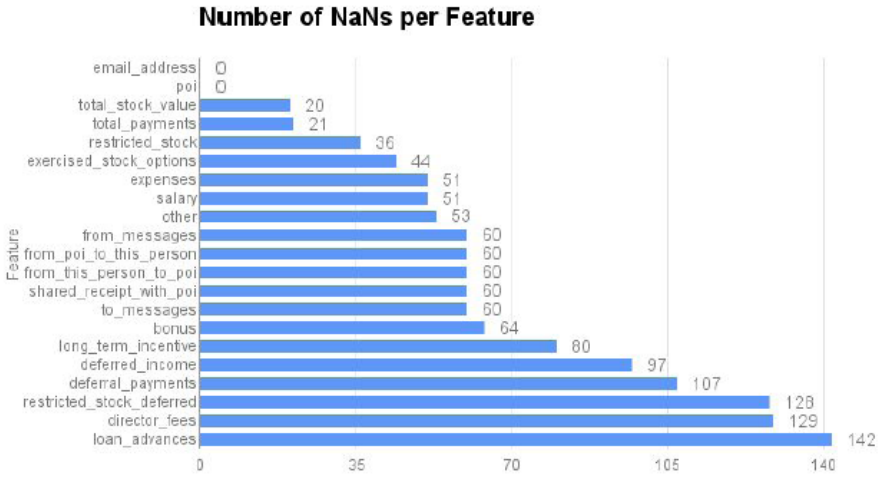
1. **Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]**

   The goal of this project is to examine financial and email data from enron employees and to try to identify employees who may have committed fraud. These employees are listed as persons of interest, or poi, and these poi's are formally defined as someone who was indicted for fraud, settled with the government, or testified in exchange for immunity. By using machine learning, the vast dataset of these employees can be utilized in order to identify these poi's. Machine learning classifiers can be trained on certain employees and then can be used to predict poi's among the rest of the dataset.

   In this dataset, there were 145 employees, each with 21 keys containing pertinent information as to that employee's salary, bonus, stock options. Email history, etc. Out of these 145 employees, 18 were POI's, which is interesting because it means that the majority of employees are not POI's and that classification of training and testing data, etc. will have to be done accordingly to ensure a fair weighting. To split the data, tools such as a stratified shuffle split can be used later on. Since there are relatively few values, when validating the algorithm later on, recall will be more important than precision, as recall indicates uncovering a higher percentage of the POI's.

   I saw outliers in the categories salary and bonus that were 'TOTAL', or could indicate total salaries and bonus instead of individually. Therefore, I manually removed these outliers using the .pop function of python dictionaries. I also saw other outliers such as NaN, meaning the particular information for that employee is not in the dataset. I dealt with these as I encountered them in creating my new variables. Below is a frequency graph showing the number of NaNs per feature. I also spotted another outlier. The employee named "LOCKHART EUGENE E" had all values as NaN so I removed all these from the dataset too.



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

I used the decision tree classifier to determine importance levels for the variables. I first looked at the email variables and noticed that some variables may be a little extraneous such as to/from email counts and to/from POI email counts. Therefore, I aimed to combine these variables into 2 variables: 'fraction_from_poi' and 'fraction_to_poi' which would be the ratio of emails for each employee to a poi or from a poi relative to the total number of emails received or sent. I discarded raw email addresses as these are not quantifiable and I did not think they were pertinent to the investigation. Therefore, I used the decision tree feature importance on the remaining 22 variables (20 existing, 2 new). Feature scaling is not necessary on decision trees. These are the results I got with the variables and the importance in parentheses:

to_messages (0.34407646792)
from_poi_to_this_person (0.133410045837)
from_messages (0.132276727205)
from_this_person_to_poi (0.118297318928)
shared_receipt_with_poi (0.111175038197)
salary (0.0930660725829)
deferral_payments (0.0609768907563)
total_payments (0.00672143857543)
loan_advances (0.0)
bonus (0.0)
restricted_stock_deferred (0.0)
deferred_income (0.0)
total_stock_value (0.0)
expenses (0.0)
exercised_stock_options (0.0)
other (0.0)
long_term_incentive (0.0)
restricted_stock (0.0)
director_fees (0.0)
fraction_from_poi (0.0)
fraction_to_poi (0.0)

From this, I decided to pick to_messages, from_this_person_to_poi, shared_receipt_with_poi, salary, deferral_payments, total_payments. However,

when I tried using these features, I noticed that the financial features were bringing the performance of the algorithm down, and that the email features by far had the most importance with the 3 financial features having minimal importance shown in the above values. I decided to test the precision and recall of combining email and financial features, and leaving them separate. These are the results, without parameter tuning.

| Features | Precision | Recall |
|---|---|---|
| email | .2 | .294 |
| Email+financial | .178 | .192 |
| financial | .161 | .152 |

From this, it is clear that the email features along provide the best performance, and that the financial features bring it down. I initially decided just to use the email features: to_messages, from_poi_to_this_person, from_messages, from_this_person_to_poi, shared_receipt_with_poi in my final analysis. However, I still was not getting satisfactory precision and recall values. It still struck me that using to_messages and from_poi_to_this_person and likewise for from_messages, seemed a little repetitive and that the new features I created would do a better job of capturing these variables. Therefore, I decided to not dismiss the new features I had created and substitute them for the 3 features. When I used a feature_list of fraction_from_poi and fraction_to_poi and shared_receipt_with_poi, the numbers were much higher. Without parameter tuning, I got precision value of .37 and recall value of .5, by far the highest out of the combinations I tested. Therefore, I decided to use this as my final features list.

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"]**

   I used a Gaussian Naïve Bayes classifier as well as a Decision Tree Classifier for preliminary analysis. The Decision Tree gave a higher accuracy, .88 repeating to .77 repeating, so I decided to select that classifier to tune and evaluate in my final analysis.

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"]**

   Parameter tuning means to look at the parameters that go into a classifier and to see which combination of or which value for certain parameters gives the best performance. If you don't tune parameters properly, algorithm performance can

be inefficient and will not focus on aspects of the data properly and give you the results you want. Parameter tuning is important to avoid overfitting and tune your algorithm in order to best solve the problem you have at hand. In order to tune the decision tree, I focused on the parameter min_samples_split and tuned it using Grid Search CV. This is an efficient way to tune parameters as grid search takes the possible parameters and optimizes them by cross validating across the parameter grid and seeing which combination works best, hence grid search. Min_samples_split refer to the minimum number of observations necessary to split a node and create 2 new ones in the decision tree. I set the min_samples_split parameter up to 100. Two is the default for a Decision Tree as you cannot consider splitting a node with less than 2 points so splitting with more points would be more accurate for the algorithm. I also tuned the criterion, entering gini or entropy into the grid_search CV. This measures the quality of the split, with gini representing Gini impurity and entropy representing information gain. I also tuned further by using the sklear.cross_validation stratified shuffle split function to modify the cross validation method in the grid search, as this is the method used in the tester. Also, since accuracy is not a good metric to measure the performance of this algorithm, I set the scoring method of the grid search equal to f1.

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"]**

   Validation is splitting a dataset into testing and training. The point of this is to evaluate your algorithm on data separate from which it was trained on. Validation is important in order to see how well your algorithm performs to a general independent dataset to the one you train. Classifiers are trained in a fixed data set and while performance within this is nice, there may also be over-fitting to that dataset which would mean that the classifier would not work as well on another generalized dataset. A classic mistake is simply looking at classifier accuracy to validate it. This is wrong, as accuracy may be artificially boosted by over-fitting and compromising the classifier just to be more accurate on the given data, which would not work as well on an independent dataset. It also ignores false positives and false negatives within the data, which is a critical part of any statistical analysis. To validate my algorithm I used the sklearn.cross_validation module. I used the train_test_split function which randomly splits the data into training and testing sets. I ran my algorithm in the tester program's test_classifier function to obtain precision and recall scores.

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"]**

   The performance of the algorithm in precision was around .534 and around .691 in recall. Precision is basically the likelihood that a person identified as a POI is actually a POI. According to the metric, 53% of my identified POI's are actually identified as POI's in the dataset as well. That means that 47% that my classified identified are false alarms and are not POI. Recall is basically how many of the POI's in the dataset were actually identified as POI's. A score of .69 means 69%

of the POI's in the dataset were identified, and 31% were not. This will obviously have to be improved on in order for the algorithm to reliably perform across other samples and datasets. One way to do this would be to actually go within the emails, as it seemed from this investigation that emails provided the most telling information, and look for text or writing patterns that could identify a POI.