Enron Submission Free-Response Questions
Miri Cho

Goal of the project:
The goal of the project was to apply machine learning algorithms to predict persons of interest (POI) based on a set of features, financial or email-related.

Description of the dataset:
Data Size
of observations: 146
# of total features: 21
POI
of POI in dataset: 18
# of non-POI in dataset: 127
non-POI vs. POI
-After removing one outlier by key value 'TOTAL', I looked at the absolute differences in median in an descending order to see which feature had the most drastic differences between POI's and non-POI's.

POI vs non-POI: absolute differences in median: (Descending)
[['loan_advances', 80325000.0], ['exercised_stock_options', 2884228.0], ['total_stock_value', 1176506.5], ['long_term_incentive', 759333.0], ['total_payments', 697935.0], ['bonus', 575000.0], ['restricted_stock', 571445.5], ['deferred_income', 141216.0], ['other', 129947.0], ['deferral_payments', 57544.0], ['salary', 26947.0], ['expenses', 5847.5], ['shared_receipt_with_poi', 995.0], ['to_messages', 931.0], ['restricted_stock_deferred', nan], ['from_poi_to_this_person', 35.5], ['from_this_person_to_poi', 9.5], ['from_messages', 3.5], ['director_fees', nan]]

Outliers:
Through EDA process, I noticed a large outlier by the key value in the dataset dictionary called 'TOTAL'. Then, I reviewed top 5 values for every feature and their name to investigate outliers as example below.

[['salary',
  [[1111258, 'SKILLING JEFFREY K'],
   [1072321, 'LAY KENNETH L'],
   [1060932, 'FREVERT MARK A'],
   [655037, 'PICKERING MARK R'],
   [510364, 'WHALLEY LAWRENCE G']]],

All the top values seemed normal and the only features with negative values were financial features with 'deferred' accounts, which seem understandable. Therefore, I proceeded with the analysis after just removing 'TOTAL', leaving the dataset with 145 observations. (data_dict.pop('TOTAL'))

Missing values:
As I was going back and forth with EDA and running machine learning algorithms, I noticed that there were some features with lots of missing values, including 'director_fees'. By dividing number of missing values for a given feature by the total number of observations, I noticed that the following features had over 50% of missing value rate. This is understandable, given that only directors would get paid 'director_fees'. At this point, I tried using only features with missing value rates less than 50%.

low_missing_rates:
['salary',
 'to_messages',
 'total_payments',
 'exercised_stock_options',
 'bonus',
 'restricted_stock',
 'shared_receipt_with_poi',
 'total_stock_value',
 'expenses',
 'from_messages',
 'other',
 'from_this_person_to_poi',
 'from_poi_to_this_person']

Features with lots of missing values: (over 50% missing rate)
[['deferral_payments', 0.7379310344827587], ['restricted_stock_deferred', 0.8827586206896552], ['loan_advances', 0.9793103448275862], ['director_fees', 0.8896551724137931], ['deferred_income', 0.6689655172413793], ['long_term_incentive', 0.5517241379310345]]

Feature Selection:

1) Process (feature importances)

First, after having eliminated the features that had a high missing value rate (over 50%), I used feature_importances of Decision Tree Classifier to come up with the best features in order. With this information (importances mapped to each feature name), I started iterations of running algorithms and observing the results.

(0.16035526306340503, 'bonus'), (0.15524149727997355, 'exercised_stock_options'), (0.13404700097193251, 'expenses'), (0.11498933668864429, 'other'), (0.089414322307277014, 'total_payments'), (0.06201219968649007, 'restricted_stock'), (0.05942210979797196, 'from_this_person_to_poi'), (0.048559715164026143, 'shared_receipt_with_poi'), (0.041494409661428935, 'total_stock_value'), (0.036430359393548682, 'from_messages'), (0.035600893889633785, 'salary'), (0.033803234158814983, 'from_poi_to_this_person'), (0.028629657936852685, 'to_messages')]

Each time I would run the algorithms (using GridSearchCV for tuning parameters and only using the best parameter for the selected features for final validation/evaluation), I would use the process of elimination to weed out features that didn't produce meaningful results. However, this approach alone didn't yield impressive results (precision and recall metrics) and I moved on to the next process of creating new features (3).

2) Scaling

I tried using MinMaxScaler() and StandardScaler() each time to smoothen out the large differences, but the results of algorithms were not affected.

3) New Feature

In creating a new feature, I first tried multiplying all email-related features by 1000, which did not affect the results at all. Then I selected more important email-related features (based on feature importances from above) and added them together to create one new feature (and multiplied by 10,000), which I then used along other financial features. This greatly improved the results. I iterated this approach with different sets of features and then finally tried creating two features with one feature representing significant email-related features, and the other feature representing less significant financial features. These two new features then would be used with other most significant financial features (based on feature importances). This final approach returned the best combination of precision and recall rates.

```
new_feature1 = ['other','total_payments','restricted_stock']
new_feature2 = [ 'from_this_person_to_poi','shared_receipt_with_poi','from_messages']

final_feature_set = [
'poi',
'bonus',
'exercised_stock_options',
'expenses',
'feature1',
'feature2'
]

[DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
        max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        random_state=None, splitter='best'), 0.8425714285714285, 0.4484848484848485,
0.444]
```

Algorithms:

For every set of chosen features, I used iterations of SVC and Decision Tree and GaussianNB. SVC continued having division errors, and GaussianNB, while returning higher precision, did not

return higher recall than Decision Tree many times. Based on the final set of features, I chose to use Decision Tree.

The best Decision Tree result from several iterations:
Highest Recall: 0.444
Highest Precision: 0.448

The bestGaussianNB result from several iterations:
Highest Recall: 0.296
Highest Precision: 0.475

Tuning Parameters:

Parameter tuning is important because given different parameters, algorithms can return drastically different results. Between different parameters, there is also a trade off between bias and variance. I used GridSearchCV to tune the parameters of both SVC and Decision Tree and used StratifiedShuffleSplit with 1,000 folds, the same way tester.py did validation. SVC continued having issues with division by zero, and DT Classifier mostly changed the min_samples_split sizes as it was tuning for different sets of features. When min_samples_split was set to 2, DT Classifier returned the best results and there was a large discrepancy in the results between different min_samples_split values.

Validation
Validation is used in the process of splitting data into training and test sets to use algorithms to train on the train set and test on the test set. The way we split data can make a lot of differences on evaluation. The classic mistake would be to fit on the train dataset and predict the same train dataset, instead of predicting on the test dataset. I tried using sklearn's train_test_split, and quickly realized that this way wasn't as effective as making k-folds. As I was tuning the algorithm parameters, I used train_test_split and noticed that tuned parameter based on this one train and one test set did not do well against StratifiedShuffleSplit validation in the tester.py. I also tried using StratifiedKFold but results did not change much.

Evaluation metrics
I primarily used 'precision', and 'recall' to evaluate both feature sets and algorithm and different parameters. 'precision' shows the rate of # of POIs that the model correctly predicted on test set over total guesses on POI's. 'recall' shows the rate of # of POIs that the model correctly predicted on test set over # of actual POI's.

The metrics were critical in choosing features and algorithms and parameters, especially due to the skewed nature of the POI numbers. Because only 14% of the total in the dataset are POI's, it's much easier to get a high recall rate if it assumes they are all POI's. This was the case for sets of features that had a pattern for the POI groups, such as director fees. Because none of the POI's had director fees values, I think the model assumed that anyone who did not have a

director fee must be a POI, raising the recall rate to 1. However, choosing features based on recall would be a trade-off with a high precision, I needed to find a balance between the two metrics.