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”]

Below is some introductory information on the dataset. There are 20 pieces of information on 146 employees. One is a feature called ‘poi’. This is the label in this project, our machine learning classifier will be trying to determine whether an employee’s ‘poi’ key is true or false. I tried to find if there were was any information to be gained by simply looking at where ‘NaN’ values showed up in the dataset. I looked at which keys had the least ‘NaN’ values, and if there was any correlations between the placement of ‘NaN’ values and whether someone was a ‘poi’. There was nothing that stood out to me.

**Number of employees:** 145

**POIs:** 18

**Number of not NA features**

{'to\_messages': 86, 'deferral\_payments': 39, 'expenses': 95, 'poi': 146, 'long\_term\_incentive': 66, 'email\_address': 111, 'deferred\_income': 49, 'restricted\_stock\_deferred': 18, 'shared\_receipt\_with\_poi': 86, 'loan\_advances': 4, 'from\_messages': 86, 'other': 93, 'deleted\_items': 146, 'director\_fees': 17, 'bonus': 82, 'total\_stock\_value': 126, 'from\_poi\_to\_this\_person': 86, 'from\_this\_person\_to\_poi': 86, 'restricted\_stock': 110, 'salary': 95, 'total\_payments': 125, 'exercised\_stock\_options': 102}

**Number of not NA features with POI=True**

{'salary': 17, 'to\_messages': 14, 'deleted\_items': 18, 'deferral\_payments': 5, 'total\_payments': 18, 'loan\_advances': 1, 'bonus': 16, 'restricted\_stock': 17, 'total\_stock\_value': 18, 'expenses': 18, 'from\_poi\_to\_this\_person': 14, 'exercised\_stock\_options': 12, 'from\_messages': 14, 'other': 18, 'from\_this\_person\_to\_poi': 14, 'poi': 18, 'deferred\_income': 11, 'shared\_receipt\_with\_poi': 14, 'email\_address': 18, 'long\_term\_incentive': 12}

**Percent features present with POI true**

{'salary': 0.17894736842105263, 'to\_messages': 0.16279069767441862, 'deleted\_items': 0.1232876712328767, 'deferral\_payments': 0.1282051282051282, 'total\_payments': 0.144, 'exercised\_stock\_options': 0.11764705882352941, 'bonus': 0.1951219512195122, 'restricted\_stock': 0.15454545454545454, 'total\_stock\_value': 0.14285714285714285, 'expenses': 0.18947368421052632, 'from\_poi\_to\_this\_person': 0.16279069767441862, 'loan\_advances': 0.25, 'from\_messages': 0.16279069767441862, 'other': 0.1935483870967742, 'from\_this\_person\_to\_poi': 0.16279069767441862, 'poi': 0.1232876712328767, 'long\_term\_incentive': 0.18181818181818182, 'shared\_receipt\_with\_poi': 0.16279069767441862, 'email\_address': 0.16216216216216217, 'deferred\_income': 0.22448979591836735}

The goal of the project is to experiment with machine learning methods in order to learn about how common machine learning algorithms work, and what can affect their performance, and to get a sense of what to expect when leaving the pattern recognition almost completely up to the program. Specifically, in this project we want to develop a classifier that is able to predict reasonably well whether an Enron employee is a person of interest from that person’s financial and email data based on the data of known people of interest.

Machine learning is a particularly nice solution to this problem for a few reasons. First, once a good machine learning algorithm is found, the classifier can update itself with new information, so the analysis does not need to be repeated whenever new information is gathered. Also, machine learning can pick up on complex patterns between many features without understanding the patterns. This is useful because even after reading up on the Enron scandal, I still did not really understand the situation enough to be able to make hypotheses that require more than minimal knowledge, so it was much easier for me to make the predictive model, all I had to do was try to maximize the performance of the algorithm by trying different things.

In terms of outliers I did not get rid of statistical outliers because I do not have reason to expect any sort of distribution, especially knowing how wide-ranging financial numbers can be. I did notice negative values, so I took the absolute value of all numbers in the dataset. There was also an employee ‘TOTAL’, so that entry was removed from the dataset.

1. 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”, “intelligently select features”, “properly scale features”]

selected\_features=['poi','expenses','from\_this\_person\_to\_poi','other']

clf.feature\_importances\_

array([ 0.43042383, 0.25687109, 0.31270509])

The process I used to select these features was by testing all the combinations of 3 features and choosing the combination that produced the best score. Doing this I am getting the best combination of at most 3 features since if there was a better combination of fewer features, the function would return a set of features with repeated features. The reason I used 3 features was a balance of validation, runtime, and performance. Since I am trying all combination of n features, the runtime increased dramatically with each added feature. I tried reducing the amount of testing I did on each feature combination but then my classifiers would not perform as expected in tester.py, and in the end, my classifier using 3 features performed quite well at least compared to the minimum requirements of this project.

No feature scaling was required since I used a decision tree classifier to do this problem which deal with one feature at a time, so the relative scales of the features do not matter.

For this project, I make a new feature called ‘deleted\_items’. It simply counts the number of email items in the deleted items directory associated with the employee’s email address. It was made by cycling through all the items in the ‘to’ and ‘from’ files of the email address associated with each employee and counted the times ‘deleted\_items’ appeared in an entry. I thought that this would help detect POIs because people who knew they were doing something illegal or immoral might not want to have incriminating information in their mailbox, whether the item was sent to them or they sent it.

This feature was not selected by my code, so I added it to the final set of features and tested with tester.py. The performance dropped to: Accuracy: 0.85169 Precision: 0.52284 Recall: 0.41200 F1: 0.46085 F2: 0.43024 with clf.feature\_importances\_ array([ 0.50136669, 0.1699064 , 0.32872692, 0. ]) where the last one refers to ‘deleted\_items’. Not great. I then removed all the features and just put in ‘deleted\_items’ and got: Accuracy: 0.82975 Precision: 0.26554 Recall: 0.20500 F1: 0.23138 F2: 0.21479.

1. 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 ended up using a decision tree classifier. I tried the decision tree classifier, the GaussianNB classifier, the KNeighborsClassifier, and different support vector machine classifiers. How I decided which algorithm to use is I used the default version of each and found the best score of all the combinations of three features for each algorithm, and the decision tree classifier came out on top with a score of 0.36.

**DecisionTreeClassifier**(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,

splitter='best')

features: expenses & from\_this\_person\_to\_poi & other

*Score: 0.356435643564*

**KNeighborsClassifier**(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=5, p=2,

weights='uniform')

features: to\_messages & deferral\_payments & bonus

*Score: 0.305636363636*

**GaussianNB**(priors=None)

features: deleted\_items & deleted\_items & deferral\_payments

*Score: 0.215111111111*

**LinearSVC**(C=1.0, class\_weight=None, dual=True, fit\_intercept=True,

intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,

multi\_class='ovr', penalty='l2', random\_state=None, tol=0.0001,

verbose=0)

features: salary & salary & salary

*Score: 0.172*

**SVC**(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

features: deferral\_payments & deferral\_payments & long\_term\_incentive

*Score: 0.0725925925926*

1. 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? What parameters did you tune? (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 items: “discuss parameter tuning”, “tune the algorithm”]

When we talk about machine learning algorithms, we are referring to a technique for code to model past behaviour. These techniques are not one size fits all though, as some techniques bring with it different choices and decisions within the overall algorithm. What algorithm, parameters combination will result in the best model depends on the nature of the data, and sometimes parameter choice can make a significant difference on the ability of the classifier to make predictions.

I tuned the parameters of my classifier simply by cycling through a set of parameter values and recording which combination did the best according to my scoring function. The combination of parameters that had the maximum score was chosen. The parameters below were selected algorithmically.

criterion=’entropy’,min\_samples\_split=5,max\_depth=None,min\_samples\_leaf=2, presort=True

1. 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 items: “discuss validation”, “validation strategy”]

Validation in machine learning is how you ensure your classifier can properly model a given situation. The most basic rule in validation is to train and test your classifier on completely different sets of data. This is because a classifier is meant to predict the labels of future data points given past features and labels. If you train and test on the same dataset, the best performance would be achieved by an algorithm that just recognizes past data points, and returns the given label, which is an extreme case of overfitting the classifier to your dataset.

To validate the classifier, I used a similar method to that in tester.py since that was what my algorithm would be graded with. I used stratifiedshufflesplit with 50 splits (to reduce compute times) and calculated the recall and precision from the combined true positives, true negatives, false positives, and false negatives. This validation technique is especially appropriate for this dataset because it imitates the unbalanced nature of the dataset (18 POIs in 145 employees) and allows you to get more predictions out of your classifier since 145 data points for training and testing does not allow for many predictions, thus making any performance score pretty random if only split once.

1. 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”]

I used one main evaluation metric to determine the performance of features, parameters etc. Previously in this project I have referred to this as my scoring function. My functional definition for performance in this project was recall multiplied by precision. Precision refers to the probability a positive prediction a classifier makes is true, and recall refers to how many of the positives in the dataset the classifier identifies. In other words, precision indicates how confident we are in a positive POI prediction by our classifier, and recall is how many POIs we should expect to be in our negative predictions. As you might be able to tell, it can be a balancing act to get a satisfactory precision and recall, as precision values a classifier that is conservative when predicting a positive, and recall values predicting as many of the true positives as possible. In order to get a classifier that had a good precision and recall while only looking at one number, I scored my classifiers by the product of their recall and precision, as a more balanced pair of numbers results in a higher product when the pairs add to approximately the same amount.

Another important metric is the f1 score. It is like just multiplying precision and recall, but it is then multiplied by 2 and divided by the sum of both, so it values balanced values more than the product of the recall and precision. For instance, let’s say an algorithm A has a precision of 1.0 and a recall of 0.26 and another, B, has precision 0.5 and recall 0.5. The product will score A 0.26 and B 0.25, thus evaluating A as better. The F1 score for A will be 0.206 and for B it will be 0.25.

The reason I used the product of both to judge the effectiveness of different classifiers that I was more concerned with better combined recall/precision, not necessarily a balance of both, but I did not want a completely awful score in one either, which ruled out using the sum of recall and precision as an evaluator.

Using tester.py, my product of recall and precision score 0.31307328 and my F1 score was 0.55319.

**Final Classifier:**

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=2, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=True, random\_state=None,

splitter='best')

Accuracy: 0.87050 Precision: 0.65088 Recall: 0.48100 F1: 0.55319 F2: 0.50749

Total predictions: 12000 True positives: 962 False positives: 516 False negatives: 1038 True negatives: 9484