Udacity Data Analyst Nanodegree

Machine Learning Project

Name: Matthew Bonilla

Date: April 19, 2018

Free Response Questions

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?

- The goal of this project is to discover who were the persons of Interest in the Enron dataset. Machine learning is useful in this instance because we are able to create a model that can iterate over many variables to most accurately label someone as a Person of Interest(POI) or not. The dataset contains 146 points, 18 of which are POIs while 128 are non-POIs. For each datapoint, there exists 21 features (initially).
- However, not all are good values, meaning that some 'features' actually have no value or 'NaN' values.These will mess up with our model. To address this issue, we want to tackle outliers. The first obvious outlier was the 'Total' column which summed all the salaries and other financial information together. Another outlier was the 'TRAVELING AGENCY' which introduced unnecessary information. Other outliers included missing values for all the features initially included. It ranged from 21 missing values for the 'total_stock_value' to 142 missing values for 'loan_advances'. Since I did not use all the features, I did not need to handle all the NaNs. However for most NaNs for features that were used, they were simply turned to zeroes. This was done because our data set of 146 points is so small and removing points would drastically affect our results. Another way would have been to turn it to averages but setting POI results and NonPOI results to the average would also muddle the results. A finer way would have been to set the NaNs as averages of each POI or NonPOI. That way, POIs and NonPOIs do not have the save average. However, the simplest way was to set each to 0.

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.

- In the end of the process, I used 5 features, 1 of which was newly created. Since I was using a decision tree to classify, I did not need to feature scale since they are not affected by scales. I chose my features based on how many missing values they had. I started off with only 'total_stock_value' which of course was not enough. I then worked my way up the list, hand selecting features to add. A very noticeable group was 'to_messages', 'from_messages' and the like. It was interesting that they all had the same amount of points so I decided to work with

that. The values did not end up being what I wanted so I decided to work around them and create a ratio of their sent/received messages to/from POIs instead of taking a flat number. People could have sent a small amount but only been talking to POIs or sent a large number but also have a large amount of communications with other people.
- With just the 'total_stock_value', 'shared_receipt_with_poi', and my two ratios, I was not getting the precision and recall in tester.py so I wanted to select more features. I again looked for a group of features that could work well together. Similar to the email messages, 'salary' and 'expenses' both had 51 missing values so they could easily be worked with. Using these 6, I only sometimes got the target I wanted (since decision trees split in random places). I wanted to consistently hit above my target so I used the D.T. Classifier's attribute of feature_importances_ which showed me that one of my ratios was not performing as well as I wanted to, going as far as showing 0.0 for importance. When I removed it, my precision and recall was well above my target and my accuracy was hitting 84%. Albeit, this is lower than I would like for accuracy, I think it is pretty good for the amount of training data we had.
- Feature Importances: (one of the many results)

```
total_stock_value: 0.361970853574
person_to_poi_ratio: 0.232689761239
expenses: 0.168854961832
shared_receipt_with_poi: 0.133949173833
salary: 0.102535249521
```

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

- I ended up using the DecisionTreeClassifier to create a model. I started off with using GaussianNB and compared it with the DecisionTreeClassifier, SVC, and KMeans. Initially, I ended up choosing the SVC since it had several parameters I could tune and also had a high initial accuracy. However, I came to realize that even with a high accuracy, my precision and recall were not doing well. With a dataset heavily skewed towards one data point (specifically non POIs), it would classify them all as non POIs and get an accuracy of 90%+. I then reverted back to a decision tree classifier when I started to look for precision and recall in the initial models. Looking back, I could have applied feature scaling to the SVC to see if I could have improved it but the DecisionTreeClassifier got me what I needed.

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).

- Tuning parameters are an important part of machine learning as these, along with proper features, are keys to improving the algorithm. Tuning parameters alters the ways, for example, a classifier classifies each point. Tuning changes the thresholds for the model. If you did not tune and just stuck with the base algorithm, while you could get a decent score, it would not be the most optimized solution. There could also be a better turned algorithm than the one you selected.

For example, Algorithm A without tuning is better than Algorithm B without tuning. However, Algorithm B with tuning could be better than Algorithm A with turning.

- For the DecisionTreeClassifier, I stuck with tuning their minimum samples split. I stuck with the gini criterion since we want to minimize misclassifications. We did not want to limit max_depth since we want all all our leaves to be pure, especially with a small dataset. I also did not want to assign weights to the features since I was not sure which weighed more than the others. I also left random_state untouched since I felt that it would cause a bias if the starting split was the same. I would be tuning it for the training data, not for an overall or any data.

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

- Validation is when you assess whether your model was accurate or validate that it is producing the correct results. A simple validation we use with algorithms is splitting our sample data into training and testing data. We training on the training and run preliminary tests on the testing to see if our algorithms are under/over fitting and whether our parameters are tuned appropriately. A classic mistake is overfitting is when your model is fit perfectly to the training data, but when it comes to testing, it performs absolutely poorly. This is because it is not generalized for datasets that are different from the training set.
- The validation I used was a StratifiedKFold cross validation using a split set of 3. I iterated the algorithm over the 3 folds and get the precision and recall of all three, then did a quick average.

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.

- The metrics used to evaluate the final model were Precision and Recall. The precision of an algorithm is the amount of the correct positives it predicted over the amount of all positives it predicted. The recall of an algorithm is the amount of correct positives it predicted over the amount of actual positives in the set. The algorithm gets a precision around .40 and a recall around .35. A 40% precision means that when the algorithm predicts someone to be a POI, they are correct 40% of the time. A 35% recall means that they only capture 35% of the POIs overall.