Identify Fraud from Enron Email

Project - Udacity Nanodegree - Tanase Mihaela

1. 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 project consists in analyzing the Enron public financial and email dataset of 146 employee who may have committed fraud.

The objective of the project is to develop a classification algorithm which can predict whether or not a person was a person of interest (POI) based on the information available - 21 financial and email features.

List of financial and email features in the dataset:

1.salary
2.to_messages
3.deferral_payments
4.total_payments
5.loan_advances
6.bonus
7.email_address
8.restricted_stock_deferred
9.total_stock_value
10.shared_receipt_with_poi
11.long_term_incentive
12.exercised_stock_options
13.from_messages
14.other
15.from_poi_to_this_person
16.from_this_person_to_poi
17.poi
18.deferred_income
19.expenses
20.restricted_stock
21.director_fees

From the 146 employees in the dataset, there are 18 POIs and 128 non-POI.

The POIs list in the dataset is incomplete. That's because the dataset was created using the file "enron61702insiderpay.pdf" which is missing some POIs. The complete list of POIs can be found in the text file "poi_names.txt" (this file has 35 POIs).
I could add the missing POIs to the dataset and put "NaN" for their financial information, but this would be problematic for the classification algorithm. After some dataset checking, I noticed that all POIs have financial information.

Number of POIs with missing financial information: 0
Number of nonPOIs with missing financial information: 21

If I add the missing POIs from the txt file with missing financial information, an algorithm which would use total_payments as a feature, will associate a 'NaN' financial value with non-POI. I decided to ignore these missing POIs.

Checking the missing information in the dataset, we can see that some features have a lot of missing values (eg. LOAN_ADVANCES has 97.26% missings).

Feature SALARY has -->51 missings. Percent : 34.93%
Feature TO_MESSAGES has -->60 missings. Percent : 41.10%
Feature DEFERRAL_PAYMENTS has -->107 missings. Percent : 73.29%
Feature TOTAL_PAYMENTS has -->21 missings. Percent : 14.38%
Feature LONG_TERM_INCENTIVE has -->80 missings. Percent : 54.79%
Feature LOAN_ADVANCES has -->142 missings. Percent : 97.26%
Feature BONUS has -->64 missings. Percent : 43.84%
Feature RESTRICTED_STOCK has -->36 missings. Percent : 24.66%
Feature RESTRICTED_STOCK_DEFERRED has -->128 missings. Percent : 87.67%
Feature TOTAL_STOCK_VALUE has -->20 missings. Percent : 13.70%
Feature SHARED_RECEIPT_WITH_POI has -->60 missings. Percent : 41.10%
Feature FROM_POI_TO_THIS_PERSON has -->60 missings. Percent : 41.10%
Feature EXERCISED_STOCK_OPTIONS has -->44 missings. Percent : 30.14%
Feature FROM_MESSAGES has -->60 missings. Percent : 41.10%
Feature OTHER has -->53 missings. Percent : 36.30%
Feature FROM_THIS_PERSON_TO_POI has -->60 missings. Percent : 41.10%
Feature DEFERRED_INCOME has -->97 missings. Percent : 66.44%
Feature EXPENSES has -->51 missings. Percent : 34.93%
Feature EMAIL_ADDRESS has -->35 missings. Percent : 23.97%
Feature DIRECTOR_FEES has -->129 missings. Percent : 88.36%

The missings are coded in the dataset with NaN or 0. I recoded all the missings to zeros.
I wanted to remove the features with many missings, but considering that our dataset is very small this will most likely lower the performance metrics.

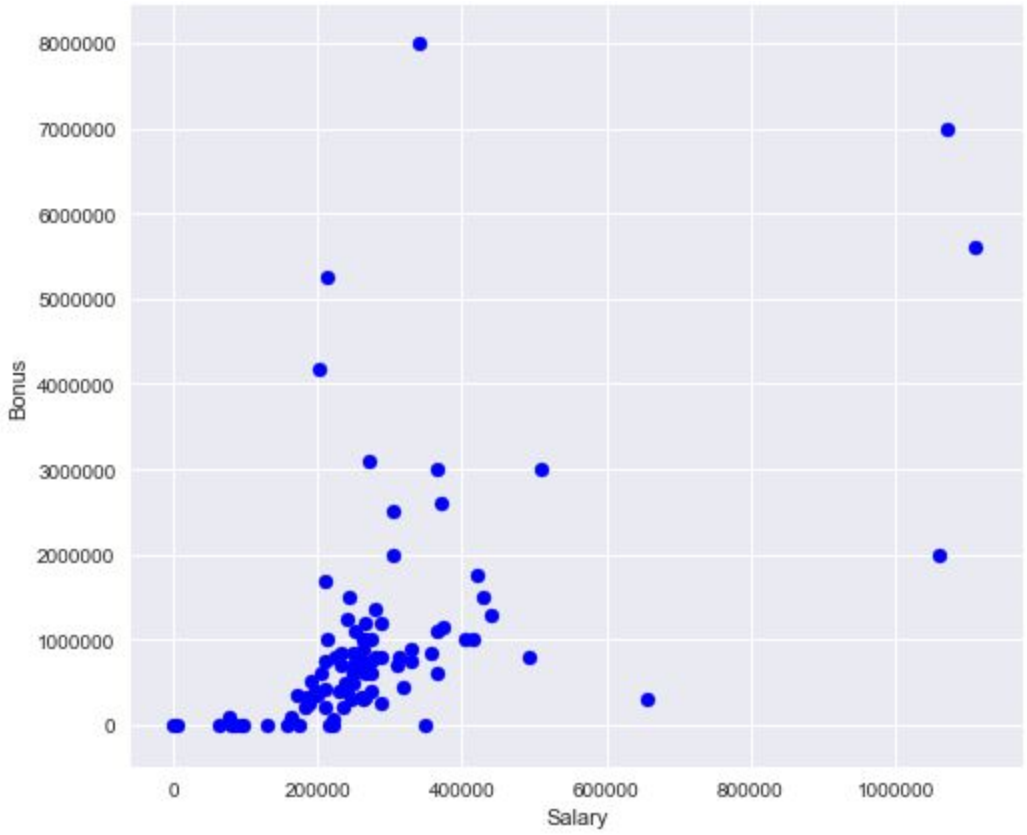
There is an outlier in the dataset that was removed: "TOTAL" data point.

The maximum total payments is: 309,886,585
The person with the maximum total payments is: TOTAL

This was an error of importing the data from the pdf file. Another data point which doesn't look and actually it isn't an Enron employee is "THE TRAVEL AGENCY IN THE PARK". I removed it as well.

There are other high values in the financial features, but these are valid data points, which can be flags of fraud activity, so I didn't remove them.

For example, in the plot below we can see the 8,000,0000 outlier for bonus and three 10,000,000+ outliers for salary.



I wanted to make a quick validation of the data, so I checked if total payments and total stocks are calculated correct.

So, salary+deferred_income+expenses+other+long_term_incentive+director_fees+deferral_payments+ loan_advances+bonus should be equal to total_payments, and restricted_stock+exercised_stock_options+ restricted_stock_deferred should be equal to total_stock_value.

Running the code, I found out that 4 employees have wrong information in the dataset.

Wrong total payments for:

	Correct	Incorrect
BELFER ROBERT	102500	-99215
BHATNAGAR SANJAY	15456290	275728

Wrong total stocks for:

	Correct	Incorrect
BELFER ROBERT	-44093	47378
BHATNAGAR SANJAY	0	15456290

It looks like the feature total_stock_value -44093 which is actually the value of restricted_stock_deferred (see the pdf);restricted_stock_deferred must be always be a negative number.

The values for these persons where replaced with values from the pdf.

2. 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", "intelligently select features", "properly scale features"]

Feature selection

The machine learning algorithm is as good as the features we put into it. We want to have the minimal number of features that capture the trends and the pattern in the dataset. In general, we want the minimum number of features that give as much information as possible. Features and information are two distinct things. The features are the number of characteristics of the data points that's attempting to access information. What we should care about is information, not the quantity of the features.

To access the best information in the features:

1. select the best features available. For this we can use the univariate feature selection which treats each feature independently and ask how much power it gives to the machine learning algorithm. Sklearn offers several automatic methods which select the best features: SelectPercentile and SelectKBest.

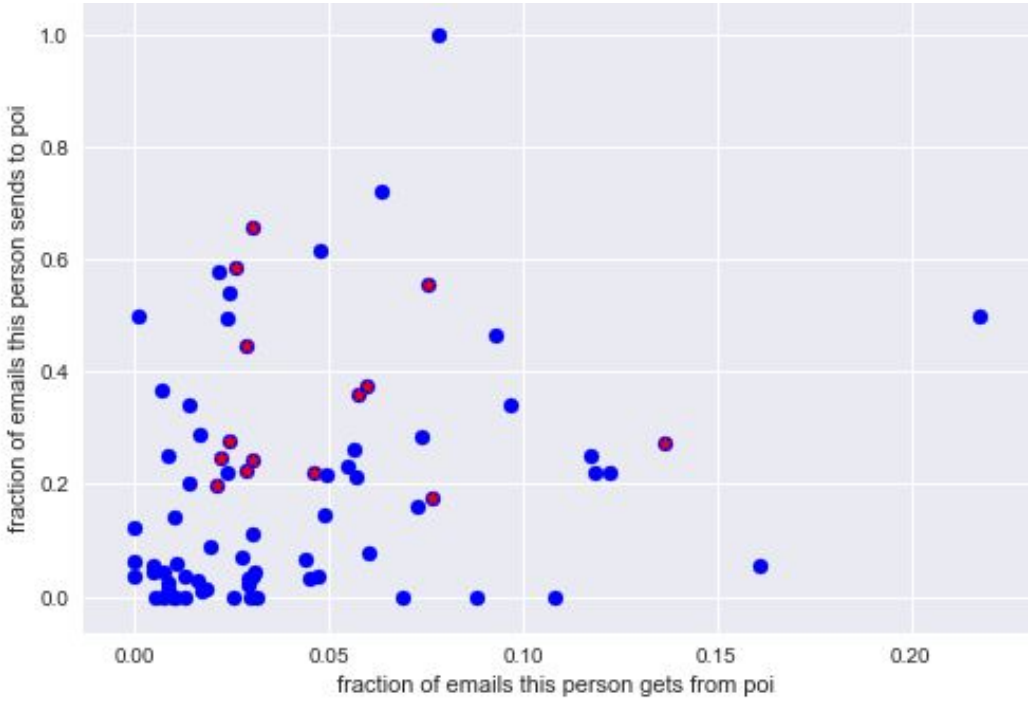
2. remove features which may add noise / cause overfitting / are strongly correlated to other features.

3. add new features using human intuition

Create new feature

In our dataset we have of emails sent to POI's and received from POI's. If an employee sends or receives a lot of emails from POIs, that employee could also be a POI. So, I created two new features:

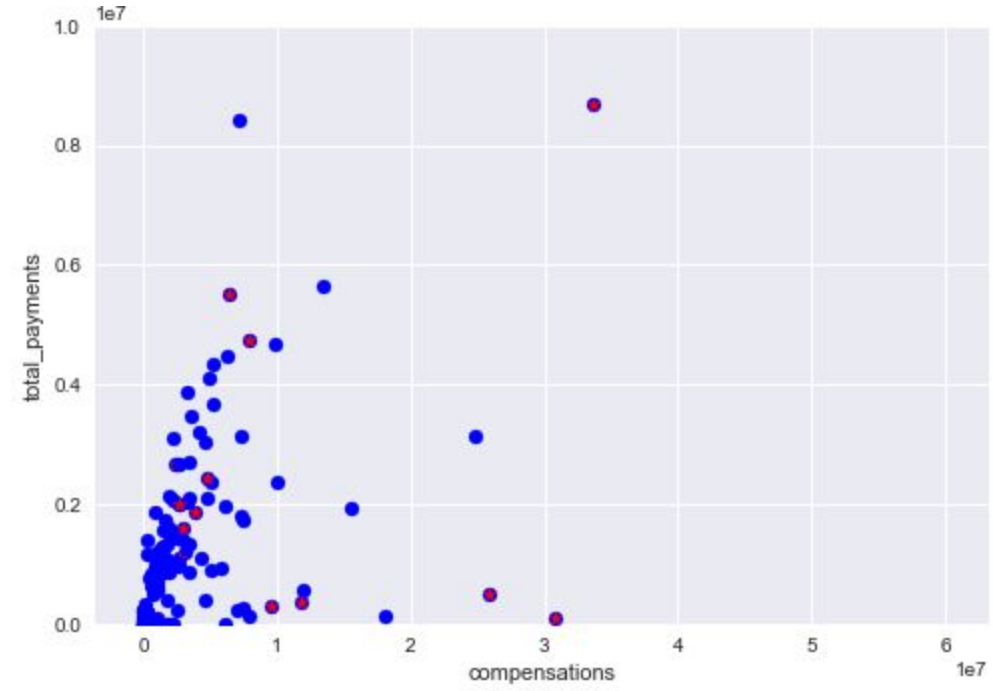
fraction_from_poi=from_poi_to_this_person/to_messages
fraction_to_poi=from_this_person_to_poi/from_messages

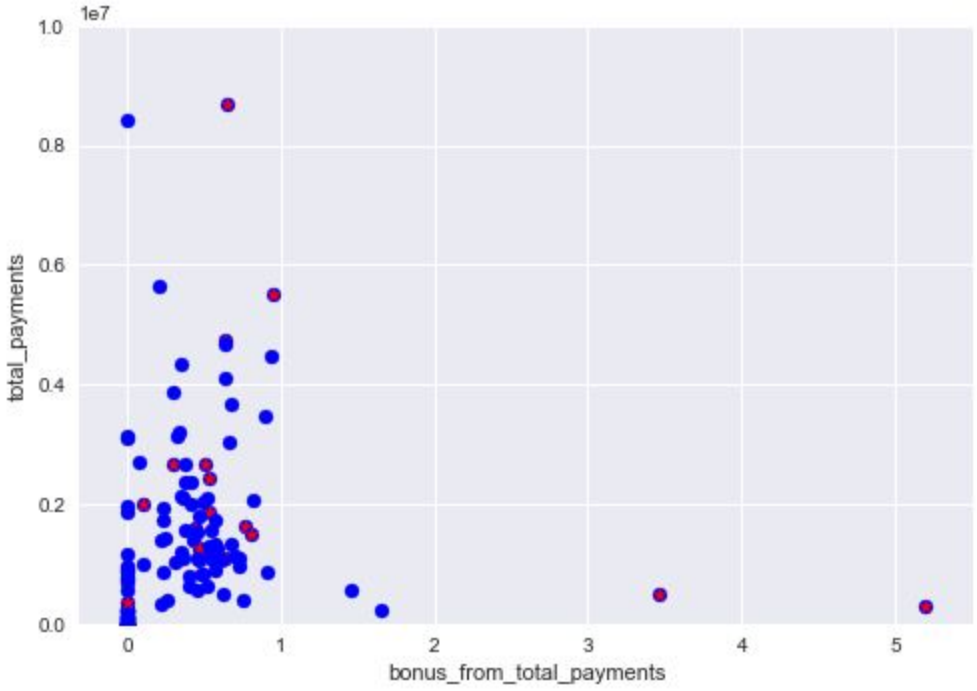


While the red points (POIs) aren't particularly clustered, if less than 20% of the messages are sent to POI that puts the sender in a band of blue points (non POIs). This means the sender is almost certainly not a POI.

I compute two more features, considering that a POI may be identified by the amount of bonuses he received. A POI most likely received higher compensations than a non POI.
So, I computed:

compensations=bonus+long_term_incentive+exercised_stock_options+restricted_stock
bonus_from_total_payments=bonus/total_payments

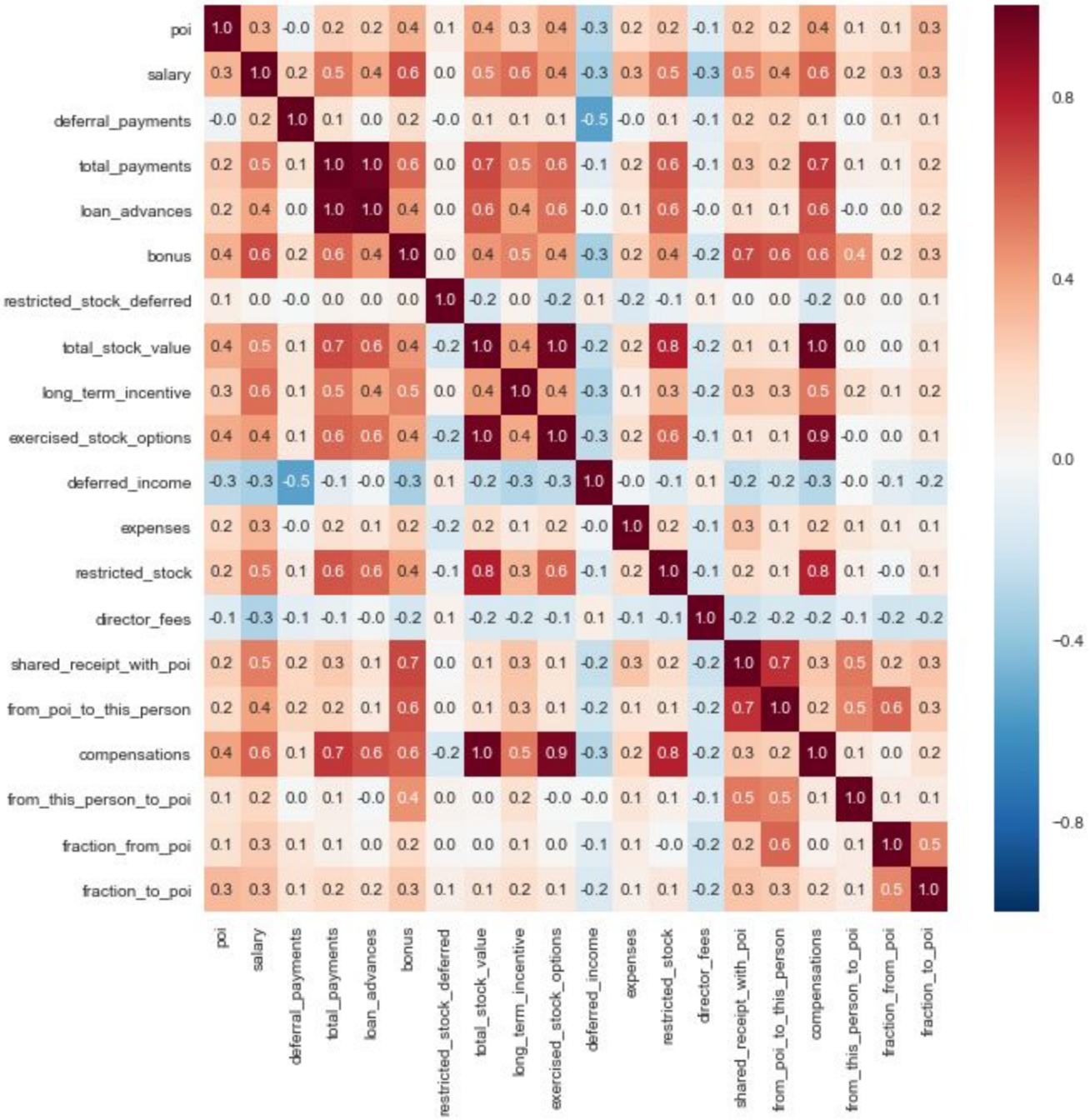




For compensation, the non-POIs (blue points) seem to be clustered in the lower band, but I'm not happy with the other feature created: bonus_from_total_payments. I will not include it in the final list of features.

There is another issue with these two new features. Because they are created from the present features in the dataset, most likely they are strongly correlated with the features they are computed from.

This was confirmed when I plotted the matrix correlation.



The final list of features used in machine learning algorithms was:

features_list = ['poi','salary','deferral_payments','total_payments','loan_advances','bonus', 'restricted_stock_deferred','total_stock_value','long_term_incentive', 'exercised_stock_options',

'deferred_income', 'expenses','restricted_stock', 'director_fees','shared_receipt_with_poi', 'from_poi_to_this_person','compensations','from_this_person_to_poi', 'fraction_from_poi','fraction_to_poi']

I removed from the list features "to_messages", "from_messages" and "others". After multiple running of the algorithms, I got the best scores without these three features.

Initial algorithms performance metrics

The first training and testing I did was on the features listed without any algorithm optimization.

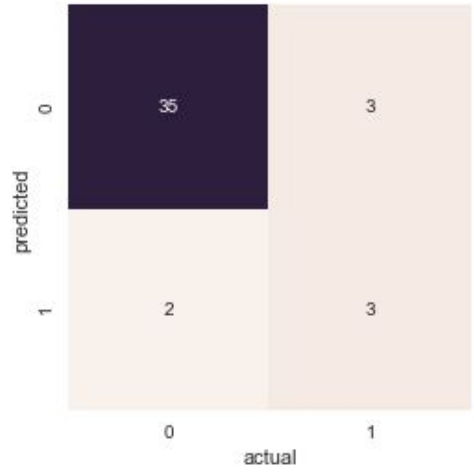
I first splitted the database into training and testing sets. 30% of the datapoints were held over for testing.

The training set contains a known output and the model learns on this data. The test dataset is used to test our model's prediction.

I tried four algorithms:

- Gaussian Naive Bayes
- Decision Tree
- KNN
- Logistic Regression

```
--------------------------------Gaussian Naive Bayes--------------------------------
Confusion matrix
```

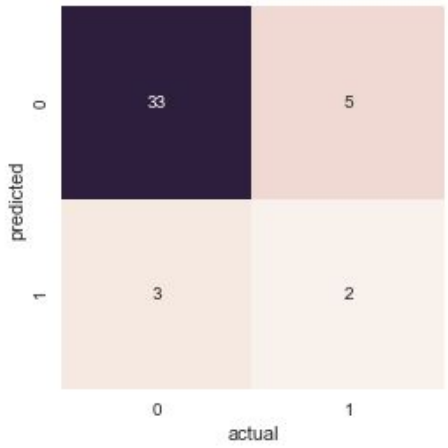



```
Accuracy: 0.883720930233
Report metrics:
             precision    recall  f1-score   support

        0.0       0.92      0.95      0.93        37
        1.0       0.60      0.50      0.55         6

avg / total       0.88      0.88      0.88        43
```

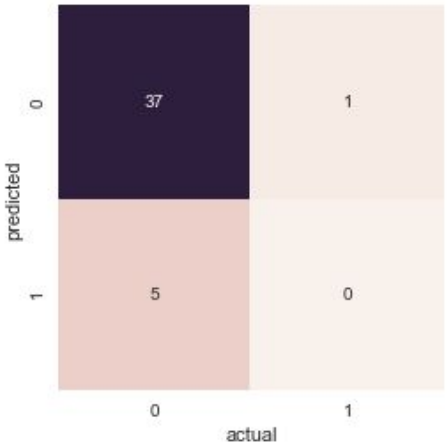
----------Decision Tree----------
Confusion matrix



Accuracy: 0.813953488372
Report metrics:

	precision	recall	f1-score	support
0.0	0.87	0.92	0.89	36
1.0	0.40	0.29	0.33	7
avg / total	0.79	0.81	0.80	43

----------KNN----------
Confusion matrix

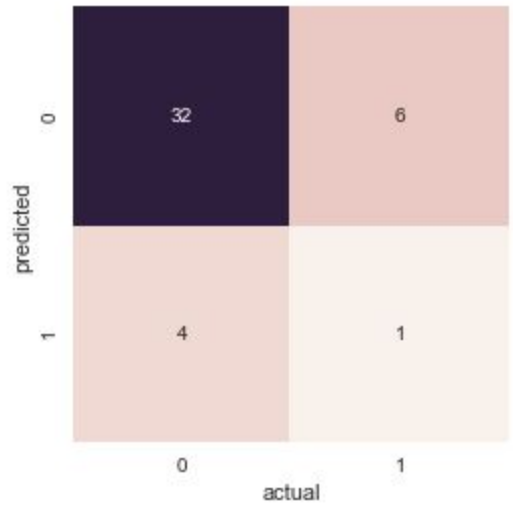


Accuracy: 0.860465116279
Report metrics:

	precision	recall	f1-score	support
0.0	0.97	0.88	0.93	42
1.0	0.00	0.00	0.00	1
avg / total	0.95	0.86	0.90	43

----------Logistic Regression----------

Confusion matrix



Accuracy: 0.767441860465

Report metrics:

	precision	recall	f1-score	support
0.0	0.84	0.89	0.86	36
1.0	0.20	0.14	0.17	7
avg / total	0.74	0.77	0.75	43

I also run the algorithms using the test_classifier() function, a more robust way of validating the results. I will explain the validation later.

----------Gaussian NB----------

GaussianNB(priors=None)

Accuracy: 0.77893 Precision: 0.28226 Recall: 0.42650 F1: 0.33971 F2: 0.38695

Total predictions: 15000 True positives: 853 False positives: 2169 False negatives: 1147True negatives: 10831

----------Decision Tree----------

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')

Accuracy: 0.80593 Precision: 0.26890 Recall: 0.26500 F1: 0.26694 F2: 0.26577

Total predictions: 15000 True positives: 530 False positives: 1441 False negatives: 1470True negatives: 11559

----------KNN----------
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform')
Accuracy: 0.87013 Precision: 0.55804 Recall: 0.12500 F1: 0.20425 F2: 0.14796
Total predictions: 15000 True positives: 250 False positives: 198 False negatives: 1750True negatives: 12802

----------LogisticRegression----------
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
Accuracy: 0.80380 Precision: 0.21197 Recall: 0.17350 F1: 0.19082 F2: 0.18004
Total predictions: 15000 True positives: 347 False positives: 1290 False negatives: 1653True negatives: 11710

Intelligently select features and Properly scale features

Because our features are unbalanced, I want to rescale them, so that they always have comparable ranges.

Many machine learning methods are more effective if the data attributes have the same scale.

For rescaling I used StandardScaling(), which is rescaling the range of features to scale the range in [0, 1].

Gaussian NB, Decision Tree and Logistic Regression don't need standardization.

From my algorithms, only KNN is a distance-based algorithms with Euclidean distance, so only for this I used StandardScaling().

To select the best features I used SelectKBest and Principal Components Analysis (PCA) dimension reduction.

StandardScaling(), SelectKBest and PCA were were chained together using GridSearchCV and pipeline. I also used the FeatureUnion estimator which process SelectKBest and PCA in parallel, then concatenates the results.

To classify data points into POI and non-POI, I used PCA to transform the features into principal components and I selected k number of best features using SelectKBest.

Then I used the Decision Tree classifier to create a model.

To get the best parameters combinations I used GridSearchCV while maximizing the F1 score.

The parameters passed to GridSearchCV are related to PCA, SelectKBest and decision tree classifier.

3. 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 tried to tune all 4 algorithms.

----------GaussianNB tuning----------

```
combined_features = FeatureUnion([("pca", PCA()), ("kbest", SelectKBest())])
pipe = Pipeline([
        ('select_features',combined_features),
        ('classifier',GaussianNB())
        ])
param_grid = ([{
        'select_features__pca__n_components':list(range(2,8,1)),
        'select_features__kbest__k':list(range( 2,8,1)),
        }])
clf_gs = GridSearchCV(pipe,param_grid,scoring='f1',cv=sss)
clf_f=clf_gs.fit(features, labels)
clf=clf_f.best_estimator_
best_parameters = clf_gs.best_estimator_.get_params()
```

----------Decision Tree tuning----------

```
combined_features = FeatureUnion([("pca", PCA()), ("kbest", SelectKBest())])
pipe = Pipeline([
        ('select_features',combined_features),
        ('classifier',DecisionTreeClassifier())
        ])
param_grid = ([{
        'select_features__pca__n_components':list(range(2,8,1)),
        'select_features__kbest__k':list(range(2,14,1)),
        'classifier__class_weight':['balanced'],
        'classifier__criterion':['gini', 'entropy'],
        'classifier__max_depth': [1,2,4,8,16,32],
        'classifier__random_state':[42],
        'classifier__min_samples_split':[2,4,8,16,32],
        'classifier__min_samples_leaf': [2,4,8,16,32]
        }])
clf_gs = GridSearchCV(pipe,param_grid,scoring='f1',cv=sss)
clf_f=clf_gs.fit(features, labels)
clf=clf_f.best_estimator_
best_parameters = clf_gs.best_estimator_.get_params()
```

```
----------------------------------------------------KNN tuning---------------------------------

    combined_features = FeatureUnion([("pca", PCA()), ("kbest", SelectKBest())])
    pipe = Pipeline([('scaler',StandardScaler()),
            ('select_features',combined_features),
            ('classifier',KNeighborsClassifier())
            ])
    param_grid = ([{
            # 'scaler__with_std': [True, False],
            'select_features__pca__n_components':list(range(2,8,1)),
            'select_features__kbest__k':list(range(2,14,1)),
            'classifier__n_neighbors':  range(2,20),
            'classifier__weights': ['uniform', 'distance'],
            'classifier__algorithm':['auto','ball_tree','kd_tree']
            }])
    clf_gs = GridSearchCV(pipe,param_grid,scoring='f1',cv=sss)
    clf_f=clf_gs.fit(features, labels)
    clf=clf_f.best_estimator_
    best_parameters = clf_gs.best_estimator_.get_params()


-------------------------------------LogisticRegression tuning---------------------------------

    combined_features = FeatureUnion([("pca", PCA()), ("kbest", SelectKBest())])
    pipe = Pipeline([
            ('select_features',combined_features),
            ('classifier',LogisticRegression())
            ])
    param_grid = ([{
            'select_features__pca__n_components':list(range(2,8,1)),
            'select_features__kbest__k':list(range(2,14,1)),
            'classifier__tol':  [0.01,0.02,0.03,0.04,0.05,0.001,0.005,0.0001],
            'classifier__C': [0.001,0.002,0.003,0.004,0.005,0.01,0.05, 0.5, 1, 10],
            'classifier__class_weight':['balanced'],
            'classifier__random_state':[42]
            }])

    clf_gs = GridSearchCV(pipe,param_grid,scoring='f1',cv=sss)
    clf_f=clf_gs.fit(features, labels)
    clf=clf_f.best_estimator_
    best_parameters = clf_gs.best_estimator_.get_params()
```

GaussianNB	Accuracy: 0.84987	Precision: 0.42410	Recall: 0.35200	F1: 0.38470
DecisionTree	Accuracy: 0.79227	Precision: 0.33788	Recall: 0.58150	F1: 0.42742
KNN	Accuracy: 0.83600	Precision: 0.38569	Recall: 0.38800	F1: 0.38684
Logistic regression	Accuracy: 0.77140	Precision: 0.30398	Recall: 0.55400	F1: 0.39256

Based on the F1 scores, Decision Tree was the algorithm with the best results.

The best parameters chosen by the GridSearchCV for Decision Tree were:

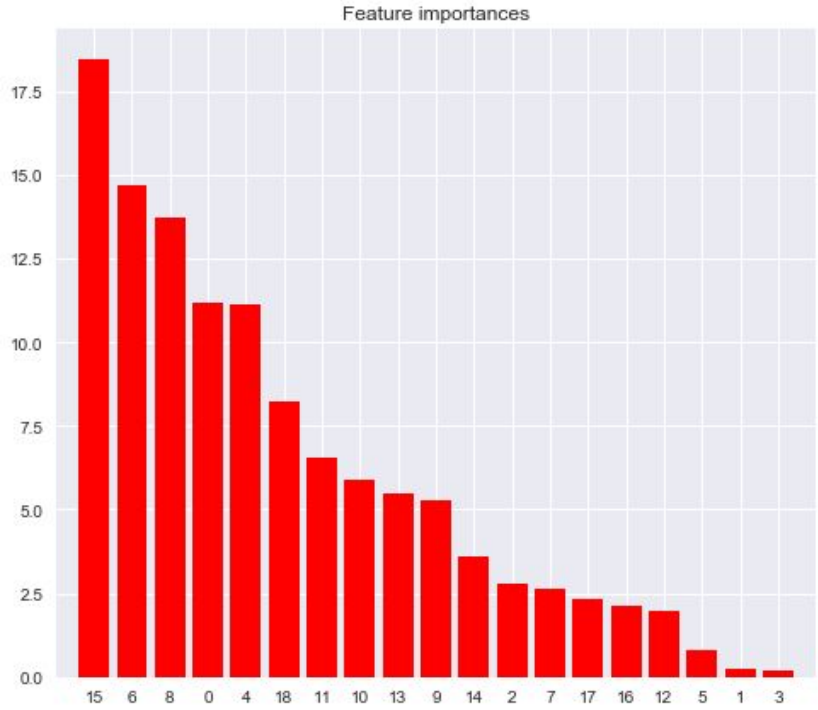
select_features__kbest__k: 9
classifier__random_state: 42
classifier__min_samples_split: 16
select_features__pca__n_components: 2
classifier__class_weight: 'balanced'
classifier__min_samples_leaf: 2
classifier__max_depth: 8
classifier__criterion: 'entropy'

Best features selected by SelectKBest: [0 4 6 8 10 11 13 15 18] which means:
[salary, bonus, total_stock_value, exercised_stock_options, expenses, restricted_stock, shared_receipt_with_poi, compensations, fraction_to_poi]

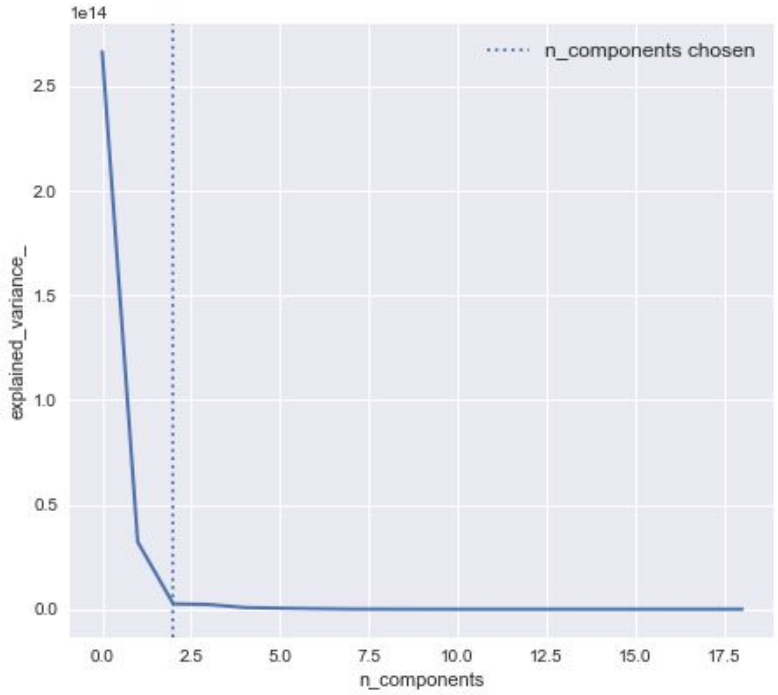
Features importance:

1. feature 15 (18.430407)
2. feature 6 (14.689864)
3. feature 8 (13.714161)
4. feature 0 (11.196268)
5. feature 4 (11.129479)
6. feature 18 (8.243016)
7. feature 11 (6.572702)
8. feature 10 (5.898734)
9. feature 13 (5.495321)
10. feature 9 (5.279110)
11. feature 14 (3.590680)
12. feature 2 (2.771634)
13. feature 7 (2.611274)
14. feature 17 (2.349338)
15. feature 16 (2.143841)
16. feature 12 (1.961608)
17. feature 5 (0.793834)
18. feature 1 (0.263968)
19. feature 3 (0.197149)

```
----------Plot the feature importances---------
```



The GridSearchCV found 2 principal components.



4. 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? What parameters did you tune? (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 items: "discuss parameter tuning", "tune the algorithm"]

In machine learning, we have hyperparameter and standard model parameters. The standard parameters are learned from the data by training a model with existing data.

Hyperparameter cannot be directly learned from the regular training process. They can be decided by setting different values, training different models, and choosing the values that test better.

For Decision Tree, the hyperparameters I tuned were:

- max_depth: The maximum depth of the tree
- min_samples_split: The minimum number of samples required to split an internal node
- min_samples_leaf: The minimum number of samples required to be at a leaf node
- criterion - The function to measure the quality of a split.
  Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- class_weight - for unbalanced classes (we have low number of POIs in the dataset)
- random_state: random_state is the seed used by the random number generator

By giving different values for hyperparameters, the classifier performs better or worse. If you don't do this correctly, you can easily overfit or underfit the data. Decision Tree classifier is especially prone to overfitting if we allow it to split to a granular degree.

5. 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 items: "discuss validation", "validation strategy"]

To get a good idea about how well our model is performing, we use what's known as the holdout set: we holdout some subsets of the data from the training of the model and then use this holdout to check the model performance. We can do this by using the train_test_split utility in sklearn. The disadvantage of using a holdout set is that we lose a portion of the data for the training. This can cause problems if the dataset is small. One way to address this is to use cross-validation. This will do a sequence of fits where each subset is used both as training and test set.In our dataset we used StratifiedShuffleSplit with 1000 folds, which means we split the data into 1000 groups and use each of them in turn to evaluate the model fit.

The best parameters obtained from the GridSearchCV are usually fitted to a training set. As the Enron dataset is very small, I trained the model on the whole dataset.

I then passed the best model to the test_classifier function which fit the classifier to train and test sets and then validated it using 1000 randomized stratified cross-validation splits.

The GridSearchCV was also using a StratifiedShuffleSplit (StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=42)). I used only 10 splits because the training time was lower. Using a splits=1000 did the GridSearchCV to run for hours.

The classic mistake is to train and validate the data on the same data, which will give an accuracy of 100% every time.

6. 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"]

For the Decision Tree model, I got:

Accuracy: 0.79227
Precision: 0.33788
Recall: 0.58150
F1: 0.42742

True positives: 1163
False positives: 2279
False negatives: 837
True negatives: 10721

Precision=true positives / (true positives + false positives)
=number of true positives divided by the total number of elements labeled as belonging to the positive class

Recall=true positives / (true positives + false negatives)
=number of true positives divided by the total number of elements that actually belong to the positive class

F1=2(precision x recall) / (precision + recall)

The recall is higher than the precision. That means that nearly every time a POI shows up in my test set, I am able to identify it. But sometimes I get some false positives, where a non POI gets flagged as POI.