Enron Submission Free-Response Questions

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

For this project, we utilize machine learning skills to choose the best algorithm to identify Enron employees who may have been involved or have vital information regarding Enron fraud based on the public Enron financial and email datasets.

The approach to arrive at a best model include:

- handling real-world dataset: Since such dataset tends to be imperfect, we may have to clean and remove the data plus make some assumptions about the data inadequacies;
- validating machine learning results using test data;
- evaluating machine learning result using quantitative metrics such as accuracy, precision, and recall scores;
- dealing with features: This includes creating, selecting, and transforming variables;
- comparing the performance of various machine learning algorithms
- fine tuning machine learning algorithms for maximum performance; and
- communicating machine learning algorithm results clearly.

We ended up removing three observations:

- TOTAL: The last line of the data table that sums up all variables.
- THE TRAVEL AGENCY IN THE PARK: Not a person.
- LOCKHART EUGENE E: Have 'NaN' values for all features.

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.

We utilized SelectKBest and GridSearchCV to arrive at an ideal number of selected features for this Machine Learning analysis. Based on this analysis, the ideal number of features is 13. We decided to exclude 'total_stock_value' and 'total_payments' because we expected them to be highly correlated with their compoenents. For instance, we expected 'exercised_stock_options' and 'total_stock_value' to be correlated because the former is a component of the latter. With 2 features excluded, we ran a simple Naïve Bayes model on top-n features (n= 3, 4, 5, ..., 11). We concluded that 5 features was the way to go given the performance metrics.

	Accuracy	Precision	Recall	F1
Top 3	0.84277	0.48281	0.3090	0.37683
Top 4	0.86307	0.53081	0.3575	0.42725
Top 5	0.86214	0.52273	0.4025	0.45480

	Accuracy	Precision	Recall	F1
Top 6	0.84550	0.45192	0.3830	0.41461
Top 7	0.85021	0.47145	0.4005	0.43309
Top 8	0.76479	0.24795	0.3180	0.27864
Top 9	0.77067	0.23194	0.3115	0.26590
Top 10	0.77753	0.23712	0.3015	0.26546
Top 11	0.77753	0.23691	0.3010	0.26514

Yes, I utilized a *MinMaxScaler()* scaling in my Support Vector Machine (SVM) algorithm. After playing around with the parameters in SVM, I didn't find any solid results. So, I toyed around with scaling. My rationale was that stocks, bonus, and salary are be on a different scale and their outliers may have different magnitudes. For instance, being cumulative based on number of years with the company, stocks may be very small or very big compared to annual salary and bonus.

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 a Naive Bayes (NB) algorithm.

```
GaussianNB(priors=None)
        Accuracy: 0.86214        Precision: 0.52273       Recall: 0.40250
        F1: 0.45480      F2: 0.42191
        Total predictions: 14000  True positives:  805     False positives:  735
        False negatives: 1195     True negatives: 11265
```

Comparing with the best performing SVM and Decision Tree Classifier (DTC), NB has the highest F1 with its precision and recall greater than 0.3. SVM and DTC performed poorly in recall at 0.134 and 0.225, respectively.

	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.86214	0.52273	0.4025	0.45480
SVM	0.85485	0.63420	0.1335	0.22057
Decision Tree Classifier (DTC)	0.83621	0.37720	0.2250	0.28187

Table: Algorithms Tested and Their Best Tuned Model

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?

Tuning the parameters means changing the model specifications to enhance a model's performance which includes, accuracy, precision, recall, and predictive strength. If we didn't do well with the parameter tuning, we may have to wait for a very long time for the fitting to be completed (e.g. a *linear* kernel in SVM) and overfitting that the estimated algorithm does well for the train data but poor with the test data.

For parameter tuning, we did extra research on the algorithm and asked questions on whether t makes sense for us to touch certain parameters. We did not go into depth on each algorithm's parameters in the class, though, the instructors encouraged us to be smart researchers by doing our own research on each

algorithm parameters. Since this is our first experience with Machine Learning, we applied maximum likelihood approach to parameter tuning. After researching on the plausible and range for these parameters, we further altered the parameter value selection if the best parameters chosen had the lowest or highest value of the ones we considered. For instance, if we had if C = 10 was the parameter for the best SVM model based on these C values *'svm__C': [0.001, 0.1, 1,10]* , I would go with higher C values until the best model's C is not the highest one in the C array.

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 the process of analyzing algorithm performance with the test data. The most common mistake is overfitting where the algorithm works great with the train data but poor with the remaining data. We have noticed that the precision and recall metrics based on confusion metrics changed as we altered the *test_size* percentage.

In the project, we run the tester.py algorithm to validate our findings. The validation method is named *stratified shuffle split* cross validation. The *test_classifier* function indicates that the performance metrics were calculated based on 1,000 runs with randomized train-test subsamples – all using the same Enron dataset. Based on tabulations of *true_negatives, false_negatives, false_positives,* and *true_positives* from these 1,000 runs, the tester.py program calculates accuracy, precision, f1, and f2 performance metrics. This method is preferred because it averages out performance metrics from a huge number of run. A train-test data sampling may be biased that we may end up with a train sample with zero POIs. This bias should be minimized with 1,000 runs with randomized train-test subsample data shuffling. A very important aspect of *stratified shuffle split* cross validation is that we stratify the sampling for POIs in both the train and test datasets. This is essential due to the small proportion of POIs – that some pure random sampling may result in the train or test subsamples getting zero POI observation.

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.

	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.86214	0.52273	0.4025	0.45480
SVM	0.85485	0.63420	0.1335	0.22057
Decision Tree Classifier (DTC)	0.83621	0.37720	0.2250	0.28187
DTC, 3 Features	0.81777	0.38879	0.3225	0.35256

Table: Algorithms Tested and Their Best Tuned Model

Definitions:

Accuracy: number of observations correctly predicted as POI or non-POI / number of all observations in the dataset

Precision = number of observations correctly predicted as POI / number of predicted POI observations

Recall = number of observations correctly predicted as POI / number of actual POI observation

Our best model is the Gaussian Naïve Bayes and its best model's performance metrics are notes in the table above. Based on this algorithm, its specifications, and data, 86.2 % of the observations in the Enron dataset were correctly predicted as POIs and non-POIs. Among those predicted as POIs by my best algorithm, 52.2 % were actual POIs. Among those who were actually POI, the model successfully predicted 45.5 % of them by my best algorithm.