Project Overview:

Enron was one of the largest companies in 2000 with projects in various sectors. By 2002, Enron had collapsed due to corporate fraud. the resulting investigation led to a lot of private emails and financial data being made public. Using this data I will create a person of interest identifier to see if I can detect key individuals from email conversations.

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 the project is to identify persons of interest based on Enron’s email conversations. I will train the machine learning algorithm to detect suspicious activity through email conversation and use that to identify our key players within the context of the Enron scandal.

Background on the data set:

* total number of data points: 146
* total number of people in dataset: 146
* total number of poi: 18
* total number of not poi: 128
* There are 21 features for each person in the dataset, and 20 features are used

The number of NaN values for features in our dataset:

|  |  |
| --- | --- |
| Feature | #NAN |
| Salary | 51 |
| to\_messages | 60 |
| deferral\_payments | 107 |
| loan\_advances | 142 |
| bonus | 64 |
| email\_address | 35 |
| restricted\_stock\_deferred | 128 |
| total\_stock\_value | 20 |
| shared\_receipt\_with\_poi | 60 |
| long\_term\_incentive | 80 |
| exercised\_stock\_options | 44 |
| from\_messages | 60 |
| other | 53 |
| from\_poi\_to\_this\_person | 60 |
| from\_this\_person\_to\_poi | 60 |
| poi | 0 |
| deferred\_income | 97 |
| expenses | 51 |
| restricted\_stock | 36 |
| director\_fees | 129 |

After running the code to see who were the highest paid we found that “TOTAL” from the data set was also included. I then removed total from the dataset as it was an outlier. Also scanning the enron PDF I saw that there is a travel agency which is not a person, so I removed that as well as Eugene Lockhart who didn’t have any numerical data associated to him.

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”]

The number of emails to POI’s and the number of emails from POI’s were an interesting feature to investigate; We also talked about this during the lessons. We took the ratio of the messages to/from POI’s because this would give us how often a POI would be contacted. Our POI identifier works on the assumption that POIs would contact each other more and would have a higher ratio of messages to each other than they would if they just sent a message to a normal employee.

I created the features:

* ratio\_frompoi
* ratio\_topoi

I did do some scaling using selectKBest and used the K value of 10. When I did this I got an organized list of scores that pertained to each feature. I ordered these scores and got the following table

|  |  |
| --- | --- |
| Feature | Score |
| exercised\_stock\_options | 24.8150797332 |
| total\_stock\_value | 24.1828986786 |
| bonus | 20.7922520472 |
| salary | 18.2896840434 |
| ratio\_topoi | 16.409712548 |
| deferred\_income | 11.4584765793 |
| long\_term\_incentive | 9.92218601319 |
| restricted\_stock | 9.21281062198 |
| total\_payments | 8.77277773009 |
| shared\_receipt\_with\_poi | 8.58942073168 |
| loan\_advances | 7.18405565829 |
| expenses | 6.09417331064 |
| from\_poi\_to\_this\_person | 5.24344971337 |
| other | 4.187477507 |
| ratio\_frompoi | 3.12809174816 |
| from\_this\_person\_to\_poi | 2.38261210823 |
| director\_fees | 2.12632780201 |
| to\_messages | 1.64634112944 |
| deferral\_payments | 0.224611274736 |
| from\_messages | 0.169700947622 |
| restricted\_stock\_deferred | 0.0654996529099 |

We see that the score for the ratio\_topoi is an important feature even though we created it. It has the fifth highest score and is necessary among the features and is behind exercised\_stock\_options, total\_stock\_value, salary and bonus.

Yes I did do some scaling and I used the MinMaxScaler() function. The scaler makes it so that there is a proper range of data. It is important that we normalize this data over a range before using any of the machine learning algorithms.

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:

* Naïve bayes
* Decision tree
* SVC
* KNeighborsClassifier
* Logistic Regression
* Random forest

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classifier | Accuracy | Precision | Recall | Parameters |
| Naïve Bayes | .8139 | .88 | .92 | {'feature\_selection\_\_k': 3} |
| Decision Tree | .81395 | .89 | .89 | {'feature\_selection\_\_k': 1, 'clf\_\_n\_estimators': 5} |
| K Nearest Neighbor | .837 | .88 | .95 | {'feature\_selection\_\_k': 1, 'clf\_\_n\_neighbors': 3} |
| Random Forest | .8139 | .88 | .92 | {'feature\_selection\_\_k': 1, 'clf\_\_n\_estimators': 5} |
| SVC | .8604 | .88 | .97 | {'feature\_selection\_\_k': 1, 'clf\_\_C': 100} |

After several rounds of trial and error I decided to go ahead and implement a pipeline using a stratified shuffle split which is a better split for smaller datasets. I also used the pipeline function in conjunction with the GridSearchCV function in order to get the best parameters for each algorithm that was implemented. It seems as though SVC seemed to perform the best overall.

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”]

Tuning the parameters means we are adjusting the algorithm during the training to improve the fit. If it isn’t tuned well the algorithm learning process might suffer. If the parameters are tuned more, they will bias the data more. Over tuning could lead to overfitting.

I used train\_test\_split() on the data. I split the data into training and testing sets using the method. The data is split into 70% train and 30% test data.

I used GridSeachCV with pipeline to find the best parameters for all algorithms tested for the classifier.

I looked on the discussion forums for help with the adjustment to GridSearchCV & pipelines using stratified shuffle split (link in the references) and once implemented the classifiers accuracy precision and recall seemed to be maximized.

When I ran poi\_id.py I saw that the decision tree classifier had the best F1 score. Because of this clf’s final value was given DecisionTreeClssifier().

When it was run with tester.py it did a good job identifying our POIs. When run with tester it had an Accuracy of 0.83453, Precision of 0.36369 and Recall of 0.32150.

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 is where the trained model is evaluated against a test set. The goal of validation is to make sure that our classifier once trained is actually performing like we want it to. We split the data into training and testing subsets because we use the training subset to train our classifier and we use the test data to validate the classifier. If we do it wrong, we can over-fit the data. One example of doing this wrong is using the same data set for both training and testing.

This data set will perform well on the training set but will fail with the test set. One way to avoid this is if we split the data into testing and training sets. Overfitting could still occur if our training data set has a lot of noise in it.

I validated by using the train\_test\_split() on the data. I split the data into training and testing sets using the method. The data is split into 70% train and 30% test data. Using this we got a decent overall accuracy, precision and recall scores.

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”]

|  |  |
| --- | --- |
| Metric | Average scores |
| Precision | .827 |
| Accuracy | .882 |
| Recall | .93 |

The Accuracy metric shows how closely the algorithm performed given the actual value. Accuracy in this case would be how well the algorithm was able to identify a POI that actually was a POI.

The precision metric shows how well your algorithm can classify true positives from all true and false positives. Precision in this case shows how well your algorithm can classify POIs from the entire list of people that are POI.

The recall metric shows how well your algorithm can classify positives from true positives and false negatives. Recall in this case shows how well your algorithm can classify POIs from the entire list of emails we have.

References:

Scikit learn documentation: <http://scikit-learn.org/stable/documentation.html>

Stratified shuffle split: <https://discussions.udacity.com/t/stratified-shuffle-split/203750/12>

Classification report: <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html>