Intro to Machine Learning Final Project

Enron POI Classifier Question Writeup

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 was to build a classification model that would identify persons of interest (POIs) in the Enron scandal based on the financial information and email corpus made available to the public following the resolution of the scandal. The financial information dataset contains information regarding salaries, bonuses, stock values, and more for a subset of 143 Enron employees. The idea is that a machine learning classification algorithm could be trained on this data and then used to determine which employees are likely to be POIs for investigation.

The dataset contained 145 total observations and 21 features for each. Of these 145, two were not actually employees, one entry was the total (which was a huge outlier) and another was a travel agency (both of which were dropped during the cleaning of the dataset). There were other outliers present in the dataset as well, but these were left in place since they seem to be valid and due to their potential for providing relevant information for classification. Of the remaining 143 entries, only 18 were actually POIs. Since there were so few POIs (only about 5.5% of the dataset) the classes were very unbalanced. This class imbalance was dealt with by doping the training set with duplicate POI observations via random upsampling.

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

The dataset contained a total of 21 features (although it is really only 20 since one is the label of POI or non-POI). Of these features, several contained a lot of missing values. Features with a high prevalence of missing values were dropped, as were features with missing values for the majority of the POI class. One of the features was email addresses and this was also dropped after it was used for getting to/from total messages and to/from/shared receipt with POI messages. These email count value features were also dropped in favor of the engineered percentage based email features of percent to/from/shared receipt with POI as the percent values are more balanced than the total values. After manual features selection and replacement of total values with percent values as described, there were 13 features remaining. The importance of these 13 remaining features was determined using the SelectKBest algorithm. Three of these 13 features had very low significance scores (<5) and were dropped leaving the 10 best features for use. The remaining 10 features included 2 of the email percentage engineered features.

The features were also scaled using a MinMaxScaler. This was not necessary for most of the algorithms that were tried, but was essential for a couple. Algorithms like KNeighbors and SVM (both of which were tested but eventually discarded) utilize Euclidean distance in their calculations and thus require scaled features so that the effects of varying magnitudes and units are eliminated. In this set of features, the magnitudes of the features varied greatly and the engineered email features were percentages rather than dollar amounts like the others so scaling evened these out.

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”]

After much testing of different algorithms, the final algorithm chosen for best performance was Random Forest. Decision Tree and Gradient Boost performed well but were slightly outperformed by the Random Forest model. Other algorithms tested were: Gaussian Naïve Bayes, AdaBoost, Logistic Regression, K Neighbors, and Support Vector Machines. The tree based methods all performed pretty similarly and all had F1 scores of 0.4 and above. Logistic Regression and K Neighbors also performed pretty well and had F1 scores of 0.4 and above. Gaussian Naïve Bayes was only able to achieve an F1 of 0.25, and SVM did not perform well at all. The tree methods all performed pretty well out of the box but were improved with tuning. Logistic Regression did not perform well out of the box but improved with tuning. K Neighbors was slightly improved by tuning. SVM could not be improved by tuning, and Gaussian Naïve Bayes does not really have room for tuning.

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? 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 (or more correctly, hyperparameters) of a machine learning algorithm means to adjust the hyperparameter inputs of the algorithm to fine tune how the model will work. This is very important because it can mean the difference between a poorly performing model or a very well performing model. Choosing the right algorithm is important, but it is equally important to tune it well. If the hyperparameters of a model are not tuned (or not tuned well) the model may perform very poorly and give very unreliable predictions. Tuning a model improperly, such as without using validation, can also be a problem as you could end up overfitting your model to the training data.

For my final Random Forest classifier, I tuned the minimum samples for splitting, the number of estimators used, and the random state for generating estimators. In order to streamline this process I used GridSearchCV by inputting a set of hyperparameters and lists of values to try for each and then having the algorithm make a grid of all the possible combinations of those hyperparameter values and test all of them using cross validation to determine the quality of fit obtained with each combination. I found that the optimal values for my hyperparameters were: random_state = 46, min_samples_split = 2, and n_estimators = 5. This process of tuning greatly improved my classifier's performance compared to the out of the box model increasing my accuracy score by 0.12, my precision by 0.47, my recall by 0.5, and the F1 score by 0.49.

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 items: "discuss validation", "validation strategy"]

Validation is the principle of testing the model to determine the performance. There are multiple correct ways to do this, but all are similar in that they involve training the model on one subset of the data and testing it on a different subset. This can be done by simply splitting the dataset into train and test sets, or by using a K fold strategy where the data is split into train and test sets k different times to cross validate. A classic mistake made is to use the same data to train and test the model which leads to overfitting of the model to the training data (which is also the testing data) thus creating a model that performs poorly when exposed to new data.

I employed a simple train test split strategy for my model validation. Before building any classifiers, I split the data into a training set consisting of 70% of the data and a testing set containing the other 30%. The models were all trained on the 70% of data in the training set so that after training the model could be tested on a completely new and different set of data to determine the performance on data it has not seen before.

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 evaluation metrics I used to gauge my model's performance were accuracy, precision, recall, and F1 score. Accuracy is simply a measure of the proportion of correct classifications made. Precision is basically a measure of how good your model is at not making false positives, or what proportion of positive classifications are correct. Recall is basically the opposite of precision, it is a measure of how good your model is at not making false negatives, or what proportion of negatives are correct. The F1 score is a combination of precision and recall.

When testing with my test set consisting of 30% of the data set aside from the beginning, my final classifier had an accuracy score of 0.977, meaning that it gave the correct classification for the test data almost 98% of the time. The precision score was 0.8, meaning that 80% of the positive classifications were true positives. The recall score was a perfect 1.0, meaning that 100% of the negative classifications were correct, or that none of the positives were misclassified as negatives. The F1 score was 0.89, the average of the precision and recall. When testing my classifier using tester.py, my accuracy was 0.93, precision was 0.88, recall was 0.97, the F1 score was 0.92, and the F2 score was 0.95. For this situation, I decided it would be more important to have a high recall score than a high precision since it would be better to falsely identify someone as a POI than to miss any POIs. The same would be true in a situation where you were using a classifier to predict if someone has a disease or not, it is better to get a false positive than to miss a case of the disease.