Identify fraud from Enron email and financial information

Question 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 goal of the project is to build a machine learning algorithm able to identify a person of interest (which will be names POI from now on) based on email and financial information. A POI is a person that is involved in the fraud that led to Enron bankruptcy.

The dataset provided to carry out the task is a collection of email and financial information that were made public as a result of the Enron scandal. A hand generated list of POI is provided to assist in the machine learning algorithm development. This list is based on the Enron fraud investigation outcome which provided individuals who were indicted, reached a settlement, plea deal with the government or testified in exchange for prosecution immunity. It is thought that the fact an individual is involved in the Enron fraud will be reflected in the email and financial information. Therefore the dataset can be used with the help of machine learning algorithm to identify patterns hidden in the email and financial information associated to POI and non-POI.

The dataset is made of 146 samples, 19 features and 1 label.

The feature list is made of
- financial information: '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 information: '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)

The label is 'poi' which can be either True (1) or False (0)

Three outliers are found in the sample data:

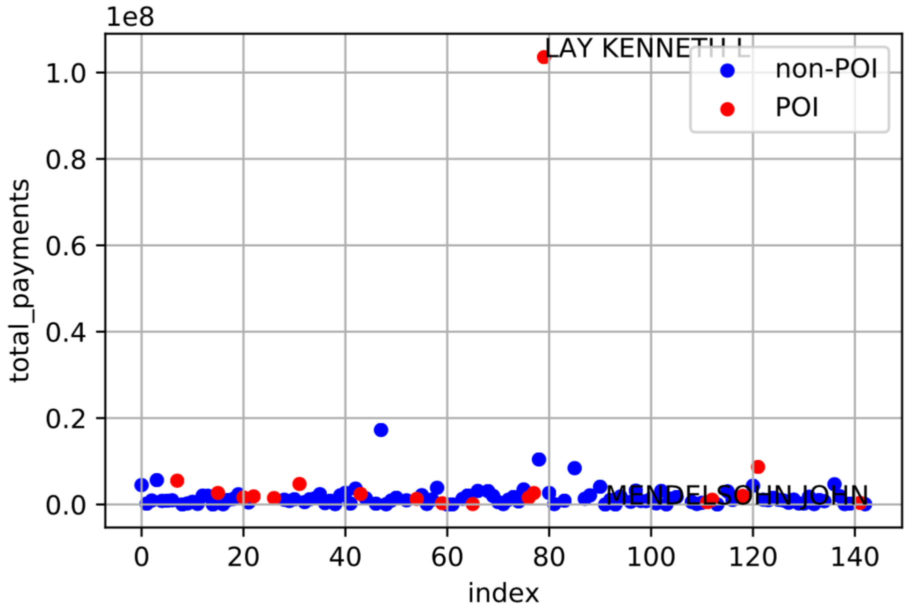
TOTAL —> sum of all entries in the financial information table
THE TRAVEL AGENCY IN THE PARK —> not a physical person
LOCKHART EUGENE E —> entry is totally empty

These outliers are removed from the dataset.

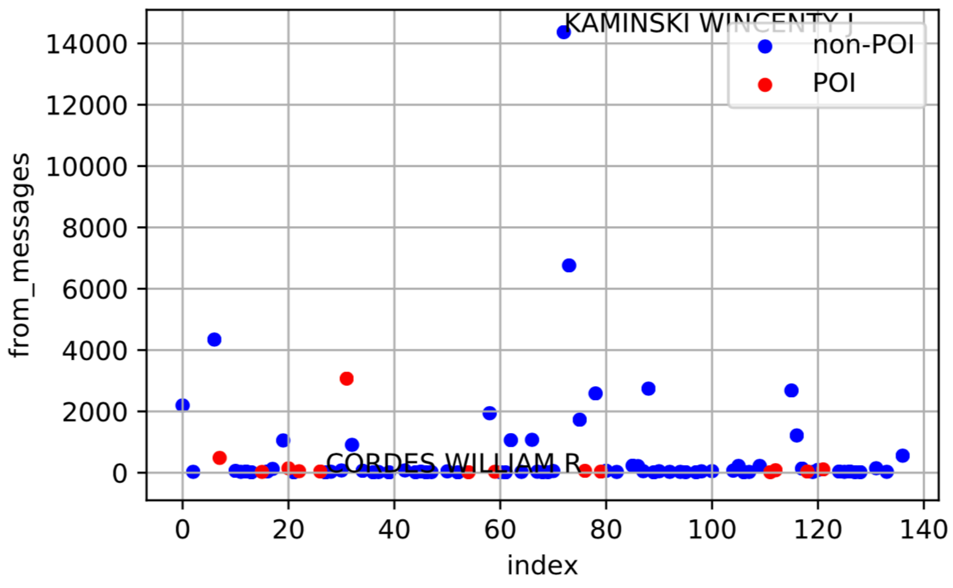
Two samples have wrong financial information when compared to 'enron61702insiderpay.pdf' file and are fixed:

BELFER ROBERT
BHATNAGAR SANJAY

Analysis of each individual feature of the dataset highlights a few outlying individuals. Plots shown below (taken from « scatter_plot_row_data.pdf » file) illustrates outliers for « total_payments » and « from_messages » features. Some belongs to the POI population (LAY and SKILLING) for the main financial items, but it is not surprising as they were CEO of the company. Other belong to the non POI population like KAMINSKI who sent 5 times more emails than any other person in the list. Again there might be some good reasons for that, like working in the communication department. Therefore it is chosen to keep the outliers in order to avoid losing information on an already quite small dataset.



proportion of null in non poi population: 16%
proportion of null in poi population: 0%



proportion of null in non poi population: 42%
proportion of null in poi population: 22%

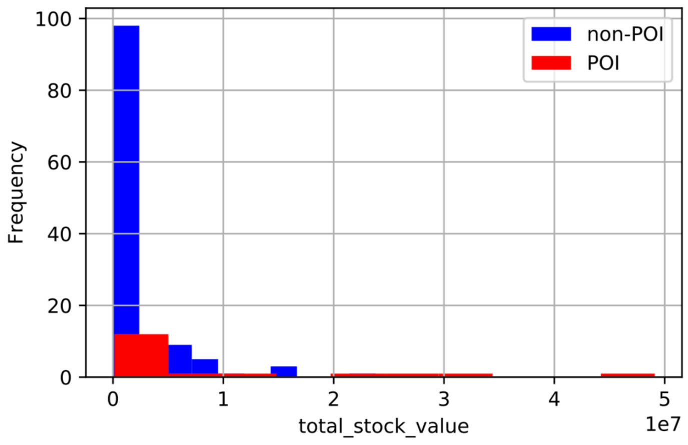
Question 2:

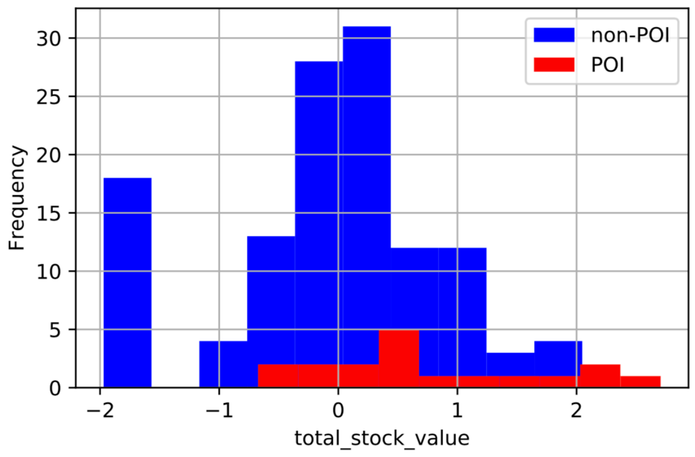
What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature 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"]

Before deciding which features shall be selected to build the POI identifier, the dataset need to be filled for missing information. Following strategy is put in place:

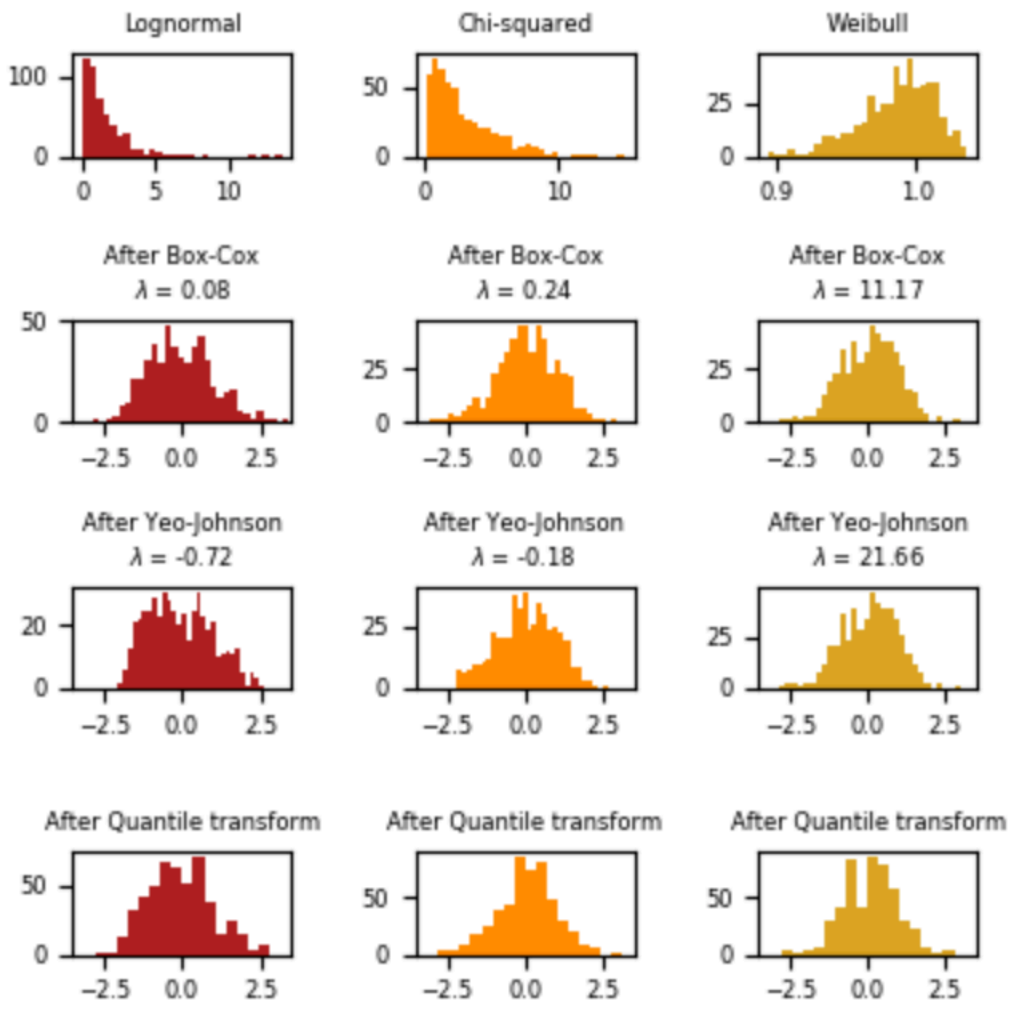
- financial information: it is checked with « total_payments » and « total_stock_value » features that missing financial information should be replaced with 0.
- email information: it is unlikely that missing email information should be replaced by 0. It would mean that respectively 22% and 42% of the POI and non POI populations did not send or receive any emails. It is chosen instead to replace missing email information by the median of the whole population (POI and non POI)

Then features are scaled in order to improve performance of some algorithms. Support vector machine for instance is performing better when each feature, used to derive the decision boundary between the label, has a normal distribution.
Some features are strongly skewed and it is possible thanks to a particular algorithm to scale and remove the skewness of the distribution. Plots below shows how « total_stock_value » can be modified with the appropriate scaling algorithm.





After some research on scikit-learn online documentation, the Yeo-Johnson algorithm provided in PowerTransformer class from sklearn.preprocessing package is chosen. As illustrated below, this algorithm is doing a great job at changing skewed distribution into normal distribution.
See https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html for more details.

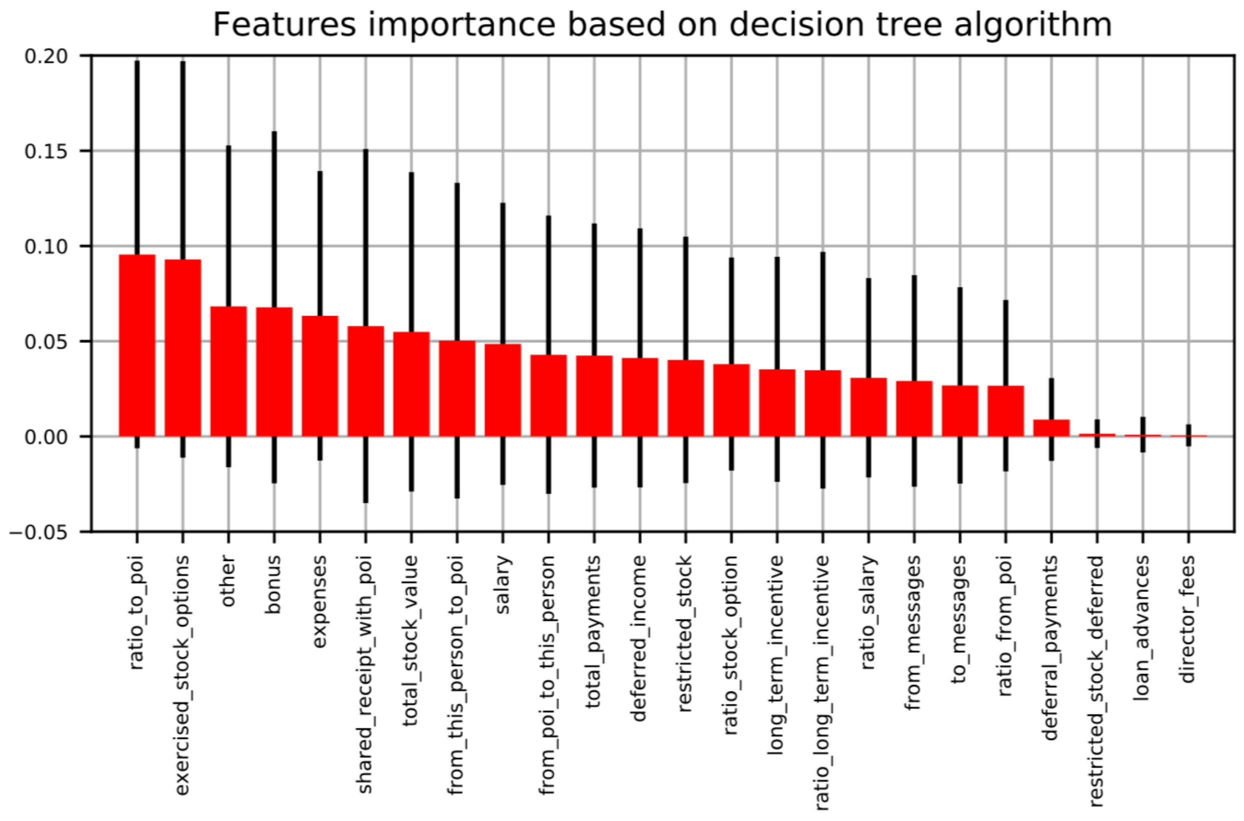


Several new features are then engineered:

ratio_salary = salary/total_payments
ratio_stock_option = total_stock_value/total_payments
ratio_long_term_incentive = long_term_incentive/total_payments
ratio_to_poi = from_this_person_to_poi/from_messages
ratio_from_poi = from_poi_to_this_person/to_messages

The aim is to normalize some data.
« salary», « total_stock_value » and « long_term_incentive » are normalized by « total_payments ». These new features tell us how much salary, stock option and long term incentive represents in the global income of an employee. It was thought that POI, who committed fraud will get in proportion more money from stock option than from salary. It turned out that these new features were not helpful.
Messages to or from POI are normalized by the total amount of messages sent or received. That way a person whose habit is to send/receive few emails will stick out of the engineered feature if its email exchange mostly concerns the POI population. « ratio_to_poi » proved to be a very strong new feature.

Finally, a decision tree algorithm is used to work out features importance. The POI identifier will only use the most important features in order to limit the number of dimension for the machine learning algorithm while keeping meaningful information.
The output of the decision tree algorithm (run with 100 trees) is the shown below.

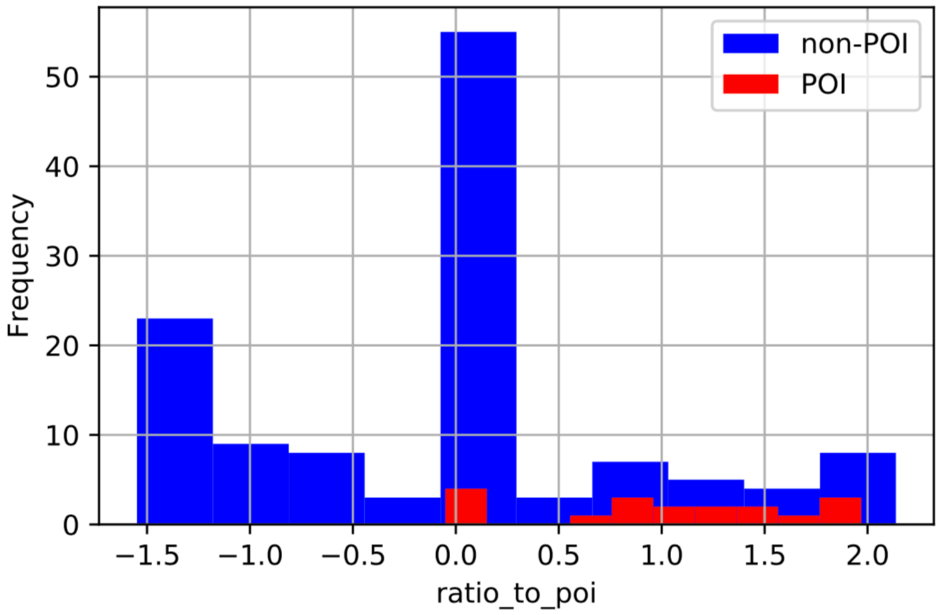


Feature ranking:
1 ratio_to_poi (9.56%)
2 exercised_stock_options (9.30%)
3 other (6.83%)
4 bonus (6.78%)
5 expenses (6.33%)
6 shared_receipt_with_poi (5.80%)
7 total_stock_value (5.49%)
8 from_this_person_to_poi (5.03%)
9 salary (4.86%)
10 from_poi_to_this_person (4.29%)
11 total_payments (4.25%)
12 deferred_income (4.12%)
13 restricted_stock (4.02%)
14 ratio_stock_option (3.80%)
15 long_term_incentive (3.52%)
16 ratio_long_term_incentive (3.48%)
17 ratio_salary (3.08%)
18 from_messages (2.92%)
19 to_messages (2.68%)
20 ratio_from_poi (2.67%)
21 deferral_payments (0.89%)
22 restricted_stock_deferred (0.15%)
23 loan_advances (0.09%)
24 director_fees (0.06%)

Ratio_to_poi and exercised_stock_options are clearly important features. Following five features are also taken onboard to build the POI identifier. List of selected features is therefore:

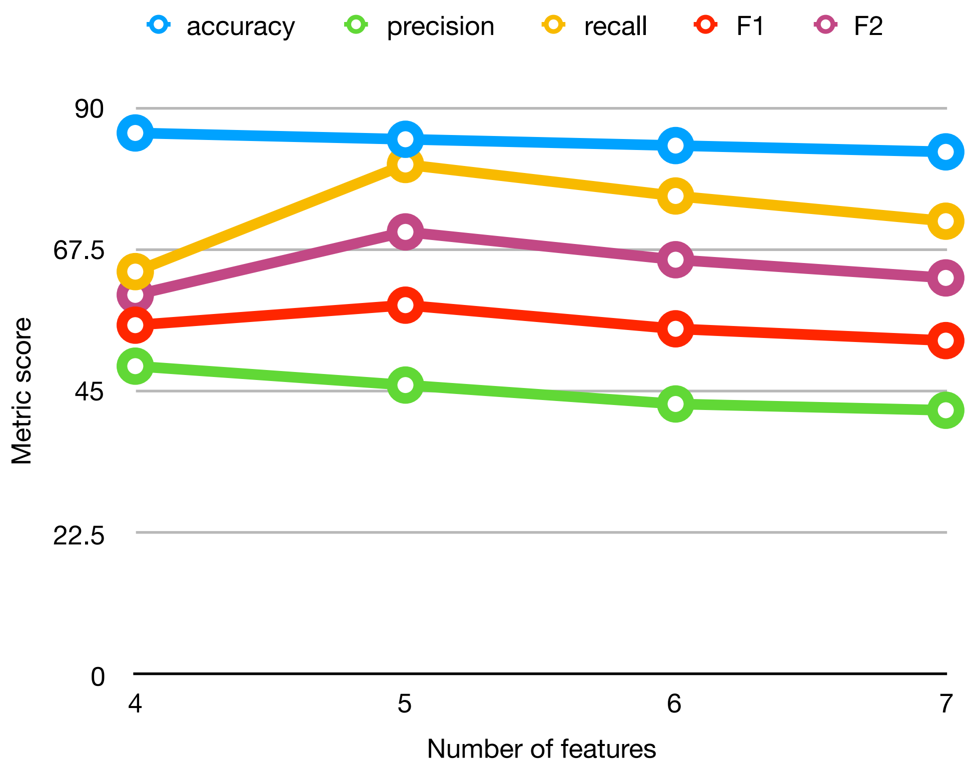
- Financial information: exercised_stock_options , other, bonus, expenses, total_stock_value
- Email information: ratio_to_poi, shared_receipt_with_poi

Analysis of the histogram plot of ratio_to_poi (after imputer and scaler transformation) shown below confirms that this feature has difference for both populations
`
- the non POI population is centered
- the POI population has a positive offset



Improvement for features selection

As suggested by Udacity reviewer, different numbers of features are tried. Tested machine learning algorithms do not behave the same way when number of features are reduced. Their hyper parameters are optimized differently and their performance varies. Plot below shows how main scores evolve with the number of features for Naive Bayes algorithm. The optimum is clearly reached for 5 features. For each algorithms the optimum can be reached for a different number of features: 5 for Naive Bayes and Adaboost, 6 for SVM and random forest, 7 or even more for KNN.



Since the best score is performed by Naive Bayes, it is chosen to keep only 5 features:

- Financial information: exercised_stock_options , other, bonus, expenses
- Email information: ratio_to_poi

Complete results of this study is provided in files GridSearchCV_resutls_x_features

Question 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"]

Five classifier algorithms from the scikit-learn library are used:
- Naive Bayes classifier: algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable
- Support Vector Machine classifier: algorithm builds a mathematical decision boundary with largest margin between the multi-label population
- K nearest neighbors: algorithm finds a predefined number of training samples closest in distance to the new point, and predicts the label from these
- Random forest classifier: a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting
- Adaboost classifier with decision tree: a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases

After tuning the hyper parameters for each algorithm, all classifiers except random forest reached the 30% criteria for precision, recall, F1 and F2 metrics. Naive Bayes classifier provides the best results with an impressive 81% recall and is selected.

« GridSearchCV_resutls.txt » provides a comprehensive comparison between all classifiers in term of GridSearchCV results and cross validation. Details of the GridSearchCV, cross validation strategy and metrics evaluation are provided in the remaining parts of the document.
Summary table is the following:

		Naive Bayes	**SVM**	**KNN**	**Random forest**	**Adaboost**
Grid search	Best F1 score	58 %	55 %	38 %	34 %	48 %
Cross validation	Accuracy	85 %	90 %	88 %	87 %	88 %
	Precision	46 %	65 %	59 %	53 %	58 %
	Recall	81 %	50 %	30 %	26 %	40 %
	F1	58 %	57 %	40 %	35 %	48 %
	F2	70 %	52	34 %	29 %	43 %
	Total predictions	15000	15000	15000	15000	15000
	True positives	1621	999	606	521	813
	False positives	1922	528	421	458	589
	False negatives	379	1001	1394	1479	1187
	True negatives	11078	12472	12579	12542	12411

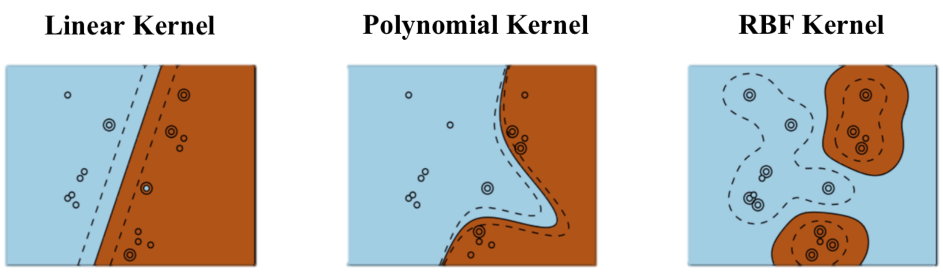
Question 4:

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"]

Most algorithms (all except Naive Bayes) need some hyper parameter tuning in order to best fit the dataset. In fact the same algorithm with two different set of hyper parameters can give totally different decision boundary and classification.

For instance hyper parameters of the support vector machine classifier affect:

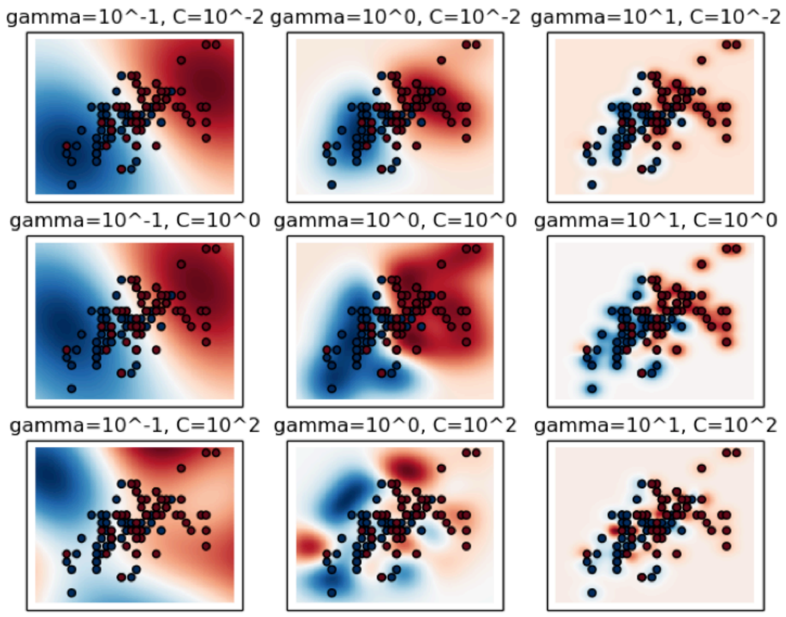
- the mathematical formulation of the decision boundary (linear, polynomial or even more complex with the rbf kernel)



C hyperparameter — *C plus gamma, degree and coefficient hyperparameters* — *C plus gamma hyperparameter*

- the margin between the decision boundary and the support vectors. C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the training error. Small C tends to emphasize the margin while ignoring the outliers in the training data, while large C may tend to overfit the training data.

- the distance of influence of a single training example. Gamma controls that distance with low values meaning 'far' and high values meaning 'close'



It is not always possible to know beforehand which hyper parameters to select. Grid search is used to figure out the best hyper parameters. The list of hyper parameters name and values for each algorithm is the following:

- Naive Bayes classifier: no hyper parameter tuning

- Support Vector Machine classifier:
    - kernel: linear, poly and rbf
    - C: 10, 100, 1000
    - gamma: 0.01, 0.1, 1, ‘scale’

- K nearest neighbors:
    - n_neighbors: 5, 10, 15
    - weights: distance and uniform
    - p: 1 and 2

- Random forest classifier:
    - min_samples_leaf: 1, 2 and 4
    - min_samples_split: 2, 4 and 6
    - n_estimators: 10, 20 and 40

- Adaboost classifier with decision tree:
    - n_estimators: 10, 20 and 40

Question 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"]

Usual practice in machine learning is to split the dataset in two parts:

- a training set which is used to fit the algorithm
- a testing set which is used to validate the algorithm

Strict application of this methodology is key to avoid an overfitting of the algorithm to the dataset. In fact, a machine learning algorithm should be able to perform well even on brand new data and not only on the dataset used to set the algorithm.

Given the small size of the dataset (only 143 samples) and the limited amount of POI (18), it is decided to use a cross validation strategy. The dataset is split several times in training and testing sets and the performance of the algorithm is based on the sum (or average) of the several evaluations. This strategy is applied both for tuning the algorithm hyper parameters with grid search and for testing the performance of the algorithm. It is recognized there is small risk of overfitting by proceeding that way but it is surely balanced by the improvement in accuracy brought by the usage of the complete dataset.

Cross validation uses the StratifiedShuffleSplit algorithm with a 10% ratio for testing. This algorithm is chosen because it respects the ratio of POI and non POI data in the training and testing set for each split. For grid searching the number of split is set to 100 to limit to computational time, while it is set to 1000 for testing.

Question 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 classifying tasks several metrics are available. The choice of the metric depends on the objective of the task and there is no general recommandation.

All metrics are derived from the confusion matrix results:

	Conditions positive	Conditions negative
Predicted condition positive	True positive	False positive Type I error
Predicted condition negative	False negative Type II error	True negative

$$\text{-}\ accuracy = \frac{true\ positive + true\ negative}{number\ of\ data}$$

In case of skewed classes (which is the case for the poi identifier dataset), accuracy is not indicated because the result of one class will out-weight the result of the other classes. Accuracy is also not able to indicate in which direction the classifier is making mistakes.

$$\text{-}\ precision = \frac{true\ positive}{true\ positive + false\ positive}$$

Out of all the items labeled as positive, how many truly belong to the positive class. This is used to evaluate performance of the algorithm in the test part.

$$\text{-}\ recall = \frac{true\ positive}{true\ positive + false\ negative}$$

Out of all the items that are truly positive, how many are correctly classified as positive. Or simply, how many positive items are 'recalled' from the dataset. This is also used to evaluate performance of the algorithm in the test part.

$$\text{-}\ f1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

The F1 score is the harmonic average of the precision and recall. It is used as an optimisation criteria in the grid search algorithm.

For the POI identifier task, my recommandation is to favor recall against precision so that the algorithm is able to identify as many POI as possible. Further investigation on a limited number of person can then confirm the real implication of the person in the Enron fraud. Average performance of the Naive Bayes algorithm is the following

	F1 score grid search	**Accuracy**	**precision**	**Recall**	**F1**	**F2**
Naive Bayes	58 %	85 %	45 %	81 %	58 %	70 %

Conclusion

I really enjoyed working on that project. At some point it became a game, almost a challenge, to figure out the best set up to capture as many POI as possible. I learned a lot on the machine learning methodology (train test split strategy, cross validation …), on the main algorithms (Naive Bayes, Support Vector Machine, KNN, Random Forest, Adaboost) and evaluation means. It also taught me how to implement such methodology in python with scikit learn. Learning by practice is definitely the way to go.

I also realized how important it is to carefully clean and prepare the dataset before rushing into the machine learning. My first algorithms difficultly reached the 30% limit because I did not properly remove the outliers, correct the errors, scale and select the features. Once all this part was fixed, the scores doubled all of a sudden!

In a last attempt to further improve the algorithm I tried to combine them in a voting classifier but the performance did not improve due to the fact that Naive Bayes is already outperforming the other classifiers. I'm convinced though that some combination could be done to take credit of both the high recall of Naive Bayes and high precision of KNN and SVM. Any suggestion?