Udacity Intro ML Final Project

Answers to Project Questions

1) The goal of this project is to classify *person's of interest* (POIs) using Enron financial and email-frequency data provided by *Udacity*. The data set includes 146 records (with 21 features), 18 records of which were labeled as POIs. By examining the data set and making use of what appear to be the most relevant features, it becomes possible to predict POIs with an accuracy rate better than guessing.

[N.B. Unless otherwise specified, GaussianNB was used as a control classifier when discussing performance metrics.]

Using all of the initial features (except for email address (which was removed as a feature due to its obvious ability to identify POIs), precision and recall scores (defined below) were 0.14720 and 0.83, respectively.

Outliers were present in the data. By reviewing those outliers, it became clear that at least one of the records did not represent an individual but the calculated totals of each column. This *TOTAL* record was removed from the data set, as well as record with the key *THE TRAVEL AGENCY IN THE PARK*. This modification caused the precision and recall scores to increase (precision: 0.14746, recall: 0.83450).

All missing values (encoded as 'NaN's) were imputed with their respective columnar medians. This modification caused the precision and recall scores to change (precision decreased: 0.14719, recall increased: 0.84200).

2) I created a single new feature hoping that it would capture the relationship between *bonus* and *salary* (in this case, their ratio), which I hoped would have a relationship with POI identification. This modification caused the precision score to increase (precision: 0.14812), while the recall score remained unchanged.

I removed apparently unhelpful features by implementing **SelectKBest** and reviewing the resulting scores. *bonus*, *deferred_income*, *exercised_stock_options*, *salary*, and *total_stock_value* all scored above 10. After removing all other features, the precision score increased (0.50161) and the recall score decreased (0.39050).

At this point, I added a **Pipeline**, **StratifiedShuffleSplit**, and **GridSearchCV**. This modification did not cause the precision or recall scores to change.

Then, I added **StandardScaler** to the **Pipeline**. This modification did not cause the precision or recall scores to change.

Then, I added **RandomizedPCA** to the **Pipeline**. This modification caused both the precision score and recall score to decrease (precision: 0.47740, recall: 0.34850).

Finally, I provided parameter options for **GridSearchCV** to tune for **RandomizedPCA**: iterated_power = [1, 2, 3], n_components = [2, 4, 6, 8, 10], whiten = [True, False]. On the first attempt, **GridSearchCV** selected the following values: iterated_power=1, n_components=2, whiten=True. These settings caused the precision score to increase (0.58291) and the recall score to decrease (0.34800).

3) As noted above, **GaussianNB** was used as the control classifier.

DecisionTreeClassifier was implemented. With its default settings, it gave lower precision and recall scores: 0.22455 and 0.24150, respectively.

The following **DecisionTreeClassifier** parameter options were provided for **GridSearchCV** to tune: class_weight = ['auto', None], criterion = ['gini', 'entropy'], max_features = ['auto', 'sqrt', 'log2', None]. On the first attempt, **GridSearchCV** selected the following values: class_weight=None, criterion='gini', max_features=None. These settings caused both the precision and recall scores to increase (precision: 0.30337, recall: 0.30200).

SVC was implemented. With its default settings, it gave lower precision and recall scores: 0.27273 and 0.04350, respectively.

The following **SVC** parameter options were provided for **GridSearchCV** to tune: C = [2**x for x in np.arange(-15, 15+1, 3)], gamma = [2**x for x in np.arange(-15, 15+1, 3)]. On the first attempt, **GridSearchCV** selected the following values: C=512, gamma=0.001953125. These settings caused both the precision and recall scores to increase: 0.44633 and 0.11850, respectively.

Out of the classifiers used, **GaussianNB** performed the best (precision: 0.58291, recall: 0.34800), so it was selected as the final algorithm.

4) Most algorithms have parameters that should be tuned in order to modify their training performance, which is in part dependent on the nature of the data set under consideration.

As discussed above, for both preprocessing and classification, **GridSearchCV** was used to tune various parameters (where applicable) for each implemented preprocessor and classifier. For suggestions related to parameter tuning for SVC(), I consulted http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

5) Breaking the data into training and testing sets is a component of a statistical assessment method called *cross-validation*, which is an effort to estimate how well a model will perform/generalize when it encounters *new* data. To this end in the current context, data sets used for training machine-learning models are broken up into separate *training* and *testing* sets. After models are trained with the training data alone, models are then evaluated using only the testing data.

As the current data set under consideration includes, as mentioned above, only 146 records, **StratifiedShuffleSplit** was used via **GridSearchCV** during the preprocessor and classifier evaluation phase since it allows for the possiblity of training and testing on all of the data available (i.e., as opposed to training only on a single training set and testing only on a single testing set), averaging the results of each learning session in order to provide a clearer picture of model performance.

Unlike in the current project, if the data separation process described above is not randomized, it is possible to partition the data in such a way that the training and testing sets do not both properly represent the data in a balanced manner (e.g., when experimenting on the Enron data set, it would be possible to create a training set containing only non-POI-related data and a testing set with only POI-related data).

6) Given the distribution of the final testing set, in particular the small number of POIs, and likely in part due to the binary nature of this specific classification problem, the generic score (i.e., the percentage of correct predictions out of the total number of predictions) for each classifier is easily made higher when POIs are not identified at all (i.e., all predicted labels are 0).

Thus, in this case, using a *confusion matrix* (and the resulting precision and recall scores) is a better indicator of model/prediction performance. In the current context, both precision and recall scores may be understood as capturing the percentage of correctly-identified POIs; however, precision scores incoporate the number of incorrectly-indentified POIs while recall scores incorporate the number of incorrectly-unidentified POIs.

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.

Aside from standard usage of scikit-learn and Python documentation/examples, none of the work above is dependent on any particular source and, thus, the work is my own.

Epilogue: email text as data

I had hoped to use email-text data along with the financial data, but clearly there is a discrepancy between the individuals represented by the financial data and the email data. Thus, it would be difficult to join the two data sets in a meaningful way due to their lack of overlap.

After this project, I may spend some time with text-vectorization of the email corpus and examine things like word frequencies; however, with such an apparent low number of known POI-emails being available (only 3!?), it's entirely unclear how useful such an endeavor would be for identifying POIs, although there are surely other interesting insights to be gleaned.