David Smith
Udacity Data Analyst Nanodegree
Project 5

Enron Submission Free-Response Questions

A critical part of machine learning is making sense of your analysis process and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be about 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your evaluator looks at your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: Link to the rubric Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that part of the rubric. If your response does not meet expectations for all rubric points, you will be asked to revise and resubmit your project. Make sure that your responses are detailed enough that the evaluator will be able to understand the steps you took and your thought processes as you went through the data analysis.

Once you've submitted your responses, your coach will take a look and may ask a few more focused follow-up questions on one or more of your answers.

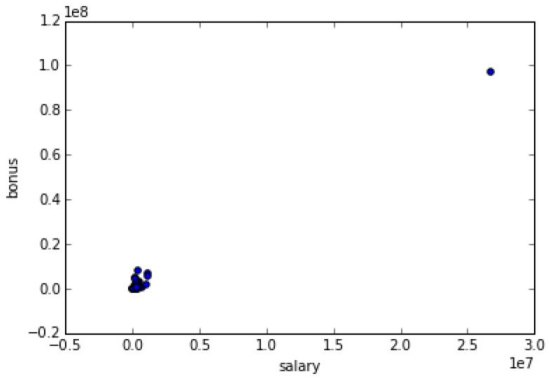
1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]

The Enron email corpus is a compilation of emails sent to and from important Enron employees during the period during which major financial fraud was being committed. By evaluating data from the Enron email corpus and public financial reports using machine learning techniques, we are trying to determine who within the Enron organization should be considered a "person of interest?" The dataset contains information about many potential POIs' financial interest in Enron as well as their email activity to and from other Enron employees, including potential POIs.

The dataset has the following characteristics:

- 146 records

- I removed one of the records because it was an extreme outlier. Somehow, the "Total" sum of all records was included in the dataset. A plot of bonus vs. salary shows this datapoint clearly (below).



- There is another record pertaining to 'LOCKHART EUGENE E' that has no numbers for financial data. I didn't need to write any code to address this issue, because featureFormat from tester.py takes care of that case. The removal of both of those records, the record count of my dataset is 144.

- 18 of the 144 records, 12.5%, are POIs
- Each record consists of a 20 features, not including POI or the features that I later generated on my own (see Question 2).
- Over the 144 records, some of the features contained more data than others. The table below outlines the percentage of each feature that is not "NaN" (not a number), meaning it contains actual data:

Feature	Pct w/ Data
to_messages	59%
deferral_payments	26%
expenses	65%
deferred_income	33%
email_address	76%
from_poi_to_this_person	59%
restricted_stock_deferred	12%
shared_receipt_with_poi	59%
loan_advances	2%
from_messages	59%

other	63%
director_fees	11%
bonus	56%
total_stock_value	86%
from_this_person_to_poi	59%
long_term_incentive	45%
restricted_stock	75%
salary	65%
total_payments	85%
exercised_stock_options	70%

The machine learning techniques involve picking an appropriate set of features, scaling those features, potentially reducing dimensionality, applying a classification algorithm, and evaluating the results on multiple dimensions.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them?

The features (with SelectKBest feature scores) I ended up using are the following:

- salary (19.025)
- bonus (30.076)
- total_stock_value (15.964)
- exercised_stock_options (15.823)
- long_term_incentive (11.365)
- shared_receipt_with_poi (10.777)
- fraction_to_poi (15.716)

I selected those features using SelectKBest as the second step in my pipeline, after having scaled the parameters with MinMaxScaler (see answer to next question below).

A very important parameter of SelectKBest in this step is “k”, the number of parameters that the algorithm is allowed to select. I started my evaluation with the following code in my “params” dictionary of pipeline parameters:

```
'SKB__k' : [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
```

I then iterated through each of the algorithms I was considering using GridSearchCV, to see both which parameters were selected (with scores) by `best_estimator_`, and what kind of F1, precision, and recall scores would come back from test_classifier. For each algorithm, the results were slightly different, as shown below:

```
NOW RUNNING NAIVE_BAYES
2 best parameters with scores:
salary 19.025
bonus 30.076

2 best parameters with scores:
salary 19.025
bonus 30.076

NOW RUNNING STANDARD_DECISION_TREE
3 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964

NOW RUNNING K_NEAREST_NEIGHBORS
2 best parameters with scores:
salary 19.025
bonus 30.076

NOW RUNNING ADABOOST
3 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964

NOW RUNNING RANDOM_FOREST
8 best parameters with scores:
salary 19.025
bonus 30.076
deferred_income 10.302
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
shared_receipt_with_poi 10.777
fraction_to_poi 15.716

NOW RUNNING LDA
5 best parameters with scores:
```

```
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
fraction_to_poi 15.716
```

The numbers returned by `test_classifier` were not high enough to pass the thresholds required for this project.

```
ALGORITHM              ACCU PREC RECA F1
Naive_Bayes            0.83 0.27 0.18 0.21
SVC                    0.87 0.55 0.12 0.20
Standard_Decision_Tree 0.84 0.18 0.06 0.09
K_Nearest_Neighbors    0.87 0.57 0.05 0.09
Adaboost               0.84 0.33 0.19 0.24
Random_Forest          0.85 0.41 0.21 0.28
LDA                    0.86 0.42 0.21 0.28
```

Since some of the algorithms were returning just two or three parameters in their `best_estimator_`, I decided to see what would happen when I removed that possibility from the list of possible values for k:

```
                'SKB__k' : [4, 5, 6, 7, 8, 9, 10, 11, 12],
```

The `test_classifier` results are as follows:

```
ALGORITHM              ACCU PREC RECA F1
Naive_Bayes            0.84 0.37 0.28 0.32
SVC                    0.86 0.40 0.09 0.14
Standard_Decision_Tree 0.84 0.15 0.05 0.08
K_Nearest_Neighbors    0.86 0.05 0.00 0.00
Adaboost               0.88 0.70 0.12 0.21
Random_Forest          0.86 0.45 0.19 0.26
LDA                    0.86 0.42 0.21 0.28
```

So the results were better, but still not where I wanted them to be. So I iterated one more time after removing four parameters as a possibility:

```
                'SKB__k' : [5, 6, 7, 8, 9, 10, 11, 12],
```

This time the results were satisfactory, as shown below. Essentially, raising the limit of the number of parameters from 2 to 5 improved the recall of each affected model.

```
NOW RUNNING NAIVE_BAYES
7 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
```

```
exercised_stock_options 15.823
long_term_incentive 11.365
shared_receipt_with_poi 10.777
fraction_to_poi 15.716

NOW RUNNING SVC
7 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
shared_receipt_with_poi 10.777
fraction_to_poi 15.716

NOW RUNNING STANDARD_DECISION_TREE
7 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
shared_receipt_with_poi 10.777
fraction_to_poi 15.716

NOW RUNNING K_NEAREST_NEIGHBORS
6 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
fraction_to_poi 15.716

NOW RUNNING ADABOOST
6 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
fraction_to_poi 15.716
```

```
NOW RUNNING RANDOM_FOREST
9 best parameters with scores:
salary 19.025
bonus 30.076
deferred_income 10.302
total_stock_value 15.964
exercised_stock_options 15.823
long_term_incentive 11.365
restricted_stock 9.364
shared_receipt_with_poi 10.777
fraction_to_poi 15.716

NOW RUNNING LDA
5 best parameters with scores:
salary 19.025
bonus 30.076
total_stock_value 15.964
exercised_stock_options 15.823
fraction_to_poi 15.716
```

And finally, I had an algorithm that met the precision and recall requirements: Naive Bayes.

ALGORITHM	ACCU	PREC	RECA	F1
Naive_Bayes	**0.84**	**0.38**	**0.31**	**0.34**
SVC	0.86	0.41	0.08	0.13
Standard_Decision_Tree	0.84	0.12	0.04	0.06
K_Nearest_Neighbors	0.86	0.05	0.00	0.00
Adaboost	0.88	0.71	0.12	0.21
Random_Forest	0.86	0.42	0.21	0.28
LDA	0.86	0.42	0.21	0.28

After doing this, I tuned configuration parameters for the algorithms themselves (not applicable to Naive Bayes) to try to boost scores further, and the numbers above represent the best scores that my analysis achieved. (See Question 4 for more discussion.)

Did you have to do any scaling? Why or why not?

Yes, as the first step of the pipeline I scaled the parameters using MinMaxScaler. Many of the parameters were large sums of money, which would have overwhelmed the scale of the other parameters that dealt with email counts.

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.)

I made three of my own features:

- *fraction_from_poi* - The percentage of all emails to a person that came from a POI. The rationale is to see if people who receive a high percentage of their emails from POIs are in fact themselves POIs. With a score of 1.433, this feature was not used in the final classification algorithm.
- *fraction_to_poi* - The percentage of all emails that a person sent that were sent to a POI. I added this to see if a person who sends a high percentage of their emails to POIs is himself or herself a POI. In fact, with a score of 15.716, this was one of the features that I used in my final classification algorithm.
- *fraction_salary_total_payments* - The percentage of a person's take-home pay that was fixed as opposed to a bonus or stock grant. My guess was that the more variable a person's compensation was, the more likely it would be that they would commit fraud. With a score of 3.084, this feature was not used in the final classification algorithm.

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 (or SelectPercentile), please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

See above

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 ended up using the Naive Bayes algorithm. I also tried Support Vector Classification (SVC), Standard Decision Tree, K Nearest Neighbors, Adaboost, Random Forest, and Linear Discriminant Analysis (LDA).

In evaluating the performance of each algorithm when running test_classifier, I looked at Accuracy, Precision, and Recall. Since Accuracy was virtually identical for all of the algorithms, I chose Naive Bayes because it had the best combination of Precision and Recall.

ALGORITHM	ACCURACY	PRECISION	RECALL
Naive Bayes	**0.84**	**0.38**	**0.31**
SVC	0.86	0.41	0.08
Standard Decision Tree	0.84	0.12	0.04

K Nearest Neighbors	0.86	0.05	0.00
Adaboost	0.86	0.71	0.12
Random Forest	0.86	0.43	0.20
LDA	0.86	0.42	0.21

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? (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 item: "tune the algorithm"]

Tuning an algorithm means finding the optimal combination of configuration parameters that yields the highest "score"--however you define that term. In this case, we are looking for the highest weighted F1 score, which combines precision and recall into one value that an algorithm can maximize. If you don't do this well, the classifier will not be optimized to be as predictive as it could be.

I used GridSearchCV, an automated way of running multiple iterations of the same algorithm using different parameter combinations in search of the one that yields the highest score. As mentioned above, in this case, the high score is based on the "F1_Weighted," During the GridSearchCV step, I used Stratified Shuffle Split on the training labels in an effort to randomize the selection of testing data due to the small sample size.

I ultimately selected Naive Bayes, an algorithm that doesn't have any configuration parameters, because it gives me the best combination of precision and recall for this data set. However, when I tested other algorithms, like Adaboost, for example, I tried to tune them as best I could to see if the F1 score could exceed that of Naive Bayes. For Adaboost in particular, I tried an array of values for both the n_estimators and learning_rate parameters. Since there is a trade-off between those two parameters, I wanted to see which combination worked the best. I ended up with a combination with a very high precision (0.71), but lower recall (0.12).

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 item: "validation strategy"]

Validation involves separating data into training and testing data sets, using the training set to train the algorithm, and then validating that the algorithm works against another set of data, the testing set. The classic mistake with doing validation wrong is accepting hypotheses suggested by a given dataset even when they are not true. By testing the algorithm on data outside the set used to train the it, we gain more confidence that the results of the algorithm are repeatable.

To validate the performance of each algorithm, I used the `test_classifier` function in tester.py with the winning classifier from the `best_estimator_` result (see feature selection answers to Question 2 above). The classifier ran against randomly selected splits of training and test data drawn from the entire data set--created through the use of Stratified Shuffle Split as the cross-validation data-splitting method. The `test_classifier` function uses Stratified Shuffle Split because the number of observations in our dataset is relatively small, and the non-POIs outnumber the POIs 7-to-1. Going with Stratified K Folds would not have given us the ability to run a large number of randomized data-splitting scenarios (folds) that give us the best chance to analyze such a small dataset. Thus, adding in the "Shuffle" component to randomize it has the desired effect. In addition to using the entire dataset, using 1,000 folds and 42 as the random state make it significantly different from the Stratified Shuffle Split configuration that was part of the GridSearchCV feature selection process. Therefore, there is little risk that the training and testing sets are too similar.

Once it subdivides the dataset into training and testing sets, `test_classifier` iterates through each fold of split data indices, allowing us to categorize all data as training features/labels or testing features/labels. Then, for each fold, it fits the classifier to the training set, makes a prediction based on the test features, and compares the prediction to the test labels. This allows us to keep a running total of true positives, true negatives, false positives, and false negatives. Once all of the iterating through different splits of training/testing data is complete, it uses the total positives and negatives to derive values for accuracy, precision, recall, F1, and F2 based on their mathematical definitions.

By comparing the results of `test_classifier` for the `best_estimator_` of different algorithms, I chose the algorithm that gave me the best combination of precision and recall. In this case, as detailed in previous sections, **Naive Bayes** turned out to be the winner.

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"]

When a classifier has a high level of **precision**, it yields a low rate of false positives, meaning when a POI is present in the data, we are good at flagging him or her. Using the Naive Bayes classifier, the precision is 0.38. Across all of the algorithms I tried, the average is 0.36, so the Naive Bayes classifier's precision is about average. My finding means that, when the Naive Bayes classifier predicts a person is a POI, that person is actually a POI 38% of the time.

When a classifier has a high level of **recall**, it yields a low rate of false negatives, meaning there is less of a likelihood we are going to ignore a person of interest when we should really be be looking at him or her more closely. Using the Naive Bayes classifier, the precision is 0.31. Across all of the algorithms I tried, 0.14 is the average. Naive Bayes's recall of 0.31 is the highest recall of any test I conducted using test_classifier. My finding means that the Naive

Bayes classifier, when comparing the number of true POIs with that of false non-POIs, identifies 31% of all POIs correctly as true POIs, and 69% incorrectly as false non-POIs.