Objective

Enron cooperation was bankrupted in 2001 primarily due to Fraud and corruption. The goal of the analysis is to build an algorithm that looks at publicly available data of employees financial and email data to predict a possible employee was involved in fraud.

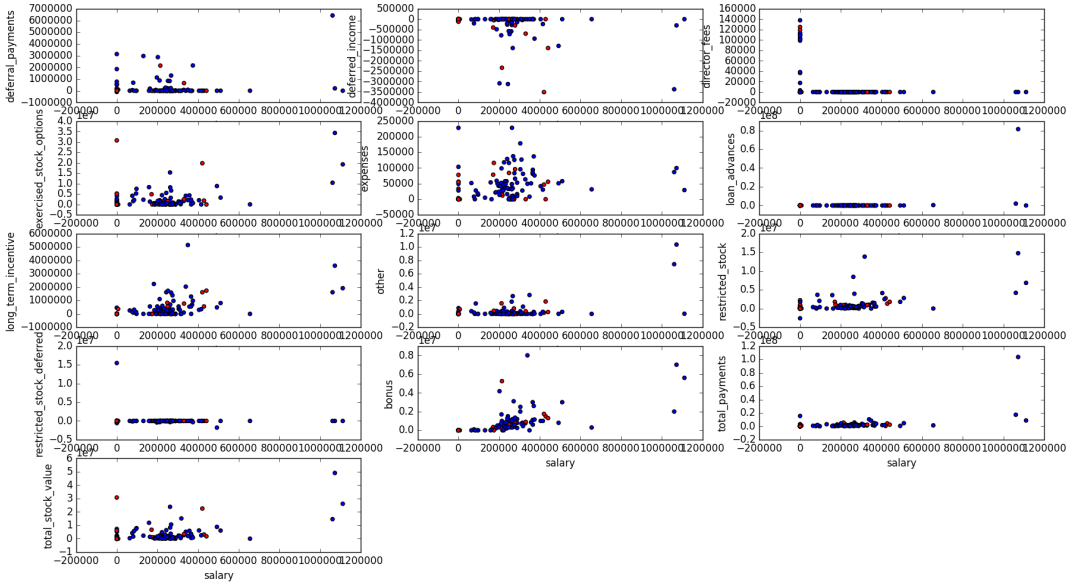
Data

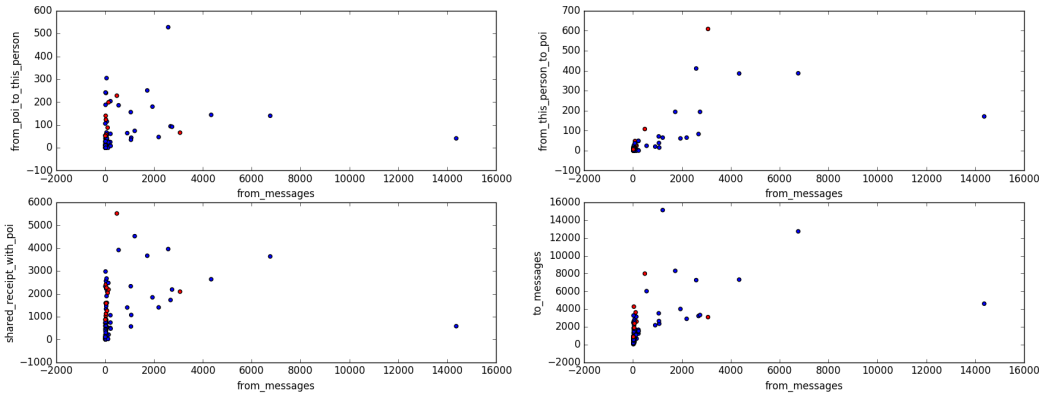
The data presented has 146 employees data with 18 marked as POI(Person of Interest) the features include a set of financial and email features. There are totally 20 features.

The initial steps were to analyze the data, identify outliers and add new features.

Data Analysis/ Outliers

There was an initial outlier which ‘TOTAL’ Column once that was removed then there seems to

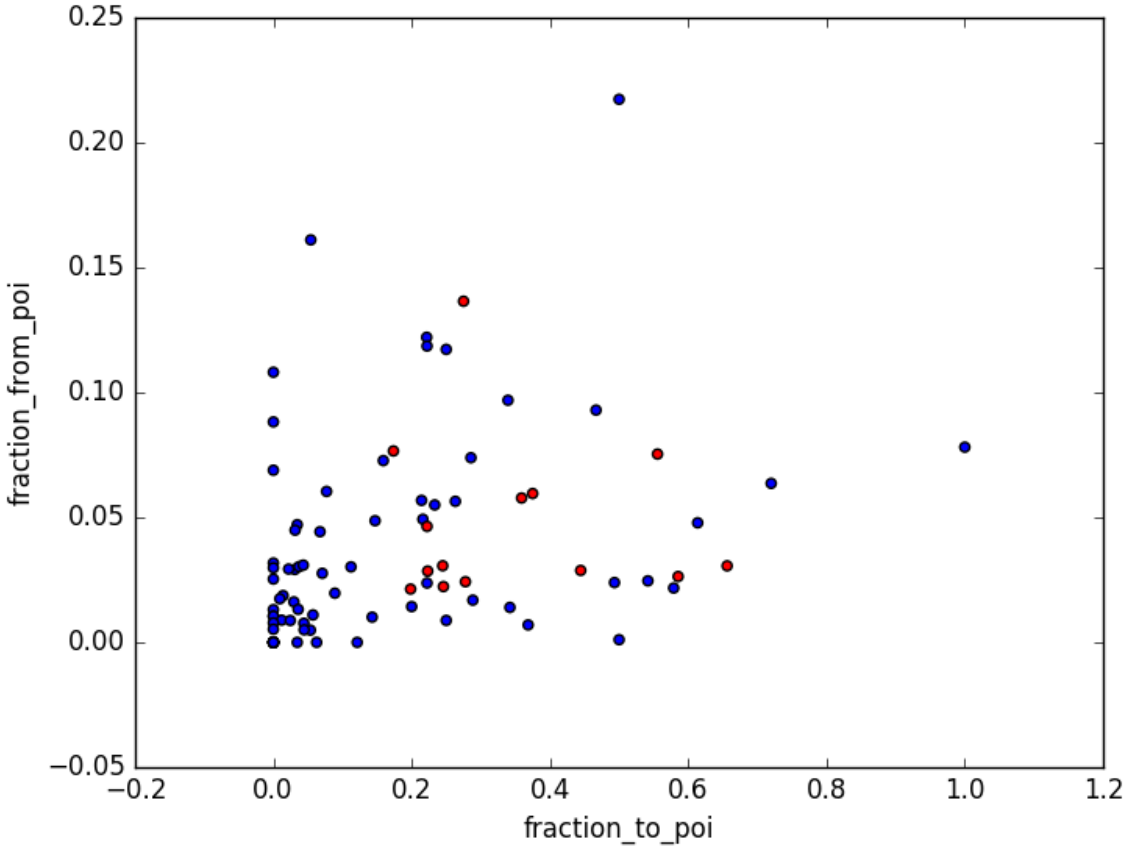




The other outlier I removed was 'LOCKHART EUGENE E' had all null values . analysis.py has other analysis done.

Feature Engineering

Added features "**fraction_to_poi" and "fraction_from_poi"** as new features
To verify these features plotted the features as below and you can see that below 20% for "**fraction_from_poi** " there are no poi's



The features also showed up in top best features

K-best features:[(True, 'exercised_stock_options', 25.097541528735491),

(True, 'total_stock_value', 24.467654047526398),

(True, 'bonus', 21.060001707536571),

(True, 'salary', 18.575703268041785),

(True, 'fraction_to_poi', 16.641707070468989),

(True, 'deferred_income', 11.595547659730601),

(True, 'long_term_incentive', 10.072454529369441),

(True, 'restricted_stock', 9.3467007910514877),

(True, 'total_payments', 8.8667215371077717),

(True, 'shared_receipt_with_poi', 8.7464855321290802),

(False, 'loan_advances', 7.2427303965360181),

(False, 'expenses', 6.2342011405067401),

(False, 'from_poi_to_this_person', 5.3449415231473374),

(False, 'other', 4.204970858301416),

(False, 'fraction_from_poi', 3.2107619169667441),

(False, 'from_this_person_to_poi', 2.4265081272428781),

(False, 'director_fees', 2.1076559432760908),

(False, 'to_messages', 1.6988243485808501),

(False, 'deferral_payments', 0.2170589303395084),

(False, 'from_messages', 0.16416449823428736),

(False, 'restricted_stock_deferred', 0.06498431172371151)]

Ran all the models with a pipeline on SelectKFeatures and aptimized the pipeline for best number of feature performance for different models Following is the output

{Pipeline(steps=[('kbest', SelectKBest(k=3, score_func=<function f_classif at 0x10be04cf8>)), ('naive', GaussianNB())]): [0.839, 0.36443221610805404, 0.24283333333333335, 0.29145829165833165, 0.26019715693978146],
Pipeline(steps=[('kbest', SelectKBest(k=5, score_func=<function f_classif at 0x10be04cf8>)), ('log', LogisticRegression(C=100, class_weight='balanced', dual=False,
fit_intercept=True, intercept_scaling=1, max_iter=100,
multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
solver='liblinear', tol=0.1, verbose=0, warm_start=False))]): [0.75, 0.2834372834372834, 0.5453333333333333, 0.37300501595987234, 0.4602745892415034],

Pipeline(steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x10be04cf8>)), ('random_tree', 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=15, n_jobs=1,
oob_score=False, random_state=None, verbose=0,
warm_start=False))]): [0.8553636363636363, 0.42485549132947975, 0.1715, 0.24436000949893136, 0.19472409355839831],

Pipeline(steps=[('kbest', SelectKBest(k=3, score_func=<function f_classif at 0x10be04cf8>)),
('tree', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
min_samples_split=20, min_weight_fraction_leaf=0.0,
presort=False, random_state=50, splitter='best'))]): [0.8262954545454545,
0.2890885750962773, 0.18766666666666668, 0.22758969176351693, 0.20182828463882418],
Pipeline(steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x10be04cf8>)),
('svm', LinearSVC(C=100, class_weight='balanced', dual=True, fit_intercept=True,
intercept_scaling=1, loss='squared_hinge', max_iter=1000,
multi_class='ovr', penalty='l2', random_state=None, tol=0.1,
verbose=0))]): [0.7503409090909091, 0.21877468125916733, 0.32316666666666666,
0.26091636950817465, 0.29501262818367163],
Pipeline(steps=[('kbest', SelectKBest(k=15, score_func=<function f_classif at 0x10be04cf8>)),
('adaboost', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=0.4, n_estimators=20, random_state=None))]): [0.8423863636363637,
0.3782869044519656, 0.24216666666666667, 0.2952951935778884, 0.2609460867066557]}

The number of features changed by the model used. As seen in the result above. The best performing on this was logistic regression with top 10 features.

I also used the PCA for dimensionality reduction and picked the best one.

Models Used

For the analysis of the data, a total of 10 classifiers were tried out, which include:

- Logistic Regression
- Decision Tree Classifier
- Gaussian Naive Bayes
- Linear Support Vector Classifier (LinearSVC)
- AdaBoost
- Random Forrest Tree Classifier

All versions of these algorithms with PCA and Select K Features

Parameter Tuning

The hyper parameter tuning is done to find the right parameters to optimize on the machine learning algorithm. As an algorithm will require different weights, constraints, learning rates that work best for different data patterns. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.

Then I optimized all the models with GridSearchCV. The parameters used for diff models are as follows.

```
params_pca = {"pca__n_components": [2, 3,  5, 10, 15], "pca__whiten": [False]}
params_tree = { "min_samples_split":[2, 5, 10, 20],
                "criterion": ('gini', 'entropy'),
                'random_state':[50]
                }
params_linearsvm = {"C": [0.1, 1, 5, 10, 100],
                    "tol":[10**-1,  10**-3, 10**-5],
                    "class_weight":['balanced']

                    }
params_adaboost = { "n_estimators":[20, 50, 100],
                    'learning_rate': [0.4, 0.6, 1]}
params_random_tree = {  "n_estimators":[2, 3, 5,10,15],
                        "criterion": ('gini', 'entropy'),
                        'min_samples_split': [1, 2, 4], 'max_features': [1, 2,
3,'sqrt',5,10]
                        }
params_log = {  "C":[0.05, 0.5, 1, 10, 10**2,10**5,],
                "tol":[10**-1, 10**-5, 10**-10],
                "class_weight":['balanced'],
                "penalty": ['l2', 'l1']
                }
```

Evaluation of model

Precision was used to validate the model but also checked for recall.
precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while **recall** (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.

The best tuned models from grid search were validated for performance and best models were picked. Validation is to ensure that the data used for testing is not being used in training the model.

The most optimized version of each model was then validated for performance. Surprisingly DesicionTree and Logistic Regression with PCA performed best.

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,max_features=None, max_leaf_nodes=None, min_samples_leaf=min_samples_split=20, min_weight_fraction_leaf=0.0,presort=False, random_state=50, splitter='best')

[accuracy:0.821659090909091, precision:0.35068714632174613,recall: 0.3615, f1:0.3560114895363151, f2:0.35928441278780854]

LogisticRegression(C=100000, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l1', random_state=None, solver='liblinear', tol=1e-10, verbose=0, warm_start=False)

[0.7978636363636363, 0.31251619590567503, 0.402, 0.35165476016912084, 0.38022573932782644]

References

- Sk learn documentation
- Starter Code & Course examples

Code

Analysis.py for the analysis and poi_id.py