Identification of Persons Of Interest (POI) from the Enron Dataset

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"]

In this work, I am using data regarding 143 of the previous employees of Enron containing the financial and email related information of the employees. Of the 143 employees, 18 of them are labelled as persons of interests (POI). POI are those people who were suspected of their involvement in the scandal.

The objective of this work is to build a classifier using machine learning algorithms, that uses these financial and email related information of the employees to predict which of the employees are POIs and which are non-POIs. The given dataset contains 143 rows (one for each employee) and 21 columns. Of these, 14 column list the financial data of the employees, and the rest tell us details about the emails sent and received by the employees. There is one column named 'POI' indicating whether the person is a POI or not. This is our target variable. All other columns except the one for email address are numeric

The typical steps of data preprocessing - such as missing value imputation, outlier detection, feature engineering, dimension reduction etc., are all performed in this work, after which, a bunch of classifiers are created and validated using appropriate metrics.

<u>Missing Value imputation</u>:

1) Every single column in the dataset has missing values, with the exception of the POI column. However, looking at the financial records that came along with the dataset, it seems the NaN values in the financial features imply non-existent, rather than not available. Hence, it would be a good idea to impute the missing values in the columns related to financial features with 0 (zero).
2) Imputing missing values for Email related features is tricky. There is no clear method to be followed here. Hence, I decide to do this: Group the dataset according to POI or non POI status. Impute missing values with the mean value of the attribute of the relevant group (POI or non-POI) which the employee with the missing data belong to

Handling Outliers

First, I made a pairplot from the seaborn package, that plotted each numeric variable against the other. This plot revealed the presence of outliers in the data, in several of the columns. Some of the outlying points were POIs whereas the others were non-POIs. Additionally, outliers are, by definition, those points that lie beyond 1.5 times the interquartile range either above the 75th and or below 25th percentile of a variable. Using this definition, I found the total outliers for each column in the dataset, as well as for each rows in the dataset. It was found that 108 out of the 143 rows had at least 1 outlying values. Hence eliminating rows with outliers in them was ruled out as an option of treating outliers. Instead, I removed few rows with the maximum number of outliers in them, that were non-POIs. Specifically, the rows corresponding to employees Frevert Mark A, Lavorato John, Baxter John, Whalley Lawrence and Headicke Mark were removed, since they had more than 8 or more outlying values in their attributes and were also non-POIs. [Please look at the accompanying Jupyter notebook Enron_POI_detection to see how I implemented this]

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"]

The selection of features was a bit tricky. First of all, I'll explain few features I engineered. Then I'll explain the rationale behind feature selection

Feature Engineering

First I created 4 new features, related to financial attributes of the records. They are:

1) ratio of bonus to total_payments: This indicates the bonus as a fraction of total payments received by the employee. This feature might help us spot POIs , since a POI might have received more than usual bonuses as part of some fraudulent activities he or she committed
2) ratio of total_stock to total_payments: indicating how much of the employees' assets are held as stock, as a fraction of total payments. A person who knows about the on-going fraudulent activities in the company might prefer to have less stock, at least when it becomes apparent that the company will crash sooner or later.
3) ratio of restricted_stock to total_stock_value: indicating how much of the employee's stock are restricted, as a fraction of total stock value. A POI might prefer to have less of their stock restricted.
4) ratio of exercised_stock_options to total_stock_value: indicating how much of the stock owned by the employee have been exercised. A POI might exercise more of their stock options than a non-POI

I also created 2 new email related features. The rationale for creating these features was already explained by Udacity.

1) fraction_to_poi: The fraction of emails sent by the employee sent, that were to a POI
2) fraction_from_poi: The fraction of emails received by the employee, that t were from a POI

Feature Selection

Having cleaned the dataset, and engineered new features, I employed the SelectKBest algorithm of sklearn to identify feature importances. I'll list below the top 15 important features as suggested by the SelectKBest algorithm

	feature	score
12	bonus_to_tot_pay	23.260487
9	deferred_income	19.994727
3	bonus	19.654948
11	from_poi_to_this_person	11.859121
0	salary	11.451845
4	shared_receipt_with_poi	11.095520
5	total_stock_value	10.551725
2	exercised_stock_options	10.375338
13	fraction_from_poi	7.452914
1	total_payments	6.260435
14	fraction_to_poi	6.257078
6	expenses	5.141294
10	long_term_incentive	4.818849
7	from_this_person_to_poi	4.616861
8	director_fees	1.945991

Interestingly, the newly created feature bonus_to_tot_pay is found to be the most important by the algorithm

However, later when I built the classifier, I tried using these top 15 features, as well as all the 25 original features (without removing any, with the exception of email address). It turned out that all the classifiers except decision trees, yielded better accuracy, precision and recall, when I used all the 25 features instead of removing any. Hence, for the final classifier, I used all the 25 features.

Feature Scaling

All features were scaled using the scale function from sklearn.preprocessing. This was required, since the different columns in our data, varied widely in their magnitudes. For better performance of our classifiers, we need a similar range of values for all our features. Scaling helps us achieve that

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 ended up using the logistic regression model, with all the 25 features (original as well as engineered, except the email ID) for my final model. This model, when tuned properly using GridSearch, was found to be giving the best values of precision, recall and accuracy. Other than logistic regression, I tried a decision tree classifier, a random forest classifier and a support vector machine.

Note that, for all the classifiers mentioned above, I tried out using all the 25 features, as well as the top 15 features, as returned by the SelectKBest algorithm. I have tabulated the results I obtained in the table below.

[Note: All the values reported below were obtained from a manual 10-fold cross validation of the algorithms on the dataset, by using the optimal parameters suggested by the GridSearchCV algorithm]

Classifier Performance while using the top 15 features:

Classifier	Accuracy	Precision	Recall
Decision Tree	0.915	0.5	0.45
Random Forest	0.915	0.5	0.4
Logistic Regression	0.812	0.345	0.55
Support Vector Machine	0.748	0.35	0.85

Classifier Performance while using all 25 features:

Classifier	Accuracy	Precision	Recall
Decision Tree	0.885	0.5	0.4
Random Forest	0.908	0.45	0.35
Logistic Regression	**0.906**	**0.61**	**0.8**
Support Vector Machine	0.883	0.58	0.8

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"]

All the algorithms I used have a set of hyperparameters that need to be tuned for best results. The performance of a classifier can be improved a lot by finding the right set of hyperparameters. A not-so-well tuned algorithm can result in less than optimal accuracy, precision and recall.

For finding the right set of hyperparameters, I used the GridSearchCV method of sklearn. In this method, we provide a set of parameters through which the algorithm should iterate over, trying out each possible combination of the hyperparameters. The algorithm will internally do a cross validation. The combination that yields the best results (in terms of the best value of a scoring

parameter we specify - could be accuracy/precision/recall/f1 score for the cross validation) will be returned.

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"]

Validation is a process of measuring how well our classifier generalises, i.e., performs over a set of data it has not seen before. A classic mistake that one can make while validating is measuring the performance of the classifier on the same data it was used for training. This is wrong, since it does not tell us anything about how well our model generalises to cases it has not seen till now. Besides, the performance will obviously be high if we measure it on the training data, because the training data itself was used for building the model.

In this work, I built a subroutine to do k-fold cross validation. The code can be found in the attached Jupyter notebook. It uses the StratifiedKFold method in scikitlearn, to generate multiple splits of train-test samples. The training samples would then be used to train the model, and the performance would be measured on the testing sample. This process would then be repeated k-times, and the average value of the performance metrics would be reported. For this work, I looked into accuracy, precision and recall. I paid more attention to precision and recall, over accuracy, since we are dealing with an imbalanced dataset.

[Note: StratifiedKFold takes care of the imbalance in the class labels in the data, and generates training and test samples representative of their proportions in the original data]

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"]

Two performance metrics I paid particular attention to are precision and recall. Since we have an imbalanced dataset, accuracy is not a good metric to look for. Precision and Recall are more sensible metrics to optimise for this kind of problems.

Precision indicates how many of the true positives predicted by the algorithm are actually truly positive, whereas recall indicates, of all the total positives in the dataset, how many were correctly identified by the algorithm. Ideally we want high precision and high recall. However, it is the case that an increase in precision is

accompanied by a decrease in recall, and vice versa. This is why we use f1-score, the optimised value of which help us attain a balance between precision and recall.

A classifier with high precision, (and hence low recall) will yield very less false positives. This implies that the classifier will most likely not wrongly identify a non POI as POI. However, to do this, it might actually fail to detect some POIs, just so as to avoid making mistakes. A classifier with high recall (and hence low precision), might catch all the POIs in the data, but in the process might implicate a lot of non-POIs are POIs as well.

In our problem, we are concerned about identifying POIs. In this particular problem, it is not clear which of the above two situations are more desirable. A POI is just a person of interest; identifying someone wrongly as POI might waste the investigating team's time following a false lead. This is what would happen if the classifier has high recall. However, it does not mean the person will be convicted. The alternative is to avoid following false leads, and investigate only those people we are sure as POI, at the cost of missing out few POIs. This would happened with a classifier with higher precision.

Personally, I feel the former is preferable, i.e., a classifier with higher Recall. But I am not sure optimising purely for recall is a good idea. Hence, I chose to go for f1 score to be optimised in GridSearchCV when finding the optimum hyperparameters. f1 score is the harmonic mean of precision and recall. f1 score tries to find a trade-off between precision and recall. Note that, a classifier that optimises for precision or recall might suffer a drop in accuracy. Optimising for f1-score gives us a balance between the 3 metrics.

I have already reported the average cross validation accuracy, precision and recall of my best classifier.