Project Overview

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, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives.

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"]

ANSWER

In this project, I am required to play the detective, build a person of interest (POI) identifier (classifier or algorithm) based on financial and email data made public as a result of the Enron scandal. A person of interest is anyone that is indicted, settled without admitting guilt or testified in exchange for immunity. The Enron dataset constitute the largest email and financial details of real people available in the open. ML will accomplish the goal of this project by building classifiers to detect if there are pattern within the emails and financial details of people who were persons of interest in the fraud case and to identify those patterns. This dataset consist of 146 data points, 18 of which were POIs while 128 were Non-POIs. There are also 21 features or attributes per data point i.e. per person. The dataset contain several missing values as detected by displayMissingFeatures() function in poi_id.py.

The following outliers were detected and removed:

- data_dict.pop('TOTAL',0),
- data_dict.pop('LOCKHART EUGENE E',0)

I removed the 'TOTAL' line because it's just a spreadsheet tally line for the other 145 data records. I suspected as outliers data points for which at least 85% of the features are either 0 or NaN and the following function, suspectDataPointForWhichMoreThanXpercentFeaturesIsNaN(), suspected the data keys below:

'WHALEY DAVID A', 'WROBEL BRUCE', 'LOCKHART EUGENE E', 'THE TRAVEL AGENCY IN THE PARK', 'GRAMM WENDY L'

Upon manual inspection, I resolve to consider only 'LOCKHART EUGENE E' as an outlier because I noticed that every single one of its features were either NaN or 0, hence, can be safely remove.

QUESTION 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 importance 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"]

ANSWER

The features I finally used were the following:

['poi', 'total_payments', 'restricted_stock_deferred', 'shared_receipt_with_poi', 'loan_advances', 'from_messages', 'other', 'from_this_person_to_poi', 'from_this_person_to_poi_ratio', 'from_poi_to_this_person', 'from_poi_to_this_person_ratio']

The features were selected by using SelectKBest imported from sklearn.feature_selection module. Yes, I did feature scaling, it is an important preprocessing techniques because it normalizes the dataset – yielding balanced features. Imbalance features create situations where one feature dominate and overshadow the effect of others, thereby compromising the trustworthiness of the experiment. Feature scaling helps re-scale the features so that they span comparable ranges, usually between zero and one. Moreover, without scaling, in my experiments algorithm like SVM was heavily impaired in time complexity. The two features I engineered were 'from_poi_to_this_person_ratio', and

'from_this_person_to_poi_ratio'. The rationale behind this is that if the ratio of messages to and from a person to POI is high, it is likely that such a person is also POI. I tested these two features alone with Decision Tree algorithm, the result was an accuracy value of 0.8333, a precision value of 0.416667 and recall value of 0.75.

The feature scores obtained from using SelectKBest are tabulated below:

FEATURES	SCORE
'salary'	0.0070633544604325306
'expenses'	0.0099556176426057411
'bonus'	0.014937147645228303
'restricted_stock'	0.018132057318754231
'long_term_incentive'	0.018204033224926044
'total_stock_value'	0.018746663578841416
'exercised_stock_options'	0.019546502219679095
'director_fees'	0.027660947474219523
'deferred_income'	0.041945576030443986
'deferral_payments'	0.050215658109185067
'other'	0.064171600407838245
'total_payments'	0.074996648782159989
'loan_advances'	0.59734317518921487
'restricted_stock_deferred'	2.72109773197006
'from_poi_to_this_person'	inf
'from_messages'	inf
'from_this_person_to_poi'	inf
'shared_receipt_with_poi'	inf
'from_poi_to_this_person_ratio'	inf
'from_this_person_to_poi_ratio'	inf

Table 1: Feature Scores

By considering SelectKBest(*score_func=<function f_classif>*, *k=10*) from sklearn documentation, in my experiment I use the default of k = 10 and f_classif as the score function. The value k = 10 is an educated experimental guess but while *chi2, f_classif* and *mutual_info_classif* are all suitable for classification problems, *chi2* only works with positive values, *mutual_info_classif* yields a constant score value for all the features but *f_ classif* produces reasonable scores as tabulated above.

QUESTION 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"]

ANSWER

Decision Tree is the algorithm I finally used. Other algorithms I tried were Naïve Bayes, Grid Search (with SVM), Support Vector Machine (SVM) and Adaboost. These algorithms produce differing accuracy, precision and recall when supplied with the same training and testing data. Table 2 details the result of these experiments.

		Accuracy	**Precision**	**Recall**
Decision Tree	E1-FSet-1	0.761904761905	0.197916666667	0.125
	E2-FSet-2	0.833333333333	0.309523809524	0.25
	E3-FSet-3	0.756756756757	0.0957207207207	0.333333333333
Naïve Bayes	E1-FSet-1	0.333333333333	0.203956582633	0.875
	E2-FSet-2	0.333333333333	0.203956582633	0.875
	E3-FSet-3	0.27027027027	0.1	1.0
Support Vector Machine (SVM)	E1-FSet-1	0.809523809524	0.190476190476	0.0
	E2-FSet-2	0.809523809524	0.190476190476	0.0
	E3-FSet-3	0.918918918919	0.0810810810811	0.0
Grid Search (with SVM)	E1-FSet-1	0.809523809524	0.190476190476	0.0
	E2-FSet-2	0.809523809524	0.190476190476	0.0
	E3-FSet-3	0.918918918919	0.0810810810811	0.0
Adaboost	E1-FSet-1	0.785714285714	0.190476190476	0.0
	E2-FSet-2	0.785714285714	0.208333333333	0.125
	E3-FSet-3	0.756756756757	0.0957207207207	0.333333333333

Table 2: Experimental Result

As tabulated above, three experiments (E1, E2 and E3) were performed for each algorithm using three different feature sets FSet-1 , FSet-2 and FSet-3 where for example E1-FSet-1 represents the first experiment using the first set of features while E2-FSet-2 and E3-FSet-3; the second and third experiments using the second and third sets of features for each algorithm respectively.

Where:

FSet-1 = ['poi', 'deferral_payments', 'total_payments', 'restricted_stock_deferred', 'shared_receipt_with_poi', 'loan_advances', 'from_messages', 'other', 'from_this_person_to_poi', 'deferred_income', 'from_poi_to_this_person'],

FSet-2 = ['poi', 'total_payments', 'restricted_stock_deferred', 'shared_receipt_with_poi', 'loan_advances', 'from_messages', 'other', 'from_this_person_to_poi', 'from_this_person_to_poi_ratio', 'from_poi_to_this_person', 'from_poi_to_this_person_ratio']

and

FSet-3 = Features ['poi', 'deferral_payments', 'restricted_stock_deferred', 'shared_receipt_with_poi', 'deferred_income']

Note:

i. FSet-1 was selected by SelectKBest from the original feature list.
ii. FSet-2 was selected by SelectKBest from the original feature list plus the two new features I engineered i.e. 'from_poi_to_this_person_ratio' and 'from_this_person_to_poi_ratio'.
iii. FSet-3 was derived by manual but intelligent one by one removal of feature from FSet-1, testing the remaining features in each algorithm, iterating the process for several times until the final set of FSet-2 was decided.

Considering the above result, it is obvious that both Support Vector Machine (SVM) and Grid Search (with SVM) produce exactly the same result. I considered Decision Tree of better performance and more suited to analyzing the skewed Enron dataset because in my judgment, it produces the best combination of accuracy, precision and recall for this dataset; justification for my assertion will be provided in later questions.

Question 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"].

ANSWER

In machine learning, a hyper-parameter is a parameter whose value is set before the learning process begins. The process of setting the values of these parameters for a learning algorithm either by using model defaults, grid search, and random search or Bayesian optimization is called **tuning**. The tunable parameters of the models can greatly affect their accuracy, without doing this well, the model will not optimally solve the machine learning problem. The more tuned the parameters of an algorithm, the more biased the algorithm will behave relative to the training and test dataset. This approach can be effective, but it can also lead to a very fragile models that over fit the test data, hence, performing poorly in practice.

My final algorithm was decision tree, studying the arrays of tunable parameters for decision; I found no strategic advantage or reason to modify the default, so I used the default parameters without further tuning. However, for other algorithm like SVM, I experimented with grid search using the following parameters set:

{'kernel': ('linear', 'rbf'), 'C': [1, 10]}

With adaboost, the parameters I tuned were :

n_estimators=1000, random_state=202, learning_rate=1.0, algorithm="SAMME.R"

Generally, the approaches I used for tuning the parameters divides into two: an automatic approach using GridSearchCV and manual intelligent search approach.

Question 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"]

ANSWER

Validation is the process of testing to see that your algorithm is doing according to specification what it is intended to do. This process determines if the algorithm will fit well outside the dataset provided i.e. it generalizes. A classis mistake in validation is overfitting. Overfitting occurs when:

i. An algorithm is trained and tested on the same dataset: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data.

ii. An algorithm is so tuned that it that it's sensitive to small fluctuations in the training set, resulting to high variance error. The algorithm model random noise in the training data, rather than the intended outputs (overfitting).

How did you validate your analysis ?

I shared available data into training, testing, and validation sets. The validation strategies I used are listed below. Note that the main difference is the number of split being done per time:

- **Holdout**: n groups = 1; Data is divided into two frames: training data frames and validation data frame. In any of the methods, one sample goes into the training set while the other into the test set. So the samples do not overlap, if they do, we just can't trust our validation.
- Sklearn.model_selection.shufflesplit
- **K-fold**: K-fold can be viewed as a repeated holdout, because we split our data into k parts and iterate through them, using every part as a validation set only one. After this procedure, we average scores over these k-folds. It is important to understand the difference between k-fold and usual holdout or bits of k-times. While it is possible to average scores they receive after k different holdouts. In this case, some samples may never get invalidation, while others can be there multiple times. On the other side, the core idea of k-fold is that we want to use every sample for validation only one. This is method is a good choice when we have have a minimum amount of data, and we can get either a sufficiently big difference in quality, or different optimal parameters between folds.

- **Leave-one-out**: A special case of K-fold where k is the number of samples in our data. This means it will iterate through every sample in our data. Each time using k-1 object is a train subset and object left is a test subject. This method can be helpful when we have too little data and just enough model to train.
- Stratification preserves the same target distribution over different folds.

Question 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"]

ANSWER

The two evaluations metric I used were **precision** and **recall**. While precision defines the ratio of true positives relative to all positives (true positive + false positive), i.e.

$$Precision = \frac{True\ Positive}{(\ True\ Positive + False\ positive\ )}$$

Recall (or sensitivity of the algorithm) on the other hand defines the ratio of true positives relative to true positives and false negatives, i.e.

$$Recall = \frac{True\ Positives}{(\ True\ Positives + False\ Negatives\ )}$$

Average performance for precision and recall for all the experimented algorithms is detailed in *experimental result* as shown in *Table 2* above.

INTERPRETATIONS OF METRICS

Precision and recall are both important metrics if reasonably applied to the situation under consideration. Precision ensures that innocent people are excluded from the list of POIs at the expense of also excluding some POIs. Recall on the other hand ensures more POIs are secured at the expense of also including some innocent people in the list of POIs. Because of the skewed or unbalance nature of available dataset and the sensitive nature of the situation under consideration (i.e. criminal investigation), precision will not be the best metric to use because we can easily miss important person of interest from being identified. However, recall will be a better metric to use; a

high recall value would ensure that truly culpable individuals were flagged as POIs and would be subject to thorough human-led investigation.

CONCLUSION

Considering my experimental result in table 2, in all the five algorithms that were experimented, Naïve Bayes gives the highest and best recall value; that is, the highest probability of getting a person of interest. However, I decided to use Decision Tree as my major and final algorithm because, it gives the most stable average result for accuracy, precision and recall for all the three experiments and feature set. Although, Naive Bayes algorithm gives the highest reall, it also seems heavily flawed with bias relative to Decision Tree algorithm.