Intro to Machine Learning

Udacity nanodegree Project 5

Diego Menin – September 2015

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.

In this project, I'll create a "person of interest" (POI) identifier based on financial and email data made public as a result of the Enron scandal, whose goal is, given a person, identify whether he or she is a "POI", in other words, if It should be investigated for fraud or not.

This analysis is divided in the following sections:

- The dataset: Will explain the dataset, the number of observations and measures;
- Outlier removal: Will go through the outlier detection and removal process;
- New Features: Will explain the new features added;
- Model evaluation: Will clarify the model performance evaluation process, especially the challenges with such unbalanced dataset;
- Feature Selection: Will explain how the features were selected to be part of the final model;
- Model build: Will go through the model build process;
- Final Considerations: Final considerations on the dataset;

The dataset:

The dataset was provided by the Udacity team and consists of 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. It contains 145 observations, 18 of each are considered POIs. Each observation contains 14 measures that indicates for example, how much was the person's salary or bonus. Due to its length, I included the complete list of the measures and its explanation on the "final considerations" section.

Outlier removal:

The outlier removal was an iterative process. Since we are not dealing with normally distributed data, is pretty impossible to define what is or is not an outlier. We'd need an assumed distribution in order to be able to classify something as "lying outside the range of expected values". Because of that, techniques like "the top 10% higher regression error" or "Interquartile range" cannot be blindly applied. So I used a mix of the latest ("Interquartile range") to suggest possible outliers and them analysed the data visually to verify if they should or should not be considered outliers.

To make that suggestion, I wrote a function that, for each column of a data frame, checks the values that are considered outliers based on the IQR rule. An IQR, or interquartile range, is the difference between the first and third quartiles of data. It's one measure of spread — how far data is spread around the mean. According to this rule, observations that fall below Q1 – "g" x (IQR), or above Q3 + "g" x (IQR) are identified as potential outliers. The value of "g" is normally 1.5. I chose to use 2.2 due to the explanation found on this YouTube video.

Very soon, I realized that the IQR rule on full dataset the wasn't such a good idea due to the high number of zeros on the columns (which were originally NaNs) so I decided to exclude them from the analysis (in nother words, the IQR rule was applied disconsidering the zeros from the calculation – I understand that is not the best technique but again, I was looking for suggestions, not a deterministic way of removing the outliers).

All the analysis is done with the "*get_outliers_list*" function documented on the *poy_id.py* file that outputs a dictionary where the key is the column and the value is a list of suggestions. The output is the following:

```
Column: salary - Number of suggested outliers (ignoring 0s): 5 Indices: [47, 79, 104, 121, 129]
Column: to_messages - Number of suggested outliers (ignoring 0s): 6 Indices: [6, 7, 73, 75, 78, 116]
Column: deferral_payments - Number of suggested outliers (ignoring 0s): 2 Indices: [47, 129]
Column: total_payments - Number of suggested outliers (ignoring 0s): 7 Indices: [11, 47, 78, 79, 85, 121, 129]
Column: long_term_incentive - Number of suggested outliers (ignoring 0s): 3 Indices: [79, 85, 129]
Column: loan_advances - Number of suggested outliers (ignoring 0s): 0 Indices: []
Column: bonus - Number of suggested outliers (ignoring 0s): 9 Indices: [0, 7, 31, 75, 78, 79, 121, 129, 138]
Column: restricted_stock_deferred - Number of suggested outliers (ignoring 0s): 0 Indices: []
Column: total_stock_value - Number of suggested outliers (ignoring 0s): 11 Indices: [3, 32, 47, 65, 79, 102, 111, 121, 129, 139, 143]
Column: expenses - Number of suggested outliers (ignoring 0s): 3 Indices: [87, 129, 131]
Column: exercised_stock_options - Number of suggested outliers (ignoring 0s): 11 Indices: [32, 35, 47, 65, 79, 102, 109, 111, 121, 129, 143]
Column: from_messages - Number of suggested outliers (ignoring 0s): 17 Indices: [0, 6, 7, 19, 31, 32, 58, 62, 66, 72, 73, 75, 78, 88, 115, 116, 138]
Column: other - Number of suggested outliers (ignoring 0s): 9 Indices: [3, 47, 69, 79, 85, 102, 118, 120, 129]
Column: deferred_income - Number of suggested outliers (ignoring 0s): 0 Indices: []
Column: restricted_stock - Number of suggested outliers (ignoring 0s): 12 Indices: [3, 47, 69, 73, 79, 102, 111, 121, 129, 138, 139, 143]
Column: director_fees - Number of suggested outliers (ignoring 0s): 5 Indices: [8, 93, 107, 129, 131]
```

Alternatively the "get_unique_outliers" function can be used to output a list on unique suggestions across all features.

```
Unique Suggestions: [47, 79, 104, 121, 129, 6, 7, 73, 75, 78, 116, 66, 11, 85, 0, 31, 138, 3,
32, 65, 102, 111, 139, 143, 87, 131, 35, 109, 19, 58, 62, 72, 88, 115, 69, 118, 120, 8, 93, 107]
```

Unfortunately it suggest 40 unique indexes, meaning that, 40 out of 145 would be considered outliers based on the IQR rule (that makes sense because there are people with huge salaries and bonuses and that may just represent the company's top executives). That's not good so I went to the second part of the analysis:

First, by just looking at the names on the dictionary I could exclude 2 key form it: "TOTAL" (which contains the total of the reports – not a valid data point) and "THE TRAVEL AGENCY IN THE PARK" (which doesn't seem to be a person – and contain mainly zeroes)

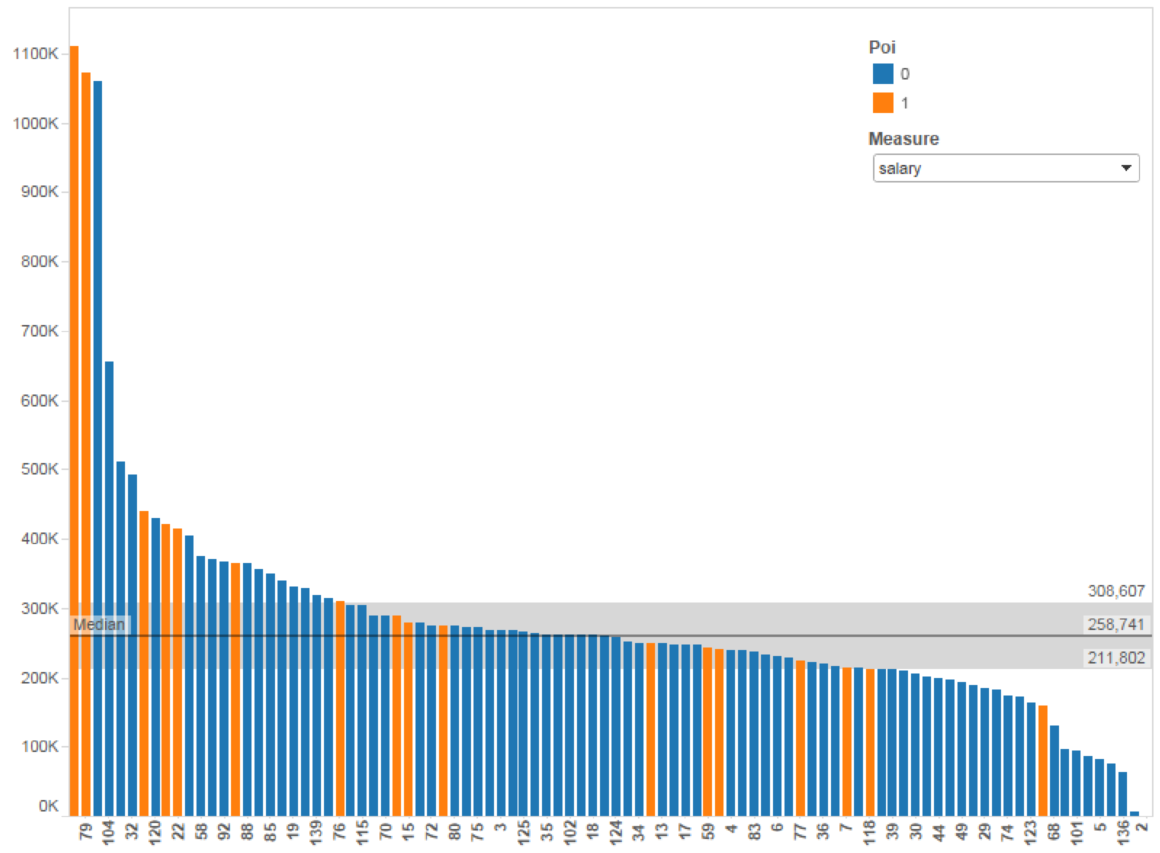
Second, looking at values on the dictionary, I found 'LOCKHART EUGENE E'', which doesn't contain any information (all values are NaN). The "*featureFormat*" function (that will be applied in the future) will remove it, so no action required.

Third, by plotting the data, I could very easily see 2 things:

- whether the indicated values were outliers or not and;
- some sort of indication on where I should or not use the feature in question on my analysis;

Important note: Since this is not a visualization project, I used the public version of an external tool called Tableau to help on the visual analysis (rather than using Python). The dashboard I built is deployed on Tableau Public. I also haven't included any person's names, I used their index on the dictionary. Here are my conclusions:

- On the fields Salary, to_messages, deferral_payments, long_term_incentive, bonus, expenses, other, deferred_income, restricted_stock, the distribution is skewed but the values identified as outliers aren't necessary outliers. For example, if we look at a bar plot of salaries, the top 5 salaries aren't necessarily outliers, they simply reflect people with much higher salaries.



- **total_payments:** Index number 79 has a value of 103,559,793 (the biggest of the dataset, much bigger than the second biggest - 17,252,530) but since it is a POI (LAY KENNETH L), I decided to keep it; Also, this feature is the sum of several other features (see the feature description, so it probably won't have much prediction power)
- **loan_advances:** only has 3 values different than zero so I decided not to do anything. This is also a good candidate to be excluded from the prediction model;
- **restricted_stock_deferred:** no outliers suggested;
- **total_stock_value:** I believe this is a very good measure for the prediction algorithm because from the top 5 values, 4 are POIS, so it will probably have quite a significant weight;
- **exercised_stock_options:** also a good measure for the prediction. The top 4 are POIS;
- **from_messages:** very skewed, probably a bad candidate for prediction. From the top 16, only one poi;
- **director_fees:** only 16 values greater different than zero, all non-poi;

In the end, given the very small number of observations, I decide not to exclude any other data points.

New Features:

Given the nature of the dataset, I didn't think I could benefit from any new features, nor that there were features to be derived from the features I chose to keep so far. Nevertheless, I decided to give it a try by creating a "ratio" between Bonus and Salary.

Model evaluation:

Evaluation method:

The model will be evaluated using the "*sklearn.cross_validation.StratifiedShuffleSplit*" function with a 1000 iterations. For each of these iterations of the splitting, the function provides train/test indices to split data into training and test sets.

The model is then built on the training set and tested on the test set and the amount of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) are summed over all 1000 iterations, which allows the computation of 5 evaluation metrics: Accuracy, precision, recall, f1 and f2. The definition of all these metrics can be found here.

Choosing the evaluation metric:

This is a pretty unbalanced data set so a lot of care was exercised when selecting the best evaluation metric. Initially I started evaluating my models using "accuracy" but I soon realized that it isn't a good measure due to the low number of POIs we have.

Since accuracy is defined by number of TP + TN divided by the total number of predictions, if we guess "non-poi" for everyone, we'll get an accuracy of 87% (which seems pretty good but clearly represents a very poor model). Accuracy by itself, especially for an unbalanced dataset, does not provide adequate information about the classifier's performance and can also be deceiving (refer to the Accuracy Paradox).

Considering we are building a model that, given a person, decides whether he or she is a "POI", in other words, if we should investigate him\her for fraud or not, we'd want to penalize "false negatives" more than "false positives". In other words, it's worst if our model tells us to investigate a person who in the end is not POI (false positive), than not investigate a person that's actually a POI (false negative).

Based on that, I thought I'd use the "recall" measure that represent the ration between TP and (TP + FN). So in this case, since the number of false negatives is on the denominator, a high number of false negatives would bring the recall metric down.

But very quickly I found that there's also ways in which this metrics can be deficient. If we just say that everyone is a POI, then our recall will be 1 since we've successfully marked all of the POIs. Of course, our false positive rate will also be extremely high, and our precision (and accuracy, as you've observed) will suffer.

So, in the end, I decided to focus on metrics that takes the FP and FN balance into account, such as the F-measures. The F1 score is an equal weighting between precision and recall performance, while other F-measures can emphasize precision or recall performance as desired by the task at hand.

Feature Selection:

From all the features available, these initial steps were performed:

- Ignored the 'email_address' feature: unique individual email, no predictive power;
- Ignored the following features: "from_poi_to_this_person", "from_this_person_to_poi", "shared_receipt_with_poi": these features represent the number of emails sent from and to a POI and the number of receipts shared with a POI. There is an extensive discussion on the Udacity forum regarding the validity of these feature, mainly related to the fact that we are trying to predict POI so we should have available features that were calculated based on already labelled POIS.

There are arguments pro and against these feature that were extensively discussed on the forums so I won't get into much detail, I tend to agree that they may constitute of Data Leakage on the model so I chose not to use them.

- Ignored the loan_advances feature: because it contains only 3 non zero values;

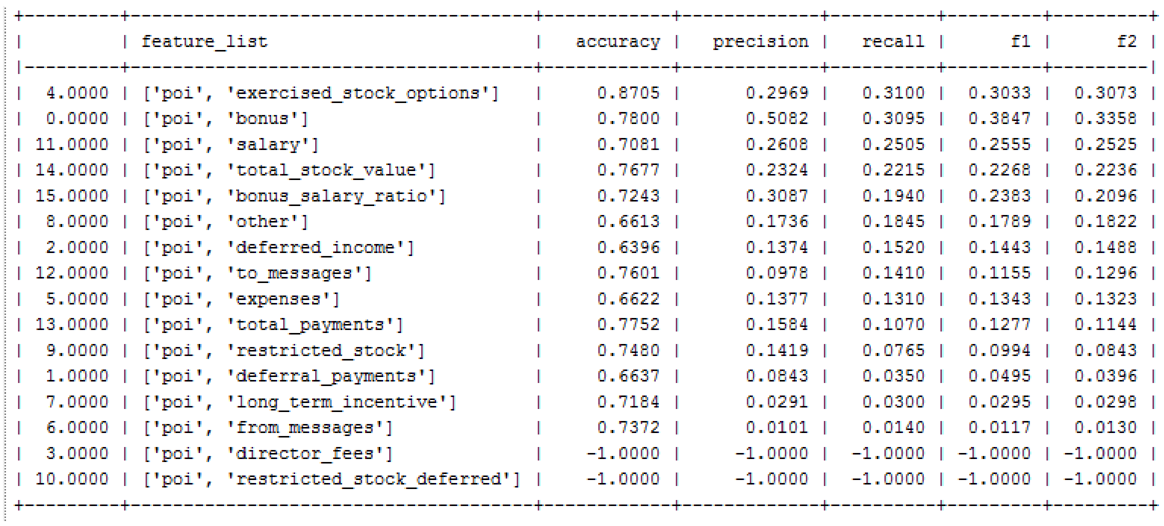
The next step was to run the "*sklearn.feature_selection.SelectKBest*" function to output the best features from our list and, from that list, select the top 5 (the reason I chose 5 was after analysing the algorithms on further steps. Basically, any feature on this list with a score less than 18 won't have enough predictive power – that includes the ratio I created on the previous step). That selection is done on the "*SelectKBestFeatures*" function.

```
('exercised_stock_options', 24.815079733218194),
('total_stock_value', 24.182898678566879),
('bonus', 20.792252047181535),
('salary', 18.289684043404513),
('deferred_income', 11.458476579280369),
('bonus_salary_ratio', 10.785573495441602),
('long_term_incentive', 9.9221860131898225),
('restricted_stock', 9.2128106219771002),
('total_payments', 8.7727777300916756),
('loan_advances', 7.1840556582887247),
('expenses', 6.0941733106389453),
('other', 4.1874775069953749),
('director_fees', 2.1263278020077054),
('to_messages', 1.6463411294420076),
('deferral_payments', 0.22461127473600989),
('from_messages', 0.16970094762175533),
('restricted_stock_deferred', 0.065499652909942141)
```

So the features selected at this point were: 'exercised_stock_options', 'total_stock_value', 'bonus', 'salary' and 'deferred_income' (decided to add deferred income as well). FYI: I didn't just decide to blindly exclude the remaining functions just based on this score. I did try to build model using all of them, it just turned out that they are not good as the score represents)

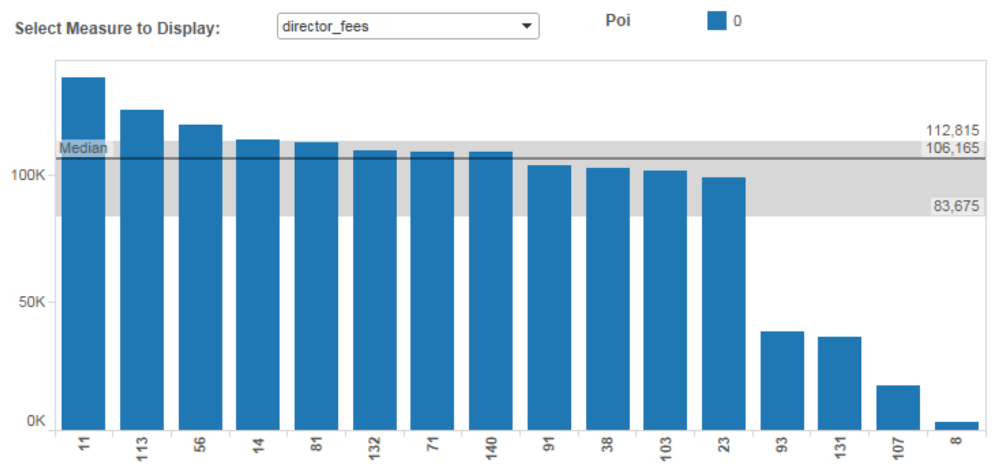
Model build:

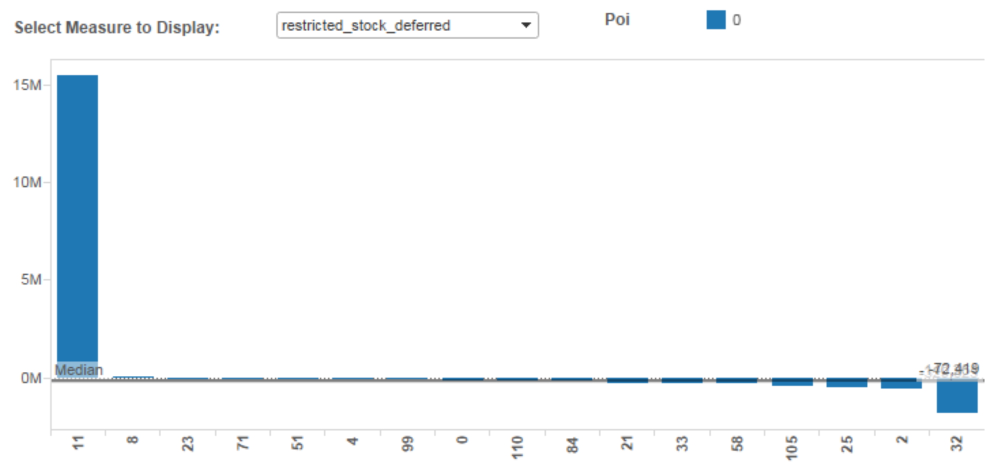
To have a baseline of measures, my first step was try to predict "poi" by running a simple decision tree using each one of the feature individually. That was done on the "*one_feature_predict*" function.



```
+---------+-------------------------------------+------------+-------------+-----------+---------+---------+
|         | feature_list                        |   accuracy |   precision |    recall |      f1 |      f2 |
|---------+-------------------------------------+------------+-------------+-----------+---------+---------|
|  4.0000 | ['poi', 'exercised_stock_options']  |     0.8705 |      0.2969 |    0.3100 |  0.3033 |  0.3073 |
|  0.0000 | ['poi', 'bonus']                    |     0.7800 |      0.5082 |    0.3095 |  0.3847 |  0.3358 |
| 11.0000 | ['poi', 'salary']                   |     0.7081 |      0.2608 |    0.2505 |  0.2555 |  0.2525 |
| 14.0000 | ['poi', 'total_stock_value']        |     0.7677 |      0.2324 |    0.2215 |  0.2268 |  0.2236 |
| 15.0000 | ['poi', 'bonus_salary_ratio']       |     0.7243 |      0.3087 |    0.1940 |  0.2383 |  0.2096 |
|  8.0000 | ['poi', 'other']                    |     0.6613 |      0.1736 |    0.1845 |  0.1789 |  0.1822 |
|  2.0000 | ['poi', 'deferred_income']          |     0.6396 |      0.1374 |    0.1520 |  0.1443 |  0.1488 |
| 12.0000 | ['poi', 'to_messages']              |     0.7601 |      0.0978 |    0.1410 |  0.1155 |  0.1296 |
|  5.0000 | ['poi', 'expenses']                 |     0.6622 |      0.1377 |    0.1310 |  0.1343 |  0.1323 |
| 13.0000 | ['poi', 'total_payments']           |     0.7752 |      0.1584 |    0.1070 |  0.1277 |  0.1144 |
|  9.0000 | ['poi', 'restricted_stock']         |     0.7480 |      0.1419 |    0.0765 |  0.0994 |  0.0843 |
|  1.0000 | ['poi', 'deferral_payments']        |     0.6637 |      0.0843 |    0.0350 |  0.0495 |  0.0396 |
|  7.0000 | ['poi', 'long_term_incentive']      |     0.7184 |      0.0291 |    0.0300 |  0.0295 |  0.0298 |
|  6.0000 | ['poi', 'from_messages']            |     0.7372 |      0.0101 |    0.0140 |  0.0117 |  0.0130 |
|  3.0000 | ['poi', 'director_fees']            |    -1.0000 |     -1.0000 |   -1.0000 | -1.0000 | -1.0000 |
| 10.0000 | ['poi', 'restricted_stock_deferred'] |    -1.0000 |     -1.0000 |   -1.0000 | -1.0000 | -1.0000 |
+---------+-------------------------------------+------------+-------------+-----------+---------+---------+
```

As expected, the features highlighted on the previous step have the best predictive power, even though it is not good enough. The -1 values represent a failure on the measure calculation, and this is due to the fact that for director_fees and restricted_stock_deferred we only have non-poi values:





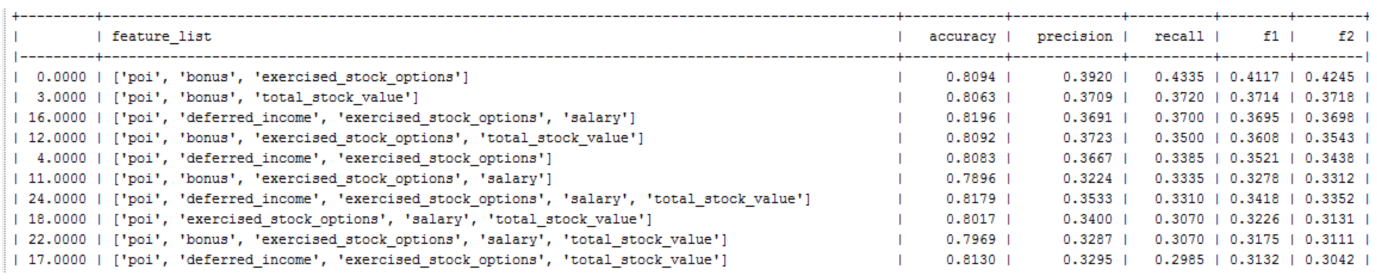
The next steps were very iterative. I ran a series of tests and algorithms trying to maximize the F1 score without scarifying the other measures too much. It was at this step I realized that 5 was the ideal number of features.

Since our dataset is quite small, I decided to take a brute force approach while fitting the models. I wrote a function called "*topk_feature_predict*" that, given a list of features (output from *SelectKBestFeatures*), produces all possible combinations from those features from 2 to n (n being the number features on the list).

By using the function I was able to fit several different models (most of them are still commented on the code) until I got to a "reasonable" one. Here are some examples of the function's output:

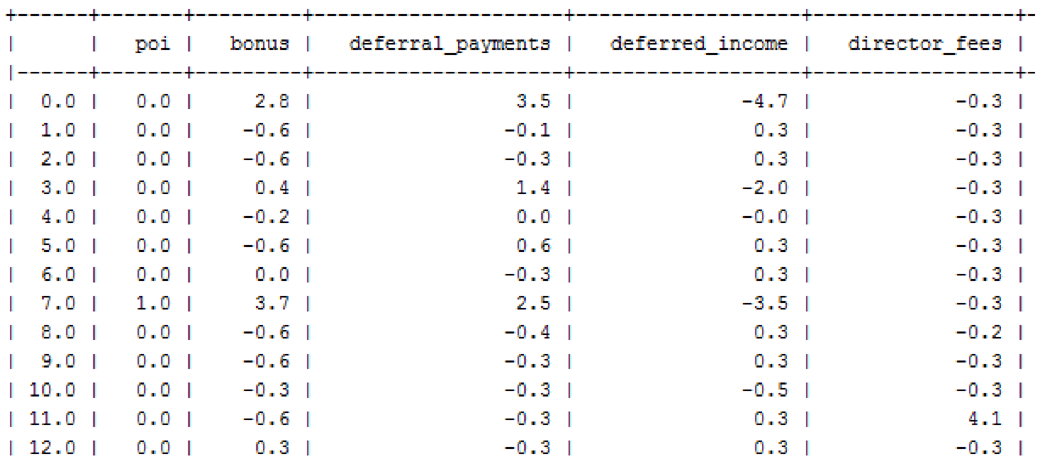
Simple tree model:

clf = tree.DecisionTreeClassifier(min_samples_split = 4)



```
+---------+------------------------------------------------------------------------------------------+------------+------------+----------+--------+--------+
|         | feature_list                                                                             |  accuracy |  precision |   recall |     f1 |     f2 |
|---------+------------------------------------------------------------------------------------------+------------+------------+----------+--------+--------|
|  0.0000 | ['poi', 'bonus', 'exercised_stock_options']                                              |     0.8094 |     0.3920 |   0.4335 | 0.4117 | 0.4245 |
|  3.0000 | ['poi', 'bonus', 'total_stock_value']                                                    |     0.8063 |     0.3709 |   0.3720 | 0.3714 | 0.3718 |
| 16.0000 | ['poi', 'deferred_income', 'exercised_stock_options', 'salary']                          |     0.8196 |     0.3691 |   0.3700 | 0.3695 | 0.3698 |
| 12.0000 | ['poi', 'bonus', 'exercised_stock_options', 'total_stock_value']                         |     0.8092 |     0.3723 |   0.3500 | 0.3608 | 0.3543 |
|  4.0000 | ['poi', 'deferred_income', 'exercised_stock_options']                                    |     0.8083 |     0.3667 |   0.3385 | 0.3521 | 0.3438 |
| 11.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary']                                    |     0.7896 |     0.3224 |   0.3335 | 0.3278 | 0.3312 |
| 24.0000 | ['poi', 'deferred_income', 'exercised_stock_options', 'salary', 'total_stock_value']     |     0.8179 |     0.3533 |   0.3310 | 0.3418 | 0.3352 |
| 18.0000 | ['poi', 'exercised_stock_options', 'salary', 'total_stock_value']                        |     0.8017 |     0.3400 |   0.3070 | 0.3226 | 0.3131 |
| 22.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary', 'total_stock_value']               |     0.7969 |     0.3287 |   0.3070 | 0.3175 | 0.3111 |
| 17.0000 | ['poi', 'deferred_income', 'exercised_stock_options', 'total_stock_value']               |     0.8130 |     0.3295 |   0.2985 | 0.3132 | 0.3042 |
```

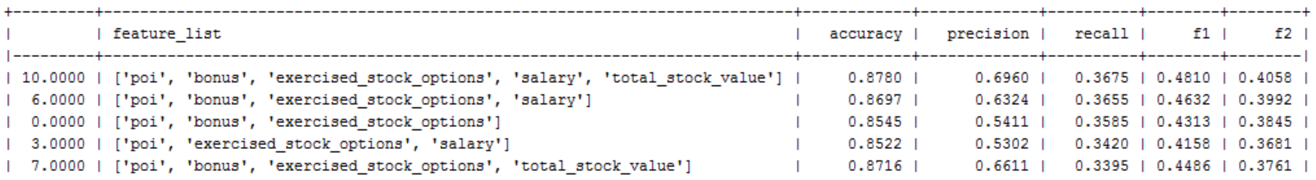
As we can see the result is a little better, but not by much. The function also allows for data normalization (subtracting the mean and dividing by the standard deviation) by setting the "*normalize_data*" parameter to true, which I tested on every scenario discussed, but unfortunately it didn't help. Here's how the normalized data looks like:



```
+------+-------+---------+--------------------+-------------------+-----------------+-
|      |  poi  |  bonus  |  deferral_payments  |  deferred_income  |  director_fees  |
|------+-------+---------+--------------------+-------------------+-----------------+-
|  0.0 |   0.0 |     2.8 |                 3.5 |              -4.7 |            -0.3 |
|  1.0 |   0.0 |    -0.6 |                -0.1 |               0.3 |            -0.3 |
|  2.0 |   0.0 |    -0.6 |                -0.3 |               0.3 |            -0.3 |
|  3.0 |   0.0 |     0.4 |                 1.4 |              -2.0 |            -0.3 |
|  4.0 |   0.0 |    -0.2 |                 0.0 |              -0.0 |            -0.3 |
|  5.0 |   0.0 |    -0.6 |                 0.6 |               0.3 |            -0.3 |
|  6.0 |   0.0 |     0.0 |                -0.3 |               0.3 |            -0.3 |
|  7.0 |   1.0 |     3.7 |                 2.5 |              -3.5 |            -0.3 |
|  8.0 |   0.0 |    -0.6 |                -0.4 |               0.3 |            -0.2 |
|  9.0 |   0.0 |    -0.6 |                -0.3 |               0.3 |            -0.3 |
| 10.0 |   0.0 |    -0.3 |                -0.3 |              -0.5 |            -0.3 |
| 11.0 |   0.0 |    -0.6 |                -0.3 |               0.3 |             4.1 |
| 12.0 |   0.0 |     0.3 |                -0.3 |               0.3 |            -0.3 |
```

<u>K Nearest neighbours:</u>

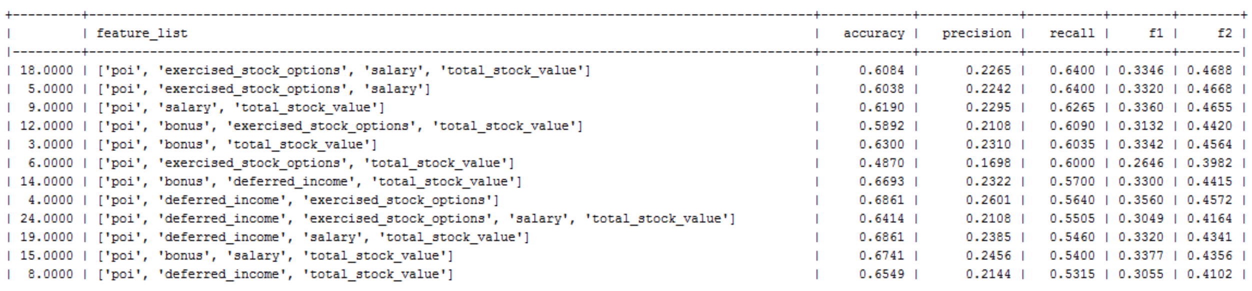
clf = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=5, p=2, weights='distance')



```
+---------+-----------------------------------------------------------------------+-----------+------------+----------+--------+--------+
|         | feature_list                                                          |  accuracy |  precision |   recall |     f1 |     f2 |
|---------+-----------------------------------------------------------------------+-----------+------------+----------+--------+--------|
| 10.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary', 'total_stock_value'] |    0.8780 |     0.6960 |   0.3675 | 0.4810 | 0.4058 |
|  6.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary']                |    0.8697 |     0.6324 |   0.3655 | 0.4632 | 0.3992 |
|  0.0000 | ['poi', 'bonus', 'exercised_stock_options']                          |    0.8545 |     0.5411 |   0.3585 | 0.4313 | 0.3845 |
|  3.0000 | ['poi', 'exercised_stock_options', 'salary']                         |    0.8522 |     0.5302 |   0.3420 | 0.4158 | 0.3681 |
|  7.0000 | ['poi', 'bonus', 'exercised_stock_options', 'total_stock_value']     |    0.8716 |     0.6611 |   0.3395 | 0.4486 | 0.3761 |
```

<u>Logistic Regression:</u>

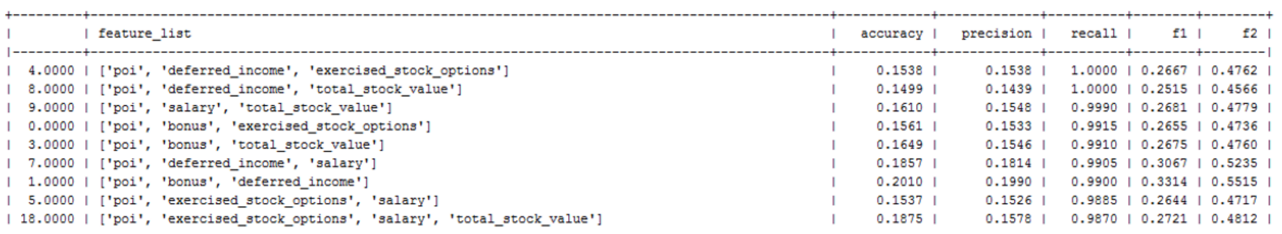
With Logistic Regression I was able to greatly improve the model's recall but of course, that impacted precision:



```
+---------+------------------------------------------------------------------------------------+-----------+------------+----------+--------+--------+
|         | feature_list                                                                       |  accuracy |  precision |   recall |     f1 |     f2 |
|---------+------------------------------------------------------------------------------------+-----------+------------+----------+--------+--------|
| 18.0000 | ['poi', 'exercised_stock_options', 'salary', 'total_stock_value']                  |    0.6084 |     0.2265 |   0.6400 | 0.3346 | 0.4688 |
|  5.0000 | ['poi', 'exercised_stock_options', 'salary']                                       |    0.6038 |     0.2242 |   0.6400 | 0.3320 | 0.4668 |
|  9.0000 | ['poi', 'salary', 'total_stock_value']                                             |    0.6190 |     0.2295 |   0.6265 | 0.3360 | 0.4655 |
| 12.0000 | ['poi', 'bonus', 'exercised_stock_options', 'total_stock_value']                   |    0.5892 |     0.2108 |   0.6090 | 0.3132 | 0.4420 |
|  3.0000 | ['poi', 'bonus', 'total_stock_value']                                              |    0.6300 |     0.2310 |   0.6035 | 0.3342 | 0.4564 |
|  6.0000 | ['poi', 'exercised_stock_options', 'total_stock_value']                            |    0.4870 |     0.1698 |   0.6000 | 0.2646 | 0.3982 |
| 14.0000 | ['poi', 'bonus', 'deferred_income', 'total_stock_value']                           |    0.6693 |     0.2322 |   0.5700 | 0.3300 | 0.4415 |
|  4.0000 | ['poi', 'deferred_income', 'exercised_stock_options']                              |    0.6861 |     0.2601 |   0.5640 | 0.3560 | 0.4572 |
| 24.0000 | ['poi', 'deferred_income', 'exercised_stock_options', 'salary', 'total_stock_value'] |    0.6414 |     0.2108 |   0.5505 | 0.3049 | 0.4164 |
| 19.0000 | ['poi', 'deferred_income', 'salary', 'total_stock_value']                          |    0.6861 |     0.2385 |   0.5460 | 0.3320 | 0.4341 |
| 15.0000 | ['poi', 'bonus', 'salary', 'total_stock_value']                                    |    0.6741 |     0.2456 |   0.5400 | 0.3377 | 0.4356 |
|  8.0000 | ['poi', 'deferred_income', 'total_stock_value']                                    |    0.6549 |     0.2144 |   0.5315 | 0.3055 | 0.4102 |
```

In fact, by setting a very low value on the "C" parameter, I was able to get 100% of recall, but again, precision was really low:

*clf = LogisticRegression(C=0.1,penalty='l1',random_state=42,tol=10**-10,class_weight='auto')*



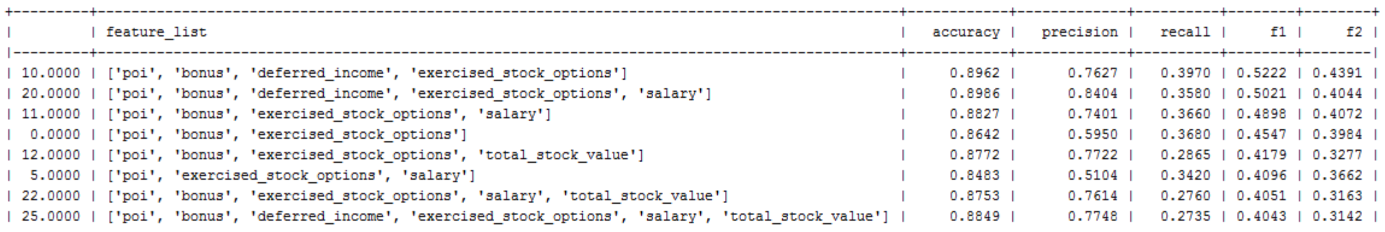
```
+---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------+
|         | feature_list                                                                            |  accuracy  | precision  |  recall  |   f1   |   f2   |
|---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------|
|  4.0000 | ['poi', 'deferred_income', 'exercised_stock_options']                                   |     0.1538 |     0.1538 |   1.0000 | 0.2667 | 0.4762 |
|  8.0000 | ['poi', 'deferred_income', 'total_stock_value']                                         |     0.1499 |     0.1439 |   1.0000 | 0.2515 | 0.4566 |
|  9.0000 | ['poi', 'salary', 'total_stock_value']                                                  |     0.1610 |     0.1548 |   0.9990 | 0.2681 | 0.4779 |
|  0.0000 | ['poi', 'bonus', 'exercised_stock_options']                                             |     0.1561 |     0.1533 |   0.9915 | 0.2655 | 0.4736 |
|  3.0000 | ['poi', 'bonus', 'total_stock_value']                                                   |     0.1649 |     0.1546 |   0.9910 | 0.2675 | 0.4760 |
|  7.0000 | ['poi', 'deferred_income', 'salary']                                                    |     0.1857 |     0.1814 |   0.9905 | 0.3067 | 0.5235 |
|  1.0000 | ['poi', 'bonus', 'deferred_income']                                                     |     0.2010 |     0.1990 |   0.9900 | 0.3314 | 0.5515 |
|  5.0000 | ['poi', 'exercised_stock_options', 'salary']                                            |     0.1537 |     0.1526 |   0.9885 | 0.2644 | 0.4717 |
| 18.0000 | ['poi', 'exercised_stock_options', 'salary', 'total_stock_value']                       |     0.1875 |     0.1578 |   0.9870 | 0.2721 | 0.4812 |
```

I won't go through all the models tested (most of them can be seen commented on the *poy_id.py* file) but I also tried: GaussianNB, SVC, RandomForestClassifier and AdaBoostClassifier. In the End the best f1 score was achieved by using a K Nearest neighbours model.

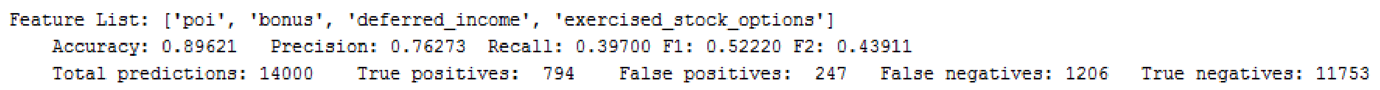
During the whole process I was constantly trying to tune the model using "*GridSearchCV"* to suggest the best parameters to be used (that happens on a function that was named after the final selected model: *getBestKNeighborsClassifier*).

In the end the best model was a K Neighbours Classifiers with a very small difference between the manhattan and minkowski metric. I choose to use the second because the f1 and f2 scores are a little better:

clf = KNeighborsClassifier(algorithm='auto', metric='manhattan', metric_params=None, n_neighbors=6, p=2, weights='distance' , leaf_size=30)

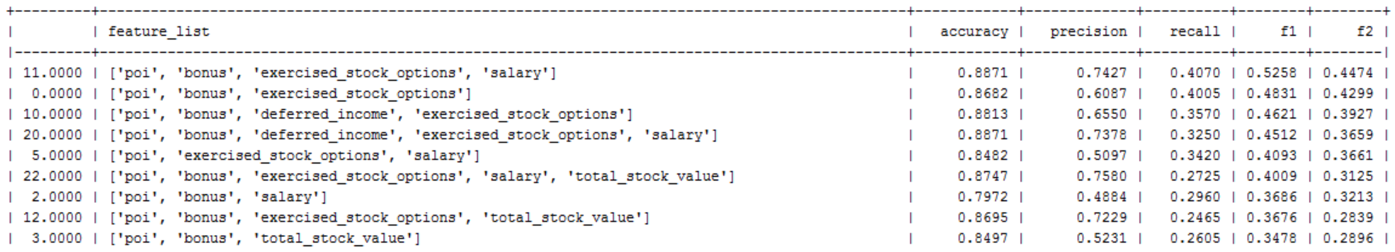


```
+---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------+
|         | feature_list                                                                            |  accuracy  | precision  |  recall  |   f1   |   f2   |
|---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------|
| 10.0000 | ['poi', 'bonus', 'deferred_income', 'exercised_stock_options']                          |     0.8962 |     0.7627 |   0.3970 | 0.5222 | 0.4391 |
| 20.0000 | ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'salary']                |     0.8986 |     0.8404 |   0.3580 | 0.5021 | 0.4044 |
| 11.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary']                                   |     0.8827 |     0.7401 |   0.3660 | 0.4898 | 0.4072 |
|  0.0000 | ['poi', 'bonus', 'exercised_stock_options']                                             |     0.8642 |     0.5950 |   0.3680 | 0.4547 | 0.3984 |
| 12.0000 | ['poi', 'bonus', 'exercised_stock_options', 'total_stock_value']                        |     0.8772 |     0.7722 |   0.2865 | 0.4179 | 0.3277 |
|  5.0000 | ['poi', 'exercised_stock_options', 'salary']                                            |     0.8483 |     0.5104 |   0.3420 | 0.4096 | 0.3662 |
| 22.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary', 'total_stock_value']              |     0.8753 |     0.7614 |   0.2760 | 0.4051 | 0.3163 |
| 25.0000 | ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'salary', 'total_stock_value'] |     0.8849 |     0.7748 |   0.2735 | 0.4043 | 0.3142 |
```

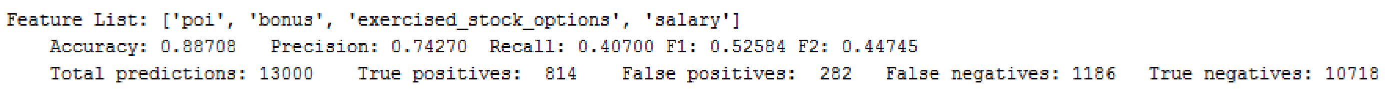


```
Feature List: ['poi', 'bonus', 'deferred_income', 'exercised_stock_options']
    Accuracy: 0.89621	Precision: 0.76273	Recall: 0.39700	F1: 0.52220	F2: 0.43911
    Total predictions: 14000	True positives:  794	False positives:  247	False negatives: 1206	True negatives: 11753
```

clf = KNeighborsClassifier(algorithm='auto', metric='minkowski', metric_params=None, n_neighbors=6, p=2, weights='distance', leaf_size=30)



```
+---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------+
|         | feature_list                                                                            |  accuracy  | precision  |  recall  |   f1   |   f2   |
|---------+-----------------------------------------------------------------------------------------+------------+------------+----------+--------+--------|
| 11.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary']                                   |     0.8871 |     0.7427 |   0.4070 | 0.5258 | 0.4474 |
|  0.0000 | ['poi', 'bonus', 'exercised_stock_options']                                             |     0.8682 |     0.6087 |   0.4005 | 0.4831 | 0.4299 |
| 10.0000 | ['poi', 'bonus', 'deferred_income', 'exercised_stock_options']                          |     0.8813 |     0.6550 |   0.3570 | 0.4621 | 0.3927 |
| 20.0000 | ['poi', 'bonus', 'deferred_income', 'exercised_stock_options', 'salary']                |     0.8871 |     0.7378 |   0.3250 | 0.4512 | 0.3659 |
|  5.0000 | ['poi', 'exercised_stock_options', 'salary']                                            |     0.8482 |     0.5097 |   0.3420 | 0.4093 | 0.3661 |
| 22.0000 | ['poi', 'bonus', 'exercised_stock_options', 'salary', 'total_stock_value']              |     0.8747 |     0.7580 |   0.2725 | 0.4009 | 0.3125 |
|  2.0000 | ['poi', 'bonus', 'salary']                                                              |     0.7972 |     0.4884 |   0.2960 | 0.3686 | 0.3213 |
| 12.0000 | ['poi', 'bonus', 'exercised_stock_options', 'total_stock_value']                        |     0.8695 |     0.7229 |   0.2465 | 0.3676 | 0.2839 |
|  3.0000 | ['poi', 'bonus', 'total_stock_value']                                                   |     0.8497 |     0.5231 |   0.2605 | 0.3478 | 0.2896 |
```

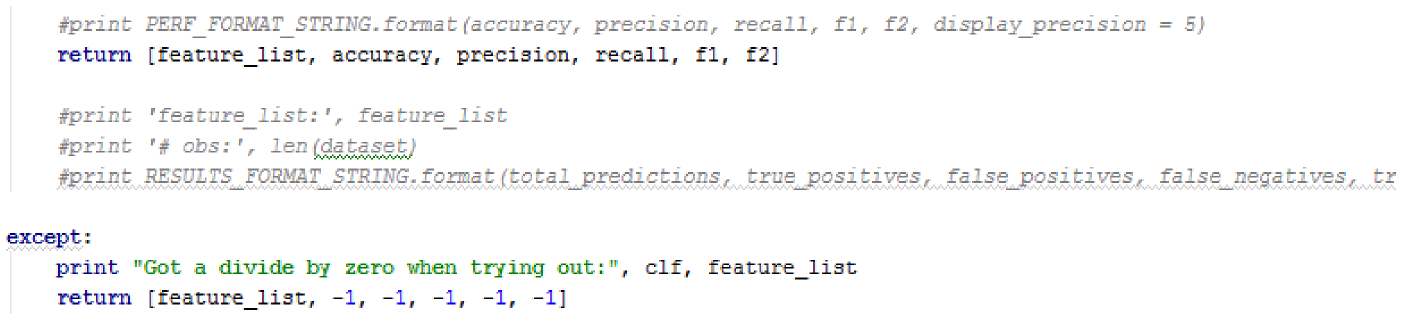


```
Feature List: ['poi', 'bonus', 'exercised_stock_options', 'salary']
    Accuracy: 0.88708	Precision: 0.74270	Recall: 0.40700	F1: 0.52584	F2: 0.44745
    Total predictions: 13000	True positives:  814	False positives:  282	False negatives: 1186	True negatives: 10718
```

Final Considerations:

Running the code:

In order for the poy_id.py file to output the results you see on this report, the *test_classifier* function on the *tester.py* file must return a list with the measures, rather than just printing it:



```
    #print PERF_FORMAT_STRING.format(accuracy, precision, recall, f1, f2, display_precision = 5)
    return [feature_list, accuracy, precision, recall, f1, f2]

    #print 'feature_list:', feature_list
    #print '# obs:', len(dataset)
    #print RESULTS_FORMAT_STRING.format(total_predictions, true_positives, false_positives, false_negatives, tr

except:
    print "Got a divide by zero when trying out:", clf, feature_list
    return [feature_list, -1, -1, -1, -1, -1]
```

Full Feature List\Explanation:

- Bonus: Reflects annual cash incentives paid based upon company performance. Also may include other retention payments.
- Deferral Payments: Reflects distributions from a deferred compensation arrangement due to termination of employment or due to in-service withdrawals as per plan provisions.
- Deferral Income: Reflects voluntary executive deferrals of salary, annual cash incentives, and long-term cash incentives as well as cash fees deferred by non-employee directors under a deferred compensation arrangement. May also reflect deferrals under a stock option or phantom stock unit in lieu of cash arrangement.
- Director Fees: Reflects cash payments and/or value of stock grants made in lieu of cash payments to non-employee directors.
- Exercised Stock Options: Reflects amounts from exercised stock options which equal the market value in excess of the exercise price on the date the options were exercised either through cashless (same-day sale), stock swap or cash exercises. The reflected gain may differ from that realized by the insider due to fluctuations in the market price and the timing of any subsequent sale of the securities.
- Expenses: Reflects reimbursements of business expenses. May include fees paid for consulting services.
- Long Term Incentive: Reflects long-term incentive cash payments from various long-term incentive programs designed to tie executive compensation to long-term success as measured against key performance drivers and business objectives over a multi-year period, generally 3 to 5 years.
- Other: Reflects items such as payments for severance, consulting services, relocation costs, tax advances and allowances for employees on international assignment (i.e. Housing allowances, cost of living allowances, payments under Enron’s Tax Equalization Program, etc.). May also include payments provided with respect to employment agreements, as well as imputed income amounts for such things as use of corporate aircraft.
- Restricted Stock: Reflects the gross fair market value of shares and accrued dividends (and/or phantom units and dividend equivalents) on the date of release due to lapse of vesting periods, regardless of whether deferred.

- Restricted Stock Deferred: Reflects value of restricted stock voluntarily deferred prior to release under a deferred compensation arrangement.
- Salary: Reflects items such as base salary, executive cash allowances, and benefits payments.
- Loan Advances: Reflects total amount of loan advances, excluding repayments, provided by the Debtor in return for a promise of repayment. In certain instances, the terms of the promissory notes allow for the option to repay with stock of the company.
- Total Payments: Sum of Salary, Bonus, Long Term Incentive, Deferred Income, Deferral Payments, Loan Advances, Other, Expenses, Director Fees
- Total Stock Value: Sum of Exercised Stock Options, Restricted Stock and Restricted Stock Deferred.

Useful Links:

- Tableau Public Dashboard
- Enron Scandal
- Evaluation Metrics
- Confusion matrix
- Accuracy paradox
- IQR Rule
- “g” value on IQR
- Sklearn Documentation
- POI features Udacity Discussion
- Data Leakage
- Scikit-learn algorithm cheat-sheet
- Tutorial on KNN Algorithm
- KNeighborsClassifier
- sklearn.cross_validation.StratifiedShuffleSplit
- sklearn.linear_model.LogisticRegression