**Enron Fraud detection using Machine Learning**

**1. Data exploration:**

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 play detective, and put our new skills to use by building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. To assist us in our detective work, the authors have combined this data with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

The goal of this project is to leverage machine learning methods along with financial and email data from Enron to construct a predictive model for identifying potential parties of financial fraud. These parties are termed “persons of interest” . The Enron data set is comprised of email and financial data (E + F data set) collected and released as part of the US federal investigation into financial fraud.

**The data set is comprised of:**

- 144 points, each theoretically representing a person
- 18 of these points is labeled as a POI and 128 as non-POI
- 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']

- email features:

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

- POI label: [‘poi’] (boolean, represented as integers)

So 'total_features_list' = POI + financial_features_list + email_features_list, in the same order.

**Outliers:**

Scatter plot of bonus vs. salary revealed the following outliers that were removed:

- TOTAL: Extreme outlier for numerical features, consisting of spreadsheet summary artifact
- THE TRAVEL AGENCY IN THE PARK: because it does not represent a person, and therefore can’t be a POI, leaving 144 data points.

**2. Feature selection and creation:**

SelectKBest was imported from sklearn.feature_selection.

Initially, K = 'all' was supplied to the SelectKBest method. 'k' stores a value of the number of features that are provided to the SelectKBest algorithm. It returned the following output for the features_list (only with existing features and not including the new features)

At any point, these features are not picked manually but used SelectKBest() to pick them randomly with varying K values.

For n_splits = 10

Naive Bayes:

- Accuracy score of NB: 0.386363636364
- Precision Score of NB: 0.15625
- Recall Score of NB: 1.0

Accuracy: 0.41020 Precision: 0.16234 Recall: 0.82300 F1: 0.27119 F2: 0.45372

Total predictions: 15000 True positives: 1646 False positives: 8493 False negatives: 354

True negatives: 4507

Decision Tree:

- Accuracy score of DT: 0.954545454545
- Precision Score of DT: 1.0
- Recall Score of DT: 0.6

Accuracy: 0.81087 Precision: 0.16168 Recall: 0.10000 F1: 0.12357 F2: 0.10826

Total predictions: 15000 True positives: 200 False positives: 1037 False negatives: 1800

True negatives: 11963

Below are the scores for total features (new and existing) with n_splits = 10

Naive Bayes:

Accuracy: 0.37533 Precision: 0.15638 Recall: 0.83850 F1: 0.26360 F2: 0.44782

Total predictions: 15000 True positives: 1677 False positives: 9047 False negatives: 323

True negatives: 3953

Decision Tree:

Accuracy: 0.80847 Precision: 0.21857 Recall: 0.16950 F1: 0.19093 F2: 0.17747

Total predictions: 15000 True positives: 339 False positives: 1212 False negatives: 1661

True negatives: 11788

On comparing we notice that the accuracy and F1 scores have fallen down a bit with SelectKBest having to choose among all the features in contrast to only the existing features. However, the new features were not selected in the final model to calculate this!!

These scores clearly indicate that the new features are not helping to achieve our goal og getting better F1 (Precision and recall) scores.

For n_splits = 100 the scores with all the features provided for the SelectKBest(), K= 'all':

Accuracy score of NB: 0.386363636364 Recall Score of NB: 1.0 Precision Score of NB: 0.15625

Accuracy: 0.37533 Precision: 0.15638 Recall: 0.83850 F1: 0.26360 F2: 0.44782

Total predictions: 15000 True positives: 1677 False positives: 9047 False negatives: 323

True negatives: 3953

Accuracy score of DT: 1.0 Recall Score of DT: 1.0 Precision Score of DT: 1.0

Accuracy: 0.78553 Precision: 0.19861 Recall: 0.20050 F1: 0.19955 F2: 0.20012

Total predictions: 15000 True positives: 401 False positives: 1618 False negatives: 1599

True negatives: 11382

Although there is a difference in the values of Accuracy, Precision and Recall run locally and by using the tester.py, we see that the F1 scores for Decision Tree classifier in a pipeline using n_splits = 100 is much better than the values obtained with n_splits = 10.

However, we will have to achieve a precision and recall values to be atleast 30 % in order to fit the model appropriately.

So let us try to tune with changing k values (number of features provided in the pipeline) of SelectKBest with n_splits = 10 to begin with.

With n_splits = 10, and the k = 14 initially with out scaling the features, we get the following for the different classifiers:

Naive Bayes: NB Recall Score: 0.4 NB Precision Score: 0.5 NB Accuracy Score: 0.886363636364

DecisionTree: DT Accuracy: 0.840909090909 DT Recall Score: 0.4 DT Precision Score: 0.333333333333

ADABOOST: AB Accuracy: 0.863636363636 AB Recall Score: 0.2 AB Precision Score: 0.333333333333

Using tester.py, following scores are obtained for Naive Bayes:

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

Total predictions: 15000 True positives: 691 False positives: 911 False negatives: 1309

True negatives: 12089

where as for Decision Tree classifier the values are not very desirable:

Accuracy: 0.79507 Precision: 0.18112 Recall: 0.15250 F1: 0.16558 F2: 0.15748

Total predictions: 15000 True positives: 305 False positives: 1379 False negatives: 1695

True negatives: 11621

An important point to note here is that the run time for n_splits = 10 was considereably higher with k = 14. So we change the feature selection with k = 12 and check for its perfomance on the scores with n-splits = 10

This time the following 12 features were randomly selected by SelectKBest() as the k value was set to 12. Features selected by SelectKBest:

['bonus', 'deferred_income', 'exercised_stock_options', 'expenses', 'loan_advances', 'long_term_incentive', 'restricted_stock', 'salary', 'total_payments', 'total_stock_value', 'from_poi_to_this_person', 'shared_receipt_with_poi']

In the final model, the following 5 features were selected in the pipeline:

Selected Features, Scores, pValues

[('exercised_stock_options', '25.10', '0.000'), ('total_stock_value', '24.47', '0.000'), ('bonus', '21.06', '0.000'), ('salary', '18.58', '0.000'), ('deferred_income', '11.60', '0.001')]

Naive Bayes: Accuracy score of NB: 0.886363636364 Precision Score of NB: 0.5 Recall Score of NB: 0.4

Decision Tree: Accuracy score of DT: 1.0 Precision Score of DT: 1.0 Recall Score of DT: 1.0

And following scores are obtained using tester.py

Naive Bayes:

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

Total predictions: 15000 True positives: 691 False positives: 911 False negatives: 1309

True negatives: 12089

Decision Tree:

Accuracy: 0.79507 Precision: 0.18112 Recall: 0.15250 F1: 0.16558 F2: 0.15748

Total predictions: 15000 True positives: 305 False positives: 1379 False negatives: 1695

True negatives: 11621

Again, the run time for k = 12 is also considerably high.

Now let us check the scores for k = 10:

Naive Bayes:

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

Total predictions: 15000 True positives: 691 False positives: 911 False negatives: 1309

True negatives: 12089

Decision Tree:

Accuracy: 0.79507 Precision: 0.18112 Recall: 0.15250 F1: 0.16558 F2: 0.15748

Total predictions: 15000 True positives: 305 False positives: 1379 False negatives: 1695

True negatives: 11621

And the run time looked slightly better. So finally, keeping this k = 10 we run for n_splits = 1000.

Following 5 features were selected in the final model,

Selected Features, Scores, pValues

[('exercised_stock_options', '25.10', '0.000'), ('total_stock_value', '24.47', '0.000'), ('bonus', '21.06', '0.000'), ('salary', '18.58', '0.000'), ('deferred_income', '11.60', '0.001')]

On running the tester.py, the scores are as follows: Naive Bayes:

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

Total predictions: 15000 True positives: 691 False positives: 911 False negatives: 1309

True negatives: 12089

Decision Tree:

Accuracy: 0.78773 Precision: 0.20693 Recall: 0.20900 F1: 0.20796 F2: 0.20858

Total predictions: 15000 True positives: 418 False positives: 1602 False negatives: 1582

True negatives: 11398

Hence only the scores of Naive Bayes is passed to be collected in the pickle files.

Using numpy 'NaN' value was added to the empty values. And a count gave that .69 % of values are NaN

Created a tuple called, features_selected_tuple using the SelectKBest algorithm's scores, pvalues for the respective features.

Here total feautures (new and existing)were provided to SelectKBest.

The following features were selected to get these scores, where I see that the new features engineered, namely, "restricted Stock Ratio" and "Money ratio" are not part of the out put estimation.

[('exercised_stock_options', '25.10', '0.000'), ('total_stock_value', '24.47', '0.000'), ('bonus', '21.06', '0.000'), ('salary', '18.58', '0.000'), ('deferred_income', '11.60', '0.001'), ('long_term_incentive', '10.07', '0.002'), ('restricted_stock', '9.35', '0.003'), ('total_payments', '8.87', '0.003'), ('shared_receipt_with_poi', '8.75', '0.004'), ('loan_advances', '7.24', '0.008'), ('expenses', '6.23', '0.014'), ('from_poi_to_this_person', '5.34', '0.022'), ('other', '4.20', '0.042'), ('from_this_person_to_poi', '2.43', '0.122'), ('director_fees', '2.11', '0.149'), ('to_messages', '1.70', '0.195'), ('deferral_payments', '0.22', '0.642'), ('from_messages', '0.16', '0.686'), ('restricted_stock_deferred', '0.06', '0.799')]

On tuning the feature selection values (K = 14), and on expanding the range of values tested for k from SelectKBest from (4-7) to (1-7) the following 5 features were finally selected automatically in the final model:

Selected Features, Scores, pValues

[('exercised_stock_options', '25.10', '0.000'), ('total_stock_value', '24.47', '0.000'), ('bonus', '21.06', '0.000'), ('salary', '18.58', '0.000'), ('deferred_income', '11.60', '0.001')]

**3. Algorithm pick**

To begin with *'Naive Bayes's GaussianNB()'* method is implemented as a base model. This gives us the following values:

- NB Recall Score = 0.4 NB Precision Score = 0.5 NB Accuracy Score = 0.886363636364

Then after implementing *'Decision Tree'* and *Adaboost* algorithms, the following values were obtained:

- DecisionTree: Accuracy = 0.840909090909 DT Recall Score = 0.4 DT Precision Score = 0.333333333333
- ADABOOST: Accuracy = 0.863636363636 AB Recall Score = 0.2 AB Precision Score = 0.333333333333

Next, because the data set is very small and imbalanced, a method called *'StratifiedShuffleSplit'* is used to cross validate the data components instead of the standard train/test model.

Finally The DecisionTree algorithm is picked to be used based on not just the accuracy but also considering the better recall and precission scores (F1)

**4. Tuning the Algorithm**

Tuning is the process of altering algorithm parameters to effect performance. Poor tuning can result in poor algorithm performance. It can also result in an algorithm that gives reasonable performance but is not efficient.

If any information from the 'test' data is used in the training of the model then it is called 'information' or 'data leakage'. It will bias the training model, so in order to avoid 'information leakage' at all costs, a method called *Pipeline* is used to perform all data processing (scaling, feature selection, etc) and parameter estimation (classifier).

The most popular method for tuning parameters, GridSearchCV is used in a pipeline. GridSearchCV searches over a 'grid' of parameters to find the optimal parameters.

For GridSearchCV, the names that are used in the parameter grid are derived from the names of the methods in the pipeline, followed by two underscores.It is a method that runs every model for each combination of parameters in the parameter grid. example parameter grid :

- parameters = {'SKB**k': range(4,7), 'pca**n_components': range(1,5), 'DT**criterion': ['gini', 'entropy'], 'DT**min_samples_split': [2, 10, 20], 'DT**max_depth': [None, 2, 5, 10], 'DT**min_samples_leaf': [1, 5, 10], 'DT__max_leaf_nodes': [None, 5, 10, 20]}

GridSearchCV performs the steps in a pipeline sequentially, feeding each result into the next step.

Once it has run all of these models, it returns the model with the highest score for the metric that we use (in this case: scoring = 'f1', where 'f1' is a weighted average of Precision and Recall).

Choosing different k values as detected by the below line (in the code of poi_id.py file) w.r.t the n_splits helped in getting better precission and accuracy. SKB = SelectKBest(k = 14)

Changing the n_splits (cv= sk_fold) value in the below line of code, a couple of times also helped to get better Recall and precision scores. Finally a value of 100 was used to get the pkl files dumped

sss = StratifiedShuffleSplit(n_splits=6, test_size=0.3, random_state=42)

Adding the Decision Tree classifier in the pipe with n_splits = 6 gives the following values:

- Accuracy: 0.81407 Precision: 0.30245 Recall: 0.30200 F1: 0.30223 F2: 0.30209

Adding the Naive Bayes Classifier in the pipe line with n_splits = 6 pipe = Pipeline(steps=[('scaling',scaler),("SKB", SKB), ("NB", GaussianNB())]

Naive Bayes classifier will not have any parameter to tune.

- Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

Clearly the Naive Bayes Classifier has better accuracy and precision and has tuned well from the initial scores with out tuning.

Since Adaboost classifier's precission and recall scores were very poor, we move on with validating the Naive Bayes and Decision Tree algorithms

ADABOOST: AB Accuracy: 0.863636363636 AB Recall Score0.2 AB Precision Score0.333333333333

**5. Validation:**

Validation is process of determining how well the tuned model performs or fits, using a specific set of criteria. Cross-validation is used to ensure that the model generalizes well, avoiding over or under-fitting, a classic mistake.

Using 1000 folds of data in tester.py as a parameter of the cross validator, StratifiedShuffleSplit(labels, folds=1000, random_state = 42) we could try to validate 10% of the data that was reserved for testing. The process of fitting the training data and predicting with test data was repeated 1000 times, and the average metrics reported.

Validation was done by changing this number of folds in getting the scores right.

*with 50 folds, k = 5*

NB Scores:

Accuracy: 0.85047 Precision: 0.42226 Recall: 0.33000 F1: 0.37047 F2: 0.34508

DT Scores:

Accuracy: 0.81407 Precision: 0.30245 Recall: 0.30200 F1: 0.30223 F2: 0.30209

*with 100 folds, k = 4*

NB Scores:

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

DT Scores:

Accuracy: 0.81407 Precision: 0.30245 Recall: 0.30200 F1: 0.30223 F2: 0.30209

**6. Evaluation**

Recall, precision, and F1 score were finally used as evaluation metrics.

Accuracy was intentionally disqualified as a metric due to the skewed nature of the data set. Since POIs are sparse, even failing to identify any POIs would give a reasonably high accuracy score.

Precision is the ratio of true positives to total positive identifications (true + false) and therefore penalizes false positive identifications. In broader terms, precision is the fraction of retrieved instances that are relevant. That is, high precision equates to a model that returns substantially more relevant than irrelevant results (more true positives per all positives).

Recall is the ratio of a true positives to true positives + false negatives and therefore penalizes false negative identification. Recall is the fraction of relevant instances that are retrieved. High recall equates to a model/classifier returning most of the relevant results, albeit they may be mixed in with irrelevant results.

F1 balances both precision and recall, which is why it is a best evaluation metrics in our case.

So, consodering F1 scores, the below values fit well using a Naive Bayes Classifier with 1000 folds.

Accuracy: 0.85200 Precision: 0.43134 Recall: 0.34550 F1: 0.38368 F2: 0.35982

And high precision value means that there is a posibbility of penalizing false positives at a higher rate than false negatives.These POIs from a large pool could then be investigated, likely leading to others who are colluding with them. That would be a more efficient and ethical use of this tool with out over fitting the data and penalising actual non POIs or under fitting by letting the actual guilty among the non POIs slip away. Also, we have to have the Human interference after a certain point of Machine dependencies in critical cases like this.

***References:***


- Udacity Forum mentor's link (@Myles): https://discussions.udacity.com/t/task-1-using-kbest/325512/26
- http://labhacker.com/articles/Enron-Fraud-Detection/
- http://scikit-learn.org/stable/modules/pipeline.html#pipeline
- http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
- http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html