ENRON Final project
By Eric Rabaute

Release note:
- Add algorithm metrics definition + meaning in that project (2-Definition)
- Add 10-Validation on validation topic
- Change final classifier algorithm from ‘K nearest Neighbors’ to ‘Decision tree’ in order to have precision and recall above 0.3 as specified (10-Validation)

1. INTRODUCTION

This analysis is part of Udacity training.

The objective is to make a data analysis applying on Machine learning methods and tools.

2. DEFINITION

- **Accuracy**
  - The ratio of well predicted test labels / overall test labels
  - Ex.: 0.77 means that 77% of POI and not POI persons in the test data set are in the good category
- **Precision**
  - The ratio between true and true+false positives
  - In this project, higher this metrics will be and less persons will be investigated by error
- **Recall**
  - The ratio between true and true+false negatives
  - In this project, higher this metrics will be and less persons needing to be investigated will not be by error
- **F1**
  - It is a mix between precision and recall helping to assess test accuracy better than Accuracy metrics (see above)
  - In this project, higher this metrics is, better is the compromise between precision and recall

3. MACHINE LEARNING OBJECTIVE

The questions we want to answer in this study are:
- How can we identify with a good accurate result the persons of interest related to Enron Fraud? (ie.: wich available features are the most valuable?)
- How can we optimize this Person of Interest Identifier thanks to Machine learning methods and tools?

4. FEATURES SELECTION

The first task is to choose among all available features the one that will helps us

financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)

email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is ‘email_address’, which is a text string)

Number of persons on the sample:146
Number of poi on the sample:18 (0.12%)
Ratio test/train data: 0.3 (meaning 100 train persons, 43 test persons)

First observations on data show a lot of features missed (ie. NaN value). I proposed to make a quantitative check on this value for all features first. Indeed, if a feature is miss for more than 50% absent of all data set, it could be not interesting to go further with it.

Hereafter the count of NaN for all features (max to min order) for a total of **146 persons**:

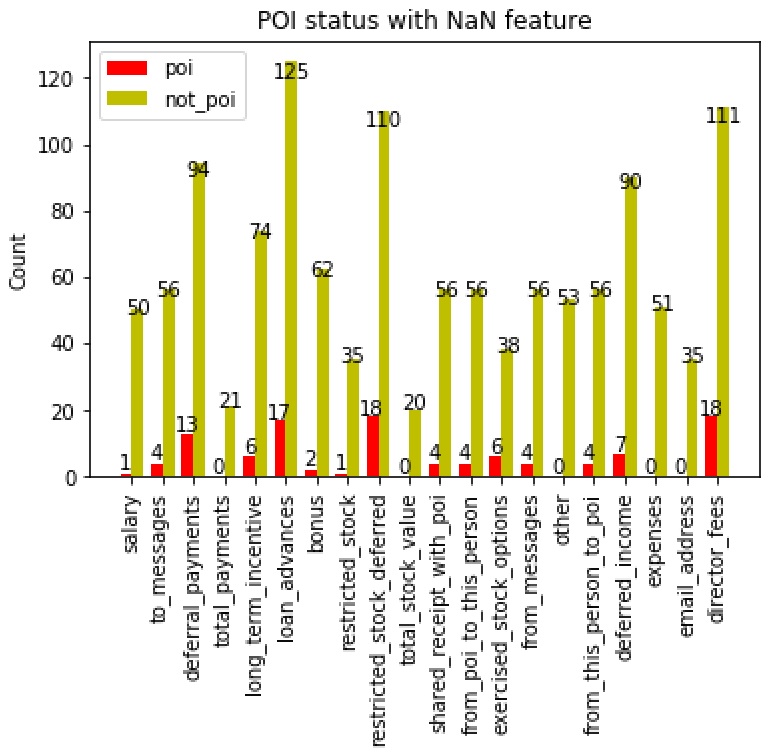
(1) loan_advances:142
(2) director_fees:129
(3) restricted_stock_deferred:128
(4) deferral_payments:107
(5) deferred_income:97
(6) long_term_incentive:80
(7) bonus:64
(8) to_messages:60
(9) shared_receipt_with_poi:60
(10) from_poi_to_this_person:60
(11) from_messages:60
(12) from_this_person_to_poi:60
(13) other:53
(14) salary:51
(15) expenses:51
(16) exercised_stock_options:44
(17) restricted_stock:36
(18) email_address:35
(19) total_payments:21
(20) total_stock_value:20

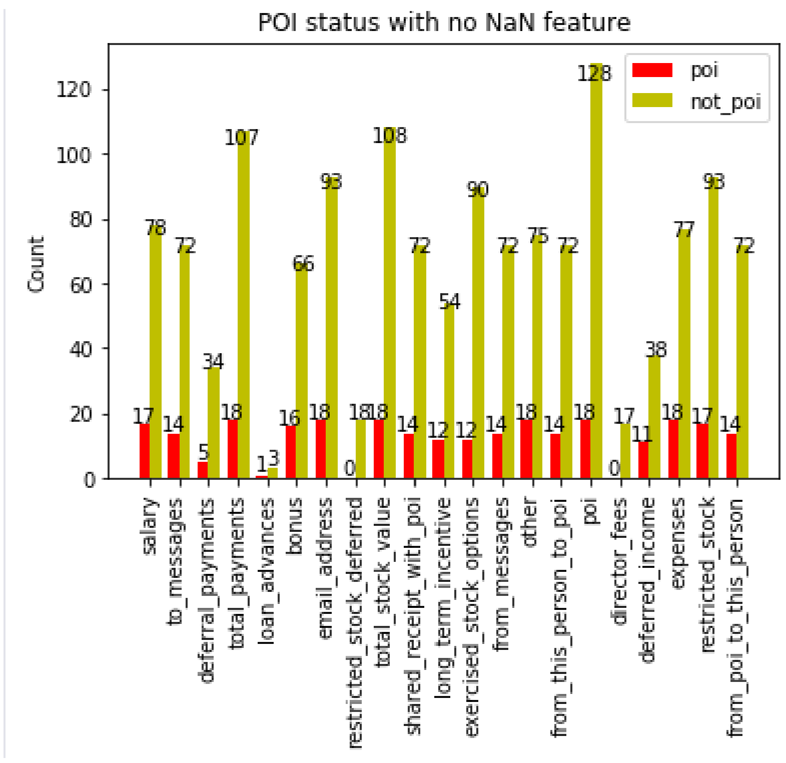
It is interesting to see that some features are more than 97% missing (loan advances). It could be explained by the fact the information is difficult to get that is not a very valuable information or this feature absence means something about the person (ex.: director fees means the person is a high ranking person and so likely a POI).

I propose to move forward this question by correlating the number of NaN and the POI status (ie. Have we got a higher POI status rate if NaN value is more or less present for one feature).

Hereafter two figures:

- The first one is number of POI and not POI persons according features equals to NaN value
- The second is the same with persons with value different of NaN





Analysis: At a first sight, no clear output is raised except that theory about 'director fees' is false, indeed, there is no 'poi' with director fees different from NaN. This is strange since we could expect to have mainly POI inside director committee.

I propose to go a last step further with the ratio between poi and not poi for none NaN value.

(1) loan_advances: poi=0.25% (n=4)
(2) deferred_income: poi=0.22% (n=49)
(3) bonus: poi=0.2% (n=82)
(4) other: poi=0.19% (n=93)
(5) expenses: poi=0.19% (n=95)
(6) salary: poi=0.18% (n=95)
(7) long_term_incentive: poi=0.18% (n=66)
(8) to_messages: poi=0.16% (n=86)
(9) shared_receipt_with_poi: poi=0.16% (n=86)
(10) from_poi_to_this_person: poi=0.16% (n=86)
(11) from_messages: poi=0.16% (n=86)
(12) from_this_person_to_poi: poi=0.16% (n=86)
(13) email_address: poi=0.16% (n=111)
(14) restricted_stock: poi=0.15% (n=110)
(15) total_payments: poi=0.14% (n=125)
(16) total_stock_value: poi=0.14% (n=126)
(17) deferral_payments: poi=0.13% (n=39)
(18) exercised_stock_options: poi=0.12% (n=102)
(19) poi: poi=0.12% (n=146)

```
(20) restricted_stock_deferred: poi=0.0% (n=18)
(21) director_fees:     poi=0.0% (n=17)
```

Analysis: On first observations, we noticed the ratio between poi and not poi in the sample is 12% (18/146). We can assume the features we would like to select have the less NaN value to provide most information as possible and the poi sample mean different from sample mean.

I propose to exclude in a first step the features where NaN values are more than 50% (ie. 73), ie:

```
(1) loan_advances:      poi=0.25% (n=4)
(2) deferred_income:    poi=0.22% (n=49)
(7) long_term_incentive:        poi=0.18% (n=66)
(17) deferral_payments: poi=0.13% (n=39)
(20) restricted_stock_deferred: poi=0.0% (n=18)
(21) director_fees:     poi=0.0% (n=17)
```

Then I propose to get three features per category (financial, email). I will prioritize features with n and ratio bigger as possible.

Consequently, I get:

```
(15) total_payments:    poi=0.14% (n=125)
(16) total_stock_value: poi=0.14% (n=126)
(18) exercised_stock_options:   poi=0.12% (n=102)

(10) from_poi_to_this_person:   poi=0.16% (n=86)
(12) from_this_person_to_poi:   poi=0.16% (n=86)
(9) shared_receipt_with_poi:    poi=0.16% (n=86)
```

Consequently, I exclude for the moments following features (I could use it afterwards).

```
(14) restricted_stock:  poi=0.15% (n=110)
 (3) bonus:      poi=0.2% (n=82)
(4) other:      poi=0.19% (n=93)
(5) expenses:   poi=0.19% (n=95)
(6) salary:     poi=0.18% (n=95)
(8) to_messages:        poi=0.16% (n=86)
(11) from_messages:     poi=0.16% (n=86)
```

5. REMOVE OUTLIERS

The best way to remove outliers is to look for the maximum value for each feature (even those not selected for the machine learning algorithm).

Hereafter the results of the maximum value:

```
Maximum value for salary: ('TOTAL', 26704229)
Maximum value for to_messages: ('SHAPIRO RICHARD S', 15149)
Maximum value for deferral_payments: ('TOTAL', 32083396)
Maximum value for total_payments: ('TOTAL', 309886585)
Maximum value for exercised_stock_options: ('TOTAL', 311764000)
Maximum value for bonus: ('TOTAL', 97343619)
Maximum value for restricted_stock: ('TOTAL', 130322299)
Maximum value for shared_receipt_with_poi: ('BELDEN TIMOTHY N', 5521)
Maximum value for restricted_stock_deferred: ('BHATNAGAR SANJAY', 15456290)
Maximum value for total_stock_value: ('TOTAL', 434509511)
Maximum value for expenses: ('TOTAL', 5235198)
Maximum value for loan_advances: ('TOTAL', 83925000)
Maximum value for from_messages: ('KAMINSKI WINCENTY J', 14368)
Maximum value for other: ('TOTAL', 42667589)
Maximum value for from_this_person_to_poi: ('DELAINEY DAVID W', 609)
Maximum value for poi: ('HANNON KEVIN P', True)
Maximum value for director_fees: ('TOTAL', 1398517)
Maximum value for deferred_income: ('BOWEN JR RAYMOND M', -833)
Maximum value for long_term_incentive: ('TOTAL', 48521928)
Maximum value for email_address: ('COLWELL WESLEY', 'wes.colwell@enron.com')
Maximum value for from_poi_to_this_person: ('LAVORATO JOHN J', 528)
```

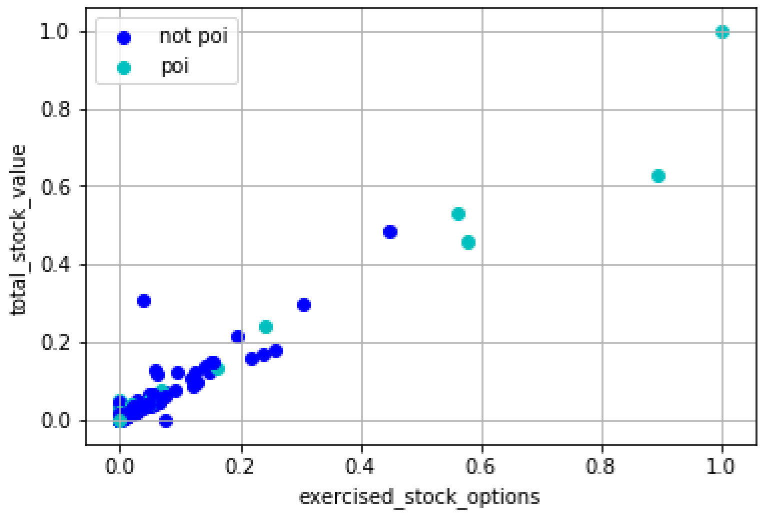
It is obvious that TOTAL person doesn’t exist. It is just an error in extracting the data from the dataset.

I propose to remove this “person” from the rest of analysis.

Hereafter the new results for maximum value:

```
Maximum value for salary: ('SKILLING JEFFREY K', 1111258)
Maximum value for to_messages: ('SHAPIRO RICHARD S', 15149)
Maximum value for deferral_payments: ('FREVERT MARK A', 6426990)
Maximum value for total_payments: ('LAY KENNETH L', 103559793)
Maximum value for exercised_stock_options: ('LAY KENNETH L', 34348384)
Maximum value for bonus: ('LAVORATO JOHN J', 8000000)
Maximum value for restricted_stock: ('LAY KENNETH L', 14761694)
Maximum value for shared_receipt_with_poi: ('BELDEN TIMOTHY N', 5521)
Maximum value for restricted_stock_deferred: ('BHATNAGAR SANJAY', 15456290)
Maximum value for total_stock_value: ('LAY KENNETH L', 49110078)
Maximum value for expenses: ('MCCLELLAN GEORGE', 228763)
Maximum value for loan_advances: ('LAY KENNETH L', 81525000)
Maximum value for from_messages: ('KAMINSKI WINCENTY J', 14368)
Maximum value for other: ('LAY KENNETH L', 10359729)
Maximum value for from_this_person_to_poi: ('DELAINEY DAVID W', 609)
Maximum value for poi: ('HANNON KEVIN P', True)
Maximum value for director_fees: ('BHATNAGAR SANJAY', 137864)
Maximum value for deferred_income: ('BOWEN JR RAYMOND M', -833)
Maximum value for long_term_incentive: ('MARTIN AMANDA K', 5145434)
Maximum value for email_address: ('COLWELL WESLEY', 'wes.colwell@enron.com')
Maximum value for from_poi_to_this_person: ('LAVORATO JOHN J', 528)
```

The result is far better (see following figure)



Moreover, in the list of features I select, there are persons with NaN value, it could disturb the algorithms.

I propose to remove all persons from the list that have NaN value for all selected features.

It only concerns 2 people.

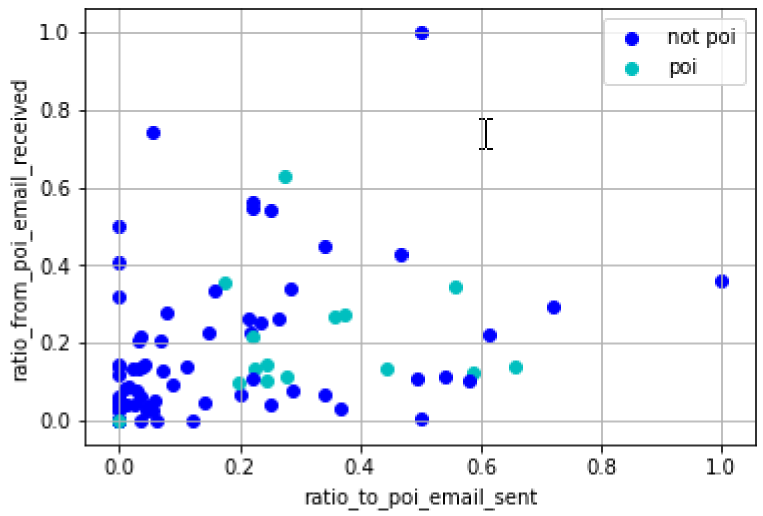
```
---> Delete: CHAN RONNIE
---> Delete: LOCKHART EUGENE E
```

6. RESCALE FEATURES

Thanks to min/max functions, we can observe that values are completely different as we compare millions of dollars with thousands/hundreds of emails.

I propose to rescale all features thanks to Python *MinMaxScaler()*.

We can see the benefit of this operation with the following figure representing a scatter plot with two new features related to email received and sent from and to POI.



Analysis: We can see that POI have received and sent more than 20% of emails from and to POI. That will leverage the power of the classification afterwards.

7. CREATE NEW FEATURE

In "*chapter 3: Feature selection*", We have chosen to select features with maximum non NAN values to get maximal information.

Now, we have to think about creating new more valuable features.

Regarding emails, my hypothesis is ratio between emails sent/received to/from poi would be far more accurate than absolute figures.
I propose consequently to add two features:

- ratio_from_poi_email_received
- ratio_to_poi_email_sent

In the same way, as the fraud has been characterized by a sudden sell of actions by persons involved in the fraud, I propose to add:

- ratio_exer_stock_total

To check the benefits of those new features, I'm making a features selection according to the k highest scores (Python *SelectKBest*).

First, I'm checking the scores:

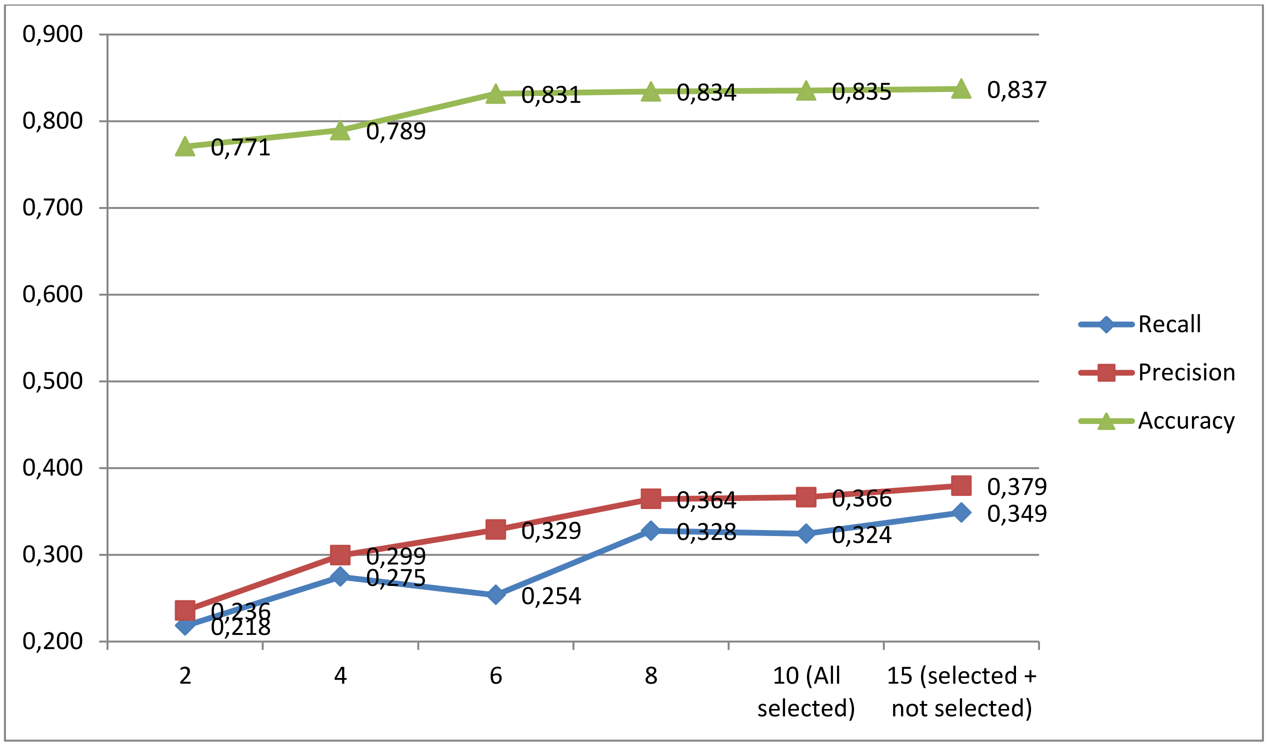
```
Scores for features:

(1) exercised_stock_options: 6.84550933503
(2) total_stock_value: 5.47661009929
(3) ratio_exer_stock_total: 4.68313270573
(4) ratio_to_poi_email_sent: 4.64455528039
```

(5) total_payments: 2.77918249429
(6) shared_receipt_with_poi: 2.43221986514
(7) from_poi_to_this_person: 1.37005929223
(8) from_this_person_to_poi: 1.0008076418
(9) ratio_from_poi_email_received: 0.827769252139

Secondly, I'm making classification tests thanks to provided test function for different value of k (selected features).

I'm using '*decision tree*' as algorithm with default settings but I could have used another one.



k (number of features)	Accuracy	Precision	Recall
2	**0.77085**	**0.23555**	**0.21800**
4	0.78946	0.29918	0.27450
6	0.83147	0.32879	0.25350
8	0.83407	0.36409	**0.32750**
10 (All selected)	**0.83507**	**0.36610**	0.32400
15 (selected + not selected)	***0.83713***	***0.37942***	***0.34850***

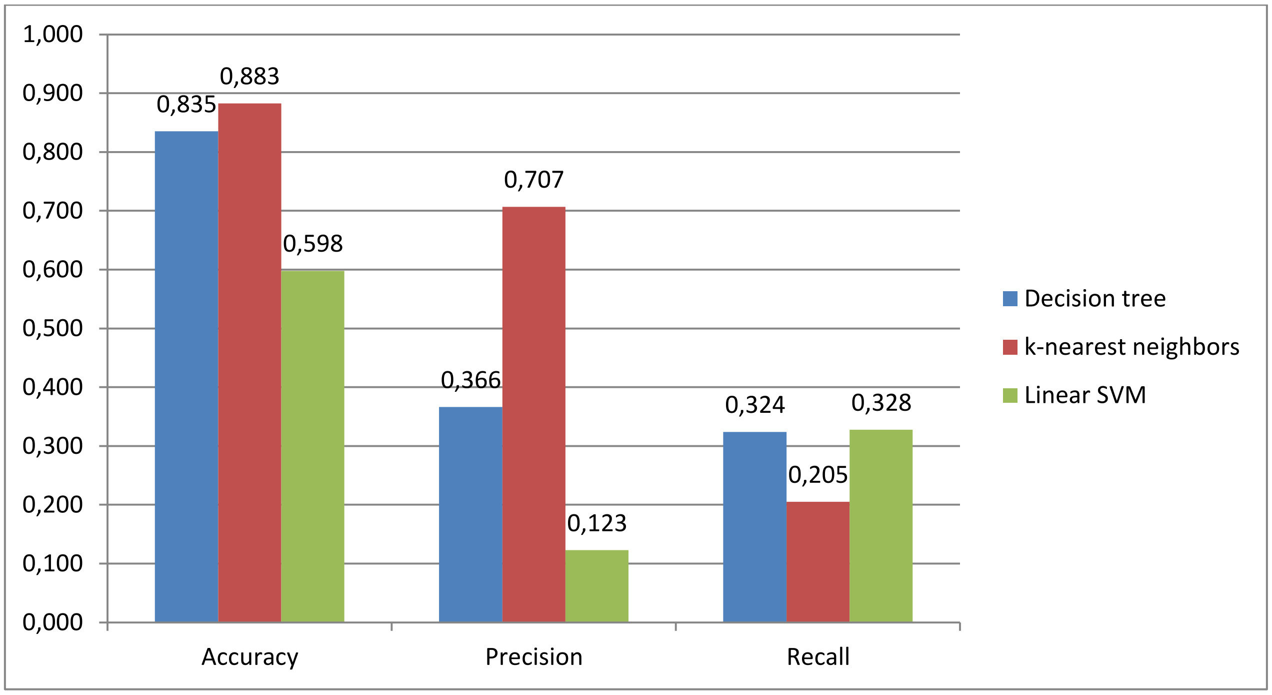
Analysis: Thanks to this table, we can see that reducing the number of features do not improve global accuracy of the classifier. Morever, we can observe that getting back the removed features improve a bit the results. Nevetheless, I keep the 10 selected features but I will assess the final tuned algorithm with all features to check that my method is the good one.

8. TRY DIFFERENT CLASSIFIER

Now the features are selected, we will test two popular algorithms to make the supervised learning:

- Decision tree classifier
- k-nearest neighbors classifier
- Linear SVM (Support Vector Machine)

In a first step, I keep default configuration, hereafter the results:



Algorithm (default)	Accuracy	Precision	Recall	Test time (secs)
Decision tree	0.83507	0.36610	**0.32400**	**1.178**
k-nearest neighbors	**0.88267**	**0.70690**	**0.20500**	2.597
Linear SVM	**0.59807**	**0.12268**	0.32750	**4.467**

Analysis: With this simple comparison, we can see that SVM is not appropriate for this set of data. I propose to remove it from the rest of the study.
Regarding Decision tree and K-nearest neigbors, it seems the second one is the best fitted for that data set with a very good precision score (~0.70) but the recall score is very flow meaning a big volume of false negative.
Consequently, I propose to keep both for tuning step.

9. TUNE THE SUPERVISED LEARING ALGORITHMS

For improving performances of both selected algorithms, I will use Python *GridSearchCV()* for exhaustive search over specified parameter values for an estimator.

Input parameters depend on the learning algorithm (bold: main paramaters):

- For Decision tree, I will play with:
  - criterion=['gini','entropy']
  - **max_depth**=np.arange(1,100,5)
  - min_samples_split=np.arange(2,20,1)
  - random_state=[5]
- For K-nearest neigbors
  - metrics= ['minkowski','euclidean','manhattan']
  - weights= ['uniform','distance']
  - **numNeighbors**= np.arange(3,12,1)

Regarding the scoring of the algorithm, I will play with 3 possibilites:

- precision
- recall
- f1

Hereafter the results:

Scoring	Algorithm	Precision	Recall	F1	Test time
Precision	K-nearest neigbors	**0.76418**	0.25600	0.38352	2.606
	Decision tree	0.46105	**0.43200**	**0.44605**	**1.766**
Recall	K-nearest neigbors	**0.76418**	**0.25600**	**0.38352**	2.417
	Decision tree	0.34506	0.25500	0.29327	**1.23**
F1	K-nearest neigbors	**0.76418**	0.25600	0.38352	2.394
	Decision tree	0.46105	**0.43200**	**0.44605**	**1.448**

Thanks with those results, we can conclude that:

- For the **best precision score**, it is better to get **K-nearest neigbors algorithm** with the following configuration (algorithm='auto', leaf_size=30, metric='manhattan', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')
- For the **best recall score**, it is better to get **Decision Tree algorithm** (class_weight=None, criterion='entropy', max_depth=6, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=5, splitter='best')
- For the best compromise (**F1 score**), it is better to get **Decision Tree algorithm** as well with the same configuration as for recall score

To chek feature selection accuracy, the same test has been made with all features (selected and removed) with no improvement observed.

10. **VALIDATION**

Eventually, for final validation, I choose to the best compromise between precision and recall , meaning the best F1 score.

As we have seen in the last chapter, the algorithm that meets this expectation is Decision Tree algorithm (see bellow the exact configuration of the classifier).

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=6,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=5, splitter='best')
        Accuracy: 0.85693       Precision: 0.46105      Recall: 0.43200         F1: 0.44605     F2:
0.43751
        Total predictions: 15000        True positives:  864    False positives: 1010   False negatives:
1136    True negatives: 11990
```

To validate the test, we use Python *StratifiedShuffleSplit()* that implement randomized K-fold cross validation method with the following configuration:

- **Test size: 0.1** (meaning 15 persons are predicted each time, 128 persons used for training)
- **Number of iterations: 1000** (meaning that we have 15 000 “persons test” on a real sample of 143 persons)

How is working this function?

- **The function stratify the data**
    - That means it split data depending on label value. In our case, it is very simple, we have only two possible values for labels: 0 (non poi) or 1 (poi)
    - It forbids having a sample with only 0 or 1 that could statically happen with 1000 iterations but could a problem for classifier assessment
- **The function extract randomized samples for test**
    - The data source is the output of last step (stratification)
    - The function returns index (representing couple of features/label) from the specified size (15 in our case)
- **The function loops the second step according the specified number of iterations**
    - Returns the number of splitting iterations in the cross-validator

⇨ We need to validate this way in order to limit overfitting and variability issues. Indeed, if we assessed our test on the same sample as we tune the classifier, we could face big different results on real data in case of overfitting algorithm.
It is the same reason as we split data into two categories: train + test, we want to have the algorithm the furthest as we can from sample specificities because that means the classifier will be accurate for most of samples and not for a particular one.

11. CONCLUSION

This study has demonstrated that no definitive response can be given to machine learning on real cases.

Indeed, the different steps of the project enabled to improve results but not with really high scores (precision 0.70, recall 0.43 at best) and not with the same method (if we improve recall score, it is at the expense of precision).

Now, in a real case, it would depend on the objectives of the data analyst.

Is it better to miss real POI in order to minimize false positive or in a opposite way would we prefer to get the most POI as possible in minimizing false negative?

Finally, to answer to initial project questions:

- How can we identify with a good accurate result the persons of interest related to Enron Fraud? (ie.: wich available features are the most valuable?)
    - ***We have identified that most of available features are valuable for identifying POI + addtitional one mainly focused on emails exchange***
- How can we optimize this Person of Interest Identifier thanks to Machine learning methods and tools?
    - ***Machine learning helped us to improve initial results thanks to:***
        - ***Feature selection***
        - ***Rescaling***
        - ***Algorithm choice***
        - ***Algorithm tuning***