Thomas Lederer
Introduction to Machine Learning,
Data Analytics Nanodegree

Free-Response Section: Enron Dataset

1. The Dataset

The goal of this project is to employ machine learning on financial and email meta-data to determine if a former Enron employee was a person of interest (POI) in the subsequent criminal investigation. Since there are numerous features for each data set and no obvious correlation between any of these features and an employee's status as a POI or non-POI, machine learning is a useful tool for accomplishing this goal. The dataset, however, also has some distinctive dimensions which determine the success and direction of the project:

i. Size: there are just 146 people in the total directory
ii. One-sidedness: of these 146, only 18 are POIs
iii. Multi-dimensionality: there are 20 features for each employee (not including 'poi': True/False which is used as a label

Of the 146 employees, there are also two outliers to be removed from the dataset. These were:

i. "Total" : an aggregate category of all employees
ii. "THE TRAVEL AGENCY IN THE PARK": a non-individual

(Beside the above, there were two more data points that, after some more exploration of the dataset's features, I also specified as outliers. These will be discussed separately under feature selection below in 2.)

1. Feature Selection

I approach the process of feature selection in two main stages: (i) feature creation and (ii) feature/dimension reduction.

Feature Creation
In the feature creation stage, I wanted to capture some more data about the logical relationship between the features that might be used to eventually improve the performance of my chosen algorithm. I found that in almost every case, the "total_payments" value was equal to the following sum:

"salary" + "deferral_payments" + "bonus" + "expenses" + "loan_advances" + "other" + "director_fees" + "deferred_income" + "long_term_incentive"

Also:
“total_stock_value” was nearly always equivalent to the sum:

“excerised_stock_options” + “restricted_stock” + “restricted_stock_deferred.”

The two instances in which this equation did not work were: 'BELFER ROBERT' and 'BHATNAGAR SANJAY' which I removed as outliers.

I then created ratio or percent values for each subcategory. My intuition was that while there may be some variance between POIs in “loan_advances” for example, we might find this number to be relatively high to their “total_payments.”

The features I created and their equations were:

“percent_salary” = “salary”\ ”total_payments”
"percent_deferral_payments" = “deferral_payments”\ ”total_payments”
"percent_bonus" = “bonus”\ ”total_payments”
"percent_expenses"= “expenses”\ ”total_payments”
"percent_loan_advances" = ”loan_advances”\ ”total_payments”
"percent_other" = “other”\ ”total_payments”
"percent_director_fees"= “director_fees” \”total_payments”
"percent_deferred_income" = “deferred_income” \”total_payments”
"percent_long_term_incentive" = “long_term_incentive”\ ”total_payments”

"percent_exercised_stock_options" = “exercised_stock_options” / “total_stock”
"percent_restricted_stock" = restricted_stock/ “total_stock”
"percent_restricted_stock_deferred" = restricted_stock_deferred / “total_stock”

(It is also worth noticing briefly that, in this respect, a zero value should not be conflated with a NaN. I believe a zero value for “loan_advances” for example, tells us something relevant about a given employee and should not be discarded as if this value was empty or could not be determined.)

Feature Selection
After the creation of these features, I now had so many features than any algorithm was prone to over-fitting. In my final algorithm, I employed scaling and PCA to reduce the number of features to only 2. It was necessary to use scaling in this case as my new features were proportions opposed to integers.

PCA variance ratio for the components were:
```
[ 0.20127772  0.17352578]
```

Although, in the finial algorithm, I used PCA to “pick” my features for me, the creation of these features was indispensible to the success of my final algorithm.

Running the same algorithm without the inclusion of the features created yields the below results:

```
Accuracy: 0.843 Precision: 0.097 Recall: 0.021, F1: .035, n: 15,000
```

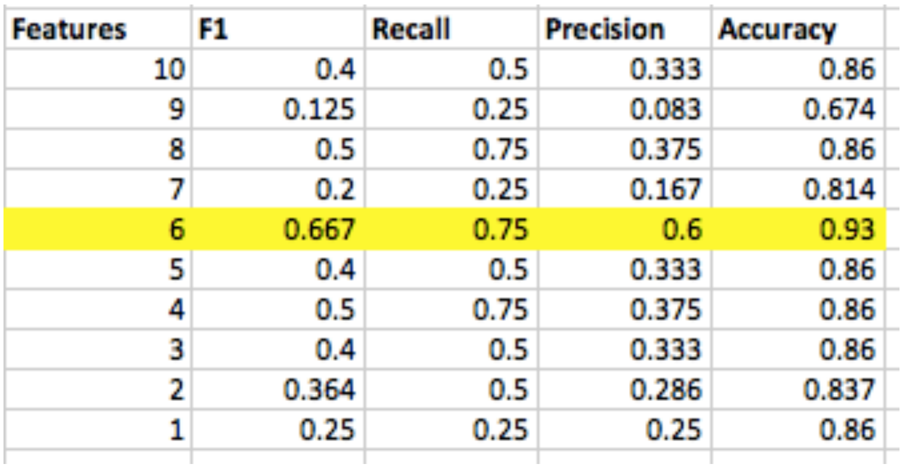
I also experimented with the SelectKBest tool, running the parameter of k between 1 and 10 to see which value worked best. I achieved the best results with five or six features depending on the method of validation. The top six feature scores were:

```
[14.145955349707767, 13.909428574104082, 12.719703253729154, 10.9001
25188189467, 10.636513552907001, 8.8050540798110113]
```

2. Algorithm Selection

Although I had early success with the Decision Tree algorithm, after more rigorous validation under Stratified Shuffle Split (sss), I achieved my best results with the Nearest Neighbor model.

Using a more standard train_test_split method of validation, and a SelectKBest tool to limit features, I got optimal results from the Decision Tree algorism at k = 6. Full results are given below:



Features	F1	Recall	Precision	Accuracy
10	0.4	0.5	0.333	0.86
9	0.125	0.25	0.083	0.674
8	0.5	0.75	0.375	0.86
7	0.2	0.25	0.167	0.814
6	0.667	0.75	0.6	0.93
5	0.4	0.5	0.333	0.86
4	0.5	0.75	0.375	0.86
3	0.4	0.5	0.333	0.86
2	0.364	0.5	0.286	0.837
1	0.25	0.25	0.25	0.86

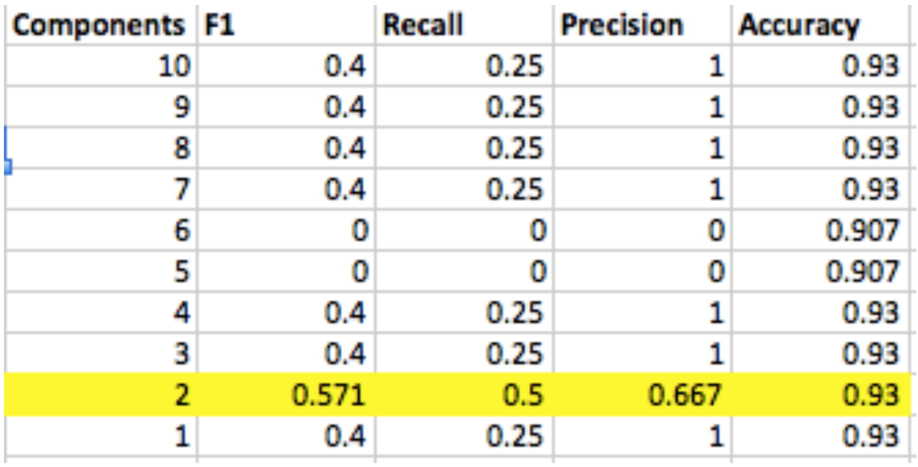
Unfortunately, using a Stratifed_Shuffle_split methods of validation with k = 6 fell just short of the results needed:

```
Accuracy: .809, Recall: .312, Precision:.294, F1: .303, n: 15,000
```

A Gridsearch cross validation, however, did improve the scores just pass the required .3 scores for precision and recall:

```
Accuracy: .815, Recall: .309, Precision:.309, f1: .309 n: 15,000
```

K nearest neighbor performed best on initial testing employing a Min_Max_Scaler and PCA with n_components at 2:



Components	F1	Recall	Precision	Accuracy
10	0.4	0.25	1	0.93
9	0.4	0.25	1	0.93
8	0.4	0.25	1	0.93
7	0.4	0.25	1	0.93
6	0	0	0	0.907
5	0	0	0	0.907
4	0.4	0.25	1	0.93
3	0.4	0.25	1	0.93
2	0.571	0.5	0.667	0.93
1	0.4	0.25	1	0.93

Although these results could not be replicated under further testing with Stratifed_Shuffle_split, the reduction in performance at 2 components was not as steep:

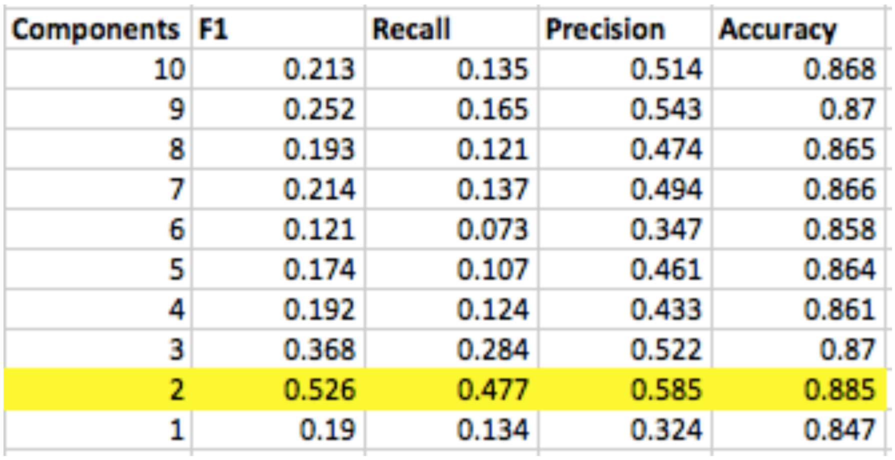
```
Accuracy: 0.885, Recall: 0.477, Precision: 0.585, F1: .526 n: 15,000
```

3. Parameters

Tuning the parameters of an algorithm is the process of iterating over multiple parameter inputs to find the optimal settings for an algorithm. Without good tuning an algorithm is prone to over-fitting, a case in which there are too many features in the training data for the algorism to form generalizations it can apply to new situations. When an algorithm is over-fitted, it exhibits high variance – it can only replicate what it has seen before. On the other hand, an algorithm might have too high a bias – where it misses important relations within the data and demonstrates a low capacity to learn anything at all. Tuning an algorithm is finding the optimal middle ground in this tradeoff between bias and variance.

Improving the decision tree algorithm under sss validation resulted changing the number of features (in SelectKBest) from 6 to 5 and the min_samples_split from a default of 2 to 3. It was interesting to me that for this algorithm the optimal number of features was different depending on the method of validation.

I tuned my K nearest neighbors algorithm to find the optimal number of datapoints to consider when classifying a new point in a coordinate system. As it turned out, the default value of 5 happened to perform better than any odd integer 1 through 9. However, this result was fortuitous and in no sense guaranteed. Also, for this algorithm, a pca with 2 components produced the best results under both the split method of validation and sss. Full results using sss methods of validation given below:



Components	F1	Recall	Precision	Accuracy
10	0.213	0.135	0.514	0.868
9	0.252	0.165	0.543	0.87
8	0.193	0.121	0.474	0.865
7	0.214	0.137	0.494	0.866
6	0.121	0.073	0.347	0.858
5	0.174	0.107	0.461	0.864
4	0.192	0.124	0.433	0.861
3	0.368	0.284	0.522	0.87
2	0.526	0.477	0.585	0.885
1	0.19	0.134	0.324	0.847

4. Validation

Validation is the process of dividing data into training and testing sets. An algorithm is tuned and fitted exclusively on the training data, its performance and capacity for generalization is then evaluated on the *not-before-seen* testing data. One classic mistake in validation is *information leakage* between these sets of data. If an algorithm is tested and trained on the same data, performance metrics will be over-stated and invalid.

The results above also demonstrate the importance of the method of validation. While employing a train_test_split method of validation is computationally friendly, the size and unbalanced character of the data largely undermine the validity of the results. This is why these results could not be replicated under the stratified_shuffle_split method adopted in my final algorithm.

5. Evaluation

```
Accuracy: 0.885, Recall: 0.477, Precision: 0.585, Test Size: 1500
```

The results of my final algorithm (K Nearest Neighbor with PCA), reproduced above, signifies that 88.5% of the time, the algorithm correctly identifies a person as a POI or non-POI. However, because there are more non-POIs than POIs, this accuracy score is not the whole picture. A recall of .477 means that when my algorithm is "shown" a POI, it will less than half the time recognize the employee as such.

Precision of .585 means that when my algorithm says “POI”, approximately 58% this is true.