Enron Submission Free-Response Questions

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 this project is to identify the person of interest involved in the Enron fraud case from 144 employees (in 146 data points) by applying machine learning algorithms on Enron email + financial dataset including 21 features for each person. There are 18 POIs, 128 non-POIs. Features with missing values include: bonus, salary, total payment, email address and so on…

   There are 2 types of outliers in the financial dataset: "Total" value which are Excel quirk and should be replaced; or people receiving far higher salary, stock option, bonus, incentives,… than other employees (these outliers are suspicious and should be kept to investigate). After investigating the "Enron insider pay" file, I also remove "THE TRAVEL AGENCY IN THE PARK" as it doesn't represent a person

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 importance 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", "properly scale features", "intelligently select feature"]**

   I believe different features lists will affect each algorithm's performance differently. Therefore, I applied two ways of features selection: manually and selectKbest. With each approach, I tune the parameters of each algorithm and apply the best_estimator on the feature list to decide on the best feature and algorithm combination. The initial features list include:

   (1) Financial features such as 'salary','bonus', 'long_term_incentive','total_payments', 'exercised_stock_options', 'total_stock_value';

   (2) Email related features: 'fraction_to_poi', 'fraction_from_poi','shared_receipt_with_poi'. There are some features that I didn't include in the financial features list (loan advances, deferred income, deferral payments…) as these features contain a lot of NaN values.

   I created 2 new features: "fraction_to_poi", "fraction_from_poi"

   "fraction_to_poi" = "number of messages from this person to poi"/"total number of receiving messages"
   "fraction_from_poi" = "number of messages from poi to this person"/"total number of receiving messages".

After creating these features; however, I didn't use them towards the end of the project because: (1) they can cause data leakage given that we are not supposed to know the number of POIs in our dataset, (2) when I used them in some algorithms, they couldn't produce a very strong performance (all algorithms report f1 score around 0.3 or less), (3) they are not even preferred by SelectKbest (they have low SelectKBest score)

I only performed feature scaling on KNearestNeighbor (KNN) because only this algorithm employs Euclidean distance to compare the distance of data points to make decision. However, after scaling is employed, the performance of KNN is driven down. Therefore, I didn't employ scaling in my final algorithm.

Before Scaling:
Accuracy: 0.86792 Precision: 0.61402 Recall: 0.38100 F1: 0.47023 F2: 0.41229
Total predictions: 13000 True positives: 762 False positives: 479 False negatives: 1238
True negatives: 10521

After Scaling
Accuracy: 0.79577 Precision: 0.33985 Recall: 0.34750 F1: 0.34363 F2: 0.34594
Total predictions: 13000 True positives: 695 False positives: 1350 False negatives: 1305
True negatives: 9650

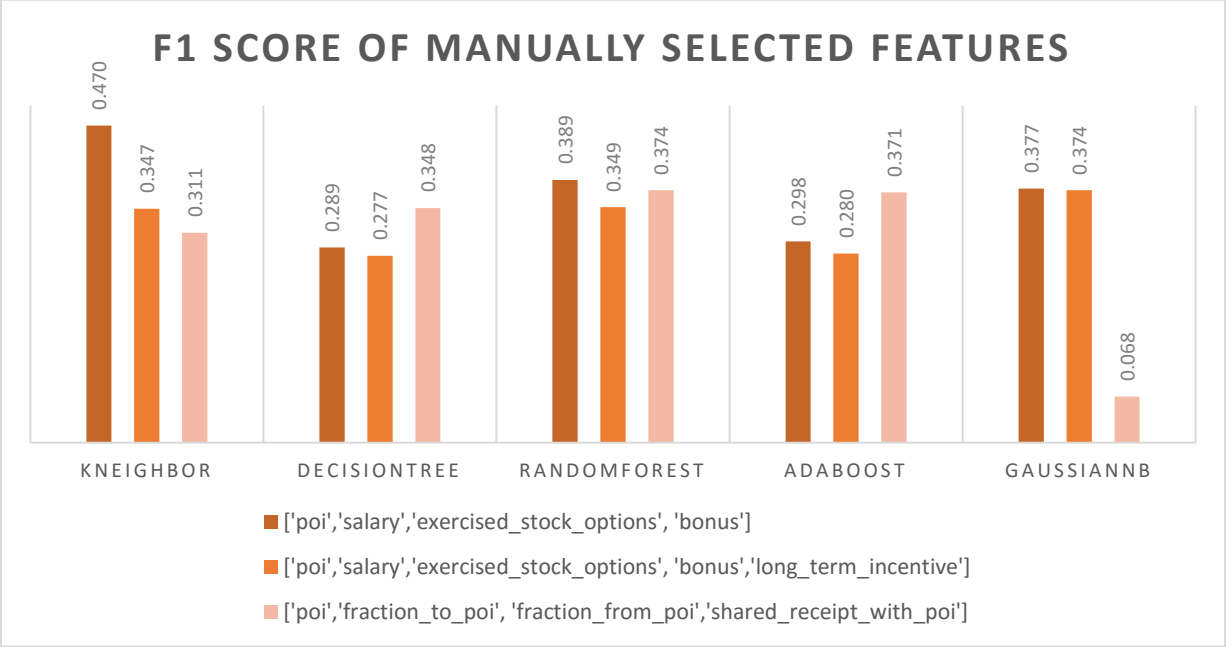
When using SelectKBest, I actually feed the whole list of features above to the algorithm using f_classif for ANOVA F-test selection. I test from 3 to 6 number of k on different algorithms and record the performance on each. Score of each feature in descending order.

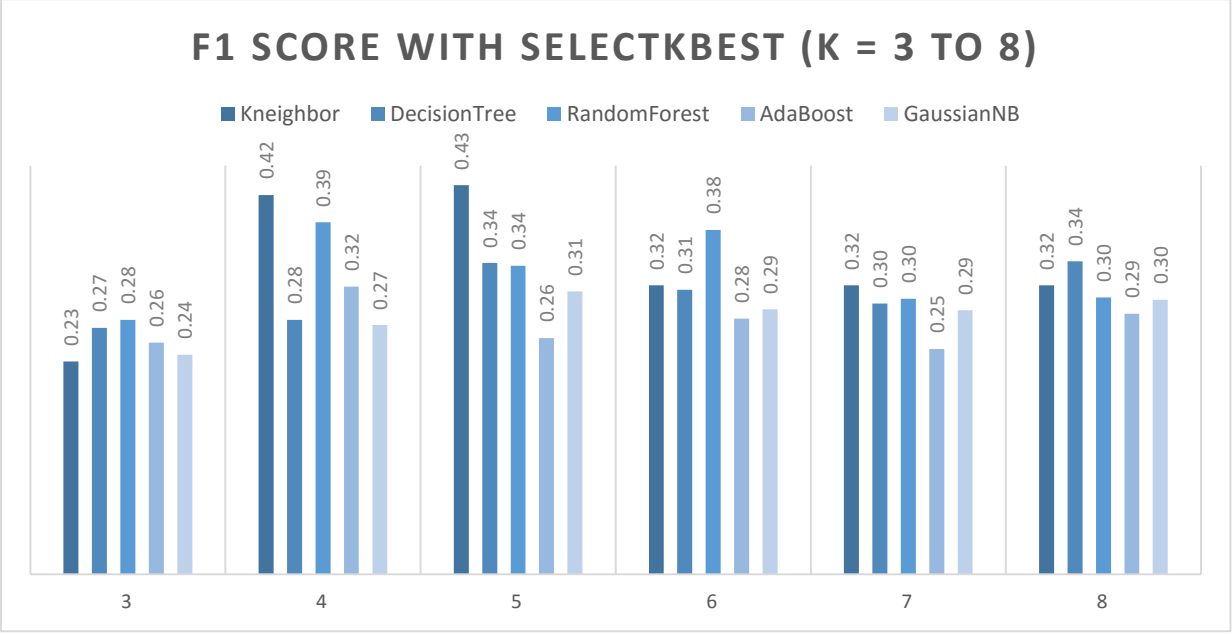
	scores
exercised stock option	24.53
total stock value	23.89
bonus	20.52
salary	18.02
fraction to poi	16.17
long-term incentive	9.77
total payment	8.67
shared receipt with poi	8.43
fraction from poi	3.05

When choosing features manually, I look at features information in EnronInsiderPay and draw scatterplot of each selected pair of features to find the correlation and outliers (as in scatterplot.py). At the initial inspection, I think scatterplots between salary and bonus, as well as long term incentive and total payment show very suspicious outliers. On the other hand, the scatterplot between exercised stock option and total stock value shows very high correlation. As total payment and total stock value are computed using other features (such as salary, bonus or stock options), I don't want to include these features in my list. Finally, I ended up with 4 features left: salary, bonus, long term incentives and exercised stock options. After fitting them to different algorithms, I only choose: salary, bonus and exercised stock options as the most effective list of features.

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"]**

This project is supervised classification learning, where a set of features and labels is used to train and another set is used to identify POIs from non-POIs. I chose multiple classification algorithms to perform the task: KNearestNeighbor, DecisionTree, RandomForest, AdaBoost and GaussianNB. I didn't use support vector machine (SVM) because SVM is not suitable for highly unbalanced dataset like this. It can easily get overfitting, which result in 0 true positive and low prediction power. After fitting these classifiers on various feature set, I realize that the model's performance differs not only between algorithms but also between different set of features.





No of K	Feature name
3	['poi', 'salary', 'total_payments', 'exercised_stock_options']
4	['poi', 'salary', 'total_payments', 'exercised_stock_options', 'total_stock_value']
5	['poi', 'salary', 'bonus', 'total_payments', 'exercised_stock_options', 'total_stock_value']
6	['poi', 'salary', 'bonus', 'long_term_incentive', 'total_payments', 'exercised_stock_options', 'total_stock_value']
7	['poi', 'salary', 'bonus', 'long_term_incentive', 'total_payments', 'exercised_stock_options', 'total_stock_value', 'fraction_from_poi']
8	['poi', 'salary', 'bonus', 'long_term_incentive', 'total_payments', 'exercised_stock_options', 'total_stock_value', 'fraction_to_poi', 'fraction_from_poi']

To my surprise, the most effective combination of feature and algorithm is Knearestneighbor(KNN) on feature list of ['poi','salary','exercised_stock_options', 'bonus'] chosen manually. And this is the combination I ended up using at the end of my project.

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? (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 item: "tune the algorithm"]**

Tuning the parameter of an algorithm is to apply different set of parameter combination within an algorithm to improve its performance. If we don't tune the parameter, we will only use the default parameters, which may not be optimal for our dataset. I determine several values of the parameters in my algorithm (eg. KNN), then I use GridSearchCV in sklearn to fit each combinations of those parameters on a set of training data. Finally, I apply the resulting best_estimator of GridSearchCV to the test_classifier to test the score of the algorithm.

The resulting best parameters for KNN: algorithm='auto', leaf_size=30, metric='chebyshev', metric_params=None, n_neighbors=4, p=2, weights='distance'.

Number of neighbors used to define label of a query point is 4 with closer neighbors of a query point will have a greater influence than neighbors which are further away. For example, among 4 nearest neighbors, if 3 of them turn out to be non-poi, that person will be defined as non-poi and vice-versa. The metric used is 'chebyshev' distance: *sum(max(|x - y|))* instead of the default Minkownski with p =2 (equivalent to the standard Euclidean Distance.

I have 1 problems with tuning parameters regarding Warm_start in RandomForest().The default warm_start is False. When warm_start is False, f1 score of Random Forest is 0.29. But warm_start set to True boosted the performance incredulously.

Accuracy: 0.94536 Precision: 0.92616 Recall: 0.67100 F1: 0.77820 F2: 0.71013 Total predictions: 14000 True positives: 1342 False positives: 107 False negatives: 658 True negatives: 11893

The reason for this is because we don't have a hold-out test set for the data, but shuffle the data 1000 times and drawn different folds for training and testing. "When warm_start set to True, RandomForest reuse the solution of the previous call to fit and add more estimators to the ensemble". This causes data leakage and hence, the model is invalid.

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 item: "validation strategy"]**

   Cross validation means separating the data to training set and test set. Then we can train a model on the training set, then test it on an independent test set. This reduces overfitting and data leakage when applying model on the same dataset that it is previously trained on. Hence, it improves reliability and validity of the model.

   Given that the dataset is small (144 data points) and highly skewed towards non-POIs (only 18 POIs), the chance of splitting the dataset into subsets that are not representative is relatively high, so some algorithms can be underperformed. That's why a mere train_test_split in cross_validation is not effective in this case. We need a technique which not only allows algorithm to fit and test with multiple different sets of data every time they iterate, but also shuffle and randomize the dataset to avoid data leakage and help achieve robustness.

   Therefore, I validated the data using StratifiedShuffleSplit from tester.py with n_iter (number of folds) set to 1000. This means that the function will shuffle and split the data 1000 times to different randomized training sets and test sets. These sets are then used to fit and test by various Classifiers.

6. **Give at least 2 evaluation metrics and your average performance or 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"]**

   Accuracy: ratio of correct predictions on total predictions. It means how many POIs the model is able to predict correctly. The accuracy of my model is 0.867. This means 86.7% predictions are made correctly for both true positives and true negatives.

   In the highly imbalanced dataset like the Enron dataset, with a lot more non-POIs than POIs. Accuracy is not a good indicator to the most effective model. Because a model can flag most of POIs as non-POIs and still get a very high accuracy score. Therefore, we need other evaluation metrics such as Precision and Recall:

   Precision: the ratio between positive predictions out of total positive predictions the model returned. Out of 1241 positive predictions, 762 are correctly identified as POIs, 479 cases wrongly identified; which equivalent to a precision rate of 61.4%.

   Recall: the ratio of positive predictions out of total number of actual correct results. That means out of 2000 correct POIs, my model can only predict 762 true POIs and fail to flag 1238 other cases (false negatives), equivalent to recall rate of 38.2%.

My model has better precision score than recall score. This means that whenever a POI gets flagged in my test set, I know with a lot of confidence that it's very likely to be a real POI and not a false alarm. On the other hand, the price I pay for this is that I sometimes miss real POIs. To harmonize Precision and Recall score, we can refer to F1 score, which is the weighted average of precision and recall score. My f1 score is 0.46, which is the good combination of precision and recall score.

$$F1 = 2 * (precision * recall) / (precision + recall)$$

Reference:

Sklearn Documentation:
http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Udacity forum
https://discussions.udacity.com/t/random-forest-performance/39911
https://discussions.udacity.com/t/combine-pca-selectkbest-and-grid-search-to-select-features/39717/12

"I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.