MACHINE LEARNING PROJECT

Introduction:

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

Our goal in this project is it to create persons of interest identifier. This will identify the people who are involved in the Enron fraud. To assist in our work, a hand-generated list of persons of interest is provided which will act as labels to train our identifier so that we can use it to identify other persons of interest.

Dataset Overview:

We have 146 data points of Enron employees.

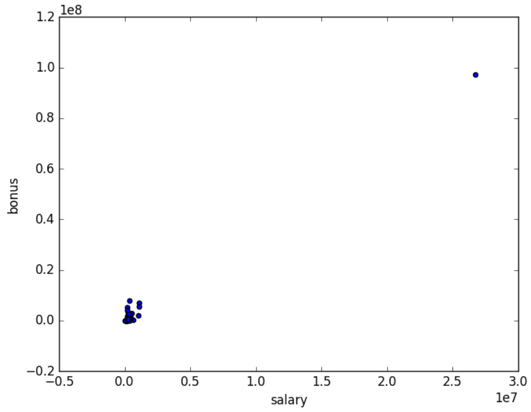
Each data point contains following 20 features. Where 'poi' which will act as our label can be either True or False depending upon the feature.

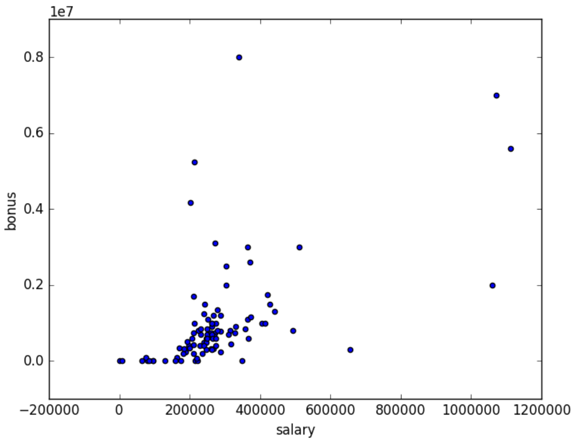
'salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercised_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi', 'restricted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances', 'from_messages', 'other', 'from_this_person_to_poi', **'poi'**, 'director_fees',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

We also see some missing values while getting an overview of the data. In financial data we replaced NaN with 0 and in email data we replaced nan with median. As the dataset is small, we preferred not to remove the data points as whole.

Outlier Removal:

From the scatter plot of the salary and the bonus we can clearly see one outlier. After examining the data we see the point name is ‘TOTAL’. We removed this point manually.





We can clearly see after removal the plot that scatter points are much more visible.

New features:

Adding new features or modifying features is a important part of machine learning. This help getting the best out of the existing features. One way to look at them is to create fractions of data. I created the following 4 features as part of my project.

1. fraction_from_poi
2. fraction_to_poi
3. expenses_per_salary
4. fraction_emails_with_poi
5. bonus_per_salary
6. deferral_per_total
7. deferral_per_loan

In our first feature we take the fraction of messages sent from person of interest with the total messages sent. In second feature we create a fraction of massages sent to person of interest with total messages sent.

- With little imagination we can think person who has done some fraud has sent or received maximum number of massages from poi. Similarly fractions of other features are also created and tested.

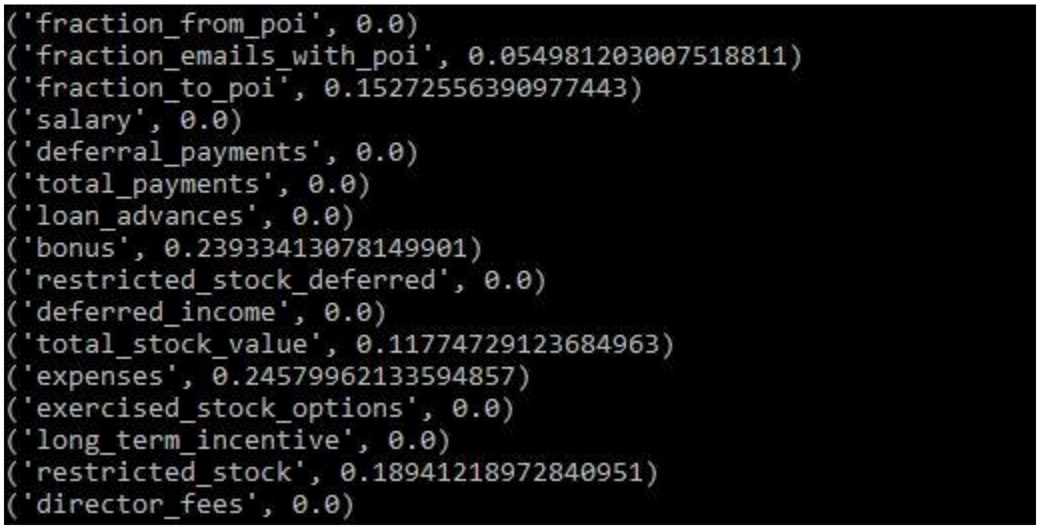
Feature Scaling:

We noticed that many features in our dataset is skewed. We used mixmaxscaler to normalize the features. This scales and translates each feature individually such that it is in the given range on the training set, i.e. between zero and one. We noticed significant change in the importance of features after implementing it.

Feature Selection:

Feature selection is one of the most important part of machine learning. How effective is our identifier depends only on how effective our our features. With too many features we generate a problem of over fitting. A model that has been overfitted has poor predictive performance, as it overreacts to minor fluctuations in the training data.

Initially I used all the features (except email features as we have created new features with them) and tried Gaussian Bayes and Decision Tree algorithm. I did not achieve the desired result with both. I decided to check the importance of features with decision tree and started removing the low importance features.



```
('fraction_from_poi', 0.0)
('fraction_emails_with_poi', 0.054981203007518811)
('fraction_to_poi', 0.15272556390977443)
('salary', 0.0)
('deferral_payments', 0.0)
('total_payments', 0.0)
('loan_advances', 0.0)
('bonus', 0.23933413078149901)
('restricted_stock_deferred', 0.0)
('deferred_income', 0.0)
('total_stock_value', 0.11774729123684963)
('expenses', 0.24579962133594857)
('exercised_stock_options', 0.0)
('long_term_incentive', 0.0)
('restricted_stock', 0.18941218972840951)
('director_fees', 0.0)
```

I selected the following high importance feature.

- fraction_emails_with_poi
- fraction_to_poi
- bonus
- total_stock_value
- expenses
- restricted_stock

Algorithm Selection:

In supervised classification, we can either have discrete or continuous output. In continuous supervised algorithm we use regression and find best fit line. But in our project our output will be discrete labels and we will make decision boundary. I tried the following supervised algorithm in my identifier.

Tested Algorithms(Before Feature Selection):

	Accuracy	Precision	Recall
Naïve Bayes	0.72693	0.23197	0.45350
Decision Tree	0.80827	0.26947	0.25600

Tested Algorithms (After Feature Selection):

	Accuracy	Precision	Recall
Naïve Bayes	0.86687	0.50111	0.34000
Decision Tree	0.82687	0.34231	0.32400

Both our algorithms are giving better than 0.3 precision and recall. We decided to tune Decision Tree Algorithm to check if we can achieve better result than Naïve Bayes.

Tune your classifier:

Algorithm parameter tuning is an important step for improving algorithm performance. It is sometimes called Hyperparameter optimization where the algorithm parameters are referred to as hyperparameters. The performance of the selected hyper-parameters and trained model is measured on a dedicated evaluation set that was not used during the model selection step. Thus tuning the classifier is an significant aspect of machine learning. We'll use GridSearch to to tune out algorithm.

GridSearchCV implements a "fit" method and a "predict" method like any classifier except that the parameters of the classifier used to predict is optimized by cross-validation. Our accuracy increased marginally (Accuracy: 0.85607) but overall result is not as good as GaussianNB. So we select GaussianNB for our final poi identifier.

Validation:

Learning the parameters of a prediction function and testing it on the same data is a mistake, This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set**.

However, by partitioning the available data into sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called cross-validation (CV for short). In the basic approach, called k-fold CV, the training set is split into k smaller sets. Because of the small size of the dataset, we used stratified **shuffle split cross validation**. This cross-validation object is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class.

Evaluation and conclusion:

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

In other words The precision can be interpreted as the likelihood that a person who is identified as a POI is actually a true POI. Recall measures how likely it is that identifier will flag a POI in the test set. We got a precision of 50.1% and recall of 34% which is greater than 0.3, Hence our goal is achieved.

Accuracy	0.86687
Precision	0.50111
Recall	0.34000