Identify fraud from Enron mail

Project 5: Data Analyst Nanodegree

Jorge Santamaría Cruzado

February 25, 2016

1 Data Exploration

The data file I'm using is "final_project_dataset.pkl". Once the data is loaded I see it has 146 entries on it. From those 146 persons, only 18 are labeled as poi. Each row of the dataset contains 23 features apart from the poi label.

First I will check the min and max of each feature to see if there is something strange there:

```
def find_outlier(df, columns=None, fun='max'):
    # Prints the row name corresponding to running fun on the selected
    # column/s
    columns = df.keys() if not columns else columns
    for c in columns:
        try:
            f = getattr(df[c], fun)
            value = f()
            print(c, df.loc[df[c] == value].index.tolist()[0])
        except TypeError:
            pass
```

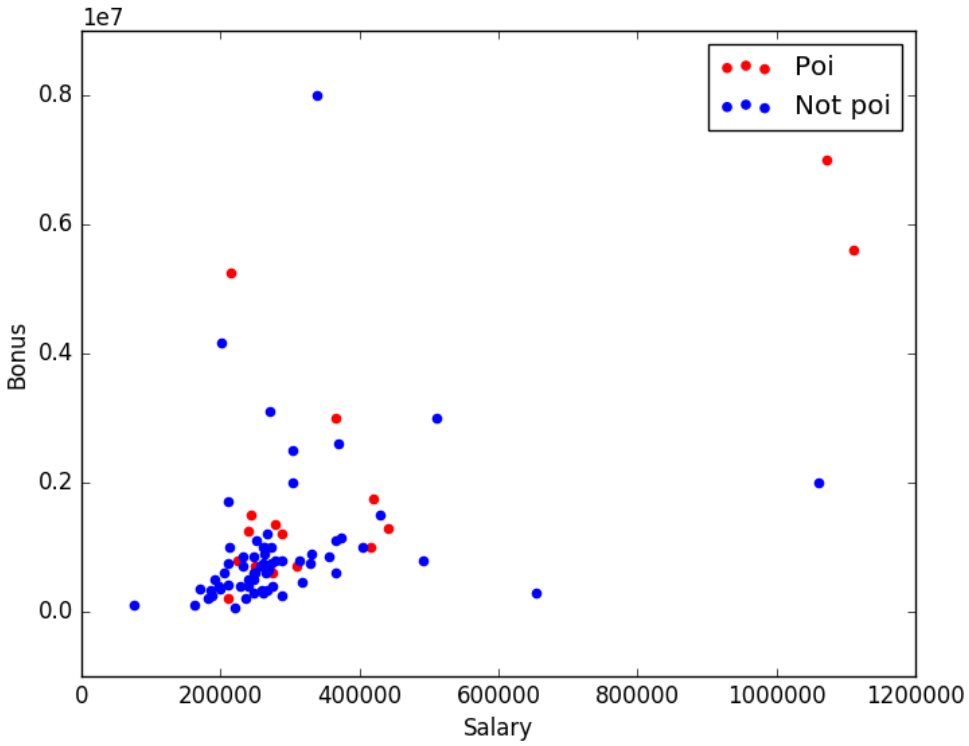
```
expenses TOTAL
exercised_stock_options TOTAL
shared_receipt_with_poi BELDEN TIMOTHY N
loan_advances TOTAL
poi_shared_proportion GLISAN JR BEN F
long_term_incentive TOTAL
director_fees TOTAL
restricted_stock_deferred BHATNAGAR SANJAY
deferred_income BOWEN JR RAYMOND M
poi BELDEN TIMOTHY N
restricted_stock TOTAL
deferral_payments TOTAL
from_messages KAMINSKI WINCENTY J
from_this_person_to_poi DELAINEY DAVID W
salary TOTAL
total_stock_value TOTAL
other TOTAL
to_messages SHAPIRO RICHARD S
total_payments TOTAL
bonus TOTAL
from_poi_to_this_person LAVORATO JOHN J
from_poi_proportion DONAHUE JR JEFFREY M
to_poi_proportion HUMPHREY GENE E
```

Looks like TOTAL wins on a lot of columns. Obviously I don't want that row on my data, so I'm removing it right away.

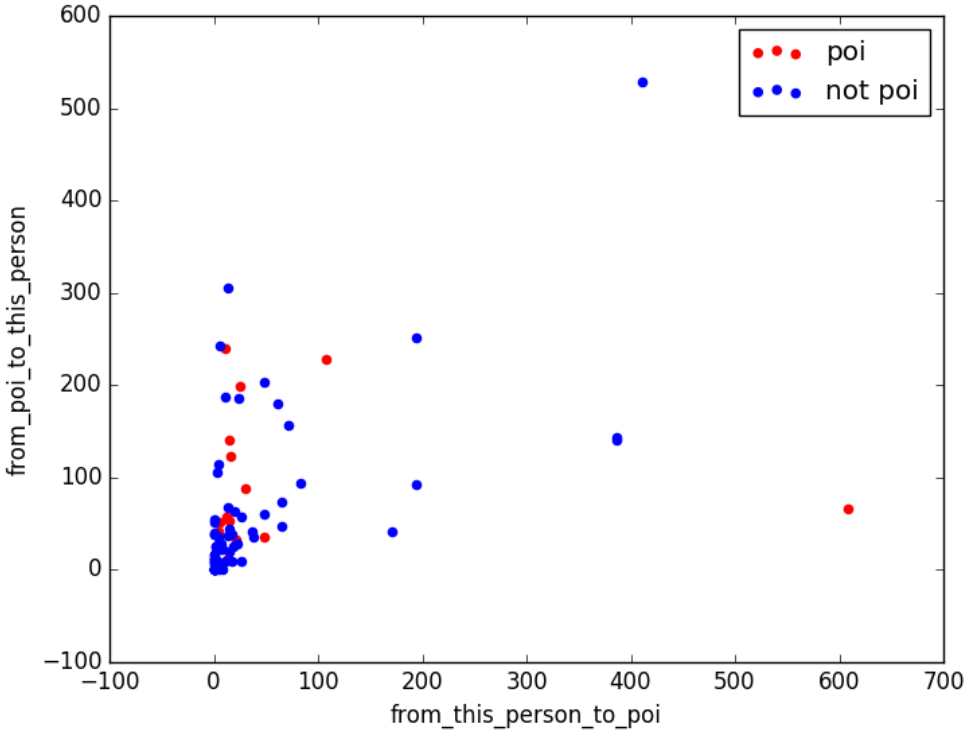
Plotting some quick scatterplots to have an overview of the data:

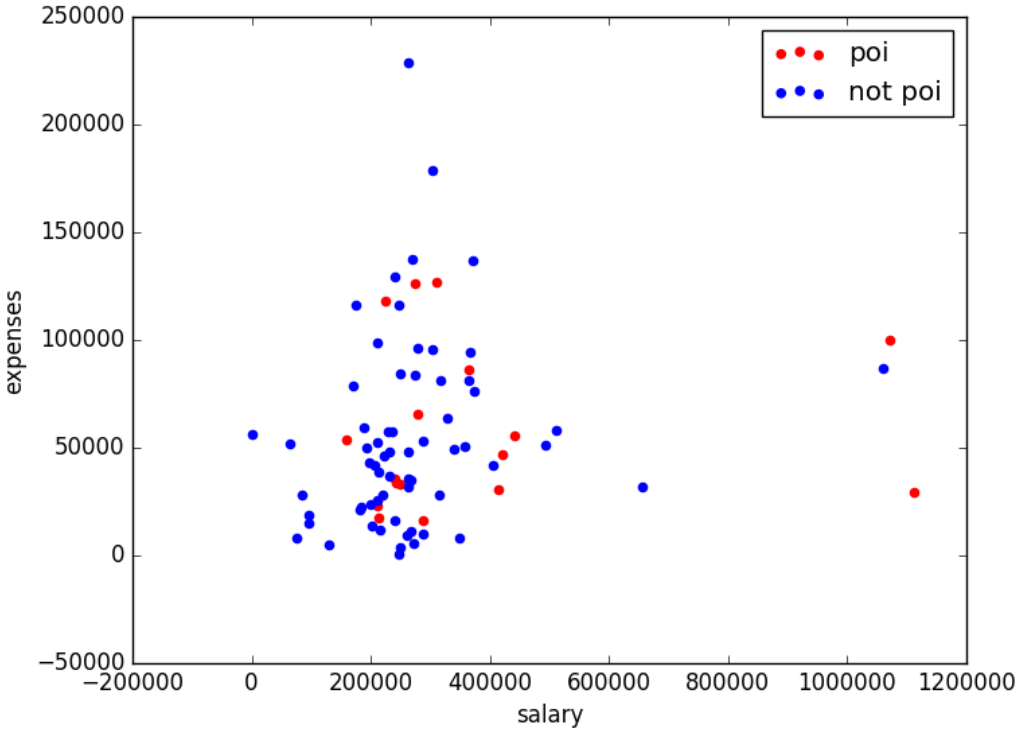
```
import matplotlib.pyplot as plt

def plot_pois(data, x, y):
    xp = [p[x] for p in data.values() if p['poi']]
    yp = [p[y] for p in data.values() if p['poi']]
    xnp = [p[x] for p in data.values() if not p['poi']]
    ynp = [p[y] for p in data.values() if not p['poi']]
    pois = plt.scatter(xp, yp, color='red')
    nopois= plt.scatter(xnp, ynp, color='blue')
    plt.legend(handles=[pois, nopois], labels=['poi', 'not poi'])
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()
```



Looks like there are still a few outliers there, however, running the find outlier function again, it looks like they are high manager from the company, so it looks coherent they have a much higher salary and bonus than the rest.


poi
not poi
from_poi_to_this_person
from_this_person_to_poi
600
500
400
300
200
100
0
−100
−100
0
100
200
300
400
500
600
700


poi
not poi
expenses
salary
250000
200000
150000
100000
50000
0
−50000
−200000
0
200000
400000
600000
800000
1000000
1200000

2 Features

There are a some NaN values in the dataset. However they are handled on the features script, so I will not process them here

First of all I will check the importance of each feature with SelectKBest:

```
selector = SelectKBest(k=19)
selector.fit(features, labels)
print(selector.scores_)
features = selector.transform(features)
```

```
('salary', 18.575703268041785),
('deferral_payments', 0.2170589303395084),
('total_payments', 8.8667215371077717),
('loan_advances', 7.2427303965360181),
('bonus', 21.060001707536571),
('restricted_stock_deferred', 0.06498431172371151),
('deferred_income', 11.595547659730601),
('total_stock_value', 24.467654047526398),
('expenses', 6.2342011405067401),
('exercised_stock_options', 25.097541528735491),
('other', 4.204970858301416),
('long_term_incentive', 10.072454529369441),
('restricted_stock', 9.3467007910514877),
('director_fees', 2.1076559432760908),
('to_messages', 1.6988243485808501),
('from_poi_to_this_person', 5.3449415231473374),
('from_messages', 0.16416449823428736),
('from_this_person_to_poi', 2.4265081272428781),
('shared_receipt_with_poi', 8.7464855321290802),
```

Now I will create two new features with the proportion of mails send and received from pois for each person. I will replace 'from_poi_to_this_person', 'from_this_person_to_poi', 'to_messages' and 'from_messages' by this two new features. This are the new scores for the features:

```
def divide(d, s1, s2):
    if isinstance(d[s1], int) and isinstance(d[s2], int):
        if d[s2] > 0:
            return d[s1]/d[s2]
    return 0

for k, v in my_dataset.items():
    my_dataset[k]['from_poi_proportion'] = divide(v, 'from_poi_to_this_person',
                                                  'to_messages')
    my_dataset[k]['to_poi_proportion'] = divide(v, 'from_this_person_to_poi',
                                                'from_messages')
    my_dataset[k]['poi_shared_proportion'] = divide(v,
                                                    'shared_receipt_with_poi',
                                                    'to_messages')
```

```
('bonus', 21.060001707536571)
('deferral_payments', 0.2170589303395084)
('deferred_income', 11.595547659730601)
('director_fees', 2.1076559432760908)
('exercised_stock_options', 25.097541528735491)
('expenses', 6.2342011405067401)
('from_poi_proportion', nan)
('loan_advances', 7.2427303965360181)
('long_term_incentive', 10.072454529369441)
('other', 4.204970858301416)
('poi_shared_proportion', 7.3088235294117645)
('restricted_stock', 9.3467007910514877)
('restricted_stock_deferred', 0.06498431172371151)
('salary', 18.575703268041785)
('to_poi_proportion', 0.14199999999999999)
('total_payments', 8.8667215371077717)
('total_stock_value', 24.467654047526398)
```

It looks like making that change of features is doing no good so I will stick to the original ones, and I will add poi_shared_proportion to them because it has the higher score of all the new ones.

The features deferral_payments, restricted_stock_deferred and from_messages looks like they are almost useless so I will try later if I can get a better score removing them.

Because I'm going to try a few algorithms, I will scale the rest of the data too. I will scale all the features train and test ones because I will use cross validation to check performance.

```
\label{code:scaling}
\caption{scaling}
  scaler = MinMaxScaler()
  scaler.fit(features)
  features = scaler.transform(features)
```

Now I will try the selected algorithm validation performance removing some of the features to check how many should I keep:

Scores with all the original features:

```
Precision 0.280555555556
Recall 0.388888888889
```

Scores with the original features plus shared_with_poi:

```
Precision 0.328703703704
Recall 0.333333333333
```

Scores removing the three features with less SelectKBest score:

```
Precision 0.134259259259
Recall 0.222222222222
```

Looks like the precision and recall drops a lot even when removing a simgle features, so I will keep all the original features plus poi_shared_proportion, the ones that are giving the best results.

3 Algorithm

By trying a few classifiers I have chosen to use a SVC. Although GaussianNB look like it's achieving the best results out of the box, it has less parameters to tune and it looks like a SVC once tuned can beat the scores of the rest.

```
clfs = {
    'bayes': GaussianNB(),
    'svc': SVC(C=3e5),
    'tree': DecisionTreeClassifier(random_state=43),
    'forest': RandomForestClassifier(random_state=43),
}

cv = cross_validation.KFold(len(features), n_folds=4, random_state=44)

for name in sorted(clfs.keys()):
    clf_curr = clfs[name]
    scores = cross_validation.cross_val_score(clf_curr, features, labels, cv=4)
    scores_precision = cross_validation.cross_val_score(clf_curr, features,
            labels, cv=cv, scoring='precision')
    scores_recall = cross_validation.cross_val_score(clf_curr, features,
            labels, cv=cv, scoring='recall')
    print('\n' + name)
    # print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
    print("Precision: %0.2f (+/- %0.2f)" % (scores_precision.mean(),
        scores_precision.std() * 2))
    print("Recall: %0.2f (+/- %0.2f)" % (scores_recall.mean(),
        scores_recall.std() * 2))
```

Now, I will run the selected SVC once with some common param values to see how it does, and then will run it a second time fine tuning a little more the parameters I get on the first iteration.

```
params = {
        'kernel': ['linear', 'rbf', 'poly'],
        'C': [1000.0, 3000.0, 10000.0, 30000.0, 100000.0,
              300000.0, 1000000.0, 3000000.0],
        'degree': [3, 4],
        'decision_function_shape': ['ovo']
        }

grid = GridSearchCV(estimator=SVC(), param_grid=params, cv=10, scoring='recall')
classifier = grid.best_estimator_
```

For kernel I suspect the best one will be rbf, but I will try the rest just in case. The C parameter if the penalty of the error and I will fine tune it on the second iteration depending on what value I get on the first. The degree if the degree of the poly kernel, the default is 3 but I want to test also 4. The decision shape will be ovo, that's one versus one, the other option would be one vs rest but there's only two classes here, so they will behave the same.

The best C value here is 3e5, I will try to fit it a little better. One I run it again, the best value for C is 2.6e5 so that's the one I will use from now.

Finally I will validate my algorithm using cross validation to be sure it will work on the test script:

```
s_p = cross_validation.cross_val_score(classifier, features, labels,
```

```
                                        cv=6, scoring='precision')
s_r = cross_validation.cross_val_score(classifier, features, labels,
                                        cv=6, scoring='recall')

print(s_p.mean())
print(s_r.mean())
```

Both values are above 0.3, so the classifier is ready to pass the test.

4 Questions

4.1 Summary

The goal of this project was to separate pois from non pois from the enron dataset, machine learning is useful for that due to the high dimensionality of the data, there is no way to plot all that dimensions and find a pattern by naked eye, but machine learning is very good on it.

The dataset contains all the emails send and received by the company employees and financial data for most of them.

4.2 Features

I handcrafted three more features containing the proportion of emails send and received from pois. I think this is more important than the raw number, because is not the same someone having received one email from a poi, if that is the only email he has received or he has another 1000 emails from non pois on his inbox..

However it looked that the new features didn't were good so I kept all the original and added the most promising one from the new features.

I also scaled all of them to use with my SVC.

4.3 Algorithm

I finally decided to use a SVC, it looked to me that it was the one that will achieve best precision and recall scores once tuned correctly. I also tried a RandomForestClassifier, GaussianNB and DecisionTreeClassifier.

4.4 Tuning

Tuning is adjusting the algorithms parameters to get better performance. In my case (SVC) that includes the penalty value for the errors, the kernel to be used, the degree of the poly kernel and the decision function shape which has proven to be useless here because there are only two classes.

4.5 Validation

Validation is measure the algorithm performance to check how it generalizes with new data that was not used on training.

One classic mistake is testing on the training data. The way to avoid it I'm using here is use cross validation. This is also more useful because there is very few rows labeled as pois compared to the full dataset.

4.6 metrics

I have used precision (from the number of predicted pois the proportion of them with are actually right) and recall(from the number of all pois in dataset, the proportion of then that are correctly predicted by the algorithm)

On my first approaches I also used f1 score that puts together the precision and recall score, but then I switched to use both of them lately