Enron decision-making

Question 1:

Description:
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 create a learning model in which to identify persons of interest from the Enron employees. A person of interest in this context indicates a person who could have been of interest in the Enron financial fraud case.

The data used in this task consisted of financial information and information derived from the email messages of a sample of Enron employees. In the dataset, each row consisted of an employee of Enron. The employees had the following fields:

salary
to_messages
deferral_payments
total_payments
exercised_stock_options
bonus
restricted_stock
shared_receipt_with_poi
restricted_stock_deferred
total_stock_value
expenses
loan_advances
from_messages
other
from_this_person_to_poi
poi
director_fees
deferred_income
long_term_incentive
from_poi_to_this_person

The dataset is very tiny, with only 145 rows, 18 of which are POIs. The lack of data and heavy bias towards the output variables being non-POIs are both important to know when creating a learning model based on this data.

This data was analysed to explore the data and to see if there are any outliers or anomalies. From the data, the "employees" TOTAL and THE TRAVEL AGENCY IN THE PARK were omitted as they were not considered Enron employees (and TOTAL was deemed to be a heavy outlier).

Additionally, a significant amount of null-values were discovered in the data. This is understandable as the data is mostly financial data of the employees, therefore many of the fields contained financial values not relevant for many employees.

Question 2:

Description:
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", "properly scale features", "intelligently select feature"]

For the dataset, I created one additional feature: email_ratio_with_poi. This feature was basically the ratio of email communication made with a poi divided by all email communication. It followed this equation:

$$X_{email\_from\_poi\_ratio} = \frac{X_{from\_poi\_to\_this\_person} + X_{to\_poi\_from\_this\_person}}{X_{from\_messages} + X_{to\_messages}}$$

Therefore the total number of features in the dataset was 21. Feature scaling was used because it made data visualisations easier (no huge changes between the values of different features) and because some of the learning algorithms (such as SVM) required for the data to normalised. And also in algorithms such as PCA, the feature scaling makes the algorithms perform significantly better. Min-max normalisation was used with the following formula:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In choosing the final features, the following strategy was used:
- Start with all features
- Run SelectKBest for all features, plot results
- Pick the optimal set of features (**k** best ones).

I also tried to discover meaningful features by using PCA using the following strategy:
- Run PCA with the whole feature-set with different amount of components (**n**).
- Pick the optimal **n**.

The optimal feature/PCA setup was chosen by running the whole setup using a hand-selected set of algorithms (more on those in Question 3) and picking the setup that gave the best values for the selected test statistics (more on that in Question 5).

Next, the optimal feature-set was selected. For this, two different methods were used: PCA and SelectKBest. For SelectKBest, the features were selected by deciding k optimal features by running the features on a score-function, which was **f_classif**. F_classif is a score-function which computes the ANOVA F-value for each feature in the feature-space and then the SelectKBest-function can select a **k** amount of features that give the best score. The test statistic F defines the following formula:

$$F = \frac{explained\ variance}{unexplained\ variance}$$

The F-statistic defines the null hypothesis that 2 or more groups have the same population mean [1].

After the Feature scores were defined, the model was trained and tested using the selected **k** best features. The optimal amount **k** was decided by iteratively running the model against each **k** features in the feature-space and choosing the amount **k** with the highest test scores.

The scores from SelectKScores can be seen in Figure 1.

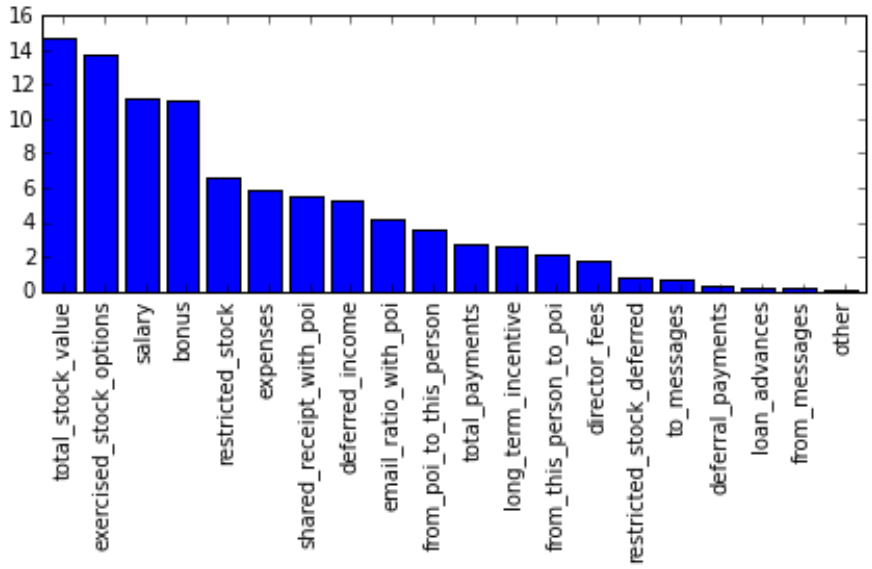


FIGURE 1 - FEATURE SCORES IN SELECTKBEST

As can be seen in Figure 1, the new feature seems to be quite average in the whole feature space, so at least it's not terrible.

For the test scores for different values of **k**, F1 and F2 was used. Also the values of Precision and Recall were plotted to make sure that the acceptable values of **k** had over 0.3 in both Precision and Recall. These values can be seen in Figure 2.

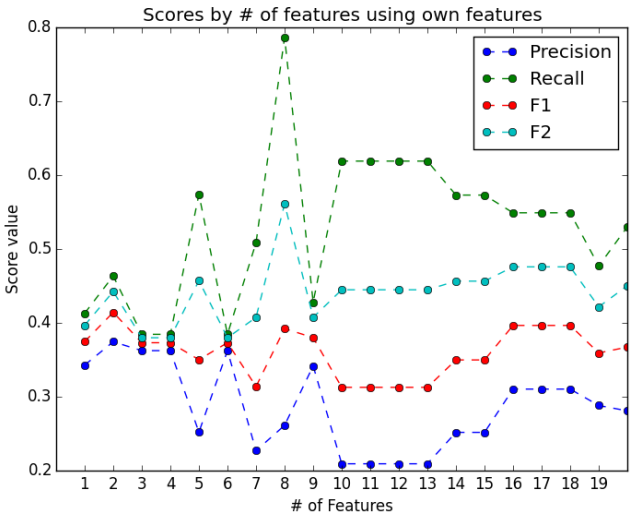


FIGURE 2 - PRECISION, RECALL, F1 AND F2 FOR DIFFERENT NUMBER OF FEATURES (K)

In Figure 2, it can be seen that with 8 features the Recall-value of the model jumps very high. Also, the Precision of the model stays relatively low. It seems that for Precision-values of over 0.3, the Recall-value plummets to very low values. What was also noticed was that the new feature added (email_from_poi_ratio) messes up the test scores quite a bit. This is evident in Figure 3 where the Precision, Recall, F1 and F2 are plotted for each **k** using the original dataset (without the new feature). In the figure, it is evident that the test scores are a fair bit higher than with the dataset including the own features. Therefore, it can be assumed that the inclusion of the feature **email_from_poi_ratio** does not do the overall model any good. Additionally, it seems impossible to gain Precision scores of over 0.3 and still maintain adequate F1 and F2-values.

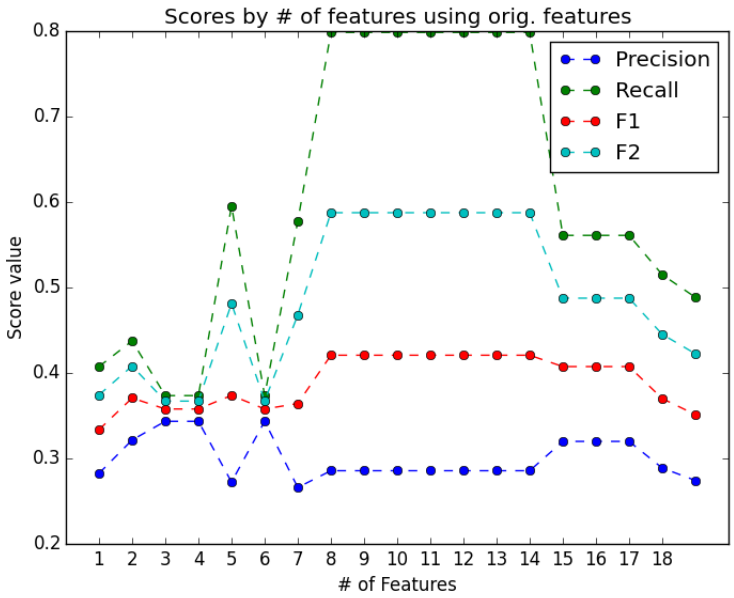


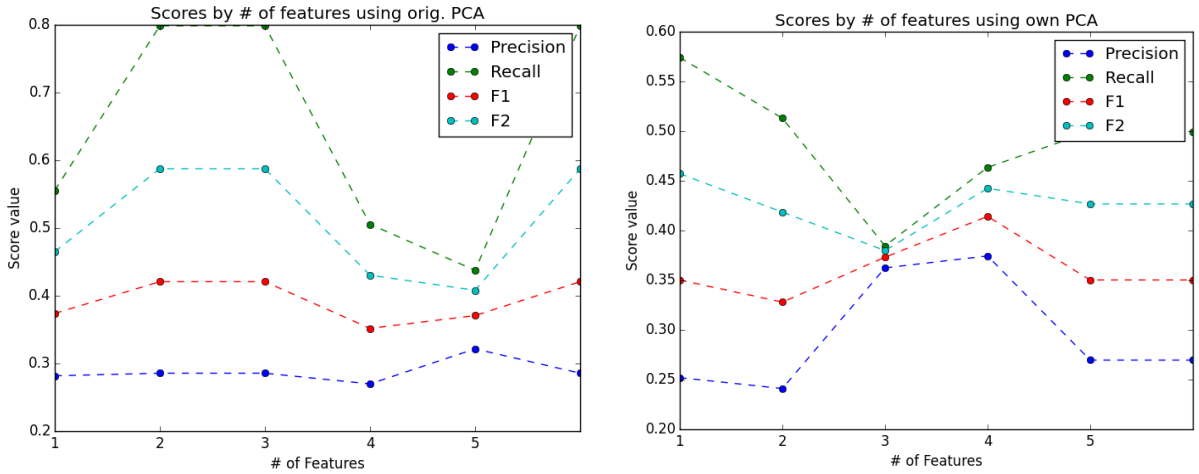
FIGURE 3 - PRECISION, RECALL, F1 AND F2 FOR EACH K USING ORIGINAL DATASET

What I found out was that the optimal setup was using k best features, when

$$k = 8$$

Next, the use of PCA was attempted to discover if using it creates a better set of features than using SelectKFeatures. The process of selecting the optimal amount of principal component **n** was similar to selecting the optimal features **k**. A value for the number of components was iteratively tested from the

value-space of [1, 2, …, 6]. The results for the original and own dataset can be seen in Figures 4&5.



FIGURES 4 & 5 - TEST SCORES FOR EACH NUMBER OF PRINCIPAL COMPONENTS IN PCA USING OWN AND ORIGINAL DATASETS

As evident in the Figures, the new feature affects the test scores in a significant way. Also, it seems to be very difficult to get Precision-values over 0.3 and still maintain adequate F1 and F2 scores. There definitely seems to be a tradeoff between Precision and Recall.

Due to project requirements, the decision on picking the best set of features has to be the following: Optimize F2-metric while making sure that the values of Precision and Recall are over 0.3. Why F2 was chosen is determined in Question 5.

With these criteria in mind, the following setup seems to be the optimal one: **PCA with the new feature, using 4 principal components**.

Question 3:

Description:
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 the model using three different algorithms:
- Naive Bayes
- SVM (Support Vector Machines)
- Random Forests with n = 10 and n = 20, where n is the amount of decision trees.

During the analysis, Random Forests were quickly dropped as the testing of the model (using the testing function in tester.py) was significantly slower than SVM or NB and consistently produced significantly worse test statistic-values than the previous two.

After running multiple tests using both different feature-setups and different amounts of PCA components, SVM seemed to always come ahead of NB. Therefore SVM was decided for the final model. The tests can be seen in folder **performance-tests/**.

Question 4:

Description:

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"]

As mentioned in Question 3, the algorithms used were Naive Bayes, SVM, and Random Forests. These were the validation methods used for the algorithms:

For Naive Bayes, no different parameters were used so no validation for the hyperparameters was necessary.

For SVM and RF, the optimal hyperparameters were selected using cross validation, namely GridSearch.

For SVM, the optimal kernel, C, and gamma parameters were selected. The parameter-space used in the optimization was:

Kernel	Gamma	C
['linear', 'rbf', 'poly']	[0.0005, 0.001, 0.005, 0.01, 0.1, 1, 10]	[0.1, 1, 50, 1e2, 5e2, 1e3, 5e3]

The following plot display the 30 best configurations of these parameters. To test the configurations, the F1-metric was used as it was the most suitable metric from the pre-defined metrics in the SKLearn's implementation of Grid Search.

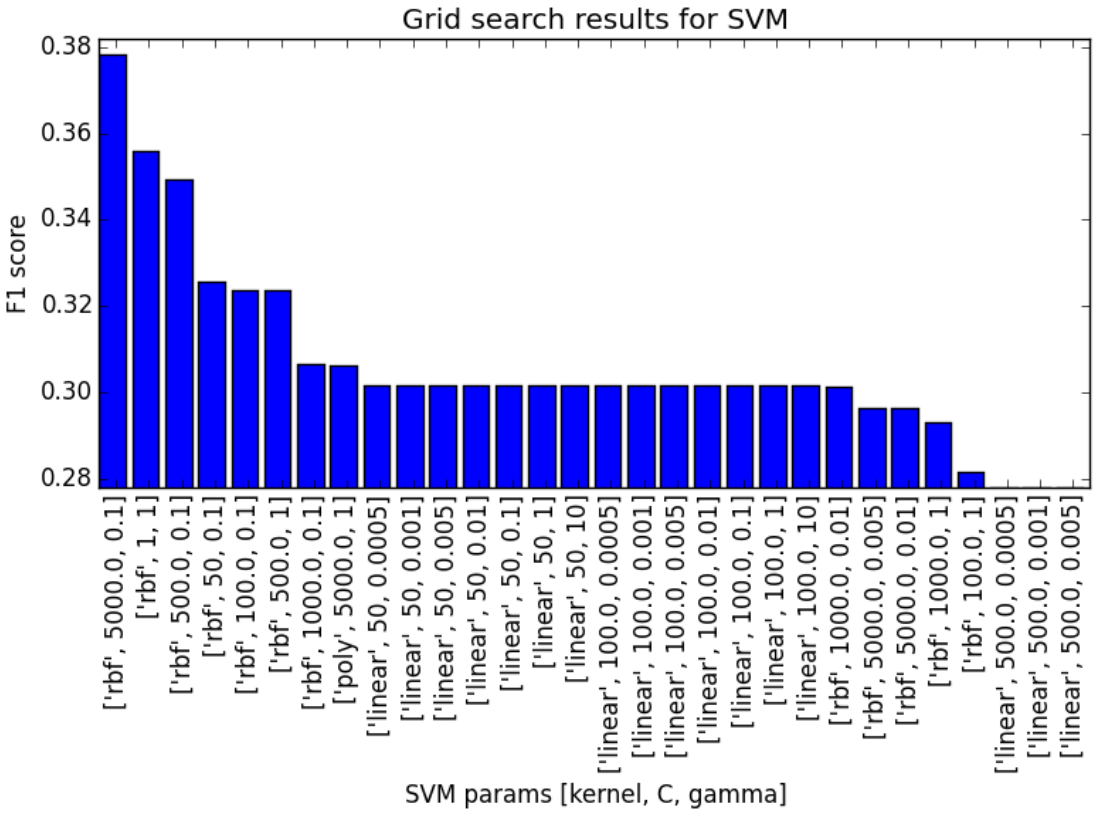


FIGURE 6 - SVM PARAMETER OPTIMIZATION SCORES

As Figure 6 shows, the rbf kernel clearly provides the best solutions. After that, the gamma-parameter seems to provide the best solutions with values of 0.1 or 1. The C-value seems to fluctuate quite a bit, with the two best solutions having C-values of 5000 or 1. The high C-value usually means a higher variance in the model [3], but in this case high C-parameter seems to have worked out well in cross-validation.

Obviously for the linear kernels, the C and gamma-values do not affect the model at all.

The following parameters were considered optimal and were hence used in the final model:

Kernel	C	Gamma
rbf	5000.0	0.1

For RF, the optimal criterion, min_sample_split, and max_features were considered. The parameter-space for the optimization was the following:

Criterion	min_sample_split	max_features	Number of trees
['entropy', 'gini']	[1, 2, 5, 10, 20, 50]	['sqrt', 'log2', None]	20

The following were considered optimal:

Criterion	min_sample_split	max_features	Number of trees
gini	1	1	20

The number of trees was selected by intuition as the amount of trees needed not be too high for a dataset this simple. Probably even using 20 trees was significantly overdoing it. The RF parameter tuning wasn't plotted due to the fact that SVM seemed to vastly outperform RF, as discussed in Question 4.

Question 5:

Description:
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"]

Validation can be said to involve the method of testing how the learning model handles data it has not previously seen. This is an effective way to create a setting in which the way a model can handle real, unknown data can be quantified using a properly chosen validation metric. To this end, the dataset is usually divided into training,test and validation sets. The training set is used to actually train the model. The validation set is used to determine the hyper parameters the model uses. And finally, the test set is used to test how well the model functions when its given data it has not seen before.

A classic mistake in validating a learning model can be to use a same subset of the data to train, validate, and test the model. This way the model will be severely prone to overfitting on that data subset. Another classic mistake is to use either training or testing data to validate hyper parameters of the model. This is a very common mistake in machine learning tasks. Actually, in this project I took the freedom to follow this anti-pattern and used the training set to validate my hyper parameters, but I only did that because during the project the dataset seemed far too small to divide it into three different subsets.

In this project, two main evaluation metrics were used to evaluate the learning model: F1 and F2. Evaluation metrics based on Precision and Recall was chosen because the datasets dependant classification variable was highly skewed towards negative classifications. Therefore, if using simple squared error-based metrics, simply creating a classifier that classifies each data row as negative would have produced good results. Therefore, to emphasize positive results, Precision and Recall-based methods were chosen as evaluation metrics.

Precision can be defined by the following formula:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$

And Recall using the following:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

Where true positives are the positive values in the test set that were correctly classified by the algorithm, false positives are the positive values that the algorithm incorrectly labeled as negative, and false negatives are negative values in the test set that the algorithm incorrectly labeled as positive.

Therefore, Precision measures the ratio of correct positive classifications and all positive classifications and Recall measures the ratio of correct positive classifications and real positive values. In another way, Precision can be said to measure the amount of type I errors and Recall to measure the amount of type II errors.

Since both Precision and Recall are important in this learning task, F1 was chosen because it combines both metrics into a single metric. The formula for the F1-metric is the following [2]:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

F2 was chosen, and in fact used as the main metric because it emphasises the Recall-value of the model. Recall was emphasized because I argue that it is far more severe to classify false negatives than it is to classify false positives. This is important in order to minimize the chance that anyone gets away with committing corporate fraud. The generic formula for F2 can be formalised as the following [2]:

$$F2 = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) * recall}$$

Where β = 2 was used in this project.

Question 6:

Description:
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"]

The model gave the following evaluation metric:

Algorithm	Features	Precision	Recall	F1	F2
SVM	PCA, n = 4	0.37439	0.46350	0.41421	0.44244

As seen in Question 3, this setup provided the optimal solution when estimated using the F2-score metric while still maintaining Precision and Recall-values over 0.3. With the model it seemed easy to optimize Recall but it was difficult get Precision-values of over 0.3. No matter how the model was tuned, there seemed to be quite a few False Positives. The abundance of False Positives means that the model is more likely to falsely label individuals as POIs.

As seen in Figures 2, 3 and 4, there is a possibility to optimize to model parameters to receive Recall-values of around 0.8. This would mean that the model will label 8 out 10 POIs correctly in the test data. But as the Precision-values would have been quite low, there would have a been a huge number of False Positives.

In practice, according to the test data, the model would classify around 37.4% of all its positive classifications correctly. And it would discover around 46.4% out of all the positive cases in the data. Therefore, the model itself should be improved as almost 2/3 of all POI classifications would be wrongly classified and over 1/2 of all real POIs would be unrecognised as POIs by the model.

The problem of the model in terms of bias-variance tradeoff would be that the model has too much variance to which using PCA proved somewhat effective to reduce that [4]. A real solution for that would be to significantly increase the size of the dataset. Currently the data 145 rows, 18 of which were labeled as positive. Additional features could be added as well, but I doubt it would help if more data is not added first.

Resources and References

[1] https://github.com/scikit-learn/scikit-learn/blob/a95203b/sklearn/feature_selection/univariate_selection.py#L119
[2] https://en.wikipedia.org/wiki/F1_score
[3] http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
[4] https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff