P5: Identify Fraud from Enron Email

/Final project for *Intro to Machine Learning*/

By Igor Buinyi, igor.buinyi@gmail.com.

The goal of this project is to implement a machine learning algorithm to support a real-world decision making problem of whether a person is a 'person of interest' (POI). After the Enron scandal, more than 500,000 emails from Enron mail servers were open to public. Such amount of text and metadata cannot be normally assessed by a person, but machine learning is capable of quickly identifying hidden patterns in the megabytes of text or other data. In this project I demonstrate how features are extracted from text and how they are used in the classification model.

Note: This is an updated version with automatic feature selection process and implemented GridSearchCV function.

Contents

1. Remove outlier

We need to clean entry 'TOTAL' from the provided dataset because it does not represent a particular person (it just contains the sum of the corresponding financial and email features. Moreover, its values are too extreme.

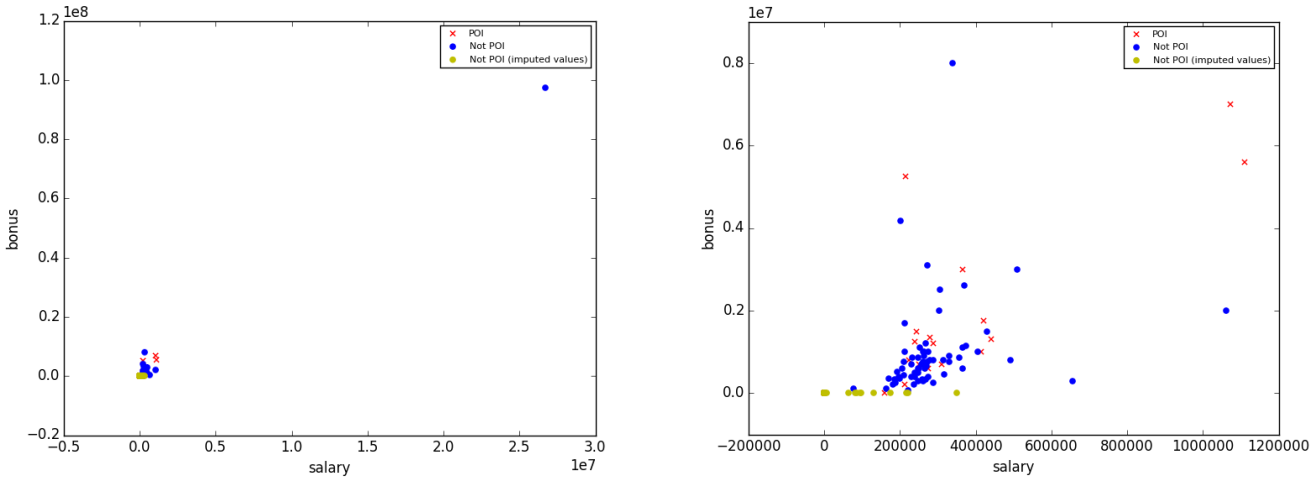


Fig1. Outlier in the data. Left plot: with outlier; right plot: without outlier.

2. Summarize data

The data summary is presented in the table below. Similar tables for POIs and Not-POIs are presented in the appendix.

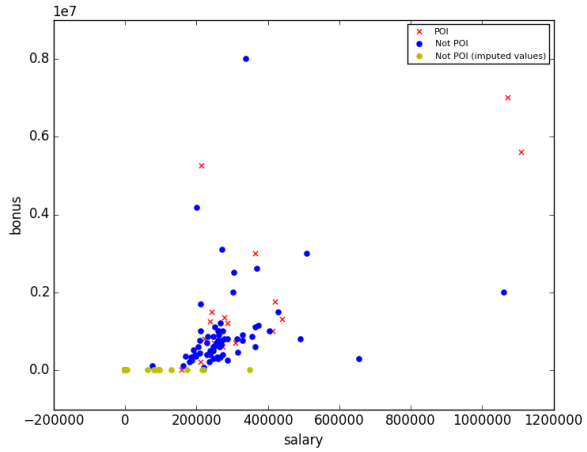
name	persons	cnt	cnt_NA	min	max	mean	sum
poi	145	145	0	False	True	0.124	18.0
bonus	145	81	64	70000	8000000	1201773.1	97343619.0
deferral_payments	145	38	107	-102500	6426990	841602.5	31980896.0
deferred_income	145	48	97	-3504386	-833	-581049.8	-27890391.0
director_fees	145	16	129	3285	137864	89822.9	1437166.0
email_address	145	111	34	None	None	0.0	0.0
exercised_stock_options	145	101	44	3285	34348384	2959559.3	298915485.0
expenses	145	94	51	148	228763	54192.0	5094049.0
loan_advances	145	3	142	400000	81525000	27975000.0	83925000.0
long_term_incentive	145	65	80	69223	5145434	746491.2	48521928.0
other	145	92	53	2	10359729	465276.7	42805453.0
restricted_stock	145	109	36	-2604490	14761694	1147424.1	125069226.0
restricted_stock_deferred	145	17	128	-1787380	15456290	621892.8	10572178.0
salary	145	94	51	477	1111258	284087.5	26704229.0
total_payments	145	124	21	148	103559793	2623421.2	325304226.0
total_stock_value	145	125	20	-44093	49110078	3352073.0	419009128.0
from_messages	145	86	59	12	14368	608.8	52356.0
to_messages	145	86	59	57	15149	2073.9	178352.0
from_poi_to_this_person	145	86	59	0	528	64.9	5581.0
from_this_person_to_poi	145	86	59	0	609	41.2	3546.0
shared_receipt_with_poi	145	86	59	2	5521	1176.5	101176.0

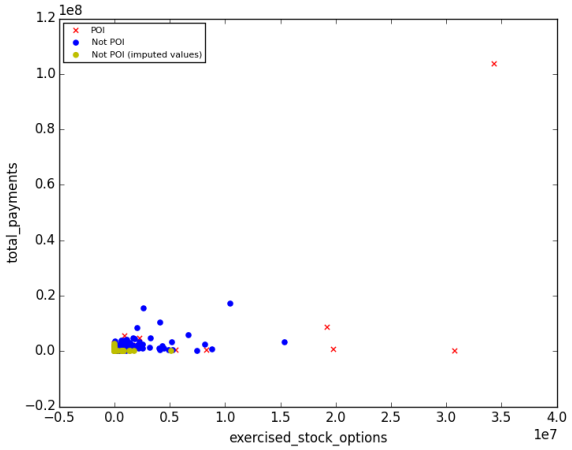
There are 145 persons in the data, including 18 POIs. There are total 20 features (not counting 'poi'), including 15 financial features and 5 email features. A few of the features have more than half missing values.

I will convert 5 email features to 3 ratios with values between 0 and 1 (shown in the table below) because the frequency of emails from/to POI is assumed to be more relevant than the number of emails.

name	cnt_all	cnt	cnt_NA	min	max	mean	sum
from_poi_to_this_person_ratio	145	86	59	0.0	0.217	0.0	3.265
from_this_person_to_poi_ratio	145	86	59	0.0	1.0	0.2	15.829
shared_receipt_with_poi_ratio	145	86	59	0.018	1.001	0.6	52.379

A few scatterplots is show below. As compared with non-POIs, POIs on average had high salaries, received hefty bonuses, exercised more stock option and owned more stock. Therefore, financial features can be used for classification.





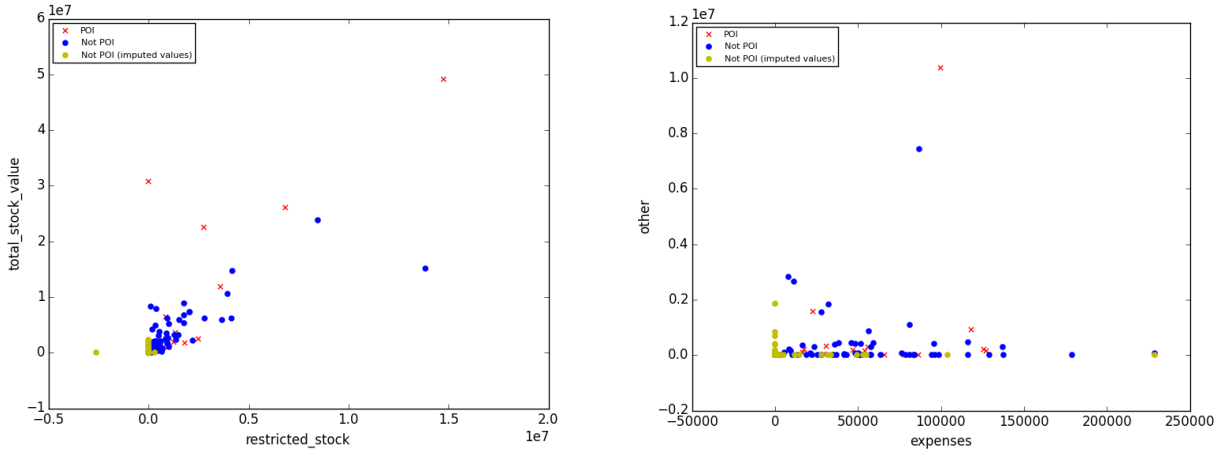


Fig.2. Scatterplots for financial features

Constructed email features are also useful for prediction as can be seen from Fig.3. The intensity of interaction between POI and POI is higher than between POI and non-POI.

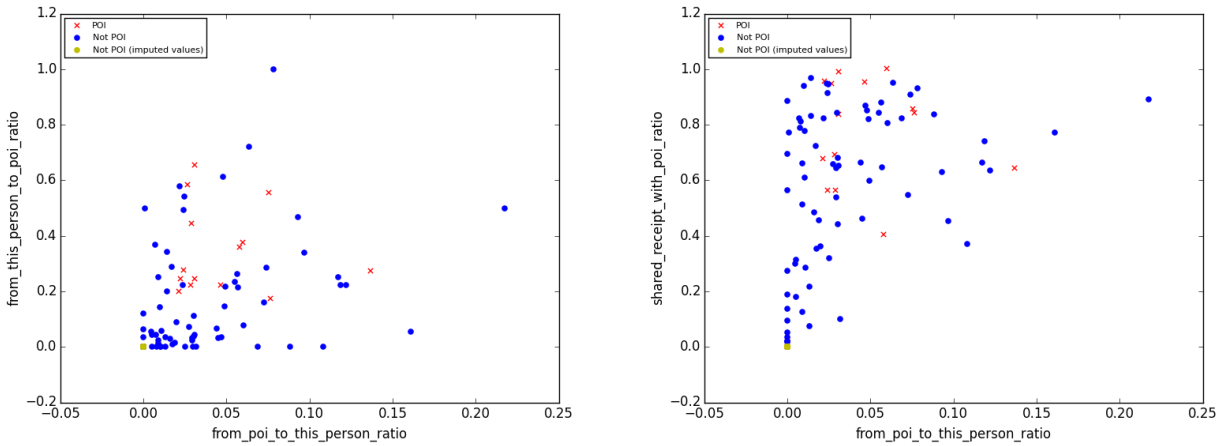


Fig.3. Scatterplots for email features

3. Mining for text features

Although it was not necessary for completing the project, I decided to explore the Enron email corpus and extract features useful for POI prediction. Basically, I would like to find out patterns in email text which differentiate POI from non-POI.

First, I checked all email addresses in “From:” field. I found 3 additional email addresses that are likely to belong to people from our dataset.

GATHMANN WILLIAM D: bill.gathmann@enron.com
LOCKHART EUGENE E: gene.lockhart@enron.com
NOLES JAMES L: james.noles@enron.com
However, I decided not to add these addresses to the dataset because I do not expect the performance to increase significantly. These three persons sent only few emails and all of them are non-POIs.

Second, I tried a variety of models to predict whether a POI was the sender or a recipient of a particular email. My goal was to extract the best words for prediction. The final algorithm is the following:

- From each of 517394 emails I extract the sender's email address and stemmed email text excluding the text after words "Original Message". So, I do not analyze the text that was replied or forwarded. Also, I exclude stopwords (names and reduced email addresses like 'vkaminsnsf') from the analysis.
- For the next stage I select only 52431 emails sent by people from our dataset, including 4211 emails sent by POIs.
- Then I use TfIdf vectorizer to extract 1500 text features from the selected emails.
- Then I implement a decision tree model to find the best features to predict whether an individual email was sent by POI. As a result, I got a list of text features. A few of the features ordered by the predictive power: **'ena', 'ect', 'guy', 'rob', 'review', 'turbin', 'forward', 'deal', 'thank', 'plan'.**

Given the small number of persons (N=145) in our dataset and a high number of potentially useful text features, we cannot use an occurrence of a word in the person's vocabulary (1 if the person used the word in one of emails and 0 if he did not) to predict whether he is a POI. Such a straightforward way would lead to overfitting, with random words becoming the best predictors of whether the person is a POI. However, using the word frequency ratio (ratio of emails with a word to the total number of sent emails) would overcome this problem. Each email with a 'poi-word' will increase the probability that the person is POI.

So, in **the third step** I calculated the number of emails from a particular email with particular predicting words. Later I will use the ratios of emails_with_the_word/total_sent_emails in the model.

All steps covered in this section are implemented in final_project_preprocess.py.

4. Building and tuning classification model (manual feature selection)

In the first submission I (not computer) decided which features are the most appealing to be included into the classification model. In this version of the report I also implement an automatic feature selection process using SelectKBest function. Please see section 5 for more details.

4.1. Feature selection

Although I tried different combinations, I present here my best selection.

Financial features: I use 8 features with the smallest number of missing values: **['expenses', 'other', 'bonus', 'total_stock_value', 'restricted_stock', 'salary', 'total_payments', 'exercised_stock_options'].** It should be noted that using the other financial features would slightly improve the decision tree model but the SVM model would become much worse with them.

Email features: As described in section 2, I transform 5 features to 3 ratios and use them: **['from_this_person_to_poi_ratio','from_poi_to_this_person_ratio', 'shared_receipt_with_poi_ratio'].**

Text features: I transform the data from the previous section to ratios emails_with_the_word/total_sent_emails (only if there are at least 10 emails sent from a particular email address). The top-3 features from previous section **['ena', 'ect', 'guy']** improve the model. The most of the others make it worse. So, I use only 3 these features.

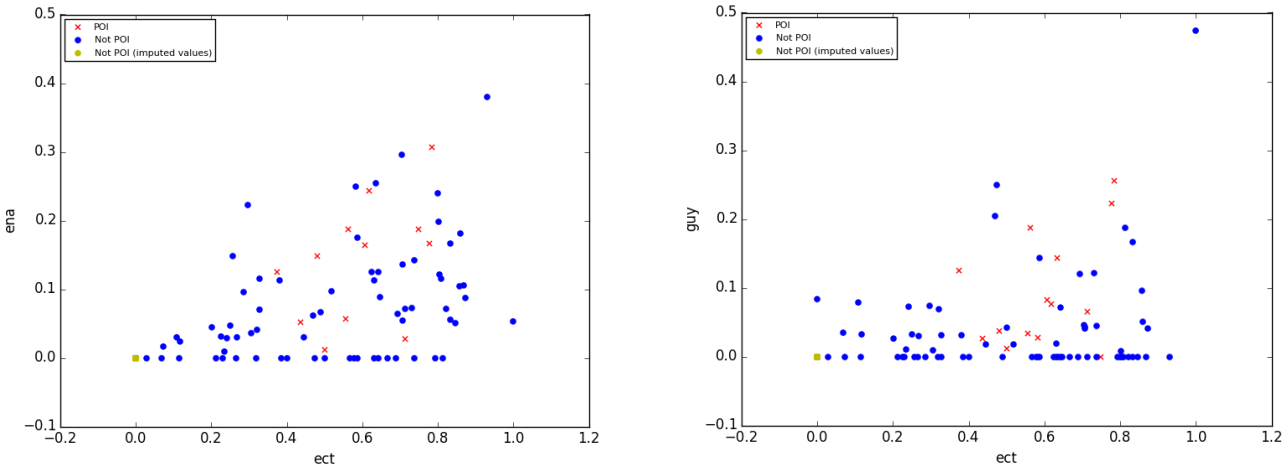


Fig.4. Scatterplots for text features

4.2. Imputing missing values.

Financial features were got from *enron61702insiderpay.pdf.* After analyzing the structure of the file, I realized that missing values correspond to just zeros. So, I replace 'NaN' with 0 in financial features.

Missing values in email and text features I decided to replace with the corresponding median values using *sklearn.preprocessing.Imputer.*

4.3. Feature scaling.

I used sklearn scaler to scale financial features. Email features are not scaled because they are already between 0 and 1.

Because of the necessity to implement imputing and scaling, I had to implement my own analog of *featureFormat* function (named *dict_to_nparray*). For the results to be compatible with *tester.py* and *featureFormat* function, I submit the dataset in the transformed form: financial features scaled and imputed, other features imputed.

4.4. Different classification models.

I used Naïve Bayes, SVM and Decision tree models on data split into training and testing sets (70%/30%).

I tried to reduce the number of features with PCA to avoid overfitting, but got worse results.

4.5. Cross validation.

Machine learning algorithms can capture important dependencies from the available data, but often algorithms learn too much. The algorithm can put much wait on occasional observations which do can not be well generalized. This is called overfitting. To avoid overfitting, we need to test the classifier performance on independent data points which were not present at the learning step.

In our case with small number of observations we need to generate many train-test splits since the classifier performance might be very different for different splits. I will use *StratifiedShuffleSplit* which is a modification of *KFold*. *StratifiedShuffleSplit* divides the sample onto K folds and preserves the percentage of samples of each class in the test and train set. Any algorithm works well only if there is a quite large absolute number of target class. It makes little sense to train a model if we have only few POI in the dataset. So, *sklearn. StratifiedShuffleSplit* ensures that there are enough POIs in both train and test sets.

If we used *sklearn.KFold,* we would get a few partitions with too small number of POIs. Such partitions would result in inferior performance of our classifiers.

4.6. Parameter tuning, cross validation and classification results.

The outcome of the classification process highly depends on the selection of algorithm and parameters. That is why tuning the parameters is a way to improve the algorithm performance. Ultimately, tuning helps us to obtain a classifier that solves a particular prediction problem.

In this section I pick a few sets of parameters and select one with the best performance. In Section 6 I will implement GridSearchCV function to do the job automatically.

Below are shown mean recall and precision obtained for 1000-fold validation (my implementation of *StratifiedShuffleSplit*, different from *tester.py*). Multifold validation is important because the results (especially in our case with quite small number of observations) highly depend on which points end up in training and testing sets. For instance, there many splits for which even the best model demonstrates zero precision and recall, so we cannot rely on model comparision for one particular data split.

Naive Bayes model	Accuracy	Recall	Precision
	0.847	0.305	0.334

SVM model tuning	Accuracy	Recall	Precision
Kernel=linear, C=1.0	0.891	0.130	0.468
Kernel=linear, C=10.0	0.877	0.168	0.445
Kernel=linear, C=100.0	0.859	0.209	0.377
Kernel=rbf, C=1.0	0.885	0.001	0.006
Kernel=rbf, C=100.0	0.836	0.289	0.289
Kernel=rbf, C=1000.0	0.824	0.313	0.271
Kernel=rbf, C=10000.0	0.824	0.313	0.271
Kernel=rbf, C=10000.0, gamma=0.0001	0.889	0.132	0.460
Kernel=rbf, C=10000.0, gamma=0.001	0.850	0.200	0.277
Kernel=rbf, C=10000.0, gamma=0.003	0.842	0.330	0.332
Kernel=rbf, C=10000.0, gamma=0.005	0.842	0.383	0.350
Kernel=rbf, C=10000.0, gamma=0.007	0.841	0.395	0.349
Kernel=rbf, C=10000.0, gamma=0.01	0.835	0.386	0.328
Kernel=rbf, C=10000.0, gamma=0.02	0.829	0.370	0.306

Rbf kernel produces better results than linear kernel due to significantly higher value of recall. Moreover, it takes much time to run SVM model with linear kernel at high values of parameter C.

Decision tree model tuning	Accuracy	Recall	Precision
max_depth=None, min_samples_leaf=1	0.828	0.328	0.290
max_depth=None, min_samples_leaf=2	0.841	0.283	0.303
max_depth=5, min_samples_leaf=1	0.832	0.321	0.299
max_depth=5, min_samples_leaf=2	0.843	0.281	0.308
max_depth=3, min_samples_leaf=1	0.849	0.297	0.328
max_depth=3, min_samples_leaf=2	0.848	0.279	0.304

Min_samples_leaf=2 improves precision at the cost of recall. Highest recall for max_depth=None, highest precision for max_depth=3.

4.7. Check whether text features improve model performance.

3 models: Naïve Bayes, SVM(Kernel=rbf, C=10000.0, gamma=0.007) and Decision Tree (max_depth=None, min_samples_leaf=1).

Model recall:

Features	**NB**	**SVM**	**DT**
Financial + email + text	0.305	0.395	0.328
Financial + email	0.253	0.242	0.297
Financial	0.238	0.177	0.290

Model precision:

Features	**NB**	**SVM**	**DT**
Financial + email + text	0.334	0.349	0.290
Financial + email	0.316	0.253	0.261
Financial	0.346	0.236	0.249

So, additional text features improve the performance a lot.

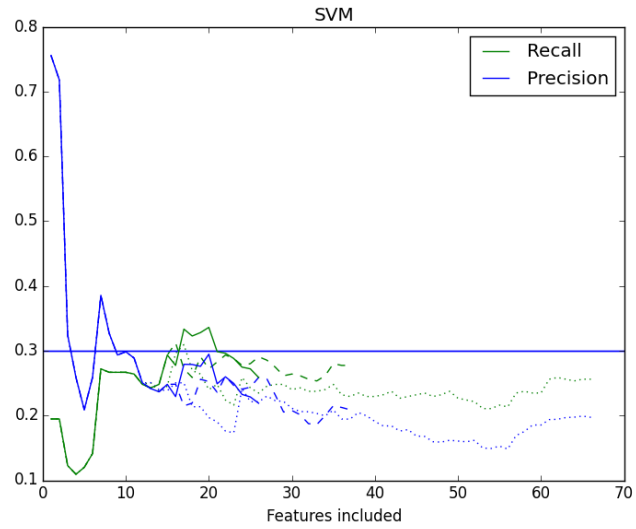
5. Automatic feature selection

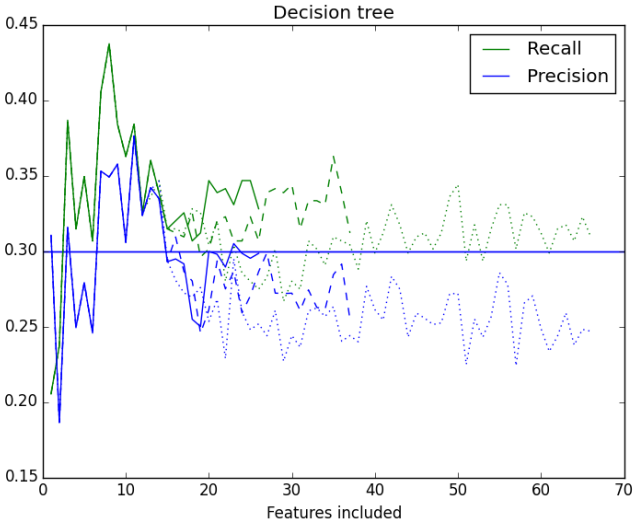
Following the reviewer comments, I implemented an automatic feature selection process. I picked a number of the most important features (the feature is more important if it describes more variance in the target variable) in the dataset using *sklearn.SelectKBest* function and tested the performance of SVM, DT and Naïve Bayes classifiers. It would be more accurate to use *GridSearchCV* or other tool to find the best parameter sets *for each combination of features and each classifier,* but such parameter optimization takes a lot of time. So, I decided to use best parameter sets from the previous steps.

Graphs for the three classifiers are shown in Fig.5. They were produced in three steps. First, N best features were select from **all features** available. Second, decision tree model were used to exclude features with small importance (delete feature if importance less than 0.33/n_features). N best features were selected **from the remaining features list.** Third, the same deletion procedure was repeated one more time.

The results are represented in the graphs with dotted (all features), dashed (after first deletion) and solid (after second deletion) curves. We can see the deletion of the least important features might improve the performance of classifiers.

The score ranking from Decision Tree classifier are presented in Appendix.





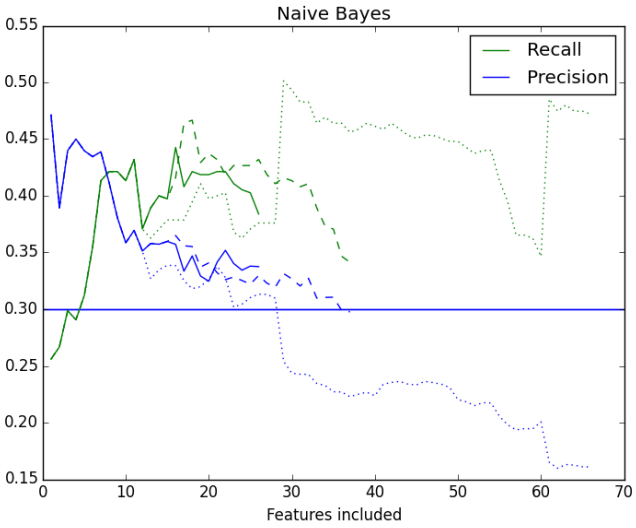


Fig.5. Classifier performance given N features are selected using SelectKBest function.

6. Automatic parameter tuning

I followed the suggestion of the reviewer and employed GridSearchCV for automatic parameter tuning. I used pipelines to tune the number of features and clf parameters simultaneously. Also, I tried different scoring functions, including those defined by me.

For SVM model with 30 best features I got the following result:

```
SVC(C=1, cache_size=200, class_weight='auto', coef0=0.0, degree=3, gamma=0.05,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
        Accuracy: 0.73060      Precision: 0.29144     Recall: 0.71300       F1: 0.41375     F2: 0.55301
        Total predictions: 15000      True positives: 1426   False positives: 3467  False negatives:  574  True
negatives: 9533
```

Unfortunately, the obtained recall is below 0.3.

For Naïve Bayes model with 5 best features I got the following result:

```
GaussianNB()
        Accuracy: 0.86000      Precision: 0.46603     Recall: 0.34300       F1: 0.39516     F2: 0.36212
        Total predictions: 15000      True positives:  686   False positives:  786  False negatives: 1314  True
negatives: 12214
```

This comparable with my final output obtained using a procedure from Section 4.

7. Results and explanation

Output of tester.py.

```
SVC(C=10000.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.007, kernel='rbf', max_iter=-1, probability=False,
  random_state=42, shrinking=True, tol=0.001, verbose=False)
        Accuracy: 0.84647      Precision: 0.42179     Recall: 0.40850       F1: 0.41504     F2: 0.41109
        Total predictions: 15000      True positives:  817   False positives: 1120  False negatives: 1183  True
negatives: 11880
```

I managed to get precision=0.42179 and recall=0.40850 from *tester.py*, well above the required 0.3.

It means that if my model identified a POI, the probability that this person is a real POI is 42% (precision). If there is a real POI in the dataset, my model identifies him as POI with a probability of 41% (recall). Therefore, in our case the output of the machine learning algorithm is not a self-contained tool for identifying POI. It can only help to support or refute other considerations.

I would also like to comment the words with the best prediction power ('ena', 'ect', 'guy'). **ECT** is an acronym for Enron Capital & Trade Resources, Inc., which changed its name to Enron North America Corp. **(ENA)** in 1999 (http://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=2963298). It is now known that this entity was at the center of financial manipulation within Enron (http://www.justice.gov/archive/dag/cftf/chargingdocs/delaineyindictment.pdf). So, our model correctly identified a place where major fraud occurred. The importance of word **'guy'** may indicate that POIs conducted fraud very cooperatively and their interaction was closer than formal work relations.

Appendix A. Summary tables

Summary table for POIs

name	cnt_all	cnt	cnt_NA	min	max	mean	sum
bonus	18	16	2	200000	7000000	2074999.9	33199999.0
deferral_payments	18	5	13	10259	2144013	519894.2	2599471.0
deferred_income	18	11	7	-3504386	-833	-1035313.5	-11388448.0
director_fees	18	0	18	None	None	0.0	0.0
email_address	18	18	0	None	None	0.0	0.0
exercised_stock_options	18	12	6	384728	34348384	10463793.7	125565524.0
expenses	18	18	0	16514	127017	59873.8	1077729.0
from_messages	18	14	4	16	3069	300.4	4205.0
from_poi_to_this_person	18	14	4	13	240	97.8	1369.0
from_this_person_to_poi	18	14	4	4	609	66.7	934.0
loan_advances	18	1	17	81525000	81525000	81525000.0	81525000.0
long_term_incentive	18	12	6	71023	3600000	1204862.4	14458349.0
other	18	18	0	486	10359729	802997.4	14453953.0
poi	18	18	0	True	True	1.0	18.0
restricted_stock	18	17	1	126027	14761694	2318620.8	39416553.0
restricted_stock_deferred	18	0	18	None	None	0.0	0.0
salary	18	17	1	158403	1111258	383444.9	6518563.0
shared_receipt_with_poi	18	14	4	91	5521	1783.0	24962.0
to_messages	18	14	4	225	7991	2417.1	33840.0
total_payments	18	18	0	91093	103559793	7913589.8	142444616.0
total_stock_value	18	18	0	126027	49110078	9165670.9	164982077.0
from_poi_to_this_person_ratio	18	14	4	0.021	0.137	0.0	0.665
from_this_person_to_poi_ratio	18	14	4	0.174	0.656	0.3	4.837
shared_receipt_with_poi_ratio	18	14	4	0.404	1.001	0.8	10.932

Summary table for Non-POIs

name	cnt_all	cnt	cnt_NA	min	max	mean	sum
bonus	127	65	62	70000	8000000	986824.9	64143620.0
deferral_payments	127	33	94	-102500	6426990	890346.2	29381425.0
deferred_income	127	37	90	-3367011	-1042	-445998.5	-16501943.0
director_fees	127	16	111	3285	137864	89822.9	1437166.0
email_address	127	93	34	None	None	0.0	0.0
exercised_stock_options	127	89	38	3285	15364167	1947752.4	173349961.0
expenses	127	76	51	148	228763	52846.3	4016320.0
from_messages	127	72	55	12	14368	668.8	48151.0
from_poi_to_this_person	127	72	55	0	528	58.5	4212.0
from_this_person_to_poi	127	72	55	0	411	36.3	2612.0
loan_advances	127	2	125	400000	2000000	1200000.0	2400000.0
long_term_incentive	127	53	74	69223	5145434	642709.0	34063579.0
other	127	74	53	2	7427621	383128.4	28351500.0
poi	127	127	0	False	False	0.0	0.0
restricted_stock	127	92	35	-2604490	13847074	931007.3	85652673.0
restricted_stock_deferred	127	17	110	-1787380	15456290	621892.8	10572178.0
salary	127	77	50	477	1060932	262151.5	20185666.0
shared_receipt_with_poi	127	72	55	2	4527	1058.5	76214.0
to_messages	127	72	55	57	15149	2007.1	144512.0
total_payments	127	106	21	148	17252530	1725090.7	182859610.0
total_stock_value	127	107	20	-44093	23817930	2374084.6	254027051.0
from_poi_to_this_person_ratio	127	72	55	0.0	0.217	0.0	2.6
from_this_person_to_poi_ratio	127	72	55	0.0	1.0	0.2	10.992
shared_receipt_with_poi_ratio	127	72	55	0.018	0.969	0.6	41.447

Appendix B. Score ranking from Decision Tree classifier

Please see section 5 for more datails.

Feature name	Score
exercised_stock_options	0.105
bonus	0.101
from_this_person_to_poi_ratio	0.064
shared_receipt_with_poi_ratio	0.063
total_stock_value	0.057
manag	0.048
expenses	0.044
other	0.042
guy	0.035
want	0.035
restricted_stock	0.028
deferred_income	0.028
total_payments	0.027
control	0.026
salary	0.019
toph	0.015
from_poi_to_this_person_ratio	0.015
corp	0.014
rob	0.014
long_term_incentive	0.014
ee	0.011
asset	0.010
pm	0.010
use	0.009
ect	0.008
given	0.007
inform	0.007
yes	0.006
plan	0.006
note	0.006
forward	0.006
howev	0.005
profit	0.005
turbin	0.005
know	0.005
ena	0.005
onli	0.005
val	0.005
ray	0.005
max	0.005
alloc	0.005
just	0.005

hard	0.005
look	0.004
ani	0.004
review	0.004
need	0.004
thank	0.004
pleas	0.004
fact	0.004
deal	0.003
ben	0.003
east	0.003
term	0.003
memo	0.003
america	0.003
goal	0.003
paul	0.002
target	0.002
view	0.002
meet	0.002
liz	0.002
deferral_payments	0.002
asap	0.001
restricted_stock_deferred	0.000
loan_advances	0.000
director_fees	0.000

Appendix C. Sources

Sklearn algorithms
http://scikit-learn.org/stable/modules/naive_bayes.html
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
http://scikit-learn.org/stable/modules/tree.html
http://scikit-learn.org/stable/modules/preprocessing.html
http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html
http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.KFold.html
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
http://scikit-learn.org/stable/modules/cross_validation.html

Parameter Tuning
http://machinelearningmastery.com/how-to-improve-machine-learning-results/

Remove empty line in csv output
http://stackoverflow.com/questions/8746908/why-does-csv-file-contain-a-blank-line-in-between-each-data-line-when-outputting

Imputation of missing values
http://stackoverflow.com/questions/11441751/how-to-get-svms-to-play-nicely-with-missing-data-in-scikit-learn
http://scikit-learn.org/stable/modules/preprocessing.html#imputation-of-missing-values

Replace value in Numpy arrays
http://stackoverflow.com/questions/19666626/replace-all-elements-of-python-numpy-array-that-are-greater-than-some-value

Array transformation
http://stackoverflow.com/questions/1966207/converting-numpy-array-into-python-list-structure

Pyplot legend
http://stackoverflow.com/questions/17411940/matplotlib-scatter-plot-legend

Enron
https://en.wikipedia.org/wiki/Enron
https://en.wikipedia.org/wiki/Enron_scandal
http://www.justice.gov/archive/dag/cftf/chargingdocs/delaineyindictment.pdf

ENA and ECT
http://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=2963298

Enron email dataset (version May 7,2015)
https://www.cs.cmu.edu/~./enron/