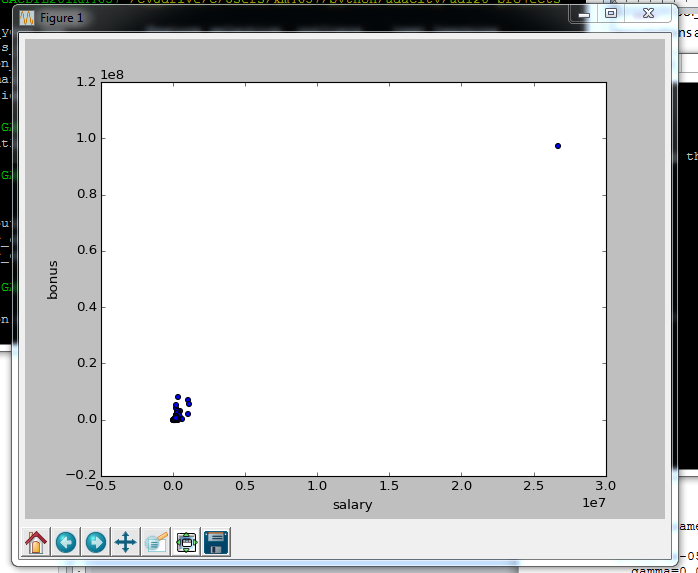
1. **PROJECT: GOAL AND BACKGROUND**

Using financial and email data of a number of Enron employees, we want to see if there is a way to predict which employees are “persons of interest”, or pois. To do this, we first look at the data to determine which features might be meaningful and design new features that might be helpful. Then we will look to see if there any outliers that need to be removed. Lastly, we pick and tune different algorithms in order to achieve a certain desired result, which for this project is a precision and recall score of greater than .3. We iterate and then evaluate these steps using cross validation to split the data into a training set and a testing set. The training set is used to “fit” the classifier, and the testing data is used to see how the classifier does predicting the test data. For this type of problem, the answer classes or labels are known (poi is 1 or 0) so using supervised learning algorithms will help to achieve our goal.

Using the script and plot from the class, we realized that the data contained a key “TOTAL” that was an outlier, seen in the plot below. This value was removed from the dataset. In the class, we looked at some of the other outliers but realized that this was valid data and not a spreadsheet calculation like this was. This is the only outlier I removed using “pop”.



1. **DATA/FEATURE SELECTION**

Our dataset contains 146 names, with 21 features per name. 18 of the names are pois. This shows that our dataset is somewhat unbalanced, as only 12% of the data is a poi.

There are 35 pois in poi\_names.txt. Since our dataset only has 18 pois, there are some pois that we don’t have data for. We can’t add these to our dataset using NaN for feature values since this could skew our results. As mentioned in class, no pois in the current dataset have ‘NaN’ for the ’total\_payments’ feature. If we add all these pois to the data using NaN for feature values (like ‘total payments’), it could be taken that having NaN as this value indicates a poi, which it does not. For this reason we are keeping the dataset as is. Also, the formatting code changes the NaN to a zero. Thinking this could skew our results, maybe we should try interpolating or taking averages?

**ADDING FEATURES**

The first features I added were those we had designed in class. The first, named ‘fraction\_from\_poi’ is defined as the ratio of emails from pois to all emails received by the employee. The other feature, ‘fraction\_to\_poi’ represents the ratio of emails sent to pois to all the emails sent by the employee. A high ratio in either of these features as compared to other employees could indicate an employee was a poi.

I also created a new feature called ‘fraction \_exercised’. This value is the ratio of the ‘exercised\_stock\_options’ to the ‘total\_stock\_value’. This feature stemmed from the idea that pois would be very likely to cash in their stock options as they would be aware of the severity of the problems at Enron before non pois.

**MANUAL FEATURE SELECTION**

I then started to manually try different combinations of features and scoring each combination using precision and recall from the provided test script. I also looked at the feature importance scores. I started with a big list of features that seemed meaningful, including the three new ones. Since I was using DecisionTree I also printed out the values of the feature importances.

['salary','total\_payments','from\_poi\_to\_this\_person','from\_this\_person\_to\_poi','exercised\_stock\_options','shared\_receipt\_with\_poi','fraction\_from\_poi','fraction\_to\_poi','fraction\_exercised']

Accuracy: 0.82980 Precision: 0.34681 Recall: 0.31300 F1: 0.32904 F2: 0.31922

Feature Importances

[ 0. 0.0754174 0.07279412 0.0940724 0.20681332 0.34554922

0. 0.20535355 0. ]

It seemed like straight ‘salary’ would not be as much of an indicator as the stock benefits or how many of those benefits got exercised before the company tanked. It had a value of 0 importance so removed it and both precision and recall went up.

Removed ‘salary’:

Accuracy: 0.83813 Precision: 0.37882 Recall: 0.33450 F1: 0.35528 F2: 0.34251

Feature Importances

[ 0.28642257 0. 0.18718487 0.07740155 0.24363745 0.

0.20535355 0. ]

Then I removed all the features with a 0 importance. Scores went up just slightly:

'total\_payments','from\_this\_person\_to\_poi','exercised\_stock\_options','shared\_receipt\_with\_poi','fraction\_to\_poi'

Feature Importances

[ 0.0754174 0.0940724 0.2380108 0.38714586 0.20535355]

Next I tried removing the first two features with the lowest importance:

'exercised\_stock\_options','shared\_receipt\_with\_poi','fraction\_to\_poi'

Accuracy: 0.82242 Precision: 0.45094 Recall: 0.30100 F1: 0.36102 F2: 0.32244

After more trial and error, the best scores for both precision and recall I could get were using these features:

'total\_payments','exercised\_stock\_options','shared\_receipt\_with\_poi','fraction\_exercised'

Feature Importances

[ 0.21962185 0.29012226 0.3435122 0.1467437

**Accuracy: 0.85607 Precision: 0.45975 Recall: 0.45400** F1: 0.45686 F2: 0.45514

To get a feel for the data, I then ran some quick stats to understand how many names had good feature data:

total\_payments=NaN (0% POI, 16.5% non-POI)

exercised\_stock\_options=NaN (33% POI, 30% NON-POI)

shared\_receipt\_with\_poi=NaN (22% POI, 43% NON-POI

**FEATURE SELECTION USING SCIKIT**

On these features, I then used both SelectKBest and VarianceThreshold from scikit. SelectKBest had the bigger impact on the scores as it selects features based on the scores.

**Accuracy: 0.85787 Precision: 0.46660 Recall: 0.46100 F1: 0.46378 F2: 0.46211**

1. **ALGORITHM**

I started with the Decision Tree algorithm on these features from above:

Accuracy: 0.85773 Precision: 0.46616 Recall: 0.46150 F1: 0.46382 F2: 0.46242

I then tried SVM using GridSearchCV to tune C and gamma. I also used cross validation to split up my data in order to look at the predicted labels. Using SVC, the classifier was predicting every name to be a non-poi, which explains a 0 prediction and recall. This is expanded in the next section.

Best parameters set found on train set:

SVC(C=1e-05, cache\_size=200, class\_weight=None, coef0=0.0, degree=3,

gamma=0.0001, kernel='rbf', max\_iter=-1, probability=False,

random\_state=None, shrinking=True, tol=0.001, verbose=False)

accuracy: 0.906976744186

precision: 0.0

recall: 0.0

Lastly, I tried RandomForestClassifier, setting class\_weights and using the default value. Precision scores went up but recall went down below .3.

Accuracy: 0.87220 Precision: 0.54179 Recall: 0.26900 F1: 0.35951 F2: 0.29912

After spending a lot of time trying to tune SVM, I ended up using DecisionTree as it gave the best results.

1. **TUNING**

Once you have selected an algorithm, tuning is the process of trying different parameter values and combinations to increase the performance of your algorithm based on the metrics you are trying to achieve. For this project, I was comparing both precision and recall. I first used the DecisionTree algorithm, with no tuning and getting pretty good results. Then I tried using SVM, with the GridSearchCV module to tune. GridSearchCV from scikit is an automated way to tune the parameters of your algorithm without having to manually try all the different combinations. It searches through the subset of parameters and values specified to achieve the best results using cross validation. It is very useful, but I noticed it can be quite slow.

First, I tuned only C and gamma. GridSearchCV picked low C, low gamma (C=1e-05, gamma=0.0001). I was getting good accuracy (.91) but zero precision and recall. I looked closer and realized that my classifier was not classifying any of the test names as pois. This may be expected when using SVM when only 12% of our data is a poi. I then added ‘class\_weights’ to be tuned to try and solve this problem of unbalanced data, but still showed same as above. I did notice that if I forced a ‘class\_weight’ of something greater than 1:6, the test\_labels would then be all pois (instead of all non-pois.)

For these reasons, I did not think this was the algorithm to use for my classifier. I then went back to Decision Tree, and started tuning it. I first tuned ‘min\_samples\_split’ as it was familiar from class. Having a low min\_samples split (default is 2) can cause overfitting, so tried a higher one (40), where I got much better precision but worse recall. Looks like in this case all the test labels were non pois. Then I used GridSearchSV to tune the min\_samples\_split parameter. I also tried setting the criterion to ‘entropy’ as discussed in class. The default value for both of these parameters produced results closer to the values I wanted.

**min\_samples\_split=20:**

Accuracy: 0.87060 Precision: 0.53846 Recall: 0.20650 F1: 0.29852 F2: 0.23554

**min\_samples\_split=2: (default)**

Accuracy: 0.85553 Precision: 0.45848 Recall: 0.46100 F1: 0.45974 F2: 0.46049

**criterion=’entropy’**

Accuracy: 0.84440 Precision: 0.41050 Recall: 0.38300 F1: 0.39628 F2: 0.38820

I also did some tuning with the class\_weights parameter when using RandomForestClassifier, but it did not change the results too much.

1. **VALIDATION**

To validate, I split the data into training and test sets using the cross\_validation module in scikit. Cross validation is a method to evaluate a model’s performance. The idea is to split the data into a training set and a test set, where the training data is used to build, or “fit” the model, and then the test set is used to evaluate the model .

Precision and recall are metrics used to evaluate the model. Precision is the ratio of true pois to all the names labeled as pois. Recall is the ratio of true pois to the total number of pois. Precision measures exactness or quality. In this case, high precision indicates that if the model identified someone as a poi, they were likely to be one. Recall measures completeness or quantity. A high recall in this case indicates that if the model predicted that someone was not a poi, they were likely to not be one. Accuracy is a ratio of the number of people that were identified correctly to the total number of people.

1. **EVALUATION**

After splitting the data using the cross\_validation module in scikit, I used the labels predicted by the model for the test set, to evaluate it based on the prediction and recall metrics explained above. My final algorithm (DecisionTree) has the accuracy, precision, and recall scores below. The accuracy is not that great, as if you just said everyone was not a poi, you would be 87.7 % accurate. But the precision and recall are pretty good.

Accuracy: 0.906976744186

Precision 0.5

Recall 0.25

When run through the tester.py script, using StratifiedShuffleSplit:

Accuracy: 0.86000 Precision: 0.47428 Recall: 0.46100 F1: 0.46755 F2: 0.46360