Identify Fraud from Enron Email

https://github.com/udacity/ud120-projects/tree/master/final_project (https://github.com/udacity/ud120-projects/tree/master/final_project)

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, 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.

In this project, we will attempt to identify "persons of interest" based on financial and email data made public as a result of the Enron scandal. We will build a "persons of interest" classifier to predict individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Resources

- **poi_id.py** : Starter code for the POI identifier, you will write your analysis here. You will also submit a version of this file for your evaluator to verify your algorithm and results.
- **final_project_dataset.pkl** : The dataset for the project, more details below.
- **tester.py** : When you turn in your analysis for evaluation by Udacity, you will submit the algorithm, dataset and list of features that you use (these are created automatically in poi_id.py). The evaluator will then use this code to test your result, to make sure we see performance that's similar to what you report. You don't need to do anything with this code, but we provide it for transparency and for your reference.

Dataset

As preprocessing to this project, we've combined the Enron email and financial data into a dictionary, where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The features in the data fall into three major types, namely financial features, email features and POI labels.

- **financial features**:
  `['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees']`
  (all units are in US dollars)
- **email features**:
  `['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']`
  (units are generally number of emails messages; notable exception is ‘email_address’, which is a text string)
- **POI label**:
  `[‘poi’]`
  (boolean, represented as integer)

Enron Submission Free-Response Questions (https://docs.google.com/document/d/1NDgi1PrNJP7WTbfSUuRUnz8yzs5nGVTSzpO7oeNTEWA/pub?embedded=true)

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"]

Background:

Enron was an energy company that at one time was one of the largest companies in the United States before going bankrupt due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

Using this financial and email data, we will build a classifier to identify whether or not an Enron employee was a "persons of interest" (POI) — individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. The classifier will be trained using features we know about employees from the public record data (eg, salary, number of emails sent) and a label of whether an employee was a POI or not. This trained classifier will then be used on a portion of the data withheld from training, where we make predictions for whether an employee is a POI or not. We measure the classifier's performance by comparing the predictions against the actual labels of the employees.

Data Exploration:

Our dataset has a total of 144 instances, with 18 that are POI's (ie, 12.5% of the data) — 3 items in `data_dict` were not utilized (1 was a TOTAL of all features, 1 had all null values). After removing 1 outlier (Enron CEO Kenneth Lay), we end up with 143 data points and 17 POI's.

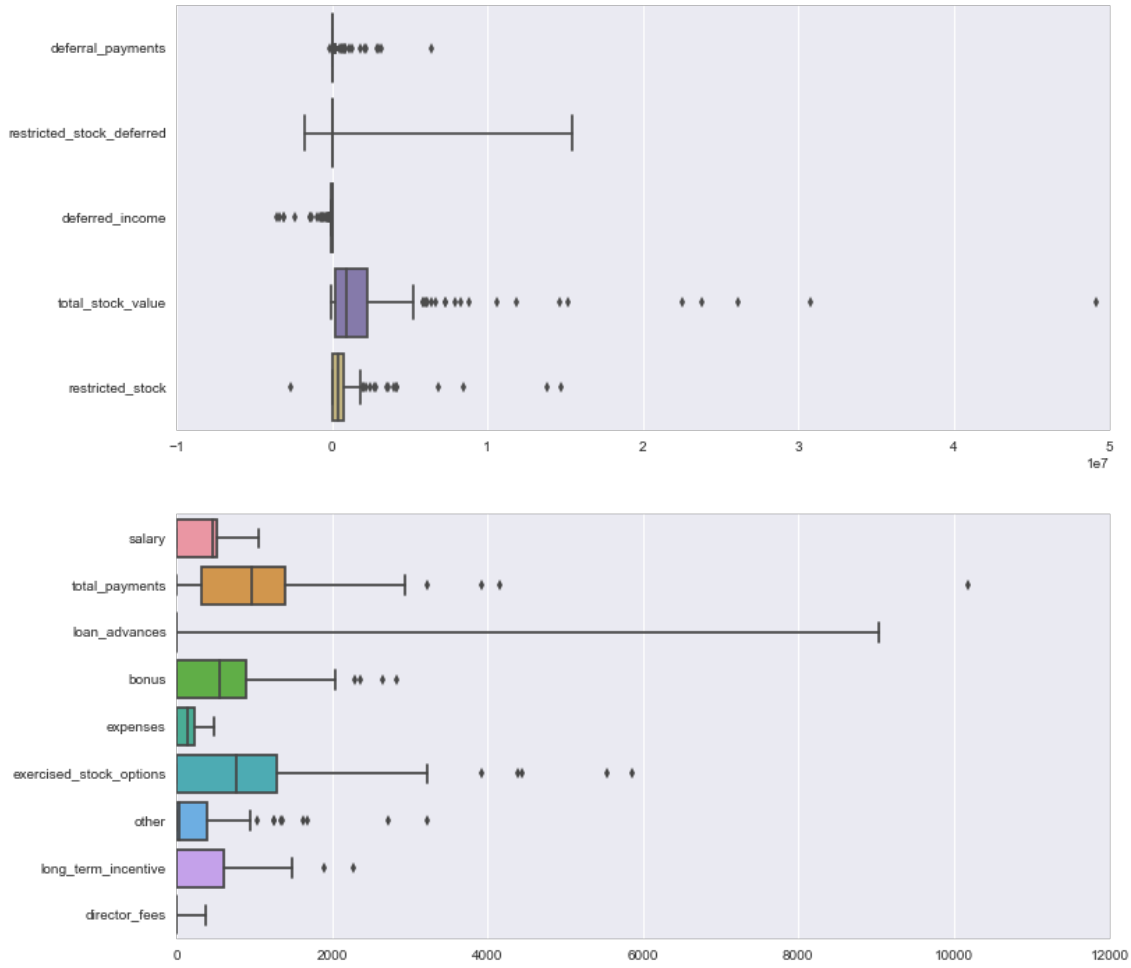
Our analysis is performed with 19 features, although the `data_dict` actually has 20 features in total (the feature "email address" is removed).

We have a number of NaN (null) values which we convert to 0 (two exceptions are noted below):

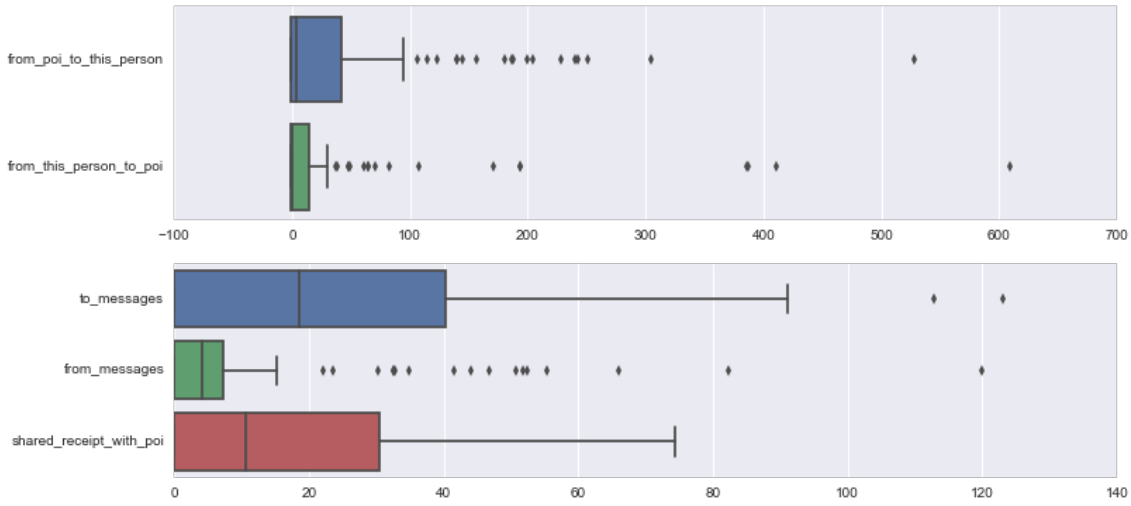
```
salary 51
to_messages 60
deferral_payments 107
total_payments 21
exercised_stock_options 44
bonus 64
restricted_stock 36
shared_receipt_with_poi 60
restricted_stock_deferred 128
total_stock_value 20
expenses 51
loan_advances 142
from_messages 60
other 53
from_this_person_to_poi 60 (nulls converted to -1)
director_fees 129
deferred_income 97
long_term_incentive 80
email_address 35
from_poi_to_this_person 60 (nulls converted to -1)
```

Outlier Investigation:

In [230]:
```
# Boxplots of financial features
```



In [80]:
```
# Boxplots of email features
```



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", "properly scale features", "intelligently select feature"]

Feature Selection:

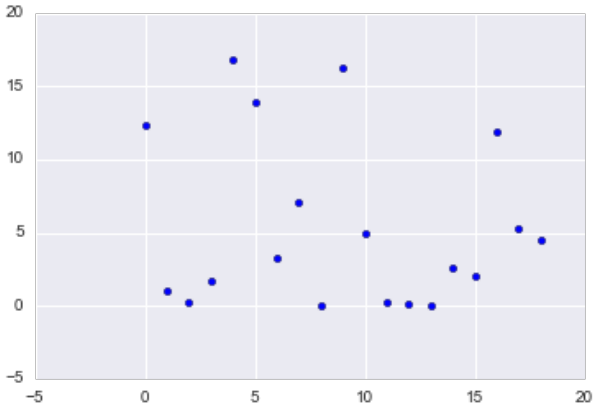
Our POI identifier ends up using 6 features, which were selected using sklearn's SelectKBest (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html). The feature scores were as follows:

In [186]:
```
# SelectKBest scores
```

	salary	to_messages	deferral_payments	total_payments	exercised_stock_options	bonus	re
0	12.381562	1.056387	0.221189	1.742171	16.800352	13.954285	3.3

The feature scores were visually inspected to spot any standout features. The below plot shows 5 features really stood out, and ultimately the top 6 were selected for use (the 6th feature is the only email feature used).

In [187]:
```
# Plot of SelectKBest scores
```



Here's a look at the feature that were ultimately used:

In [189]:
```
# Display head of our dataset of 6 highest scoring features
```

	salary	exercised_stock_options	bonus	shared_receipt_with_poi	total_stock_value	deferred_i
0	201955.0	1729541.0	4175000.0	1407.0	1729541.0	-3081055.(
1	0.0	257817.0	0.0	0.0	257817.0	0.0
2	477.0	4046157.0	0.0	465.0	5243487.0	-5104.0
3	267102.0	6680544.0	1200000.0	0.0	10623258.0	-1386055.(
4	239671.0	0.0	400000.0	0.0	63014.0	-201641.0

Feature Engineering:

New polynomial features (scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html) were created to detect any interaction between features. Model performance with these features was not as good as simply using the 6 best features.

In [192]:
```
# Test set results of models fitted on polynomial features
```

```
GaussianNB:

             precision    recall  f1-score   support

          0       0.88      0.18      0.30        38
          1       0.11      0.80      0.20         5

avg / total       0.79      0.26      0.29        43

---------------------------------

LogisticRegression:

             precision    recall  f1-score   support

          0       0.89      0.87      0.88        38
          1       0.17      0.20      0.18         5

avg / total       0.81      0.79      0.80        43

---------------------------------

DecisionTreeClassifier:

             precision    recall  f1-score   support

          0       0.90      0.92      0.91        38
          1       0.25      0.20      0.22         5

avg / total       0.82      0.84      0.83        43

---------------------------------

SVC:

             precision    recall  f1-score   support

          0       0.88      1.00      0.94        38
          1       0.00      0.00      0.00         5

avg / total       0.78      0.88      0.83        43

---------------------------------
```

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"]

Pick an Algorithm:

We end up using Naive Bayes, which likely performed well given the small amount of data.

We also looked at other classifiers, including Logistic Regression to use a linear separator, and SVM and Decision Tree to try out non-linear separators. Our features were scaled for Logistic Regression and SVM, but neither they nor the Decision Tree could outperform Naive Bayes.

Given that the Naive Bayes model has no parameters to tune, we will attempt to tune the SVM.

In [211]:
```
# Test set results of models fitted on 6 highest scoring features
```

```
GaussianNB:

             precision    recall  f1-score   support

          0       0.93      1.00      0.96        38
          1       1.00      0.40      0.57         5

avg / total       0.94      0.93      0.92        43

--------------------------------

LogisticRegression:

             precision    recall  f1-score   support

          0       0.82      0.61      0.70        38
          1       0.00      0.00      0.00         5

avg / total       0.73      0.53      0.62        43

--------------------------------

DecisionTreeClassifier:

             precision    recall  f1-score   support

          0       0.88      0.95      0.91        38
          1       0.00      0.00      0.00         5

avg / total       0.78      0.84      0.81        43

--------------------------------

SVC:

             precision    recall  f1-score   support

          0       0.88      1.00      0.94        38
          1       0.00      0.00      0.00         5

avg / total       0.78      0.88      0.83        43

--------------------------------

LogisticRegression (scaled features):

             precision    recall  f1-score   support

          0       0.90      1.00      0.95        38
          1       1.00      0.20      0.33         5

avg / total       0.92      0.91      0.88        43

--------------------------------

SVC (scaled features):

             precision    recall  f1-score   support

          0       0.89      0.84      0.86        38
```

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 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 item: "tune the algorithm"]

Parameter tuning:

Most machine learning algorithms have parameters that can be adjusted with different values to produce different fits to a given dataset. For example, with SVMs we can adjust the `gamma` parameter that defines how far the influence of a single training example reaches, and the `C` parameter that trades off misclassification of training examples against simplicity of the decision surface (see: RBF SVM parameters (http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html)).

Default parameters for many sklearn estimators were benchmarked (http://partiallyderivative.com/podcast/2016/06/27/randy-olson-interview) with the MNIST (http://yann.lecun.com/exdb/mnist/) dataset, which isn't necessarily translatable to predicting other types of data and datasets (also see new project to create sklearn benchmarks (https://github.com/rhiever/sklearn-benchmarks)). This makes it very important to try out other parameter settings to find the best fit of the model to the data being examined.

With grid search we can specify a set of different parameter values to see if different settings can improve the performace of the model, in effect "tuning" the model to be most effective with our data. Here, the grid search successfully improves the test results of the SVM.

Untuned model results:

```
precision    recall    f1-score     support
0.14         0.20      0.17         5
```

Tuned model results:

```
precision    recall    f1-score     support
0.33         0.40      0.36         5
```

In [208]:
```
from sklearn.grid_search import GridSearchCV
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

# Xcale the data
scaler = StandardScaler()
X_train_pre = scaler.fit_transform(X_train)
X_test_pre = scaler.transform(X_test)

# Create a scoring function
f1_scorer = metrics.make_scorer(metrics.f1_score)

# Setup parameters and cross validation for model optimization through Grid Search
C_range = np.logspace(0, 1, 16)
gamma_range = np.logspace(-1, 1, 16)
parameters = dict(gamma=gamma_range, C=C_range)

# Fit the grid object
grid_obj = GridSearchCV(SVC(class_weight='balanced'), parameters, scoring=f1_scorer,
grid_obj.fit(X_train_pre, y_train)

# Select the best settings for classifier
best_clf = grid_obj.best_estimator_

# Show the tuned model's parameters
print best_clf, "\n"

# Test the algorithm's performance
print 'Test set results:\n', classification_report(y_test, best_clf.predict(X_test_pr
print 'Train set results:\n', classification_report(y_train, best_clf.predict(X_trair
```

```
SVC(C=1.0, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.46415888336127786,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)

Test set results:
             precision    recall  f1-score   support

          0       0.92      0.89      0.91        38
          1       0.33      0.40      0.36         5

avg / total       0.85      0.84      0.84        43

Train set results:
             precision    recall  f1-score   support

          0       0.97      0.88      0.92        88
          1       0.48      0.83      0.61        12

avg / total       0.91      0.87      0.88       100
```

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"] 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"]

Validate and Evaluate:

Validation is important in order to ensure that our model generalizes well to data that's not being trained or fitted on. A classic mistake here is to validate a model with data that the model was actually trained on. It's important to keep these "training" and "test" data separate, and that's what we've done here. Using sklearn's `train_test_split` method we have created a test set that contains ~ 30% of our original data (ie, 43 data points). We have also stratified the train and test sets to preserve the class imbalance across the two sets.

Our model's performance is evaluated by looking at 3 measures: precision, recall, F1 score.

- **Precision** allows us to measure how accurate the model is when it actually predicts that an individual is a POI. It equals the number of correct hits divided by total guesses that a person is a POI.
- **Recall** shows how many of the POI's we identified out of all the POI's that exist. It equals the number of correct hits divided by total POI's that exist.
- **F1 score** gives an interpretation of precision and recall together so that we balance the ability to make accurate predictions (precision) vs not missing out on identifying possible POI's (recall). It is equal to 2 times the product of precision and recall, divided by the sum of precision and recall.

$$F_1 = 2 \times \frac{(Precision)(Recall)}{Precision+Recall}$$

Here are the results of the Naive Bayes model:

```
precision    recall    f1-score    # of poi's in test set
1.00         0.40      0.57        5
```

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.