Machine Learning Algorithm on Enron Dataset

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"]

Enron was an American energy company which went bankrupt after scandal broke in 2001. In this project we will analyze enron dataset and apply machine learning algorithm to identify person who have involved in this scandal. The dataset has total of 146 data points and total of 20 features and label identifying person of interest (boolean value representing whether the person involved in the scandal or not). The Features with more than 100 missing values are director fees, loan advances and restricted stock deferred. In the dataset we have total of 18 person of interest. We will process these features using machine learning algorithm to identify the person of interest.

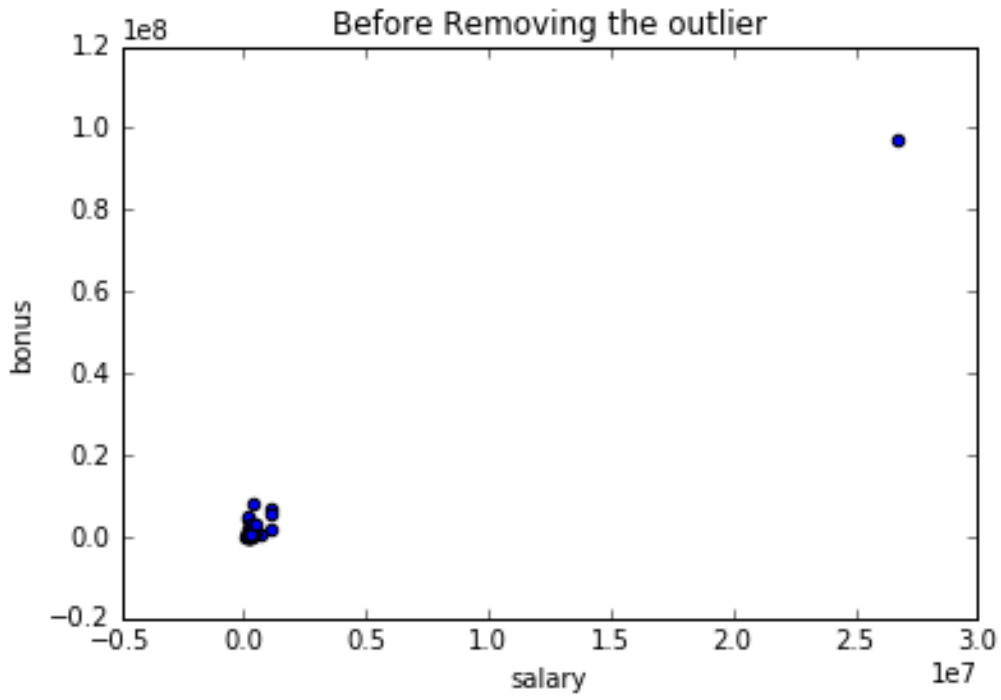

Fig1: Scatter Plot on Salary and Bonus with outlier

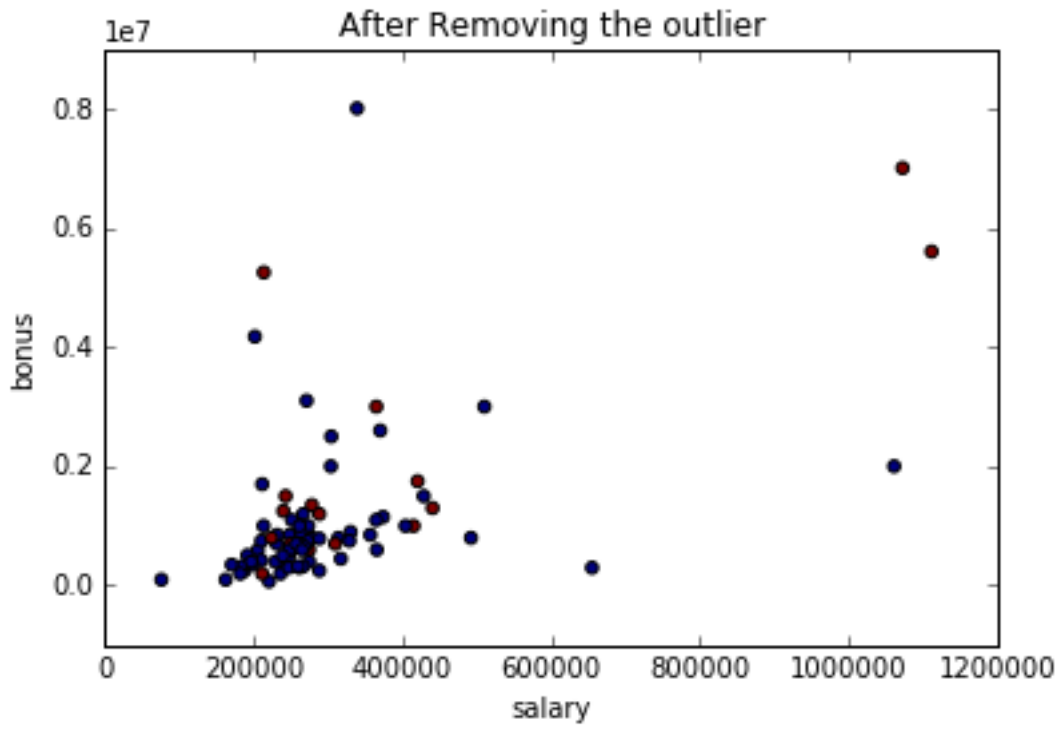

Fig 2: Scatter Plot on Salary and Bonus after removing Outlier

From the Fig 1 I can see that there is one value which is a clear outlier, on further investigation I found out that value is total and it is not a person. After removing the outlier I found other data points are in more reasonable range which is shown in Fig 2. The value is removed from the dataset and will not be part of further analysis and machine learning processing. During my analysis of deffer payment feature I found one value to be negative on further investigation I found out that value had lot of NAs and it is removed from the dataset.

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"]

Apart 20 features which are already available in the dataset I created two more features, they are named as *to_ratio* and *from_ratio*, the *to_ratio* is ratio between message from person of interest to this person and total messages received by this person. Similarly *from_ratio* is ratio between from this person to person of interest and total messages sent by this person. I am creating this ratio instead of taking absolute values of message sent or received from person of interest this is because there could be some individuals who could have send lot of message to person of interest but relative number of message they sent to person of interest could be quiet low. For instance, an admin person will be sending many generalized messages to all the staffs in the company so he will have very high number of messages sent to person of interest, but relative messages he sent to person of interest could be low. For this purpose I took ratio instead of absolute value of number of messages sent and received from person of interest.

There are some features with too many NA's those values are not taken into consideration for machine learning predictions. All the features with more than 50% values of NAs such as *deferral_payments, deferred_income, loan_advances, long_term_incentive and restricted_stock_deferred* are removed from the further analysis. One feature that warrant feature scaling is *deferred_income*, since all the values in this feature is negative, but this feature had lot of NAs and it is not taken into consideration. I used decision tree's feature importance to identify importance features. I used the features *salary, bonus, from_ratio, to_ratio, exercised_stock_options, total_stock_value, other, expenses, shared_receipt_with_poi, restricted_stock and total_payment* for feature selection.

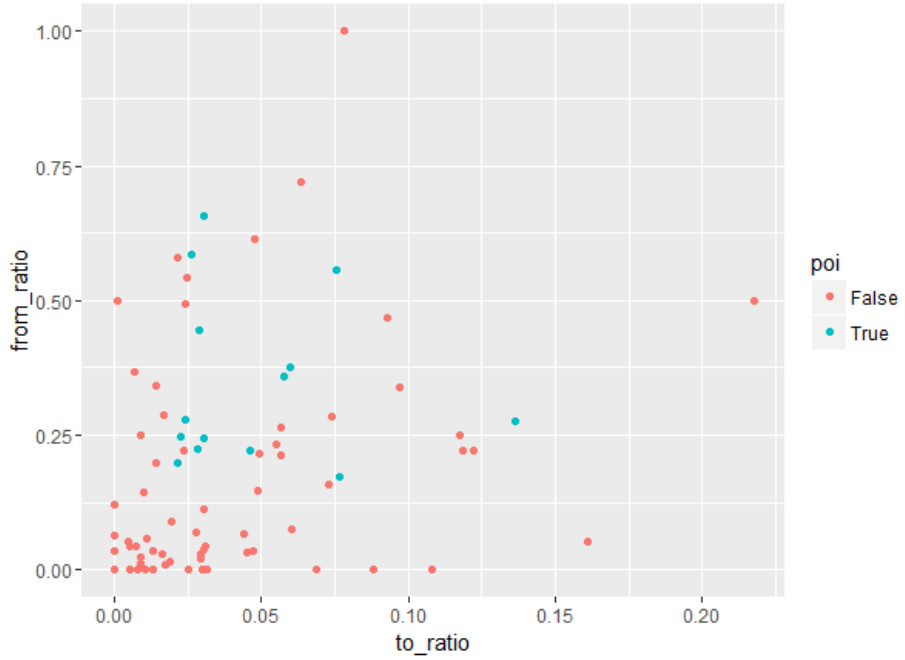

Fig 3: From message ratio and to message ratio for person of interest

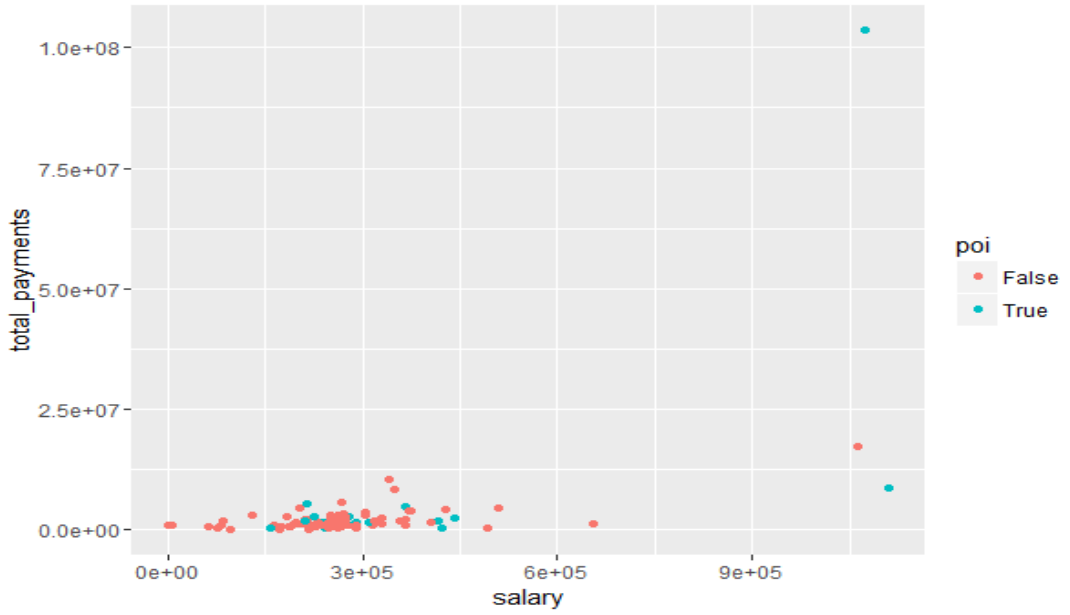

Fig 4: Total Payment and salary

I analyzed different features by looking into their graphs and checking their max, min and median values. All the extreme values for all these features looked real and are all used by the machine learning algorithm to find person of interest. For instance I looked into outlier of salary and bonus from Fig 2, like looking into salary greater than 1000000 and bonus greater than 5000000. Similarly extreme values of Fig 4 and those values looked acceptable as they maybe from people holding high position such as CTO or CEO and hence having high salary, bonus and total payment. Furthermore I found interesting patterns among features such as from Fig 3 I found that person of interest has all their *from_ratio* value value greater than 20 %.

I looked into feature importance of decision tree to get top features that can be used for our machine learning algorithm. The feature importance is tabled as follows

Features	**Importance**
to_ratio	0.0
salary	0.0
bonus	0.0
from_ratio	0.19
exercised_stock_options	0.0
total_stock_value	0.0
other	0.112
expenses	0.32
shared_receipt_with_poi	0.196
restricted_stock	0.0775
total_payments	0.10

Table 1: Feature Importance

From table 1 I can see 5 of the features has importance value of 0, only 6 features has non-zero values. At this stage I ran tester.py to get the other performance metric and I found accuracy to be 0.83, precision to be 0.35176 and recall value as 0.35000. I believe the performance metric can be improved by removing the features with value equal to 0.

Features	**Importance**
from_ratio	0.385
other	0.3169
expenses	0.09
shared_receipt_with_poi	0.208
restricted_stock	0.0
total_payments	0.0

Table 2: Feature Importance after removing some features

Same test is performed by removing all the features with performance metric equal to zero I got the results as shown in table 2, I found some features importance value increased while for restricted stock and total payment the value become zero. The value for accuracy is 0.84864, for precision is .469 and for recall is 0.45. Again the features with value zero are removed and I re-ran the test.

Features	**Importance**
from_ratio	0.347
other	0.245
expenses	0.078
shared_receipt_with_poi	0.33

Table 3: Feature Importance after removing restricted stock and total payment

After removing the features restricted stock and total payment, I got feature importance as shown in table 3. The accuracy value is 0.83, precision is 0.45 and recall is 0.48. All these four features are used by the machine learning algorithm to do the analysis.

The new feature I created *from_ratio* seems to influence the final algorithm, with *from_ratio* the accuracy, precision and recall score are 0.88, 0.62 and 0.52 respectively. When *from_ratio* is removed from the final analysis all performance metrics accuracy, precision and recall score goes down to 0.84, 0.48 and 0.25 respectively. This gives the importance of new feature on final algorithm analysis.

Note: Fig 3 and 4 are generated using R programming and their code is attached in Appendix A. I did this as exploratory data analysis is easier using R and to give some variety to this project.

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 used the four features selected above and deployed it with naïve bayes, I found the algorithm performed pretty badly. The accuracy is 0.80, precision is 0.003 and recall is 0.001. So I decided naïve bayes is not an algorithm to be used on this dataset. One reason for this might be

dataset is quiet small for naïve bayes. The accuracy of naïve bayes based on cross validation split is 0.79, cross validation uses 70% test data and 30% training data. It is important to split the data into training and test data. If we use all the data to train it will cause the model to overfit and it will not perform well with unknown data. So it is always important to split the data into training and test data, fit the model using training and test the performance of the model using test data. I changed the machine learning algorithm from naïve bayes to decision tree and I could see the improvement immediately, the accuracy is 0.83, the precision is 0.45 and recall is 0.48. So decision tree is used as final algorithm.

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"]

Parameter tuning is an important aspect of machine learning algorithm. If we don't tune the machine learning algorithm, then the algorithm may not perform as well as it could. For instance some values to the parameters might make machine learning algorithm to overfit (high variance) or underfit (high bias). The art of machine learning is to find right balance between high variance, high bias and make the model to perform to best of its abilities. I will discuss how I do parameter tuning for decision tree algorithm on this dataset.

The decision tree by default will have minimum sample leaf of 1. The minimum sample leaf represent number of samples leaf node of a decision tree has. This dataset is not large and hence I believe that it is causing model to overfit (high variance) for default value of 1. So when minimum sample leaf is increased to 2 the accuracy score increased to 0.86, similarly precision and recall score increased to 0.56 and 0.51 respectively. The accuracy based on cross validation split is 0.815. I increases the minimum sample leaf to 4 the accuracy score, precision remained the same but the recall value reduced to 0.46. The cross validation score is 0.789. I increased the minimum sample leaf to 8, at this value accuracy increased to 0.84, but the precision and recall both the metric reduced to 0.46 and 0.37 respectively. With higher minimum sample leaf the model starts to underfit (high bias). So the best value for minimum sample leaf is 2 for this dataset.

I changed minimum sample split to 4 from the default of 2, with minimum sample leaf to be 2. All the performance metrics such as precision, recall and accuracy reminded almost the same at 0.56, 0.51 and 0.86 respectively. When I further increased the minimum sample split to 8 precision and accuracy remained the same but the recall reduced to 0.47. I further increased the minimum sample split to 16, I found precision and accuracy to remain the same but recall increased to 0.55. But the accuracy based on cross validation split is 0.79, but it increases to 0.82 when I used 60% test data and 40% training data. So for future analysis I will keep validation split to be 60% for test data and 40% for training data so that it closer with accuracy score of tester.py. When

I increased the minimum sample split to 32, the accuracy, precision and recall score all went down to 0.83, 0.47 and 0.54 respectively. So I fixed the minimum sample split to 16.

For minimum sample leaf of 2, minimum sample split to 16 when I changed criterion from gini to entropy, the accuracy and precision increased to 0.88, 0.62 respectively, but recall reduced to 0.53. The cross validation split accuracy remains at 0.82.For minimum sample leaf of 2, minimum sample split to 16 and criterion to entropy, I changed splitter from best to random. The performance was quiet bad with accuracy, precision and recall score of 0.83, 0.38 and 0.14 respectively. Since the splitting of samples is done on random features rather than based on best possible features we can see performance degradation.

What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is a method of determining how well our predictive model perform on independent data. The machine learning algorithm is given set of known data (training data) on which algorithm train itself. Then the model is given set of unknown data (test data) to determine the performance of the model. In test data we will have the actual value for each data point, then we use predictive model to predict the values, then we compare the actual value and predicted values to understand the performance of our predictive model.

Main objective of the validation is to determine the performance of the predictive model on independent data i.e. on data the model has not seen before. The validation will help us to avoid the problem such as overfit, where the predictive model will perform exceptionally well on known data (training data), but poorly on other unknown data. For my analysis as stated in the previous answer I have divided the data set into 60% test data and 40% training data. Then I validated my model using accuracy score.

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 this dataset precision and recall are more important measure of performance than accuracy. As accuracy gives total correct overall predictions to the total values in the dataset. On the other hand, precision gives the measure of how many of our predictions correctly identified the person of interest to the total number of person of interest predicted and the recall is measure of correctly identified person of interest to total number of person of interest. For our dataset I would be more interested in increasing precision, since whosoever identified as person of interest is more likely to be person of interest and they can be scrutinized and identify other person of interest. But at the same time recall shouldn't be too low as we might be missing many persons of interest, which might hamper our investigation. So for decision tree at minimum sample leaf of 2, minimum sample split of 16 and criterion at entropy we have precision of 0.62 and recall of 0.53

at accuracy of 0.88 seems to be a good deal. For other scenarios like cancer treatment recall should be much higher as cancer should not be gone undetected.

Appendix A

R code for Figure 3 from_ratio and to_ratio

```
ggplot(aes(x = to_ratio,y = from_ratio),data =  enron)+
 geom_jitter(aes(color = poi))
```

R code for Figure 4 salary and total_payment

```
ggplot(aes(x = salary,y = total_payments),data = enron)+
 geom_jitter(aes(color = poi))
```

Summary Analysis of some of the features in R

```
summary(enron$expenses)
summary(enron$shared_receipt_with_poi)
summary(enron$total_payments)
summary(enron$salary)
summary(enron$deferred_income)
summary(enron$deferral_payments)
summary(enron$restricted_stock)
summary(enron$exercised_stock_options)
summary(enron$total_stock_value)
```