Project 4 – Identifying Fraud from Enron Email

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”]

Enron was an energy and energy services company best known for its on-line energy trading division. Enron collapsed in 2001 after the exposure of accounting fraud. The fraud involved reporting earnings on investments and operations before taking in any receipts. When that practice started generating losses the company, abetted by the auditing firm Arthur Andersen, began hiding the losses in special purpose entities.

The goal of this project is to identify persons connected to the fraud from a set of financial and email features. There are only 145 persons in the database and 18 of them are labeled as persons of interest. There is an additional database row that totals the values associated with the features in all the other rows.

The data set is a Python dictionary indexed by the names of the Enron employees. The value of each dictionary entry is a dictionary. The dictionary value has features 'poi', 'email_address', and of financial and email features.

Feature	**Description**
poi	0/1 for non-poi/poi. Used as a label in supervised learning.
salary	Periodic cash compensation subject to an employment contract.
deferral_payments	A loan with a delayed first payment.
total_payments	salary+bonus+deferral_payments+expenses+other-| deferred_income|+long_term_incentive+loan_advances
loan_advances	A loan of future salary? Most of Kenneth Lay's compensation in 2001.
bonus	Cash compensation not subject to an employment contract.
restricted_stock_deferred	Unvested restricted stock. (A negative number).
deferred_income	Cash payment with a vesting period. (A negative number).
total_stock_value	restricted_stock+exercised_stock_options-|

Feature	Description
	restricted_stock_deferred|
expenses	Expenses reimbursed.
exercised_stock_options	A stock option is a grant of the right to purchase stock at a below market price subject to a vesting period. Exercised stock options are those options that have been purchased.
other	A money grant of some kind.
long_term_incentive	A performance-based cash award to an executive.
restricted_stock	Grant of stock at a nominal price subject to a vesting period.
director_fees	Fee paid to the board of directors.
to_messages	Number of emails received.
from_poi_to_this_person	Number of emails received from a person of interest.
from_messages	Number of emails sent.
from_this_person_to_poi	Number of emails sent to a person of interest.
shared_receipt_with_poi	Number of emails received that are also received by (another) person of interest.
email_address	Email address.

There are many possible ways in which these features could be predictors that a person is a person of interest.

Enron persons of interest accomplished huge insider trades once their company began unraveling. This might show up as large exercised_stock_options. The opposite is true of restricted_stock_deferred amounts. These look to be grants of unvested stock and should correlate negatively with an individual being a person of interest in the year 2001; they seem to be new grants, possibly to newer employees.

Large compensation packages are another signal that an individual knew about Enron financials or belonged to senior management, so salary, bonus, expenses (reimbursements), long_term_incentive, or even loan_advances might be a signal that an individual is a person of interest. The 'other' column which also records income transfers.

Email from or to another person of interest might mean that the email owner is also a person of interest. So, from_poi_to_this_person, from_this_person_to_poi, and shared_receipt_with_poi are candidates for being predictors that an Enron employee is a person of interest.

There is one outlier in the dataset corresponding to the row that totals the values associated with the features in all the other rows. This row has name 'TOTAL'. It can be removed by invoking pop("TOTAL",0) on the data dictionary.

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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

The features that predicted persons of interest the best, in some sense of the word best, were 'exercised_stock_options', 'bonus', 'salary', 'long_term_incentive', and 'deferred_income'. Scaling the variables decreased the valuation metrics, so feature scaling was not used.

The first step in feature selection was to start with a list of all of the financial and email features and remove 'total_stock_value' and 'total_payments' from consideration. The analysis which produced the table of features (question 1) discovered that these were sums of other financial features. Their presence softens the effect of adding or dropping features from consideration by the machine learning algorithm.

NaNs assigned to 'total_payments' and 'total_stock_value' were replaced using the formulas from the above table.

NaNs assigned to the remaining financial variables were replaced with 0 because there were many examples for which this was appropriate (judged by using the total_payments formula from the above table).

The variables 'frac_salary', 'frac_bonus', 'frac_expenses', 'frac_deferral_payments', 'frac_loan_advances', 'frac_deferred_income', 'frac_other', and 'frac_long_term_incentive' were obtained from the corresponding financial variable by dividing by 'total_payments'.

The variables frac_restricted_stock_deferred', 'frac_deferred_income', and 'frac_exercised_stock_options' were obtained from the corresponding financial variable by dividing by 'total_stock_value'.

The variables 'fraction_from_poi' and 'frac_shared_receipt_with_poi' were obtained from 'from_poi_to_this_person' and 'shared_receipt_with_poi', respectively, by dividing by the value of 'to_messages'.

The variable 'fraction_to_poi' was obtained from 'from_this_person_to_poi' by dividing by the value of

'from_messages'.

The sklearn SelectPercentile with percentile set to 25 was used to find the best variable combination using the raw variables. The variables were ranked in decreasing order. This was used to construct a sequence of longer and longer feature_lists that were fed into the full algorithm. The best feature list found with the GaussianNB algorithm was ['exercised_stock_options', 'salary', 'bonus', 'deferred_income', 'long_term_incentive'] with Accuracy: 0.86667, Precision: 0.50000, and Recall: 0.396000. Turning off scaling of the variables improved the score to Accuracy: 0.86214, Precision: 0.52273, and Recall: 0.40250.

The sklearn SelectPercentile with percentile set to 25 was used to find the best variable combination using the scaled, synthetic variables (plus 'director_fees'). The variables were ranked in decreasing order. This was used to construct a sequence of longer and longer feature_lists that were fed into the full algorithm. The best feature list found with the GaussianNB algorithm was ['frac_bonus', 'fraction_to_poi', 'frac_long_term_incentive', 'frac_shared_receipt_with_poi'] with Accuracy: 0.81117, Precision: 0.30030, and Recall: 0.10000.

The sklearn SelectPercentile with percentile set to 25 was used to find the best variable combination using the raw financial variables and the scaled, synthetic email variables (plus 'director fees'). Scaling was applied before the call to SelectPercentile to all variables. One of the best feature lists found with the GaussianNB algorithm (and scaling of the variables) was ['exercised_stock_options', 'bonus', 'salary', 'fraction_to_poi', 'deferred_income'] with Accuracy: 0.86220, Precision: 0.47713, Recall: 0.34950. An alternative was ['poi', 'exercised_stock_options', 'bonus', 'salary', 'fraction_to_poi', 'deferred_income', 'long_term_incentive'] with Accuracy: 0.85613, Precision: 0.45117, and Recall: 0.3650. Leaving the feature scaling out was attempted but it led to worse results.

3. What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: "pick an algorithm"]

The algorithm I ended up using is Naive Bayes.

A Support Vector Machine was tried. For the SVM runs, the data was scaled. It was assumed that the features discovered using naïve Bayes were the best choice with SVM. The best result obtained used C=2.438, gamma=3.047. Validation, using independent data, reported Accuracy: 0.87247, Precision: 0.57895, and Recall: 0.15950.

The SVM algorithm was combined with a PCA pre-processor. The SVM parameters were the same as those used without the PCA pre-processor. The Accuracy, Precision, and Recall were identical to those found for SVM alone.

A DecisionTree algorithm, using the sklearn object DecisionTreeClassifier, was used. The results were

Accuracy: 0.77814, Precision: 0.24886, and Recall: 0.27400. The decision tree classifier reports out the importance of the various imput variables. In this case

Variable	**SelectPercentile score**	**DecisionTree importance**
exercise_stock_options	1.0	0.443
salary	0.782	0.548
bonus	0.866	0.307
deferred_income	0.529	0.000
long_term_incentive	0.473	0.195

which means that it did not use deferred income. With that sole exception, the ranking of the input variables was the same as that SelectPercentile and a naïve Bayes algorithm reports.

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? (Some algorithms don't 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 if you used, say, a decision tree classifier). [relevant rubric item: "tune the algorithm"]

GaussianNB does not have parameters to tune.

The parameters to the SVM algorithm (object SVC) were essential. Without finding good assignments for gamma and C, the fit found no persons of interest at all. This caused the scoring in tester.py to abort with a divide by zero error. The settings for gamma and C were found using GridSearchCV. GridSearchCV has parameters that specify a range for each parameter (gamma and C) that it is attempting to find good values for. It also takes a cross validation generator; I used StratifiedShuffleSplit.

To get results, I tuned the parameters to SVC for a sequence of cases. Each attempt required examining the C and gamma found using GridSearchCV to see if the found value was in the interior of the grid that was specified.

Features	**C**	**gamma**	**Accuracy**	**Precision**	**Recall**
exercised_stock_options	1	2.683	0.840	1.000	0.120
exercised_stock_options salary	51.79	2.683	0.868	0.917	0.161
exercised_stock_options salary	51.79	1	0.861	0.703	0.167

bonus					
exercised_stock_options salary bonus deferred_income	2.683	2.683	0.873	0.581	0.170
exercised_stock_options salary bonus deferred_income long_term_incentive	2.683	2.683	0.872	0.572	0.162

Once the SVM was working with the full feature list used in the naïve Bayes classifier, the grid search algorithm was tightened up by increasing the number of grid points to 32x32. The best result obtained used C=2.438, gamma=3.047. Validation, using independent data, reported Accuracy: 0.87247, Precision: 0.57895, and Recall: 0.15950. A 64x64 grid performed no better.

The code for finding C and gamma looks like:

```
C_range = np.logspace(0, 3, 32)
gamma_range = np.logspace(0, 3, 32)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(labels, n_iter=5, test_size=0.2, random_state=42)
grid = GridSearchCV(SVC(kernel='rbf'), param_grid=param_grid, cv=cv)
grid.fit(features, labels)
return grid.best_params_
```

The majority of the manual tuning involved changing the lines that set C_range and gamma_range.

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 item: "validation strategy"]

Validation is checking your learning algorithm on data that is independent of the data that the learning algorithm trains on. The classic mistake is to validate using training data. Machine learning algorithms designed using this classic anti-methodology will probably work on their training data but fail to generalize when applied to new and never-before-seen data. The training data may or may not be over-fit; without validation it is not possible to tell.

The strategy used here is to split the data using StratifiedShuffleSplit into a set of random folds. The code for this is in tester.py. The number of random folds tried is 1000, a remarkably high number for a data set with only 145 rows. For each fold, the machine learning algorithm fits and predicts on different data sets. Each fold yields a count of true positive, true negative, false positive, and false negative predictions. These prediction counts are accumulated.

The accumulated predictions are used calculate accuracy, precision, and recall. Precision and recall were used as the validation statistics used in this study. I paid some attention to the accuracy as well. The code from tester.py reads:

```
total_predictions = true_negatives + false_negatives + false_positives + true_positives
accuracy = 1.0*(true_positives + true_negatives)/total_predictions
precision = 1.0*true_positives/(true_positives+false_positives)
recall = 1.0*true_positives/(true_positives+false_negatives)
```

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"]

The best naïve Bayes classifier run reported an average accuracy, precision, and recall of 0.86667, 0.50000, and 0.39600, respectively. This study used precision and recall to perform its validation. The precision of 0.5 and recall of 0.396 are better than the minimum of 0.3 and 0.3.

Accuracy is the fraction of correct predictions.

Precision is the ratio of true positives to the sum of true positives and false_positives. This is the fraction of the persons identified as poi's that are in fact poi's.

Recall is the ratio of true positives to the sum of true positives and false_negatives. This is the fraction of poi's that were actually detected.