Ryan Nelson
Udacity - Machine Learning Final Project
Github\Raginwombat

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”]

   My goal for the project was to end up with an automated classifier that took would build automated as much as possible the Classifier choosing and parameter tuning. The data use used is fairly broad and didn’t look like a ton of varied data, it also wasn’t strictly numeric. The majority of it was clustered in similar ways with ridiculous outliers. The first steps were to convert the relevant data to numerals (when applicable) to allow for easy analysis, Next the outliers had to be stripped out so the classifiers didn’t spend all of their time handling those exceptions.

   For entries that has NaN for values 0’s were written in it’s place. The following keys were removed completely for having no data or not being a person:
   TOTAL
   THE TRAVEL AGENCY IN THE PARK
   LOCKHART EUGENE E

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 importance’s 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”]

   Initially I started with a large features set and plotted them in Pyplot to get a feel for the shaped of the data. I chose the features by using judgment for areas where people could easily manipulate payments without raising too many eyebrows. I ended up with a features list of ('poi','salary', 'total_payments', 'loan_advances', 'bonus','restricted_stock_deferred', 'deferred_income', 'from_poi_to_this_person', 'exercised_stock_options', 'long_term_incentive', 'from_this_person_to_poi') based on that premise. I then ran the features list through my classifiers to determine their performance. Due to poor results I changed my approach to using SelectKBest to select the features with ANOVA F-score to correlate the features to my POI ID. I started with all of the features and got a max score close

to 1 for each. I then went back and reduced the feature set until I got a decent score without using all of the features on the list.

Using all of the features in my list yielded:
Accuracy: 0.93347 Precision: 0.92243 Recall: 0.54700 F1: 0.68675 F2: 0.59547
Total predictions: 15000 True positives: 1094 False positives: 92 False negatives: 906 True negatives: 12908

19 Features:
Accuracy: 0.93540 Precision: 0.93066 Recall: 0.55700 F1: 0.69690 F2: 0.60563
Total predictions: 15000 True positives: 1114 False positives: 83 False negatives: 886 True negatives: 12917

18 Features:
Accuracy: 0.94033 Precision: 0.93607 Recall: 0.59300 F1: 0.72605 F2: 0.63991
Total predictions: 15000 True positives: 1186 False positives: 81 False negatives: 814 True negatives: 12919

17 Features:
Accuracy: 0.77373 Precision: 0.20441 Recall: 0.24100 F1: 0.22120 F2: 0.23267
Total predictions: 15000 True positives: 482 False positives: 1876 False negatives: 1518 True negatives: 11124

My final Parameters were:

- scaler: StandardScaler(copy=True, with_mean=True, with_std=True),
- pca': PCA(copy=True, n_components=11, whiten=False),
- classifier: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=3, max_features='auto', max_leaf_nodes=None, n_jobs=4, oob_score=False, random_state=None, verbose=0, warm_start=False))

My final features list was :

- 'director_fees'
- 'from_messages'
- 'other'
- 'shared_receipt_with_poi'
- 'deferred_income'
- 'total_payments'
- 'to_messages'
- 'long_term_incentive'
- 'deferral_payments'
- 'exercised_stock_options'
- 'restricted_stock'
- 'salary'

- 'total_stock_value'
- 'loan_advances'
- 'from_this_person_to_poi'
- 'restricted_stock_deferred'
- 'bonus'
- 'poi'

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”]
   I initially tested the following classifiers with no parameters and ranked them based on performance.,
   - LinearSVC
   - LogisticRegression
   - Decision Tree
   - RandomForest
   - Niaeve Bayes
   - Adaboost
   - KNeighbors

   I then created a loop and a Pipeline to tune the parameters of the top 4 classifiers.

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”]

   I used a few mechanisms to choose and tune the algorithm. Firstly I ran the basic classifiers, Niaeve Bays, Decision Tree and SVM with no parameters on my training data then used the testing data to score them. I then took the top two performers and used Grid Search to tune the parameters. I then reran the comparison between these two classifiers and then reran the tuning for the Adaboost with the optimized Decision tree.

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”]

For validation I initially tried to use the accuracy scoring, however it isn't sufficient for the increasing for precision and recall required by the project. I then tried a variety of combinations such as precision for my initial classifier selection and recall for my parameter tuning.

I eventually settled on using Select K Best and my classifier selection with the precision recall scorer, and grid search with the recall scorer. The idea was since the sample set was so I trained on the whole sample set cross validated through Select K Best and the GridSerachCV's default cross-validation strategy.

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 classifier test initially reported the following metrics

Accuracy: 0.77320 Precision: 0.15468 Recall: 0.15700 F1: 0.15583 F2: 0.15653

Total predictions: 15000 True positives: 314 False positives: 1716 False negatives: 1686 True negatives: 11284

After optimizing and filtering through Adaboost the following metrics were observed.

Accuracy: 0.82033 Precision: 0.28563 Recall: 0.23150 F1: 0.25573 F2: 0.24062

Total predictions: 15000 True positives: 463 False positives: 1158 False negatives: 1537 True negatives: 11842

We can see the accuracy increases by 5% and precision by 13%. In real world the number of True positives increased while false positive and false negatives dropped. This makes the classifier more reliable and reflects the increase in accuracy.

Rubric requirements:

Understanding the Dataset and Question

total number of data points: 146 people with 21 features (not including names) which yields 3066 data points. Out of which 1708 data points are actual values and 1358 are NaN

allocation across classes (POI/non-POI): 18 POI identified in the dataset

number of features used: 18 features were used

are there features with many missing values?: Yes, many features include missing values, almost 1/5 of the data set is missing values.

Outlier Investigation (related lesson: "Outliers")

Outliers were identified initially by graphing some data and using a linear regression to cut off the top 10%. During the actual processing the values were sorted by size and the top 5% had a 0 written in to them. Dropping the actual data would prevent analysis of the features due to dataframe length mismatches, and setting the values to the mean would complicate the computation since the outliers skew the mean just as much as a 0 possibly could.

Optimize Feature Selection/Engineering

Create new features (related lesson: "Feature Selection")

Two features were created: 'from_poi_frac', 'to_poi_frac'. They are the percent of emails from and to an identified POI respectively.

Orignal list
Fit took: 0.122 s
Prediction took: 0.001 s
Scoring took: 0.001 s
(precision, recall, fbeta_score, support) : (0.13904338153503892, 0.17241379310344829, 0.15394088669950737, None)

Created list
Fit took: 0.114 s
Prediction took: 0.0 s
Scoring took: 0.001 s
(precision, recall, fbeta_score, support) : (0.11494252873563218, 0.13793103448275862, 0.12539184952978055, None)

It looks the created features actually reduce the precision and recall scores by .01-.05 points.

Intelligently select features (related lesson: "Feature Selection")

Feature selection was automated after investigating the performance of manual selection using EDA vs automated selection. Select K Best was utilized with an ANOVA Fscore to choose the features. The maximum number of features were used and incrementally reduced and iterated through until the final score met the threshold for his project.

Properly scale features (related lesson: "Feature Scaling")

In my pipeline the StandardScaler was utilized prior to passing it to PCA.

Pick and Tune an Algorithm

Pick an algorithm (related lessons: "Naive Bayes" through "Choose Your Own Algorithm")

Three initial classifier algorithms were chosen and their results printed to the console:

classifer LinearSVC
Fit took: 0.002 s
Prediction took: 0.0 s
Scoring took: 0.001 s
(precision, recall, fbeta_score, support) : (0.0, 0.0, 0.0, None)

classifer LogisticRegression
Fit took: 0.002 s
Prediction took: 0.0 s
Scoring took: 0.001 s
(precision, recall, fbeta_score, support) : (0.375, 0.42857142857142855, 0.39999999999999997, None)

classifer Decision Tree
Fit took: 0.001 s
Prediction took: 0.0 s
Scoring took: 0.0 s
(precision, recall, fbeta_score, support) : (1.0, 1.0, 1.0, None)

classifer RandomForest
Fit took: 0.014 s
Prediction took: 0.001 s
Scoring took: 0.0 s
(precision, recall, fbeta_score, support) : (1.0, 0.42857142857142855, 0.5999999999999998, None)

classifer Niaeve Bayes
Fit took: 0.001 s
Prediction took: 0.0 s
Scoring took: 0.0 s
(precision, recall, fbeta_score, support) : (0.14583333333333334, 1.0, 0.25454545454545457, None)

classifer Adaboost
Fit took: 0.002 s
Prediction took: 0.0 s
Scoring took: 0.0 s
(precision, recall, fbeta_score, support) : (1.0, 1.0, 1.0, None)

classifer KNeighbors
Fit took: 0.001 s
Prediction took: 0.001 s
Scoring took: 0.0 s
(precision, recall, fbeta_score, support) : (1.0, 0.14285714285714285, 0.25, None)

Discuss parameter tuning and its importance.

Parameter tuning was conducted using a pipeline which scaled the features, used PCA to reduce the feature dimensions and finally Grid Search The pipeline was created to provide the highest possible precision and recall by reducing the features as much as possible then testing a variety of classifiers to yield the best performing one. The importance is that each classifier is a tradeoff between overfitting and specificity. Tuning the parameters allows the balances to be adjusted to meet the project goals. Grid search automates this search by exhaustively testing using the defined metric to rank the parameter combinations and return the highest performing one.

Tune the algorithm (related lesson: "Validation")

Grid search is used in rounds to discover the optimized parameters for the classifiers.

Validate and Evaluate

Usage of Evaluation Metrics (related lesson: "Evaluation Metrics")

Select K best used the ANOVA F Score for feature selection. The initial classifier selection was done using the score function which uses binary precision_recall for scoring. The GridSearch was completed using recall as the ranking metric. Classifier ranking was done using the precision and recall score

Discuss validation and its importance.

Validation Strategy (related lesson "Validation")

Implemented in code. Intermediate results for classifier selection was done using the recall and precision for the binary Precision Recall scorer. This was used to measure the performance of the GridSearchCV cross validation approach. The final validation was done by passing the data was split using train_test_split and passed the resulting data to tester.py which computes Accuracy, Precision, Recall, F1 and F2

Algorithm Performance

The algorithm has the following performance:

Accuracy: 0.92873 Precision: 0.91304 Recall: 0.51450 F1: 0.65814 F2: 0.56371

Total predictions: 15000 True positives: 1029 False positives: 98 False negatives: 971 True negatives: 12902