': …….}, , , }

The length of the dictionary is 146. Hence we have 146 people to consider. Whenever dealing with a large set of data, one should always be wary of outliers. The simplest way to check for outliers in this case would be to plot a simple bonus vs salary scatter plot. Any outlier with abnormal value would be at once visible.



So, we do find one outlier. A little search inside the data dictionary reveals that this outlier is the **'TOTAL'** value at the end of the list. We go through the list of names once manually, and find one wrong entry which does not belong to any person. It is named **'THE TRAVEL AGENCY IN THE PARK'**. We also check if there are any entries with all values missing. And sure enough, we find an entry of a person called **'LOCKHART EUGENE E'**.

So we promptly remove all outliers and re-plot the data.



Now, the data looks more realistic, with a few large values. I have also colored the persons of interest with red. This gives us an idea of how the POIs and non POIs are distributed.
We also take this time to look at how many poi and non-poi person are present in the list. We find the following:

1. **Number of poi = 18**
2. **Number of non-poi = 125**

FEATURE SELECTION

Feature selection is as important as the selection of the algorithm that these features will be used within. While good features can improve the training and prediction of an algorithm, not all algorithms will give the same results with the same set of features.

But before we start working with features, let us look at some of the important properties of the features:

1. There are a total of 21 features.
2. The features present are:
   salary, to_messages, deferral_payments, total_payments, exercised_stock_options, bonus, restricted_stock, shared_receipt_with_poi, restricted_stock_deferred, total_stock_value, expenses, loan_advances, from_messages, other, from_this_person_to_poi, poi, director_fees, deferred_income, long_term_incentive, email_address, from_poi_to_this_person
3. The features and their count of missing values are as given below:

Feature	No of missing values	Feature	No of missing values
Loan_advances	140	Director_fees	127
Restricted_stock_deferred	126	Deferral_payments	105
Deferred_income	95	long_term_incentive	78
bonus	62	To_messages	57
shared_receipt_with_poi	57	From_messages	57
From_poi_to_this_person	57	From_this_person_to_poi	57
Other	52	Salary	49
expenses	49	Exercised_stock	34
Email_address	32	Total_payments	20
Total_stock_value	18		

4. Some features have a high number of missing or 0 values. The top three features with a very high number of null values are:
   - Loan Advances (140 null values)
   - Director Fees (127 null values)
   - Restricted Stock Deferred (126 null values)

FEATURE SCALING:

Feature Scaling is a method used to standardize the range of independent values or features. Since the range of raw data can vary significantly, it is better to normalize them for use in machine learning applications. This is because a lot of the classifiers use Euclidean distances to measure the distance between two points. If the distances are large for a particular feature, then the distance between the two points will be primarily governed by this feature.

In this project, as we will find later on, that the best performance is given by Decision Trees. Since for Decision Trees feature scaling do not provide much added benefits, we have not included any sort of Feature scaling.

In this step, what we have essentially endeavored to do is lay a ground work for our features and algorithms. We start by selecting a wide range of features, which we intuitively feel may be of some

importance and then use a set of algorithms to test them. This step is essentially a bit of trial and error, as we don't know what parameters would give us good results, nor do we have any idea as to which algorithm will be best suited for this data set.

INITIAL SET OF ORIGINAL FEATURES:

Let us try a broad range of classifiers with a preliminary set of features and test out our algorithm. This will form the base line for future modifications of the algorithm. It needs to be noted that we do not include the three features with high number of missing values we find above. The feature list for our preliminary analysis consists of the following:

```
'poi','salary','deferral_payments','total_payments','loan_advances','bonus',
'deferred_income','total_stock_value','expenses','exercised_stock_options',
'other','long_term_incentive','restricted_stock','director_fees','to_messages',
'from_poi_to_this_person','from_messages','from_this_person_to_poi',
'shared_receipt_with_poi'
```

The performance is as follows:

SL No	Algorithm	Accuracy	Precision	Recall
1	Decision Tree	80.91	23.57	19.25
2	Logistic Regression	78.41	17.88	17.25
3	Ada Boost	83.92	33.59	21.1
4	Random Forest	85.78	38.54	10.95
5	Linear SVC	75.53	13.2	14.95

NEW FEATURES:

In the above results although the accuracy is not bad, but the precision and recall values are not too good. I would like to include some added non linearity within the feature set, as well as some new ratios between some related features. The new features are as follows:

```
ratio_of_to_poi_messages = from_this_person_to_poi / from_messages
ratio_of_from_poi_messages = from_poi_to_this_person / to_messages
total_poi_messages = from_this_person_to_poi / from_poi_to_this_person
salary_square = salary ^ 2
bonus_square = bonus ^ 2
total_stock_value_square = total_stock_value ^ 2
incentive_cube = long_term_incentive ^ 3
bonus_salary_ratio = bonus / salary
```

While we use most of the features from the original data set, with the added new features we end up having the following 27 distinct features listed below:

```
'poi', 'salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus',
'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options',
'other', 'long_term_incentive', 'restricted_stock', 'director_fees', 'to_messages',
'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi',
'shared_receipt_with_poi', 'ratio_of_to_poi_messages', 'ratio_of_from_poi_messages',
'total_poi_messages', 'salary_square', 'bonus_square', 'total_stock_value_square',
'incentive_cube', 'bonus_salary_ratio'
```

This results in the following performance:

SL No	Algorithm	Accuracy	Precision	Recall
1	Decision Tree	83.3	34.07	26.85
2	Linear SVC	49.82	11.43	40.95
3	Ada Boost	84.62	39.85	30.05
4	Random Forest	85.9	41.95	14.35
5	Logistic Regression	48.8	11.18	40.9

The general trend after including the new features can be summarized as follows:

For Decision Tree and Ada Boost, the performance has increased over all. But for the rest of the classifiers, we find an increase in performance of only one of the performance parameter at the cost of the others. Since, at the ground level these parameters all depend upon the predictions which can be broken down to being true positive, true negative, false positive or false negatives, the inclusion of the new features seems to be assisting some of the above while detrimentally affecting the rest. These 4 predictions mentioned above, will be discussed in detail later on, for the time being it is important to remember that adding of the new features has been beneficial as well as detrimental.

INTRODUCING PCA:

Now, for any algorithm, 28 features are an overkill. Thus, we use the Principal Component Analysis to bring the number of features down, and thus remedy the issue of some really bad performance parameters. The help of the pipeline functionality of Python is taken to add the PCA feature to each and every classifier. Since we are also using Grid Search in tandem with PCA and pipeline, we initially set a range of parameter values for the classifiers and the PCA to test each of their performances and then choose the best ones based upon their performances.

The performance can be summarized by the performance metrics we are most interested in, and is shown below:

SL No	Algorithm	No of components	Accuracy	Precision	Recall
1	Decision Tree	10	82.21	33.56	34.1
2	Linear SVC	5	79.56	27.09	31.5
3	Ada Boost	15	84.1	30.92	15.6
4	Random Forest	20	85.56	35.52	10.15
5	Logistic Regression	20	82.4	32.86	30.4

We note, that the performance of the classifiers has significantly improved due to the use of the PCA in reducing the number of features. It is also to be noted that best classifier, which is the Decision Tree uses far less number of features at 10 than the total features present.

A question may arise in the mind of the readers as to **how were the number of components for the PCA chosen?** There are a few ways that help us decide this, like plotting the cumulative sum of Eigen values in descending order to look at the fraction of variance retained with decrease in Eigen values. But here we simply use the Grid Search and trial and error to get the best performance. The use of grid search is explained in the next section.

For this project, we set out with a target of around 30% precision and recall and a high value of accuracy. From the above table we find that both Decision Tree and Logistic Regression provide pretty good results.

The question is can we do better? Also, we have used PCA to combine our features. Should we also include a few original features to our list? Would that be any better?

PICKING AND TUNING AN ALGORTIHM

Picking the algorithm:
We have already picked the algorithms we wanted to look at and tuned them accordingly. But we have not yet explained our reason for choosing the algorithms we have chosen or the tuning process. In this section we aim to remedy that.

The choosing of a classifier depends upon a basic knowledge of the algorithm and the data we want to classify. This process is more of an intuition developed upon existing knowledge of how the classifiers perform. For example, we have excluded one of the simplest and widely used algorithms: The Naïve Bayes. This is an algorithm that performs well with a small amount of data which are independent. But for cases that have a lot of correlated features the performance can actually degrade. Thus we felt that this classifier will not be a good fit for this project.
The ones that are chosen are based upon similar thought flow.

Parameter Tuning:

Tuning refers, essentially, to selecting the best parameters of an algorithm to optimize its performance given certain pre-existing conditions or constraints. Hyper-parameters are generally able to take a wide range of values, and are thus in general left up to the user to specify.
Tuning an algorithm thus involves going through different values of hyper parameters of an algorithm to find the perfect match that gives the performance we want.

In context of machine learning, the goal is to set the parameters in such a way as to allow the learning to be completed in the best possible way. The goal could be to increase the performance based upon specific performance parameters.

Python makes this process easier. Python provides a function called **Grid Search.**
In grid search we can provide a list of values for the hyper-parameters and the grid search does an exhaustive search by considering all parameter combinations to select the best set of parameter values.
When we say best set of parameters, in reference to Grid Search, we actually refer to the score function that it uses to determine this. For scoring, Grid Search uses an 'accuracy score' for classifications and an 'r2' score for regressions.

VALIDATION:

Validation refers to the process of evaluating a trained model by using a testing data set. In general, the testing data set is a portion of the same data set from which the training data sets were also picked, to train the model.

Its importance lies in the ability of judging a trained algorithm's generalization abilities from its performance by using validation.

In our project we have used Gird search which uses a cross validation evaluation method that provides the added benefit of avoiding over fitting.
If an algorithm was made to learn the parameters on a set of data and then it was tested on the same set of data, it would probably give a 100% accuracy. But the caveat lies in the fact that, if it were given data to classify that it had never seen before, its performance would drop by a great margin. This situation is called over fitting. To avoid this, the common practice is to hold out a certain percentage of the data for testing purposes and the remaining data is used for the training phase.

But while evaluating different settings of hyper-parameters, there is still a chance of over fitting as the parameters can be tweaked until the estimator performs optimally. Essentially this causes the knowledge of the test to leak into the model. The solution is to create another test set called validation set from the training set. But this causes the data set for the training set to decrease a lot. To avoid this, the k fold cross validation was developed. Here, the training set is split into *k* smaller sets and k-1 sets are used for training and the last set for validating. The performance measure reported by *k*-fold cross-validation is then the average of the values computed in the loop.

Coming back to our problem, the following table provides a list of the parameters that we tuned for the different algorithms that we used in our project:

Sl. No	Algorithm	Parameters and their tuning list of values

1	Decision Tree	`"min_samples_split": [2,5,10],`
		`"criterion": ["gini", "entropy"],`
		`"random_state": [47]`

2	Logistic Regression	`"C": [0.05, 0.5, 1, 10, 10**2,10**5,10**10,10**20],`
		`"tol": [10**-1, 10**-5, 10**-10],`
		`"random_state": [47]}`

3	Ada Boost	`"n_estimators": [20, 30, 40, 50],`
		`"algorithm": ["SAMME", "SAMME.R"],`
		`"random_state": [47]`

4	Random Forest	`"n_estimators": [20, 30, 40, 50],`
		`"criterion": ["gini", "entropy"],`
		`"min_samples_split": [2, 5, 10],`
		`"random_state": [47]}`

5	Linear SVC	`"C": [1, 5, 10, 15, 20],`
		` "tol": [0.1, 0.01, 0.001, 0.0001],`
		` "dual": [False],`
		` "random_state" : [47]`

More tuning:
Moving forward we have two useful piece of information.

1. We have identified the best algorithms so far
2. We also know that the parameters for these algorithms were chosen based upon their accuracy and r2 values.

Decision Tree and Logistic Regression seem to give better results than the other classifiers we have selected. And even then Decision Tree clearly is the winner here. So, we will remove the others and go forward with only Decision Tree, to see if we are able to make any further improvements.

In our trial and error of changing the parameter values to improve the performance, most parameters do not seem to cause improvement, on the contrary they degrade the performance. However, changing the random_state to 46 does improve the performance a little.

It is to be noted that practical decision trees are based on heuristic algorithms, where returning a globally optimal decision tree depends upon training multiple trees in an ensemble and learner where the features and samples are randomly sampled. We set a seed value to make these random selections more consistent so that we get the same performance every time.

The new performance results are as given below:

SL No	Algorithm	Accuracy	Precision	Recall
1	Decision Tree	82.7	35.12	35.05

PERFORMANCE WITHOUT PCA:

Just out of curiosity, I wanted to remove PCA and only use some of the original features and the added features in the Decision Tree.

An important function in python that we can use for looking at the relative importance between features is the **"feature_importances_"** function. We use this to decrease our features list further and settle on the following list:

```
'poi', 'salary', 'bonus', 'total_stock_value', 'from_poi_to_this_person',
'from_this_person_to_poi', 'shared_receipt_with_poi', 'ratio_of_to_poi_messages',
'salary_square', 'incentive_cube'
```

Surprisingly enough, we get quite good results:

SL No	Algorithm	Accuracy	Precision	Recall
1	Decision Tree	81.64	35.62	35.48

We do not use PCA any more as the number of features are now under a manageable limit, and using PCA at this point would only decrease our performance further.

NOTE: Manually playing with the features can probably give better results in any one of the performance parameters at the cost of another. For example, setting the feature list to the following:

```
'poi', 'ratio_of_to_poi_messages', 'ratio_of_from_poi_messages','bonus_salary_ratio',
'salary'
```

provides a precision and recall of 38%, but decreases the accuracy to 77%.

However, manually selecting features is an exhaustive process, especially in cases with a lot more features and more complex data. In this case, I was simply fortunate to be manually able to find the correct set of features. Even then their performance is comparable to the one we found using Grid Search and PCA.

PROJECT EVALUATION

Based on the above analysis, we finally settle on a Decision Tree with the following parameters:

```
Pipeline(steps=[('pca', PCA(copy = True, iterated_power = 'auto', n_components = 10,
                    random_state = 47, svd_solver = 'auto', tol = 0.0, whiten = False)),
```

('DecisionTreeClassifier', DecisionTreeClassifier(class_weight = None, criterion = 'gini',
max_depth = None, max_features = None, max_leaf_nodes = None, min_impurity_split = 1e-07,
min_samples_leaf = 1, min_samples_split = 10, min_weight_fraction_leaf = 0.0,
presort = False, random_state = 46, splitter = 'best'))])

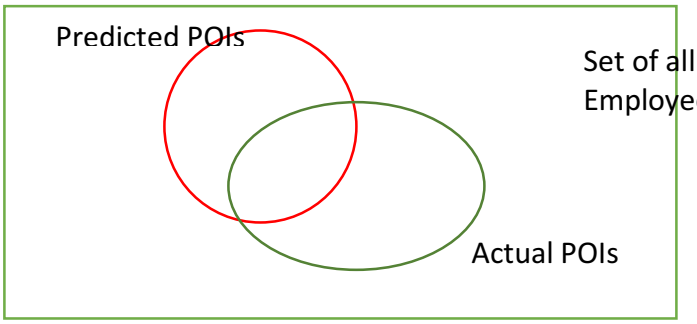
The above parameters were decided upon after running a list of parameters through Grid Search using pipeline, and the features were minimized by the use pf PCA.

The full performance of the above algorithm is as follows:

Accuracy: 0.82707 Precision: 0.35120 Recall: 0.35050 F1: 0.35085 F2: 0.35064
Total predictions: 15000 True positives: 701 False positives: 1295 False negatives: 1299
True negatives: 11705

PERFORMANCE RUBRIC:

We have based our performance upon 3 parameters: Accuracy, Recall and Precision. So what do they mean?
The concept can be easily understood by a Venn-diagram.



True Positive: This refers to the common area between the two regions above. These are the correct POI predictions made.
False Positive: The area inside the red region, but not overlapping with the green region. These are all the non-POI members predicted as POIs.
False-negatives: The area inside the green region, but not overlapping with the red region. These are all the real POIs, not recognized by the predictions.

One the basis of the above four parameters, we can define accuracy, precision and recall.

Total_predictions = true_negatives + false_negatives + false_positives + true_positives
Accuracy = (true_positives + true_negatives) / total_predictions
Precision = true_positives / (true_positives + false_positives)
Recall = true_positives / (true_positives + false_negatives)

In context to this project, the recall and precision can be defined in the following manner:
Precision score gives the confidence level of a classifiers evaluation of a person to be a POI. It measures how certain we can be that a prediction of POI is actually correct.
Recall score provides the fraction of POIs that the classifier has been able to identify from all the existing POIs in the data set.

From the above discussion, we get a better view of why only accuracy is not a singular parameter for measuring of performance. And how the values of precision and recall hold importance.

CONCLUSION

In this project we have given an equal weightage to the precision and recall values and not just based our decision on Accuracy. The reason behind this is that accuracy only gives us an idea as to how many predictions are correct. But recall and precision gives us an even deeper understanding based on the correct predictions of not only the true positives but also the false negatives as percentages of false positives and true negatives.

A high value of accuracy, combined with decent values of precision and recall makes our algorithm truly robust.