Enron Submission Free-Response Questions

By Edward Lee

Question 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?*

The goal of the project is to see how we could use financial and email data about Enron employees that was publicly made available during the Enron scandal to identify possible persons of interest. Using machine learning, we are able to train and optimize algorithms to predict possible persons of interest based on other features provided to it.

The initial dataset consisted of 146 data points, which each represented an Enron employee and 21 features for each employee, which was a merging of financial and email data collected from various sources in the Enron fraud case. Of the 146 data points, 18 of them (12%) were known persons of interest and the remaining 128 were not persons of interest. Some features that had high percentages of missing values. Features with over 50% missing values were deferral_payments, deferred_income, director_fees, loan_advances, long_term_incentive, restricted_stock_deferred.

Through data exploration, several outliers were identified:

Outlier	**Treatment of Outlier**
Dataset included a data point for “TOTAL”	This data point was removed because it was the total values left behind from “enron61702insiderpay.pdf” and would skew our model.
The key for a data point was “THE TRAVEL AGENCY IN THE PARK”	This data point was removed because it represented an organization and not a person. To ensure consistency, we should ensure that each data point is representative of an individual.
“LOCKHART EUGENE E” contained all missing values	This data point was removed because having all missing values would skew our model.
The restricted_stock value for “BHATNAGAR SANJAY” contained a large negative value.	Cross-referencing “enron61702insiderpay.pdf”, it was found that the value was correct, but the sign was wrong. Therefore, the restricted_stock value for “BHATNAGAR SANJAY” was changed from a negative to a positive value.
All of the feature values for “BELFER ROBERT” were flagged as outliers.	Cross-referencing “enron61702insiderpay.pdf”, it was found that values for BELFER ROBERT were inputted incorrectly because they contained the values from adjacent columns. These values were manually corrected.

Question 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.*

I selected my features using a Decision Tree algorithm and choosing features with the highest importance values. When I ran the Decision Tree classifier, I observed the following features and feature importances:

Feature	Importance
exercised_stock_options	0.19984
total_payments	0.16141
bonus	0.10913
long_term_incentive	0.10895
shared_receipt_with_poi	0.10015
expenses	0.09667
from_poi_to_this_person	0.05561
from_this_person_to_poi	0.05549
salary	0.05296
other	0.03001
restricted_stock	0.02977
deferral_payments	0.00000
deferred_income	0.00000
director_fees	0.00000
from_messages	0.00000
loan_advances	0.00000
restricted_stock_deferred	0.00000
to_messages	0.00000
total_stock_value	0.00000

After, I incrementally increased the size of the feature list to include features from high to lower importance (feature importance had to be greater than 0.0) and compared the performance for each feature list. The following were the performances of the feature lists:

# of Features	1	2	3	4	5	6
Accuracy	0.8616	0.799	0.7965	0.78807	0.811	0.8272
Precision	0.31954	0.30204	0.30869	0.27932	0.29280	0.34867
Recall	0.34	0.3105	0.3425	0.306	0.295	0.341

# of Features	7	8	9	10	11
Accuracy	0.82553	0.83133	0.8254	0.80666	0.8006
Precision	0.34582	0.35874	0.34761	0.27897	0.25482
Recall	0.346	0.3365	0.353	0.284	0.2575

The best performances occurred when the feature list included the top 6 to 9 feature importances and a sharp drop in performance was observed when the feature list size was less than 6 or greater than 9. I decided to use the top 9 feature importances to optimize for recall performance.

Feature scaling was implemented using *MinMaxScaler* inside my test function *test_stratified_shuffle_split()* to optimize the performance of the KNN classifier that I will be evaluating in the next section.

As part of the assignment, I engineered a new feature called "percentage_salary" which represents what percent a person's total compensation (i.e. total_payments + total_stock_value) consists of their salary. My rationale was that salaries would be kept relatively consistent but POIs would get large compensations through other channels (e.g. bonuses, stock options, etc.). Therefore, I suspected that POIs would have a "percentage_salary" value that was much lower than non-POIs.

The following effects of "percentage_salary" on the performance of my classifier were observed:

Performance Metric	**Without "percentage_salary"**	**With "percentage_salary"**
Accuracy	0.822466666667	0.823933333333
Precision	0.336296296296	0.339025615269
Recall	0.3405	0.3375
F1	0.338385093168	0.338261087447
F2	0.339650872818	0.337804023621

Since adding "percentage_salary" had little or no effect on the performance of my classifier, I chose not include it in my final feature list.

Question 3: *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?*

I tested the performance between 2 different algorithms: Decision Tree and KNN. Since I used Decision Tree feature importance for my feature selection, which would likely favor the Decision Tree algorithm, I felt that I should also test with a different feature set to be fair. For the second feature set, I used SelectKBest to choose which features I tested with.

The following were the results:

Performance Metric	Decision Tree	KNN with Feature Scaling	KNN without Feature Scaling
Features with Feature Importance			
Accuracy	0.822466666667	0.857066666667	0.867333333333
Precision	0.334828101644	0.180327868852	0.509293680297
Recall	0.336	0.0165	0.137
Features with SelectKBest			
Accuracy	0.796357142857	0.853142857143	0.876571428571
Precision	0.284992420414	0.437219730942	0.687327823691
Recall	0.282	0.0975	0.2495

Surprisingly, applying feature scaling to the KNN classifier decreased performance in terms of both precision and recall. It could be that I have certain features in my feature list that have much more predictive power than others and when feature scaling is applied, it decreases the influence of these features and as a result, decreases performance. While the KNN classifier had good precision, the recall was much lower than the Decision Tree algorithm for both feature sets. Since the goal of this project is to create a classifier that could aid with investigations by identifying possible persons of interest, I felt it was important to have good recall performance. Therefore, I chose the Decision Tree algorithm for my final classifier.

Question 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?*

To tune the parameters of an algorithm is to optimize the settings of your algorithm so that it you're your dataset without being overly complex or simplistic. An algorithm that isn't tuned well can result in models that have high bias (i.e. does not effectively learn from your training data) or high variance (i.e. overfitted to the training data, but does not produce good real-world predictions). Tuning the parameters of your algorithm helps you find a balance between high bias and variance.

In this project, I was mostly focused on tuning 2 parameters that were important to the Decision Tree algorithm: min_sample_split, max_depth. Both were important parameters in tuning how the algorithm fit the data, so to avoid overfitting. Initially, I ran GridSearchCV on min_samples_split and max_depth, which returned low values for the parameters such as 'min_samples_split' = 2, 'max_depth' = 5. Then, I manually tested both parameters independently with values ranging from small to very large. The combination that I found that produced the optimal performance in terms of precision and recall was 'min_samples_split' = 2, 'max_depth' = 15, which I used for my final classifier.

Question 5: *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 a method of evaluating the performance of your model, typically by testing the model against testing dataset, which is independent of the data used to train the model. The classic mistake for validation is when the same data used for training the classifier is also used for testing, which usually results in inaccurate and unrealistically good performances.

For validating my analysis, I used a cross-validation technique, specifically Stratified Shuffle Split, which is the same validation method used by the tester.py function provided by the course. There are 2 reasons for why Stratified Shuffle Split was an appropriate cross-validation technique for the dataset. Firstly, with only 146 data points in our dataset, using the shuffle split technique would allow us to maximize the data that the Decision Tree algorithm can train with. Secondly, since we have a disproportionate number of non-POIs (88%) vs. POIs (12%) in our dataset, Stratified Shuffle Split ensures that we keep that characteristic consistent in the data when training and testing the algorithm.

Question 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.*

The 2 metrics that I used to evaluate the performance of my model was precision and recall. For my final classifier, the average performance was the following:

- Precision = 0.344091360477
- Recall = 0.3465

In the context of this project, the precision value tells us that if my model identifies a person as a POI, then approximately 34% of the time that person will truly be a POI. The other 66% of the time, the person will actually not be a POI. The value for the recall metric tells us that if a dataset is provided to my model, it will be able to identify approximately 35% of the POIs that exist in the dataset.