Intro to ML – Project document

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?

Goal:

The goal of this project is to identify poi's (Persons of interests) from the Enron financial dataset.

Poi's are individuals who were indicted, reached a settlement, or plea deal with the government, or testified in exchange for prosecution immunity.

Data Overview:

The dataset used in this project is from 146 Enron employees including their email and financial data.

About Enron:

Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives.

ML Role in achieving the goal:

This data will be used to build a POI identifier based on the financial and e-mail data made public as a result of the Enron scandal

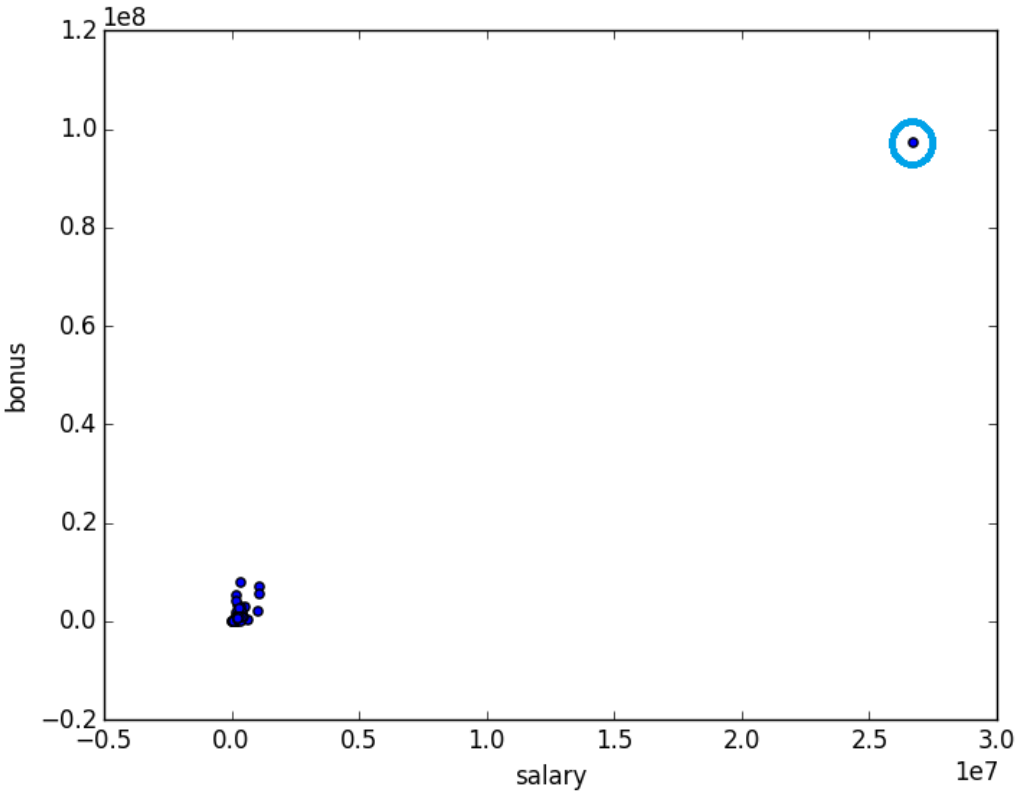
Data exploration and outliers:

First let's explore some basic characteristics about the data set:

- Number of data points (Enron employees) is 146
- There is a total of 18 POI in the data set
- Total number of features is 21, 14 financial features and 7 email features
- Each employee has a number of features, these include financial features like(all units are in US dollars):
    - 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
- Along with the financial features, there is some email features for each employee, these include (units are number of email messages except for 'Email address'):
    - To messages
    - Email address
    - From poi to this person
    - From messages
    - From this person to poi
    - poi – boolean, represented as an integer
    - shared receipt with poi

Outliers:

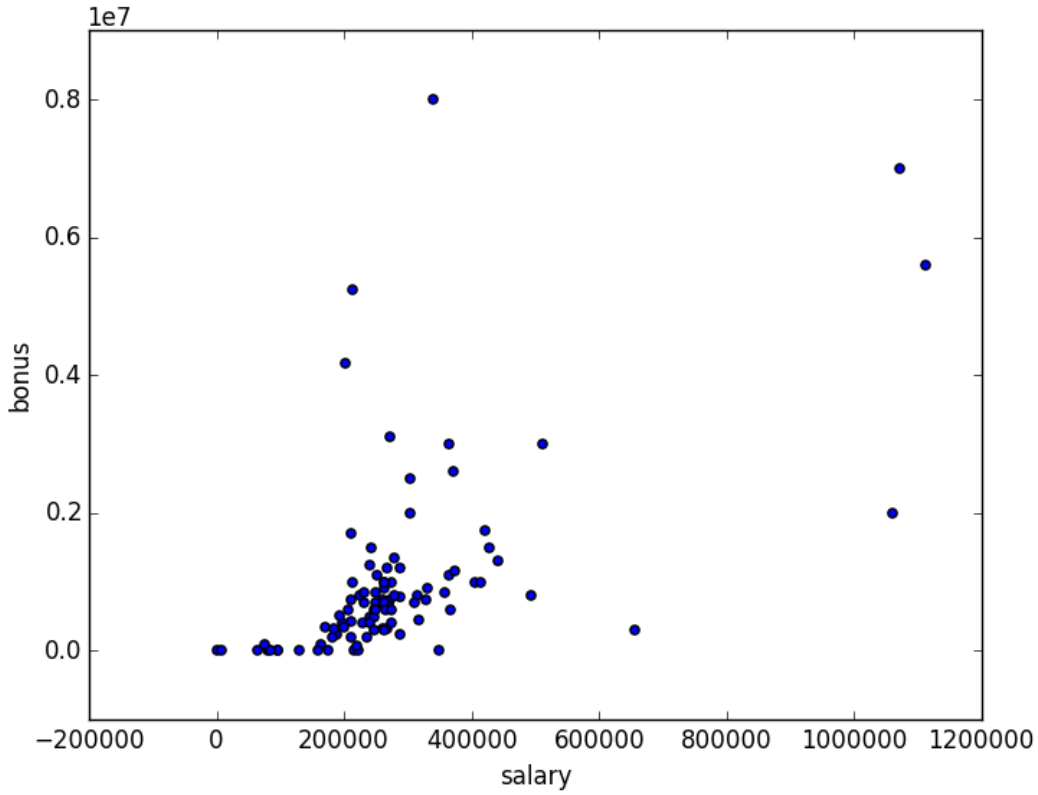
Data visualization is often the best way to detect outliers, here's a plot of salary vs. bonus



Clearly we can conclude that the circled point falls way far from the other data points which are clustered together at the bottom left corner.

Querying the data set, we find that this point corresponds to an employee named 'TOTAL', which certainly isn't a valid employee name. It is in fact a spread sheet error, where the total sum of the financial features got included as an employee name.

After removing that outlier, we re-plot the relation between salary and bonus.



Much better! Now we can see how our data points are scattered.

An important note to mention here that while there appear to be some outlier points even after we removed the biggest outlier ('TOTAL'), it's not correct to consider these points as outliers because they may be of interest (i.e they may be poi's)

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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use

Features:

I ended up using only 5 features in my poi identifier; those are:

1. exercised stock options
2. total stock value
3. bonus
4. salary
5. deferred income

Feature Selection:

Those features were selected using an automated feature selection process (SelectKbest), which selects the top k features based on a scoring criterion.

I tuned up the k parameter a bit; I tried higher numbers like 7 and found that it incremented the recall by a tiny amount but at the cost of decreasing the precision considerably. And I find it wiser in this application to value precision more that recall (check the Validation and evaluation section to find out why)

Scaling:

Scaling was not a concern, since the algorithm I ended up using doesn't care about scaling.

Engineering my own feature:

The feature I created was 'fraction_to_poi' which represents the fraction of messages sent from a person to poi. It is related to the email features. The rationale behind this feature is that if this fraction is high, which means a large portion of your messages is sent to a poi, then you yourself

are probably a poi also. It's very similar to the feature named 'from_this_person_to_poi' but it is scaled to be more consistent.

Testing my own feature:

In selecting the best k features out of all features, the newly engineered 'fraction_to_poi' feature got a very low score compared to other features

In testing, I added in my feature along with the previously selected features, and I found out that the precision went down to 0.15 (from 0.49, a huge amount!) and recall went up to 0.9 (from 0.38). A pretty high recall but a very low precision, so this feature will not be selected in the final prediction algorithm.

Important Note: Although this feature does not seem to produce better results, it might be good if we use it with a different algorithm, and as it turns out Decision trees make good use of this feature while Naïve Bayes (the one I used) doesn't.

3 – What algorithm did you end up using? What other one(s) did you try?

I have tried 2 algorithms, Naïve Bayes and Decision trees. Naïve Bayes got the best performance so I ended up using it.

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 don't 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 if you used, say, a decision tree classifier).

Trying out Decision Trees:

Using the same features that we mentioned above (those determined by SelectKbest), the Decision tree classifier gave 0.36 for precision and 0.31 for recall, so it passes the minimum requirement of 0.3.

Parameter Tuning:

A very important step to get the best of your algorithm is parameter tuning, which is basically playing around with different parameters that the algorithm takes as input until you get the desired performance.

The only parameter I tuned for this algorithm is the 'min_samples_split' which by increasing it too much, our classifier will be highly biased, and if we lower it too much our classifier will over fit to the data. So by manual tuning of the parameter, I found that a value of 11 is the best, higher and lower values will result in worse performance.

The interesting thing is when I added my feature ('fraction_to_poi'); I got a 0.37 for precision and 0.32 for recall. It's not a significant improvement, but it proves that different algorithms uses the same feature differently, Naïve Bayes was influenced very badly when using this feature, while decision trees was positively influenced (but not by much).

Final comparisons:

Metric	Naïve Bayes	Decision Trees
Precision	0.499	0.375
Recall	0.397	0.32

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

Validation:

Validation is a way of properly testing your algorithm; see how well it generalizes to other unseen inputs. This is generally done by splitting the data into training and testing sets, train on the training set, and test on the testing set.

Benefits of validation:

- Gives an estimate of performance on an independent (unseen) dataset
- Serves as a check for over fitting.

The classic mistake:

A classic mistake when doing validation is when the majority of outcomes of the training data are in one class and the majority of outcomes of the testing data are in a different class. In other words the algorithm is trained to see a certain outcome, but tested to produce a completely different one. This can be solved by randomizing or shuffling the training and testing data.

I used stratified shuffle split cross validation (already implemented in tester.py)

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

Final algorithm performance:

Metric	Naïve Bayes
Precision	0.499
Recall	0.397

Precision is ~ 0.5:

That means that if our algorithm flags some as poi, then there's 50% chance that he actually is poi.

Recall is ~ 0.4:

That means that if person is actually a poi, there's a 40% chance that our algorithm identifies/flags him as poi.

I think in this particular dataset and the application we are trying to achieve, which is identifying poi's, it's better that precision gets a higher value because if we flag someone as poi, then we need to be very confident that this flagged person is actually a poi especially if there's very few of them in the data set (only 18 poi of 146 Enron employees).