Identify Fraud from Enron Email

---

1. Overview of the project.

One of the most interesting bankrupt cases in the world — 2002 Enron scandal, for a whopping $23 billion in liabilities from both debt outstanding and guaranteed loans. The dataset was acquired by the Federal Energy Regulatory Commission during its investigation after the company's collapse. A copy of the email database was subsequently purchased for $10,000 by Andrew McCallum, a computer scientist at the University of Massachusetts Amherst. He released this copy open on the web for research purposes.

The dataset contains actual e-mail messages. As part of this ND project, I will be using ML algorithms to find patterns of e-mail communication amongst employees, and finally to predict the culprits behind the company's downfall. The steps taken to achieve the aim of the project :-

* Data wrangling and exploration, removal of outliers.
* Feature selection based on accuracy testing of respective algo.
* Iterative feature selection process helps us decide which ML algorithm is the best fit.
* The algorithm is used to make predictions of POI responsible for fraud.

This way, Machine Learning can be helpful to be a detective! :)

2. Understanding the Dataset and Question.

Data Exploration

* Total number of data points — 146
* Allocation across classes :
    * In the Sample — 18
    * In the Non-PoIs — 128
    * In the Population — 35
* Number of features used — 21
* Are there features with many missing values?

    NaN indicates missing values in the dataset — Features like Salary, Email Addresses and Total Payments. An interesting thing to note is that, none of PoIs have any missing values in Total Payments. This can be used to predict or to estimate Total Payments values of Non - PoI(s).

Outlier Investigation

* While trying to parse the dictionary of dataset, I encountered an error with one of the keys called — 'TOTAL'.
* A few other outliers are valid data entries, therefore not eradicating them.
* Below code snippet explains how it handles the outliers.

```
### Task 2: Remove outliers
temp_data = {}
for x,y in zip(data_dict.keys(),data_dict.values()):
  if not x=='TOTAL':
    temp_data [i] = y
```

* This process was useful to understand limitations of the dataset. :)

3. Optimize Feature Selection/Engineering

Create new features

When my thinking process is involved to creating something new, my ideas were related to computation of correlation of already existing features. I think the rate of communication involving the insiders in Enron matters. Therefore I decided to calculate ratios of communication rate amongst PoIs and Non - PoIs resulting in three new features which has values ranging between zero and one :

* Ratio of communication from PoI.
* Ratio of communication to PoI.
* Ratio of all communications by PoI. (50% of first ratio + 50% of second ratio)

The following code snippets demonstrates creation of new features.

A function to calculate the ratio of messages.

```
def messages_ratio(messages_by_poi, messages_by_all):
    """
     Computes ratio of messages to/from PoI and rest of them.
    """
    init = 0.0
    if messages_by_all !='NaN' and messages_by_all!=0.0:
          init = messages_by_poi/(1.0* messages_by_all)
    return init
```

Iterating through the dictionary and creating newly defined features

```
for x,y in zip(temp_data.keys(), temp_data.values()):

        from_poi_ratio = messages_ratio(y["from_poi_to_this_person"], y["to_messages"])
        to_poi_ratio = messages_ratio(y["from_this_person_to_poi"], y["from_messages"])

        my_dataset[x]["from_poi_ratio"] = from_poi_ratio
        my_dataset[x]["to_poi_ratio"] = to_poi_ratio
        my_dataset[x]["all_poi_ratio"] = (to_poi_ratio*0.5) + (from_poi_ratio*0.5)
```

	Accuracy	Precision	Recall
With created features	0.71682	0.11317	0.183
Without new features (Based on my final set of features)	0.80427	0.4646	0.50300

It is clear that the newly created features don't result in good enough performance.

Therefore, chucking the new features in feature selection.

Intelligently select features

Before deciding about feature selection, it was important to decide algorithm to be the best fit and based on the algo picked, the features were selected accordingly. I have chosen Decision Tree Classifier algorithm based on the results. (See the section : '*Pick and Tune an Algorithm'* for more info.)

One of the most popular Natural Language Processing algorithms — TfidfVectorizer was used to choose the best features.

This was an iterative job of testing and eliminating features and calculation of its importance values with Tfidf function. There was almost ~8 accuracy rate when the list of features were *poi*, *expenses, other.*

Let me show you how the performance is affected based on below feature list (Last part of iterative feature selection work) — :

Feature	Tfidf Value
other	0.76
expenses	0.24

Accuracy : 0.79272

Precision : 0.43924

Recall : 0.477450

Different combinations of features are attempted, and the performance is documented for each one :-

Features	High Tfidf Importance Values (Features)	Accuracy	Precision	Recall
poi, other, salary, loan, total_payments, bonus, restricted_stock, total_stock_value, director_fees, expenses	*salary, total_payments, expenses, restricted_stock, other.*	0.7653	0.214	0.232
poi, salary, total_payments, expenses, restricted_stock, other.	*total_payments, expenses, other*	0.7732	0.258	0.253
poi, total_payments, expenses, other	*expenses, other*	0.781	0.297	0.31
poi, expenses, other	*expenses, other*	0.792	0.439	0.477

Importance values (Tfidf) for *expenses* and *other* were ~ 0.78 and ~ 0.2 and its respective Accuracy, Precision and Recall values were — 0.792, 0.439 and 0.477.

Since values were higher than previous other features and Precision and Recall being > 0.3 . I decided to use the features — *poi*, *expenses, other* .

Why the selected feature-list is selected and is the best?

* *poi* is needed as it's a deciding feature dealing with True or False.

* *expenses* and *other* resulted in best accuracy, precision and recall rate. Before making it to the final selection list, All the features which involve money were considered for experimenting. Different combinations of features were chosen and eliminated based on Tfidf values. The initial features list were : *other*, *salary*, *loan*, *total_payments*, *bonus*, *restricted_stock*, *total_stock_value, director_fees, expenses.*

* For every iterative process of eliminating features with least Tfidf values and in turn considering for the next cycle, features *other* and *expenses* always resulted in high values, mostly non-zero. As we can see from the above table.

* Also, it makes sense that PoI usually tend to spend very abrupt and abundant amount of money costing in expenses (*expenses* feature) and secretly venture in side businesses and other activities — hence *other* makes sense too.

Why other features don't work and is not considered?

* Using too many features results in slow performances.

* Noisy values

* Mainly because of overfitting as well

Properly scale features

I don't think scaling will affect Decision tree algos. Therefore did not perform any scaling on the features.

4. Pick and Tune an Algorithm

Pick an algorithm

The algorithms that I tested out were the following :

* Naive Bayes
* Decision Tree Classifier
* Random Forest Classifier

Note : *I also considered Random Forest Classifier because it uses many of the Decision Tree classifier functions. And it will be good to compare.*

* Naive Bayes :

NB is an efficient algo and executes very fast. It has its own set of limitations, one of them being the inability to process due to strong independence feature assumptions.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
```

I got DivideByZero Error — Precision or recall may be undefined due to a lack of true positive predictions.

* Decision Tree Classifier :

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

Output :

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=None, splitter='best')
           Accuracy: 0.80100     Precision: 0.45599     Recall: 0.48950        F1: 0.47
215        F2: 0.48241
           Total predictions: 11000         True positives:  979   False positives: 11
68         False negatives: 1021  True negatives: 7832
```

* Random Tree Classifier :

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

Output :

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
           Accuracy: 0.80473      Precision: 0.43236     Recall: 0.23650        F1: 0.30
575        F2: 0.26006
           Total predictions: 11000          True positives:  473   False positives:  6
21         False negatives: 1527  True negatives: 8379
```

Performance comparison of all the algorithms

Algorithm	Accuracy	Precision	Time taken
Naive Bayes	NA - (Error)	NA - (Error)	2.795 seconds
Decision Tree	0.801	0.46	1.7 seconds
Random Forest	0.804	0.43	13.89 seconds

Since Decision Tree takes just ~ 1.7 seconds. Decision Tree Classifier is picked as the algorithm.

Discuss parameter tuning and its importance

Tuning or fine-tuning an algorithm means to improve the performance of an algorithm by selectively choosing the best parameters which works for the algorithm. :)

Without tuning the algorithm, many problems arise — especially for Decision Tree classifiers which is based on entropy resulting in the information gain.

It’s important to focus and improvise the info gain from the iterative process! We will also have to make sure that there is no overfitting.

There are many parameters for a DecisionTreeClassifier. You can check them out here : http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Of all the params from that, I have chosen three main params :

* *max_depth*
* *max_features*
* *random_state*

Now let me explain how each of the param affect the performance :

* *max_depth*

This is basically responsible for the max depth the tree can grow. If the number is too high, then it can lead to problems causing over-fitting. The below table shows accuracy, precision and recall for every depth level.

max_depth	Accuracy	Precision	Recall
Default	0.80	0.51	0.49
4	0.77	0.2	0.16
8	0.81	0.469	0.479
12	0.79	0.443	0.49

max_depth = 8 gives best performance.

* max_features

The number of features to consider when looking for the best split. Search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

max_features	Accuracy	Precision	Recall
Default	0.80	0.45	0.49
sqrt	0.77	0.2	0.41
log2	0.78	0.469	0.42
auto	0.79	0.443	0.41

Looks like default option (without any params) gives best performance.

* random_state

random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

max_features	Accuracy	Precision	Recall
Default	0.80	0.45	0.49
42	0.81	0.46	0.489
60	0.78	0.469	0.48
39	0.79	0.443	0.477

Using random_state = 42 leads to best performance.

Code :

```
clf = DecisionTreeClassifier(max_depth=8, random_state=42)
```

Output :

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=8,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best')
        Accuracy: 0.80764       Precision: 0.47157      Recall: 0.48100 F1: 0.47624     F2: 0.47908
        Total predictions: 11000        True positives:  962    False positives: 1078   False negatives: 1038   True negatives: 7922
```

Therefore, this is the FINAL fine tuned param used to exhibit best performance results! :)

---

5. Validate and evaluate

Usage of evaluation metrics, validation, its importance and algorithm performance.

* Accuracy

It is ratio of the count of the data points that are identified correctly divided and the total number of data points. DecistionTreeClassifier algorithm has an accuracy of **0.80764**. It is important to have minimum of this accuracy to predict the fraudsters correctly.

* Precision

Precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). High precision means that an algorithm returned

substantially more relevant results than irrelevant ones. In this work, the precision means the rate of a person being an actual PoI when the person is identified as a PoI by DTC algorithm. This algo has precision of **0.47517**

***** Recall

It is the fraction of relevant instances that have been retrieved over the total amount of relevant instances — Number of true positives divided by the total number of elements that actually belong to the positive class. In this case, recall is about correctly identifying the actual PoI. The algo has recall value of **0.50300**.

Validation Strategy.

Validation i.e. assessing how the results of a statistical analysis will generalize to an independent data set. It is important to validate else, we can't keep track of performance of the algo and there should be a metric that we need to understand. This why we need to validate. I have used cross-validation technique. The goal of cross-validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

Cross validation was done with *sklearn.cross_validation*. It allows us to create random split into training and test sets can be quickly computed with the *train_test_split* helper function. :)