Identify fraud from Enron dataset

Sam Cumarasamy

19th June, 2018

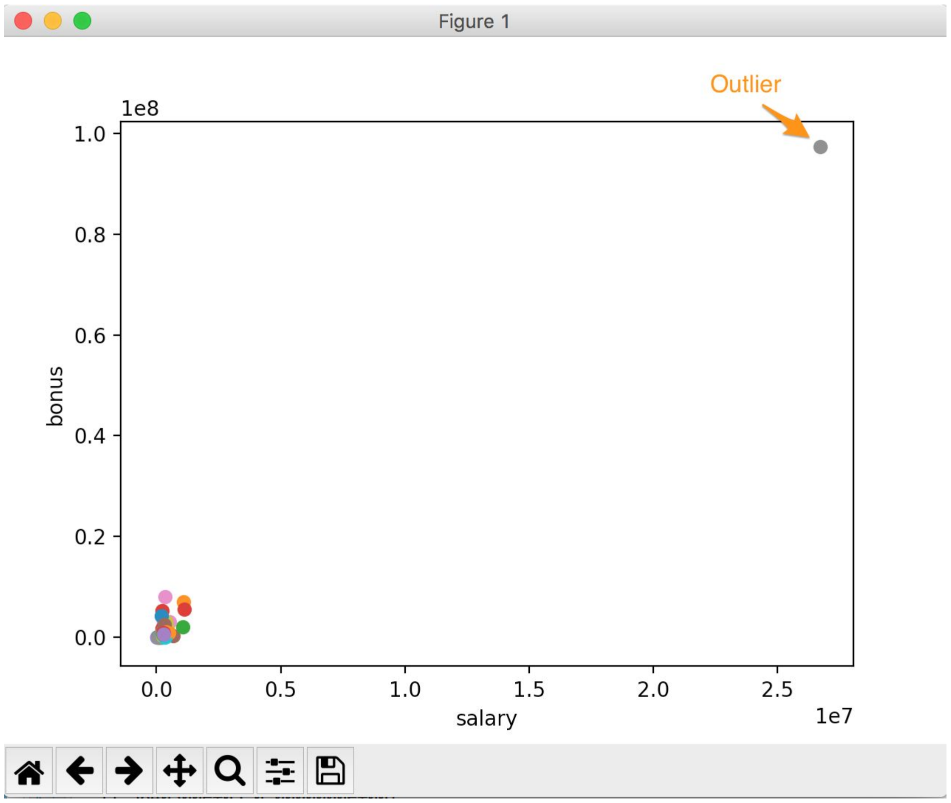
Understanding the Dataset and Question

> *"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?"*

Enron is a perfect example of corporate greed, fraud and corruption. As a result, it's failure would go on to be one of the biggest corporate failures in American history if not the world. This project is about the dataset that was made available after the demise of Enron. The main aim here is to apply Machine Learning algorithms to help predict as to who are the Persons of Interest who were particularly involved in the downfall of Enron. The goal of this project is to find an algorithm that will help identify these Persons of Interest (poi) with the highest possible accuracy. The dataset for this project comes from a public document (enron61702insiderpay.pdf) showing the financial details of individuals high up in the organisation as well a number of email communication between these individuals. This dataset can be summarised as follows:

```
Number of records in the dataset:  146
Number of fields in the dataset:  21
Variables in dataset:  ['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']
number of Persons of Interest:  18
number of Non Persons of Interest:  128
```

From the above dataset, there were 3 outliers identified and removed. The first outlier was identified in the lesson on "Outliers" when plotting the values of "salary" and "bonus" in the dataset which is shown below:



From here, it was identified that the above outlier has a salary larger than 2.5 * 10^7 and this corresponded with the record “TOTAL” in the dataset. Furthermore, by just looking at the PDF file ie enron61702insiderpay.pdf, a further two more outliers were identified and they were "LOCKHART EUGENE E" and "THE TRAVEL AGENCY IN THE PARK". The following are the details of the records in the dataset:

```
TOTAL:  {'salary': 26704229, 'to_messages': 'NaN', 'deferral_payments': 32083396,
'total_payments': 309886585, 'exercised_stock_options': 311764000, 'bonus': 97343619,
'restricted_stock': 130322299, 'shared_receipt_with_poi': 'NaN', 'restricted_stock_deferred':
-7576788, 'total_stock_value': 434509511, 'expenses': 5235198, 'loan_advances': 83925000,
'from_messages': 'NaN', 'other': 42667589, 'from_this_person_to_poi': 'NaN', 'poi': False,
'director_fees': 1398517, 'deferred_income': -27992891, 'long_term_incentive': 48521928,
'email_address': 'NaN', 'from_poi_to_this_person': 'NaN'}

LOCKHART EUGENE E:  {'salary': 'NaN', 'to_messages': 'NaN', 'deferral_payments': 'NaN',
'total_payments': 'NaN', 'exercised_stock_options': 'NaN', 'bonus': 'NaN', 'restricted_stock':
'NaN', 'shared_receipt_with_poi': 'NaN', 'restricted_stock_deferred': 'NaN', 'total_stock_value':
'NaN', 'expenses': 'NaN', 'loan_advances': 'NaN', 'from_messages': 'NaN', 'other': 'NaN',
'from_this_person_to_poi': 'NaN', 'poi': False, 'director_fees': 'NaN', 'deferred_income': 'NaN',
'long_term_incentive': 'NaN', 'email_address': 'NaN', 'from_poi_to_this_person': 'NaN'}

THE TRAVEL AGENCY IN THE PARK:  {'salary': 'NaN', 'to_messages': 'NaN', 'deferral_payments':
'NaN', 'total_payments': 362096, 'exercised_stock_options': 'NaN', 'bonus': 'NaN',
'restricted_stock': 'NaN', 'shared_receipt_with_poi': 'NaN', 'restricted_stock_deferred': 'NaN',
'total_stock_value': 'NaN', 'expenses': 'NaN', 'loan_advances': 'NaN', 'from_messages': 'NaN',
'other': 362096, 'from_this_person_to_poi': 'NaN', 'poi': False, 'director_fees': 'NaN',
'deferred_income': 'NaN', 'long_term_incentive': 'NaN', 'email_address': 'NaN',
'from_poi_to_this_person': 'NaN'}
```

Record	**Outlier Reason**
“TOTAL”	The values for each variable are cumulative of all the records in the dataset. Hence the values are quite large.
"LOCKHART EUGENE E"	This individual has no values recorded in this dataset. So the algorithm is not going to be useful for the classifier
"THE TRAVEL AGENCY IN THE PARK"	Rather than being an individual it’s actually a company. Therefore it’s not a person of interest. Additionally, it holds only one value which is “Total Payments”. Hence an outlier.

So the above records were removed from the dataset in the code:

```
TOTAL_element = data_dict.pop("TOTAL")
lockhart_element = data_dict.pop("LOCKHART EUGENE E")
agency_element = data_dict.pop("THE TRAVEL AGENCY IN THE PARK")
```

Optimize Feature Selection/Engineering

> *"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."*

There are 2 types of features that were available for this project. There are the finance-based 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']
```

Then there's the email-based features:

```
['to_messages', 'email_address', 'from_poi_to_this_person',
'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']
```

For this project, 4 new features were created where 2 of them were finance-based and the other 2 were email-based. The new features added were:

- The following financial features were created because the thought process was that Persons of Interest (poi) would stand to gain financially either through stock options or being generously compensated:
    - 'Sal_pkg_pct' - Ratio of salary package to total monies received (finance feature)
    - 'Pay_stock_ratio' - Ratio of total monies received to the total stock holdings (finance feature)
    - **NB:** The above ratios are meant to show how much financially an individual is incentified by trying to drive up the share price (e.g. share price, stock prices) e.g. bonuses and stock options. It's an indication that an individual is more likely to commit fraud if they've got a high bonus componenet or a stock option.
- The following email-based features, in that
    - 'Ratio_to_msg' - ratio of messages received from poi to the total number of emails received (email feature)
    - 'Ratio_from_msg' - ratio or email messages sent to a poi to the total number of emails sent out (email feature)

- **NB:** The choice of the above features is that they measure the amount of times the individual communicates with a “poi”. The above email-based features were taken from the lesson “Feature Selection” - Creating a new Enron feature

The code to calculate the above new features:

```
for key in data_dict:
   bonus = data_dict[key]["bonus"]
   salary = data_dict[key]["salary"]
   total_payments = data_dict[key]["total_payments"]
   total_stock_value = data_dict[key]["total_stock_value"]
   to_messages = data_dict[key]["to_messages"]
   from_messages = data_dict[key]["from_messages"]
   from_poi = data_dict[key]["from_poi_to_this_person"]
   to_poi = data_dict[key]["from_this_person_to_poi"]

   if total_payments == "NaN":
       data_dict[key]["sal_pkg_pct"] = 0
       total_payments = 0
   else:
       if bonus == "NaN":
           bonus = 0
       if salary == "NaN":
           salary = 0
       data_dict[key]["sal_pkg_pct"] = round((float(bonus + salary)/total_payments), 4)

   if total_stock_value == "NaN":
       data_dict[key]["pay_stock_ratio"] = 0
   else:
       data_dict[key]["pay_stock_ratio"] = round((float(total_payments)/total_stock_value),
4)
   if from_poi == "NaN" or to_messages == "NaN":
       data_dict[key]["ratio_to_msg"] = 0
   else:
       data_dict[key]["ratio_to_msg"] = (float(from_poi)/to_messages)

   if to_poi == "Nan" or from_messages == "NaN":
       data_dict[key]["ratio_from_msg"] = 0
   else:
       data_dict[key]["ratio_from_msg"] = (float(to_poi)/from_messages)
```

The feature **'loan_advances'** was removed from the initial feature list as this particular feature was producing the following warning message when trying to evaluate the performance of the model using cross-validation::

```
/Users/samcumar/anaconda/lib/python2.7/site-packages/sklearn/feature_select
ion/univariate_selection.py:113: UserWarning: Features [3] are constant.
  UserWarning)
```

```
/Users/samcumar/anaconda/lib/python2.7/site-packages/sklearn/feature_select
ion/univariate_selection.py:114: RuntimeWarning: invalid value encountered
in divide
  f = msb / msw
```

So in summary, the feature list was used for this model:

```
features_list =  ['poi', 'salary', 'deferral_payments', 'total_payments', 'bonus',
                 'restricted_stock_deferred', 'deferred_income', 'total_stock_value',
                 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive',
                 'restricted_stock', 'director_fees', 'to_messages',
'from_poi_to_this_person',
                 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi',
                 'sal_pkg_pct','pay_stock_ratio', 'ratio_to_msg', 'ratio_from_msg']
```

For this project, the “SelectKBest” algorithm was used to select the features that would contribute to the optimal performance of the model. It was used as part of the pipeline below:

```
pipeline = Pipeline([("select", SelectKBest(f_classif)),
                   ("clf", AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))
                    ])
```

For this project, the “SelectKBest” algorithm was used to select the top k number of features from the list “features_list” that would contribute to the optimal performance of the model. In the case of the top performing model that was found, in order to find the top optimal “k” number of features, a numerical range of k-values was passed onto the RandomizedSearchCV algorithm like so:

```
param_grid_rs = {
   "select__k": range(10, 15), #----> Range of k-values
   "clf__base_estimator__criterion" : ["gini", "entropy"],
   "clf__base_estimator__splitter" : ["best", "random"],
   "clf__base_estimator__max_depth" : [2, 3],
   "clf__base_estimator__min_samples_split" : [5, 6, 7],
   "clf__base_estimator__min_samples_leaf": [1, 2, 3, 4, 5, 6, 7],
   "clf__algorithm": ["SAMME", "SAMME.R"],
   "clf__n_estimators": range(55, 70),
   "clf__learning_rate": [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1]
            }

rndm_srch = RandomizedSearchCV(pipeline, param_distributions=param_grid_rs, n_iter=70, cv=7,
n_jobs=-1)
```

From the the above code, (as identified by “select_k” in the param_grid_rs variable) the range of k-values of 10 to 15 were passed onto RandomizedSearchCV which then identified the optimal k-value. See below for a summary of the k-value that was chosen by the RandomizedSearchCV algorithm:

```
****************
SelectK summary:
****************
k value -> 13

Rank 1 -> Score: 21.53 - Feature: sal_pkg_pct
Rank 2 -> Score: 18.79 - Feature: exercised_stock_options
Rank 3 -> Score: 17.70 - Feature: total_stock_value
Rank 4 -> Score: 16.01 - Feature: bonus
Rank 5 -> Score: 12.90 - Feature: deferred_income
Rank 6 -> Score: 12.60 - Feature: salary
Rank 7 -> Score: 12.35 - Feature: ratio_from_msg
Rank 8 -> Score: 10.12 - Feature: shared_receipt_with_poi
Rank 9 -> Score: 8.62 - Feature: expenses
Rank 10 -> Score: 7.66 - Feature: long_term_incentive
Rank 11 -> Score: 7.65 - Feature: total_payments
Rank 12 -> Score: 5.77 - Feature: restricted_stock
Rank 13 -> Score: 5.42 - Feature: from_poi_to_this_person
```

Output above was taken from running the model_perf.py program for the AdaBoostClassifier algorithm.

Due to the optimal model being a tree-based classifier, there was no requirement for scaling of features. Additionally, no Principal Component Analysis (PCA) algorithm was used as this was found to degrade the performance of the model when trying to tune the AdaBoostClassifier which provided the best performing model.

Also it was good to see that the engineered feature “sal_pkg_pct” had the highest rank amongst the features that were next passed onto the AdaBoost algorithm.

In regards to the “features_list”, the following table summarises the model performance with and without the newly engineered features:

Model	**Nested-f1 score**	**Precision**	**Recall**	**f1**
With the 4 newly created features	0.133	0.1	0.33	0.15
Without the 4 newly created features	0.747	0.833	0.75	0.79

NB: The above scores were taken with KFolds=10.

Pick and Tune an Algorithm

Picking an Algorithm

> *"What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?"*

Three algorithms were chosen and tested. They were:

- AdaBoostClassifier
- RandomForestClassifier
- GaussianNB (Gaussian Naive Bayes)

In tuning the algorithms, their best performances are summarised below:

Algorithm	**Nested-f1**	**Precision**	**Recall**	**f1**
AdaBoost	0.747	0.833	0.75	0.789
RandomForest	0.547	0.65	0.448	0.531
GuassianNB	0.403	0.4	0.533	0.457

From the above summary, it's clearly seen that the AdaBoostClassifier performed the best. The above results are an output from the file model_perf.py, using KFolds value of 10. So that's the algorithm that was chosen for this model.

Tuning the Algorithm

> *"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?"*

Tuning the algorithm helps to find the most optimal set of parameters that will help solve a machine learning problem. In this particular case, finding poi's in an accurate manner. Failure to properly tune the algorithm can result in an algorithm that will either underfit (bias) or overfit (variance). Depending on the algorithm, tuning these parameters can be a time-consuming task. Hence there are algorithms that can help us in wading through this parameters and help us choose the optimal values. Two of these were used for this project:

- GridSearchCV - This algorithm searches in an exhaustive manner throughout a manually set range of parameters.
- RandomizedSearchCV - This algorithm randomly selects a set of parameter values to be used by the algorithm from a set range of parameters.

In this project, the RandomizedSearchCV was used to tune the AdaBoostClassifier (see poi_id_AdaBoost.py). It was also used to tune the parameters for the RandomForestClassifier (see poi_id_RandomForest.py). However, for the Gaussian Naive Bayes algorithm, the GridSearchCV algorithm was used as there wasn't much parameters to tune (see poi_id_GNB.py file). The reason for choosing the RandomizedSearchCV was due to the large number of parameters that I was seeking to tune. Using the GridSearchCV would take a long time and would use a lot of CPU resources for this project.

For the top performing algorithm in this project, the following parameters were tuned using the RandomizedSearchCV(see poi_id_AdaBoost.py) to find the optimal model:

```
pipeline = Pipeline([("select", SelectKBest(f_classif)),
                    ("clf", AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))
                    ])
.
.
.
param_grid_rs = {
   "select__k": range(10, 15),
   "clf__base_estimator__criterion" : ["gini", "entropy"],
   "clf__base_estimator__splitter" : ["best", "random"],
   "clf__base_estimator__max_depth" : [2, 3],
   "clf__base_estimator__min_samples_split" : [5, 6, 7],
   "clf__base_estimator__min_samples_leaf": [1, 2, 3, 4, 5, 6, 7],
   "clf__algorithm": ["SAMME", "SAMME.R"],
   "clf__n_estimators": range(55, 70),
   "clf__learning_rate": [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1]
            }

rndm_srch = RandomizedSearchCV(pipeline, param_distributions=param_grid_rs, n_iter=70, cv=7,
n_jobs=-1)
```

Validate and Evaluate

Validation and Strategy

"What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?"

Validation is a technique by which an error rate/accuracy is found of the chosen Machine Learning model when applied to a sample dataset of a population. This error rate/accuracy of the model is indicative of when it's applied to the entire dataset of a population. Failure to validate the model can lead to the problem of overfitting. So when the model is applied against data that is unseen, the error rate will be high or accuracy is found to be low. Validation is basically about separating the data into training & testing data.

One of the more common techniques of validation is K-Folds Cross-Validation. K-Folds Cross-Validation is where data is randomly split into K-Folds where each Fold is approximately the same length. Each Fold is split into training and test datasets. The model is then trained K times over each fold. Then the validation score or error rate is then the average of the accuracy or error rate of each Fold. In this project, the method of Cross-Validation used was StratifiedShuffleSplit. The algorithm StratifiedShuffleSplit was used because of the small dataset as well as the small ration of poi records to non-poi records. In such an algorithm, there is a greater chance of a poi record, in both training and test datasets in each Fold.

Two techniques using StratifiedShuffleSplit were employed. The first technique was using a nested validation method using the cross_val_score() function. The f1 score was used to validate the model. The second technique again used the StratifiedShuffleSplit algorithm to split the data into training and test data. Then from there the scores of precision, recall and f1 were used to validate the model. In both techniques, a K-Fold value of 10 was used. Below is the function that was written that shows both techniques (taken from model_perf.py):

```
def model_perf(estimator, feature_list, dataset, cv):
   data = featureFormat(dataset, feature_list, sort_keys=True)
   labels, features = targetFeatureSplit(data)
   features = np.array(features)
   labels = np.array(labels)
   # Using nested cross validation (cross_val_score) to calculate the nested f1 score
   f1_nested_scores = cross_val_score(estimator, features, labels, cv=cv, scoring="f1")
   f1_nested_scores_avg = f1_nested_scores.mean()
   print "************************************************************"
   print "Calculating nested f1 score using cross_val_score() function"
   print "*****************************************************************"
   print "Nested cross_val f1 scores array: ", f1_nested_scores
   print "cross_val f1 avg score: {:.3f}".format(f1_nested_scores_avg)
   print "****************************************************"
   print "Performing Cross Validation split ie cv.split(X, y)"
   print "****************************************************"
   tn_sum = 0
   fn_sum = 0
   tp_sum = 0
   fp_sum = 0
   for train_idx, test_idx in cv.split(features, labels):
       f_train, f_test = features[train_idx], features[test_idx]
       l_train, l_test = labels[train_idx], labels[test_idx]
       estimator.fit(f_train, l_train)
       pred = estimator.predict(f_test)
       # using confusion matrix to find true -ve, false -ve, true +ve, false +ve
       tn, fp, fn, tp = confusion_matrix(pred, l_test).ravel()
       tn_sum = tn_sum + tn
       fn_sum = fn_sum + fn
       tp_sum = tp_sum + tp
       fp_sum = fp_sum + fp
   precision = 1.0 * tp_sum / (tp_sum + fp_sum)
   recall = 1.0 * tp_sum / (tp_sum + fn_sum)
   f1 = 2.0 * ((precision * recall) / (precision + recall))
   print "CV Split Precision: {}".format(precision)
   print "CV Split Recall: {}".format(recall)
   print "CV Split F1: {}".format(f1)
```

Evaluation Metrics

> *“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.”*

Due to the imperfect balance of non-poi/poi records i.e. 18 poi records vs 128 non-poi, the “accuracy” scoring method was not used. This is because of being due to the existence of a large number of non-poi to poi records. So the model will always have a high accuracy in picking non-poi records. To properly evaluate the models, the metrics of precision, recall and f1 were chosen and would better represent the performance of the model.

In the context of this project, this is how the metrics would evaluate the model:

- Precision - This score will show how well the model correctly picked a “poi” from those people that it identified as a “poi”.
- Recall - The recall score will show how well the model was able to correctly identify people who are “poi” from the dataset.
- Nested f1-score or f1 - This is a weighted average of both precision and recall scores.

The scores of the best performing model (AdaBoost) is as follows:

Algorithm	**Nested-f1**	**Precision**	**Recall**	**f1**
AdaBoost	0.747	0.833	0.75	0.789