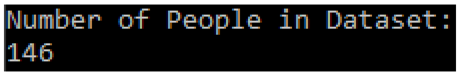
Project 5: Identifying Fraud from Enron Emails and Financial Data

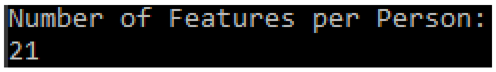
By: Adam Fink

Project Overview

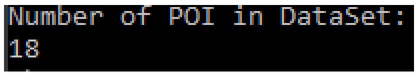
In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.



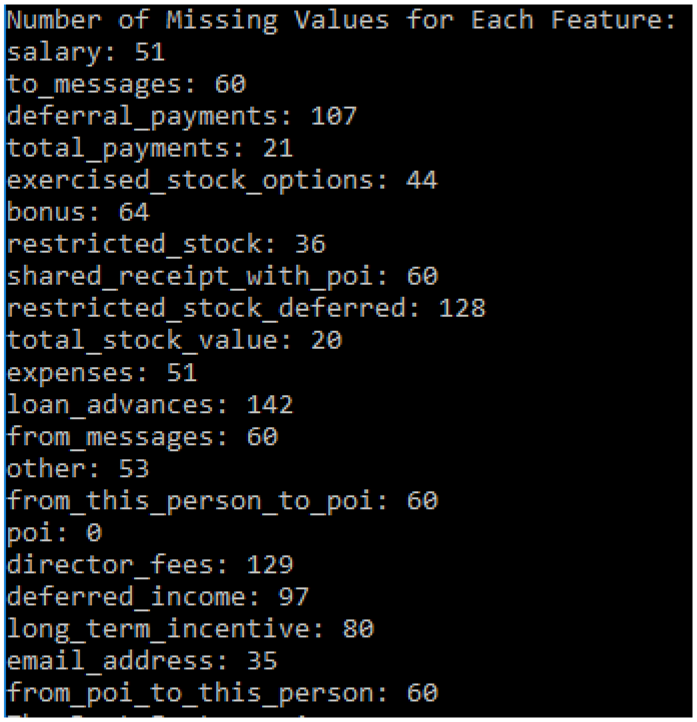
```
Number of People in Dataset:
146
```



```
Number of Features per Person:
21
```



```
Number of POI in DataSet:
18
```



```
Number of Missing Values for Each Feature:
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
poi: 0
director_fees: 129
deferred_income: 97
long_term_incentive: 80
email_address: 35
from_poi_to_this_person: 60
```

Questions

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"]

The goal of this project is to use Machine Leaning to find trends in POI in the Enron Dataset to try to Identify them. **See above for Overview. There were 3 outliers in the Dataset. I handled these outliers by removing shown below.

```
def remove_outlier(dict_object, keys):
    """ removes list of outliers keys from dict object """
    for key in keys:
        dict_object.pop(key, 0)
```

The 3 outliers where:

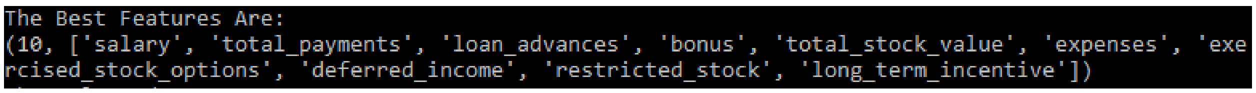
<u>Total</u>: Removed because it was total of all data points.

<u>The Travel Agency In The Park</u>: Removed because it was a data entry error, wasn't a person

<u>Lockhart Eugene E</u>: Removed because it only had NaN values.

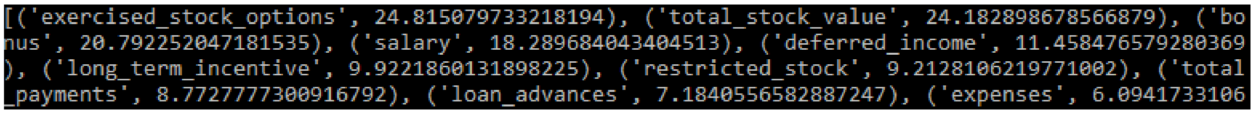
What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature 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"]

By using SelectKBest I came up with a list of 10 best features.



```
The Best Features Are:
(10, ['salary', 'total_payments', 'loan_advances', 'bonus', 'total_stock_value', 'expenses', 'exe
rcised_stock_options', 'deferred_income', 'restricted_stock', 'long_term_incentive'])
```

And their scores.

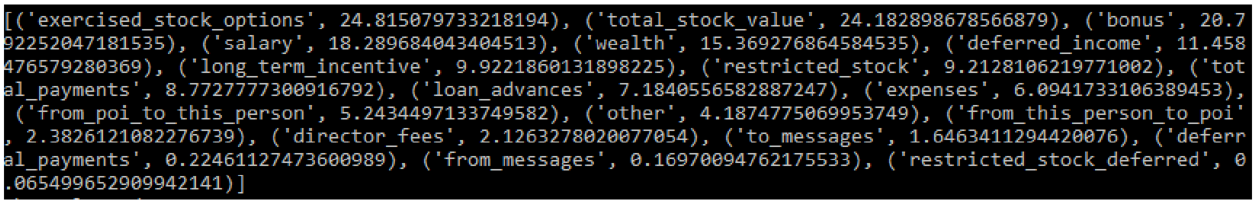


```
[('exercised_stock_options', 24.815079733218194), ('total_stock_value', 24.182898678566879), ('bo
nus', 20.792252047181535), ('salary', 18.289684043404513), ('deferred_income', 11.458476579280369
), ('long_term_incentive', 9.9221860131898225), ('restricted_stock', 9.2128106219771002), ('total
_payments', 8.7727777300916792), ('loan_advances', 7.1840556582887247), ('expenses', 6.0941733106
```

The feature I made was Wealth. It is the salary, total stock options and bonus added up.

```
for item in my_dataset:
    worker = my_dataset[item]
    if (all([worker['salary'] != 'NaN',
                worker['total_stock_value'] != 'NaN',
                worker['exercised_stock_options'] != 'NaN',
                worker['bonus'] != 'NaN'])):
        worker['wealth'] = sum([worker[field] for field in ['salary',
                                                             'total_stock_value',
                                                         'exercised_stock_options',
                                                                         'bonus']])
    else:
        worker['wealth'] = 'NaN'
```

I chose select K best. I played around with the k value and came to pick 10. Under 10 the Precision was great but the Recall was too low. Above 10 the Precision was to low and the recall was great. With 10 give or take 1 it evened out the Precision and Recall at there peaks.



```
[('exercised_stock_options', 24.815079733218194), ('total_stock_value', 24.182898678566879), ('bonus', 20.7
92252047181535), ('salary', 18.289684043404513), ('wealth', 15.369276864584535), ('deferred_income', 11.458
476579280369), ('long_term_incentive', 9.9221860131898225), ('restricted_stock', 9.2128106219771002), ('tot
al_payments', 8.7727777300916792), ('loan_advances', 7.1840556582887247), ('expenses', 6.0941733106389453),
 ('from_poi_to_this_person', 5.2434497133749582), ('other', 4.1874775069953749), ('from_this_person_to_poi'
, 2.3826121082276739), ('director_fees', 2.1263278020077054), ('to_messages', 1.6463411294420076), ('deferr
al_payments', 0.22461127473600989), ('from_messages', 0.16970094762175533), ('restricted_stock_deferred', 0
.065499652909942141)]
```

I scaled all features using MinMaxScaler.

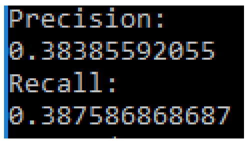
```
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)
```

I used feature scaling to make sure features would be weighed evenly with certain classifiers such as SVC. I also, used feature scaling because the features were significantly varied by magnitude.

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"]

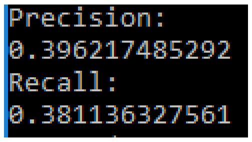
I ended up using **Logistic Regression.**

Before Wealth Feature:



```
Precision:
0.38385592055
Recall:
0.387586868687
```

After Wealth Feature:

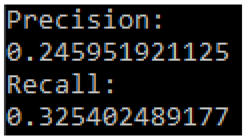


```
Precision:
0.396217485292
Recall:
0.381136327561
```

I also tried:

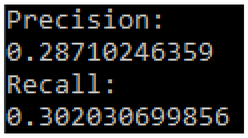
KMeans:

Before Wealth Feature:



```
Precision:
0.245951921125
Recall:
0.325402489177
```

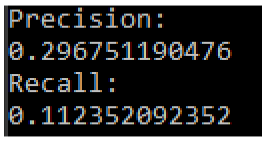
After Wealth Feature:



```
Precision:
0.28710246359
Recall:
0.302030699856
```

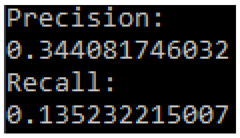
SVC:

Before Wealth Feature:



```
Precision:
0.296751190476
Recall:
0.112352092352
```

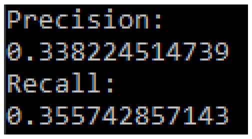
After Wealth Feature:



```
Precision:
0.344081746032
Recall:
0.135232215007
```

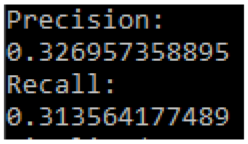
GaussianNB:

Before Wealth Feature:



```
Precision:
0.338224514739
Recall:
0.355742857143
```

After Wealth Feature:



```
Precision:
0.326957358895
Recall:
0.313564177489
```

It was a close call between Logistic Regression and GaussianNB. Behind them was KMeans and lastly was SVC.

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"]

Tuning an algorithm is when you change parameters of an algorithm. If you don't do this well you won't get the results your looking for. Scaling is typically required when using algorithms that rely on Euclidean (x,y) distance.

```
GaussianNB()
```

```
KMeans(n_clusters=2, tol=0.001)
```

KMeans uses Parameters *n_clusters* and *tol*. N_clusters is 2 for POI and non-POI, and the tolerance is .0001.

```
SVC(kernel='rbf', C=1000)
```

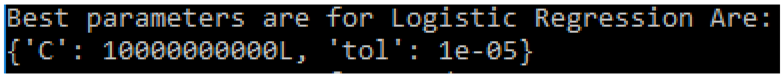
SVC uses parameters *kernel* and *C*. The Kernel is set to 'rbf' and the Penalty Parameter is set to 1000.

```
rf_clf = RandomForestClassifier(max_depth = 5,max_features = 'sqrt',n_estimators =
10, random state = 42)
```

Random Forest uses Parameters *max_depth, max_features, n_estimators*, and *random state*. Max_depth is the maximum depth of tree. Max_features is the number of features to consider when looking for best split. Random state is the seed used by random number generator. N_estimators is the number of trees in forest.

I checked the best parameters with GridSearchCV for Logistic Regression:

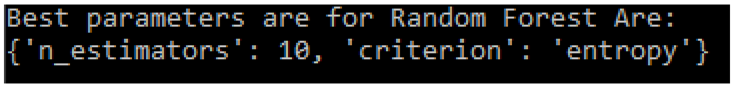
```
l_tuning_parameters = {'tol': [10**-1,10**-5,10**-10], 'C': [
0.5,1,10,10**5,10**10]}
LF = GridSearchCV(LogisticRegression(), l_tuning_parameters, scoring =
'recall')
LF.fit(features, labels)
print("Best parameters are for Logistic Regression Are:")
print(LF.best_params_)
```



```
Best parameters are for Logistic Regression Are:
{'C': 10000000000L, 'tol': 1e-05}
```

And also Random Forest:

```
r_tuning_parameters = {'n_estimators': [2,3,5,10], 'criterion':
['gini','entropy']}
RF = GridSearchCV(RandomForestClassifier(), r_tuning_parameters, scoring =
'recall')
RF.fit(features, labels)
print("Best parameters are for Random Forest Are:")
print(RF.best_params_)
```



```
Best parameters are for Random Forest Are:
{'n_estimators': 10, 'criterion': 'entropy'}
```

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"]

It gives estimate of performance on an independent dataset and it serves as a check on overfitting. I split the dataset with a 3:1 training to test ratio and took the average recall over 1000 trials.

```
def evaluate_clf(clf, features, labels, num_iters=1000, test_size=0.3):
    print clf
    accuracy = []
    precision = []
    recall = []
    first = True
    for trial in range(num_iters):
        features_train, features_test, labels_train, labels_test =\
            cross_validation.train_test_split(features, labels,test_size=test_size)
        clf.fit(features_train, labels_train)
        predictions = clf.predict(features_test)
        accuracy.append(accuracy_score(labels_test, predictions))
        precision.append(precision_score(labels_test, predictions))
        recall.append(recall_score(labels_test, predictions))

    print "Precision:"
    print (mean(precision))
    print "Recall:"
    print (mean(recall))
    return mean(precision), mean(recall)
```

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"]

I used Precision and Recall. As seen above you can see Logistic Regressions Precision is .3838 and its Recall is .3875. Precision is defined by true positives divided by total positives. With a Precision of .3838 that means about 38% of total positives are true positives. With a Recall of .3875 rounded to 39, means that it guesses 39% of all POI's. These numbers would offer a great start to an investigation into the persons who committed fraud at Enron.