Machine Learning Free Response

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?

Enron, an American energy company, is famous for its bankruptcy and tremendous audit failure. In this project, we are given a dataset from Enron to detect fraud among certain people with financial and email data.

We are using machine learning in order to classify all employees in the dataset to either be a person of interest or not. Machine learning is useful in doing this because of its many classifiers and methods, feature input, and speed.

The only outlier I chose to remove was the entry “Total” because it is a spreadsheet quirk that shouldn’t be used for training or testing data. I left everything else in the dataset because we are more than likely looking for outliers as persons of interest here.

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 sed an automated feature selection like SelectKBest, please report the feature scores and reasons for your choice of parameter values.

Feature selection began by looking at how many missing data points are in each feature and created feature. I ended up changing all NULL values to 0.

After outlier removal, there were **145 data points** left, and **18 persons of interest**.

Printout of how many missing data points per feature:
```
bonus                    64
deferral_payments            107
deferred_income              97
director_fees            129
email_address            146
exercised_stock_options       44
expenses                  51
from_messages              60
from_poi_to_this_person       60
from_this_person_to_poi       60
loan_advances             142
long_term_incentive          80
```

other	53
poi	0
restricted_stock	36
restricted_stock_deferred	128
salary	51
shared_receipt_with_poi	60
to_messages	60
total_payments	21
total_stock_value	20
to_msg_ratio	60
from_msg_ratio	60
bon_plus_expenses	75
bon_sal_ratio	64

Then I checked the feature importances in DecisionTreeClassifier's attribute, `feature_importances_`. The following is a print out of the features and importances values.

DecisionTreeClassifier importances values:

total_payments : 0.220710506425
bon_plus_expenses : 0.207373271889
restricted_stock : 0.120556584967
exercised_stock_options : 0.117942168517
expenses : 0.116851004716
long_term_incentive : 0.108843537415
from_poi_to_this_person : 0.0846560846561
to_msg_ratio : 0.0230668414155
salary : 0.0
bonus : 0.0
deferred_income : 0.0
other : 0.0
to_messages : 0.0
email_address : 0.0
from_messages : 0.0
from_this_person_to_poi : 0.0
shared_receipt_with_poi : 0.0
from_msg_ratio : 0.0
bon_sal_ratio : 0.0

I also created features along the way after reading the PDF document containing descriptions of the finance features. `bon_plus_expenses` is the addition of bonus and expenses. `bon_sal_ratio` is the ratio of bonus divided by salary. I chose to create these ratios because the bonus numbers in the provided PDF document were enormous for certain people. Expenses were also enormous for certain people and nonexistent for others. I figured one way to identify persons of interest was to look for two things they would likely both have large numbers spending. Salary was the next thing I thought would highly affect identification, but it turned out to be pretty useless.

`to_msg_ratio` is the ratio of a person's messages to a person of interest to his or her total messages. `from_msg_ratio` is a ratio of a person's messages to his or her total messages. While it seems like these features are cheating because it inherently means we already know who the persons of interest are, neither of these were as useful as finance features in my classifiers. I would have imagined people who conspired with one another to have talked to each other more. One reason this may not have worked is because they didn't use email as a form of contact for their dastardly deeds.

Afterwards, I added a feature one by one until my recall and precision scores started dropping. I started with bonus plus expenses. Next I added exercised stock options, total payments, and long term incentive one by one. I ended up using three features, bonus plus expenses, exercised stock options, and total payments. I chose mostly based on DecisionTreeClassifier's importance values print outs and intuition from reading the PDF document.

Here is how the classifier performed with each feature addition and GridSearchCV optimizations after tuning:

features list = bon_plus_expenses
accuracy: 0.71600
precision: 0.50361
recall: 0.41800
f1: 0.45683

features list = bon_plus_expenses, to_msg_ratio
accuracy: 0.78867
precision: 0.53775
recall: 0.34900
f1: 0.42329

features list = bon_plus_expenses, from_msg_ratio
accuracy: 0.76960
precision: 0.38199
recall:0.24600
f1: 0.29927

features list = bon_plus_expenses, bon_sal_ratio
accuracy: 0.74611

precision: 0.40506
recall: 0.30400
f1: 0.34733

features list = bon_plus_expenses, exercised_stock_options
accuracy: 0.81292
precision: 0.44068
recall: 0.45500
f1: 0.44772

features list = bon_plus_expenses, exercised_stock_options, total_payments
accuracy: 0.83400
precision: 0.43165
recall: 0.51150
f1: 0.46819

features list = bon_plus_expenses, exercised_stock_options, total_payments, restricted_stock
accuracy: 0.83073
precision: 0.36572
recall: 0.36700
f1: 0.36636

features list = bon_plus_expenses, exercised_stock_options, total_payments, long_term_incentive
accuracy: 0.83229
precision: 0.42169
recall: 0.46850
f1: 0.44387

These are before I changed one parameter, `random_state`, from 2 to 42. The new accuracy, precision, and recall scores are 0.86336, 0.51860, and 0.60650, respectively.

I used StandardScaler to scale the data for KNeighborsClassifier since it wouldn't work very well with data that isn't scaled. In addition, specifically for KNeighborsClassifier, I used SelectKBest in the entire list of features for feature selection. Here is a printout of the scores:

```
SelectKBest scores:
exercised_stock_options : 25.0975415287
deferred_income : 11.5955476597
email_address : nan
bon_plus_expenses : 23.297519463
bonus : 21.0600017075
salary : 18.575703268
bon_sal_ratio : 10.9556270745
long_term_incentive : 10.0724545294
restricted_stock : 9.34670079105
total_payments : 8.86672153711
shared_receipt_with_poi : 8.74648553213
```

expenses : 6.23420114051

from_poi_to_this_person : 5.34494152315

from_msg_ratio : 5.20965022058

other : 4.2049708583

to_msg_ratio : 4.16908381529

from_this_person_to_poi : 2.42650812724

to_messages : 1.69882434858

from_messages : 0.164164498234

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

I ended up with DecisionTreeClassifier. It provided the best precision and recall scores together after tuning. It also worked much faster, so I had more opportunities to explore its parameters within the dataset.

I also tried RandomForestClassifier and AdaBoostClassifier with DecisionTreeClassifier as its base estimator. AdaBoost didn’t do anything special in my parameters besides take a lot longer.

I tried KNeighborsClassifier last in a pipeline with StandardScaler and SelectKBest because it worked in a different way than the tree classifiers I’d tried before. I also tried its performance with PCA in the pipeline, but it only made precision and recall scores worse.

After tuning all three classifiers with GridSearchCV, here are the accuracy, precision, and recall scores.

RandomForestClassifier:
Accuracy: 0.85186
Precision: 0.48013
Recall: 0.44700

DecisionTreeClassifier:
Accuracy: 0.86336
Precision: 0.51860
Recall: 0.60650

KNeighborsClassifier:
Accuracy: 0.83479
Precision: 0.42241
Recall: 0.42600

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 a model that was not your final choice or a different model that does utilize parameter tuning e.g. a decision tree classifier).

Tuning the parameters of an algorithm is adjusting how the algorithm works on a more polished level. If you don't tune your parameters, you can either get a classifier that is overfitted to a set of training data or one that is too general that it's useless to set loose.

For my tree classifiers, I tuned its minimum samples split, minimum number of samples to be a leaf node, and its maximum number of features at each split. I added all these to a bunch of different options with GridSearchCV and picked the best parameters based on that. I continued to add parameters one by one and checking the evaluation scores until I was satisfied with my results.

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 way to test your trained model. It gives an estimate of its performance and checks for overfitting. If you don't do validation properly, you may end up with a classifier that is perfectly fine for one set of training data but flops on another set, unable to have a general use.

I used StratifiedShuffleSplit for my cross validation. This splits the data in training and testing data, while maintaining there are some POI in each set of test data. Since this is a highly imbalanced data set with a small number of POI, I want each training and test set to have some POI in it. If I sampled the data randomly without a stratified method, I'd likely have many sets with zero POI and do a poor job training my classifier. In my code, I chose 100 different splits and shuffles to be exhaustive when tuning.

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.

In this project, we used both precision and recall as evaluation metrics. Since this data set is imbalanced, accuracy is a poor metric. There are many, many more opportunities to label a data point a non person of interest which can skew our accuracy score to be higher than its actual effectiveness.

Precision is the ratio of true positives to the sum of true positives and false positives. It is a measure of how well the classifier found positives in the dataset. In this project, the true positives are persons of interest classified as persons of interest. The false positives are non persons of interest classified as persons of interests. The higher the score, the better the classifier performed. If the score is low, it could mean the classifier identifies too many persons of interest, more than actually present in the data.

Recall is is the ratio of true positives to the sum of true positives and false negatives. In this project, the false negatives are persons of interest who are classified as non persons of interest. Recall tells the ratio of how may correct results the classifier found in relation to how many correct results existed in the dataset. Ideally, the recall score should be 1 or close to 1. If the score is low, then the classifier is doing a poor job actually finding persons of interest in the data.