Project Overview and Objectives:

The goal of the project is to use machine learning techniques to create an algorithm to identify persons of interest in a data set consisting of individuals related to the Enron Corporation, which collapsed in the early 2000s as a result of fraud. The data set consists of 146 entries and each entry has 21 features, although not every entry has the required data for every feature. The features for each entry include data related to financial aspects such as 'salary' and 'exercised_stock_options' and also email related features such as "email_address' and 'to_messages'. This is because the data is derived from combining both financial data and publicly available email data for a large number of individuals related to Enron. The data set consists of 144 individuals related to the corporation and 2 items which are outliers.

Each of the entries also has a feature called 'poi' which states whether the entry is of a person considered a person of interest (POI). Of the 144 individuals, 18 individuals are POIs in that they have either been convicted or charged with a crime in the Enron related fraud or are of notably higher than average interest in relation to their role in the Enron fraud. The first outlier is the entry named 'TOTAL' which presumably is an aggregation of the 144 persons in the data set. The second outlier is an entry named 'THE TRAVEL AGENCY IN THE PARK', which seems to be an organization rather than a person. Since the focus of this project is to identify POIs based on their financial or email data characteristics, the data from these two entries is not relevant to creating and training an algorithm, also called an identifier, that will help identify POIs and thus both of these entries have been removed.

Features Used and Feature Selection Process:

I used the 'deferred_income', 'restricted_stock_deferred', and 'director_fees' features with a Naïve Bayes algorithm. I wrote code to run many combinations of features with the Naïve Bayes (and other) algorithm(s) and selected these features based on the satisfactory performance of the algorithm with these features.[1] Please see accompanying document named "Naïve Bayes Combinations Tried" – this document shows the full range of combinations tried using the Naïve Bayes algorithm. I chose these identifier features since among these combinations tried I did not find a combination that offered better performance than my chosen features.

I only tried combinations that included up to four features in this document since once I came across the feature combination I chose noted above I felt adding additional features would not be more useful. This is particularly so since trials with four feature combinations (see "Naïve Bayes Combinations Tried" document) did not result in a four feature combination with more impressive performance than the one I chose.

Reasoning Behind Using Naïve Bayes:

I ended up using a Naïve Bayes algorithm because I was able to develop a POI identifier that used only 3 features and provided a satisfactory performance in that both precision and recall metrics (discussed in more detail below) were over the required 0.3, but with the additional benefit that the recall metric was 1.0 – essentially guaranteeing that a POI would be identified, while also maintaining a reasonable precision score. Since I used the Naïve Bayes algorithm, I did not have to tune parameters on that identifier.[2]

I also tried the decision tree algorithm, with a large variety of features, and in many trials I was able to come up with identifiers that resulted in precision and recall scores over 0.3 but none of these had a performance I would consider as superior to the Naïve Bayes algorithm I chose and thus I did not use these algorithms. In the case of Support Vector Machine algorithms, my trials with several features did not result in even one instance of the precision and recall metrics meeting the requirement of being above 0.3.

Validation and Evaluation Metrics for the Identifier:

The key to validation is to split the data into training data and testing data. The training data is used to choose and tune the algorithm on the basis of which the identifier will be built. The identifier is then tested on separate testing data.

I used Udacity provided code in the tester.py file, specifically the 'test_classifier' function to validate my identifier. The 'test_classifier' function uses the sklearn 'StratifiedShuffleSplit' method to shuffle data, break it into stratified folds—this involves preserving the same percentage for each target class in each of the folds as is present in the entire data set—and split the data into training and testing data and then develop performance metrics by taking average performance after testing the algorithm multiple times with different folds in the training and testing area each time.[3][4]

A classic mistake made during validation is to test and validate an algorithm using data that was already used to train the algorithm. This results in overfitting, which means that the algorithm has already been trained on the test data, which will likely mean that the algorithm's performance on the that test data will be better than its performance on test data (or a real world non training/testing situation data) on which the algorithm has not already been trained.

Most machine learning algorithms are built to work on data that is not used to train the algorithm and thus performance should also be validated on the basis of data not used in training if one wants a useful picture of algorithm performance.

Evaluation Metrics and Model Performance:

The validation method described above provided in tester.py prints out the following evaluation metrics:

Accuracy: 0.70733 Precision: 0.36284 Recall: 1.00000 F1: 0.53248 F2: 0.74008 Total predictions: 6000 True positives: 1000 False positives: 1756 False negatives: 0 True negatives: 3244

The data above for Accuracy, Precision, and Recall are average performances for the POI identifier after using the 'StratifiedShuffleSplit' method described above, which validates the identifier multiple times—each time with different folds—and then comes up with the averages.

Three metrics are worth discussing:

Accuracy: This is the proportion of predictions that is accurate. In heavily imbalanced data such as the Enron fraud data set in which there are only a few POIs relative to the entire data set, this metric can be relatively high simply because an algorithm predicts that the vast majority of entries in the data set are not POIs. This could be the case even in a situation where the identifier in a very large majority or in all instances incorrectly classifies a POI as a non-POI.

Thus, while the metric is useful on a general level, other metrics discussed below are arguably more important. In this case, the average for accuracy is 0.70733 – The average accuracy would have been much higher if the identifier simply characterized all entries in the data set as being non-POIs. However, the goal of the identifier also includes maximizing both Precision and Recall as discussed below and thus that leads to a lower accuracy rate.

Precision: This metric is derived from dividing True Positives by the sum of True and False Positives. Essentially 1000/(1000 + 1756) = 0.36284. This metric is focused on providing the proportion of times the identifier predicts that an entry in the data set is that of a POI and is correct. This metric comes in over 0.3 but it is not very high, in contrast to the recall metric (discussed below) which is 1.0. If this metric were very high it would mean that the identifier is infrequently wrong when it predicts that an entry is a POI. In this instance that is not the case since the identifier is correct only slightly over a third of the time.

Recall: This metric is derived from dividing True Positives by the sum of True Positives and False Negatives. Essentially 1000/(1000 + 0) = 1

This metric essentially is the proportion of times the identifier characterizes a POI as a POI. In this case the metric is 1.0, which means that in no instance was a POI not identified by the identifier.

If an identifier simply categorized every entry as a POI, the Recall metric would also be 1. However, in such an instance Accuracy would be much lower since only 18 out of 144 entries are POIs and the accuracy would fall dramatically to approximately 0.12. Similarly in such an instance the Precision metric would be much lower than 0.36 since there would be many more false positives.

My identifier, however, maintains a perfect Recall rate even while maintaining a relatively high Accuracy rate and a decent Precision rate at 0.36. The benefit of using my identifier in a practical setting is that if one wants to find POIs in new similar data, then one ought to focus on the entries flagged by my identifier as POIs and not focus on entries not flagged as POIs. This can help save substantial resources with the help of machine learning.

[1] "python – How to get all possible combinations of a list's elements? – Stack Overflow" - https://stackoverflow.com/questions/464864/how-to-get-all-possible-combinations-of-a-list-s-elements; Author(s): Multiple Contributors; Date of Publication: Contributions from 2009 through 2017; Access Date(s): Approximately February – April 2018; Used to write code on deriving combinations of features.

[2] "sklearn.naive_bayes.GaussianNB --- scikit-learn 0.19.1 documentation" - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html; Author: scikit-learn developers (Includes multiple contributors); Date of Publication: November 2017 (estimated – based on document available on 4/12/2018 at http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf); Access date(s): Approximately February – April 2018; Viewed as background information on Naïve Bayes.

[3] "sklearn.cross_validation.StratifiedShuffleSplit --- scikit-learn 0.19.1 documentation" - http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html; Author(s): scikit-learn developers (Includes multiple contributors); Date of Publication: November 2017 (estimated – based on document available on 4/12/2018 at http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf); Access date(s): April 2018.

[4] "3.1. Cross-validation: evaluating estimator performance --- scikit-learn 0.19.1 documentation" -http://scikit-learn.org/stable/modules/cross_validation.html; Author(s): scikit-learn developers (Includes multiple contributors); Date of Publication: November 2017 (estimated – based on document available on 4/12/2018 at http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf); Access date(s): April 2018; Viewed as background on cross-validation.