Whitney King
Machine Learning
7/12/2017

ENRON FRAUD PERSON OF INTEREST EMAIL ANALYSIS

(1) Introduction

The fall of Enron occurred in December of 2001, after fraud and corruption gave way to bankruptcy. This massive failure, and the disaster it left in its wake, has gone down as one of the largest and most notorious corporate collapses in history. When the information from Enron's corporate records was made available to the public during the investigation, so was one of the largest open sets of complete emails available. This analysis focuses on a preprocessed dataset derived from these emails (provided via Udacity resources), which is aimed at creating a classifier that can determine persons of interest (POI) connected to the fraud/corruption.

The dataset consists of 146 employee data points. When these were investigated for outliers, it was found that the **TOTAL** line was being treated as an employee instead of an aggregate, so this entry was popped. Additionally, one entry (**THE TRAVEL AGENCY IN THE PARK**) was not an employee, and was also popped from the list. Finally, an employee was found that contained only NaN values (**LOCKHART EUGENE E**), so he was also removed putting the total employee data points at 143. Out of these, 18 were pre-categorized as POI and 125 were not. To start off, each employee contains 12 features (either financial or email), and the POI label. Many features contained NaN or missing values (Appendix I), so these were handled in the Python script to ensure they didn't impact the outcome.

(2) Features

As the data is further cleaned and processed, two additional features were added, bringing the total features up to 14. *Initially the number of features was higher (20 total), however the list was shortened to include only the best features to improve run time.* The feature **to_poi_message_ratio** was added to measure the frequency someone sends emails to POIs, and **from_poi_message_ratio** was added to measure the frequency someone receives emails from POIs. These were interesting features to create and investigate, however from_poi_message_ratio didn't end up being important enough for the final feature selection.

To determine the best features, cross validation was performed using **SelectKBest** with the **f_classif** function. First, this was performed on the original 19 selected features ($k = 19$), then it was reran on the features to include the two newly generated features ($k = 21$), where k is the number of features. Scores were determined for each feature, for each of the chosen classifiers. Each possible value of k from 1 to 21 was tested, one for each feature. The classifiers were fit first with the original features, and then were fit with the newly generated features. **RandomForest** and **AdaBoost** both returned 3 best features and scores (Appendix II): **bonus** (18.29), **total_stock_value** (.22), and **exercised_stock_options** (8.77).

(3) Algorithms

The algorithms chosen to fit to this dataset were **RandomForest** and **AdaBoost**. The classifiers were fit with the data, then used to evaluate best features, accuracy, precision, and recall. The scores were plotted (Figure 1 and Figure 2) to visualize the output. Finally, averages of these scores were returned for an idea of which classifier fit the dataset best overall (Appendix III).

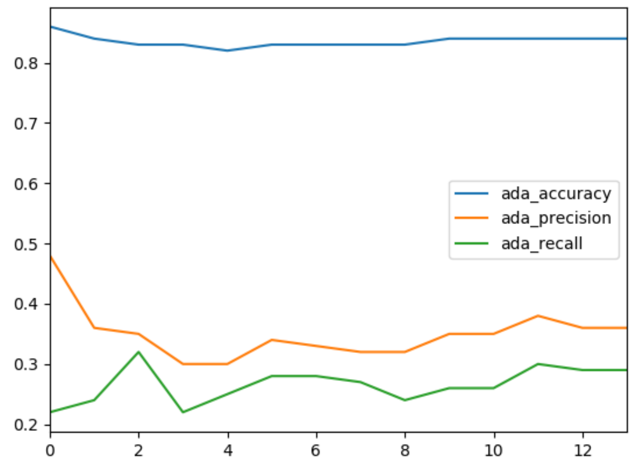

Figure 1 - AdaBoost

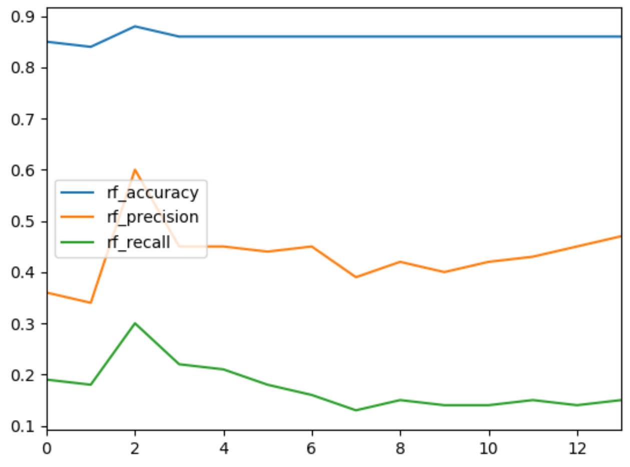

Figure 2 - RandomForest

Those calculated averages were then used to figure out which algorithm might be the better fit for the selected best features. While AdaBoost had a lower average accuracy and precision, the recall was much closer to the benchmark of .3 that needs to be attained. Both classifiers were tuned to figure out which one would perform the best.

(4) Tuning

Since including the newly generated features returned higher overall classifier scores for both AdaBoost and RandomForest, the new features were included during the tuning even though neither new feature ended up as a best feature. Tuning the parameters for each classifier is a means in which to ensure you're going to get the best performance and reliability from the decisions it's making on future sample data. Prior to being tuned, the average recall scores for AdaBoost and RandomForest (.27 and .16 respectively) were both lower than we needed (.3). **GridSearchCV** was used to permutate through various parameter values for each classifier to figure out what the best case for performance going forward is.

The parameters used in this tuning process were **n_estimators[70, 80, 90, 100]** and **learning_rate[.4, .6, 1]** for AdaBoost, and **n_estimators[80, 90, 100, 125], min_samples_split[2, 4, 6]**, and **max_features[1, 2, 3]** for

RandomForest. For AdaBoost, the best parameter fo*r n_estimators was 100* and *learning_rate was 1*. For RandomForest, the best parameter for n_estimators was 125, max_features was 3, and min_samples_split was 2. In general, higher values for n_estimators and max_features will lead to a more complicated classifier. Conversely, the higher the value for min_sample_splits is, the less complicated the classifier is.

(5) Validation

StratifiedShuffleSplit was used as a form of cross validation to understand more statistics about our best features with tuned classifiers. This method was chosen due to the relatively small nature of the dataset, as it's well suited for this type of scenario. What makes StratifiedShuffleSplit an ideal choice is the ability to iterate through all of the different possible sample sets of data, folding the various splits for training and testing data to find the best split to use. Ensuring that different splits of the data are stratified (which uses folds to reserve a percentage of sample data) makes sure we don't end up with a validation set of data that skews for or against a particular subset of data, and instead end up with the best representative split between testing and training data that will fit across the whole dataset.

Performing this evaluation gives us an idea of how we can expect the algorithm to do when fed future data, and is overall a better-quality measurement than just using sample data alone. Splitting the data into training and testing sets ensures that we won't fall into the trap of overfitting the data to the sample, which allows us to use it as a predictive model for future data. Reserving some data for testing allows us to verify how the model is likely to perform going forward. If validation isn't performed, it's easy to fall into scenarios where you won't be getting the best results you could be.

The counts being examined are true positives (correctly identified POIs), true negatives (correctly identified non-POIs), false positives (non-POIs identified as POI), and false negatives (POIs identified as non-POIs). The statistics being examined are accuracy, precision, and recall.

(6)Metrics

After the tuning and validations was completed, the output was investigated both without the two added features (Appendix IV) and with the two new features (Appendix V), and the most important statistics are compared below, showing tuning is significantly impacted by including the new features:

	k = 19			*k = 21*		
Classifier	**Accuracy**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
AdaBoost	*0.79938*	*0.30711*	*0.242*	*0.79962*	*0.33832*	*0.32705*
RandomForest	*0.84446*	*0.4902*	*0.275*	*0.84177*	*0.48248*	*0.3925*

To calculate precision (P), we can use the following equation employing true positive (TP) and false positive (FP) counts, which tells us the proportion of 'positive' decisions that were true positives out of a sum of true and false positives:

$$P = \frac{TP}{TP + FP}$$

To calculate the recall (R), we can use the following equation employing true positive (TP) and false negative (FN) counts, which tells us the proportion of true positives that were returned out of the sum of all actual positives (true positives and false negatives):

$$R = \frac{TP}{TP + FN}$$

Once both classifiers have been tuned, it's clear that RandomForest out performs AdaBoost when it comes to accuracy, precision, and recall. Without including the new features in the tuning, neither classifier reached a recall high enough to meet the minimum of .3, however including these made a big difference in the end results, and **RandomForest** is the clear best classifier to pick.

Conclusion

This was an interesting project that I didn't fully grasp the concept of until almost all the way through coding the required Python script. Having all of the POIs preidentified didn't make sense until I really dissected what the classifier tuning was doing with the different parameters and the shuffle split. Once I related this back to the data structure, it made sense why we'd want to know who was a POI and who wasn't going into training the algorithm, since we'd be concerned with classifying future or different sample data than what we'd use in this set down the line. I did struggle a lot with getting the recall for the classifiers over .3, but once I changed the seed and made some code updates, I was able to get it where it needed to be for both classifiers.

Overall, I feel I learned a lot, and I'm looking forward to finding way to employ the features of scikit-learn to other datasets in the future, since the statistical part of this whole course has been very engaging for me.

Appendix I: NaN Values

```
# NaNs in Each Feature:
      poi:  0
      salary:  51
      deferral_payments:  107
      total_payments:  21
      loan_advances:  142
      bonus:  64
      restricted_stock_deferred:  128
      deferred_income:  97
      total_stock_value:  20
      expenses:  51
      exercised_stock_options:  44
      long_term_incentive:  80
      restricted_stock:  36
      director_fees:  129
      to_messages:  60
      from_poi_to_this_person:  60
      from_messages:  60
      from_this_person_to_poi:  60
      shared_receipt_with_poi:  60
```

Appendix II: Best Features (with new Features)

```
SelectKBest Features - Ada: 3
bonus with a score of: 18.29
total_stock_value with a score of: 0.22
exercised_stock_options with a score of: 8.77
--------------------------------------------------------------------------------
RF Feature Importance:
[ 0.32070123  0.29495759  0.38434118]
Best Features - RF:
bonus with a score of: 18.29
total_stock_value with a score of: 0.22
exercised_stock_options with a score of: 8.77
```

Appendix III: Average Classifier Scores

		Adaboost			**RandomForest**		
k	**Feature**	**Accuracy**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
1	poi	*0.86*	*0.48*	*0.22*	*0.85*	*0.38*	*0.2*
2	salary	*0.84*	*0.36*	*0.24*	*0.84*	*0.33*	*0.18*
3	deferral_payments	*0.83*	*0.35*	*0.32*	*0.88*	*0.59*	*0.26*
4	total_payments	*0.83*	*0.3*	*0.22*	*0.86*	*0.48*	*0.25*
5	loan_advances	*0.82*	*0.3*	*0.25*	*0.86*	*0.46*	*0.21*
6	bonus	*0.83*	*0.34*	*0.28*	*0.86*	*0.46*	*0.16*
7	restricted_stock_deferred	*0.83*	*0.33*	*0.28*	*0.86*	*0.43*	*0.15*
8	deferred_income	*0.83*	*0.32*	*0.27*	*0.85*	*0.35*	*0.12*
9	total_stock_value	*0.83*	*0.32*	*0.24*	*0.86*	*0.42*	*0.16*
10	expenses	*0.84*	*0.35*	*0.26*	*0.86*	*0.45*	*0.16*
11	exercised_stock_options	*0.83*	*0.32*	*0.24*	*0.86*	*0.42*	*0.15*
12	long_term_incentive	*0.84*	*0.36*	*0.29*	*0.86*	*0.46*	*0.16*
13	restricted_stock	*0.84*	*0.35*	*0.26*	*0.86*	*0.42*	*0.14*
14	director_fees	*0.84*	*0.36*	*0.29*	*0.86*	*0.47*	*0.15*
15	to_messages	*0.83*	*0.35*	*0.27*	*0.86*	*0.41*	*0.12*
16	from_messages	*0.83*	*0.35*	*0.27*	*0.86*	*0.42*	*0.14*
17	from_this_person_to_poi	*0.83*	*0.34*	*0.27*	*0.86*	*0.45*	*0.15*
18	from_poi_to_this_person	*0.84*	*0.34*	*0.27*	*0.86*	*0.46*	*0.15*
19	shared_receipt_with_poi	*0.83*	*0.34*	*0.27*	*0.86*	*0.42*	*0.13*
20	to_poi_ratio	*0.83*	*0.34*	*0.27*	*0.86*	*0.43*	*0.14*
21	from_poi_ratio	*0.83*	*0.34*	*0.27*	*0.86*	*0.43*	*0.12*
	Average	***0.83***	***0.34***	***0.26***	***0.86***	***0.44***	***0.16***

Appendix IV: Tuned Scores without New Features

```
Beginning tuning AdaBoost...
Please wait...
Best Parameters:
{'learning_rate': 1, 'n_estimators': 100}
Time Spent Tuning AdaBoost: 1179.06s
Tuned AdaBoost Metrics:
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1,
          n_estimators=100, random_state=None)
        Accuracy: 0.79938       Precision: 0.30711      Recall: 0.24200
        F1: 0.27069     F2: 0.25272
        Total predictions: 13000
        True positives:  484
        False positives: 1092
        False negatives: 1516
        True negatives: 9908

--------------------------------------------------------------------------------
Beginning tuning RandomForest...
Please wait...
Best Parameters:
{'max_features': 3, 'min_samples_split': 4, 'n_estimators': 125}
Time Spent Tuning RF: 4169.0s
Tuned RandomForest Metrics:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=3, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=4, min_weight_fraction_leaf=0.0,
            n_estimators=125, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
        Accuracy: 0.84446       Precision: 0.49020      Recall: 0.27500
        F1: 0.35234     F2: 0.30147
        Total predictions: 13000
        True positives:  550
        False positives:  572
        False negatives: 1450
        True negatives: 10428
```

Appendix V: Tuned Scores with New Features

```
Beginning tuning AdaBoost...
Please wait...
Best Parameters:
{'learning_rate': 1, 'n_estimators': 100}
Time Spent Tuning AdaBoost: 1183.27s
Tuned AdaBoost Metrics:
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1,
          n_estimators=100, random_state=None)
        Accuracy: 0.79962       Precision: 0.33832      Recall: 0.31650
        F1: 0.32705     F2: 0.32064
        Total predictions: 13000
        True positives:  633
        False positives: 1238
        False negatives: 1367
        True negatives: 9762

--------------------------------------------------------------------------------
Beginning tuning RandomForest...
Please wait...
Best Parameters:
{'max_features': 3, 'min_samples_split': 2, 'n_estimators': 125}
Time Spent Tuning RF: 4999.96s
Tuned RandomForest Metrics:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=3, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=125, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
        Accuracy: 0.84177       Precision: 0.48248      Recall: 0.39250
        F1: 0.43286     F2: 0.40771
        Total predictions: 13000
        True positives:  785
        False positives:  842
        False negatives: 1215
        True negatives: 10158
```