Identifying Persons of Interest in the Enron dataset

Mohamed Khodeir

October 17, 2015

1 References

1. Kohavi, Ron, and George H. John. "Wrappers for feature subset selection." Artificial intelligence 97.1 (1997): 273-324.

2 Introduction

The goal of this project is to create a reliable model for identifying persons of interest (POI's) in the Enron dataset. POI's are those employees that have accepted a plea bargain, or been indicted. This is posed as a binary classification task over the set of 145 employees.

The original dataset contains all of the email exchanges between Enron employees. For the purpose of this project a small number of employee-specific statistics (e.g. the number of email exchanges with persons of interest) have been extracted from the email dataset and combined with financial data (e.g. salary).

2.1 Dataset

The resulting dataset contains around 145 employees with up to 20 features each, so the task will be to a create a model that uses the observed data for each employee to infer their POI status.

There is an inherent imbalance in this classification task since there are only 18 POI's. This has implications for the expected accuracy, as simply predicting 'Not POI' for all cases would give around 87.5% accuracy.

2.2 Outliers

Several outliers were detected in the initial exploration of the dataset. Two entries were removed from the dataset. The first containing the total of all financial fields, and the second seemingly completely unrelated (named 'THE TRAVEL AGENCY IN THE PARK'). Two other entries which were outliers on several different fields turned out to be the result of transposition errors; so these were re-entered from the findlaw source pdf.

3 Feature Selection/Engineering

3.1 Engineering

I added several features that are nonlinear combinations of given ones. This included the proportion of sent/received emails that came from known POI's, the proportion of emails received that were also addressed to POI's, and others less well-defined.

The feature selection problem was then to reduce this augmented feature set to a subset that does no worse at test-time.

3.2 Scaling

Some models are sensitive to the relative scaling of features. For example, K-nearest neighbours' distance metric will be dominated by very large features (e.g. bonus, total_payments).

Therefore I found normalizing the relative scaling of features was necessary for comparing performance accross models.

There was one complication when scaling test data. Namely, the test data could contain much more extreme values than the training data in some features. To address this issue, I applied a combination of the [0,1] scaler (with min and max values learned from the training data) with a sigmoid transform. The sigmoid transform is nearly linear in the range [0,1] so properly scaled data is relatively unchanged while values outside this range decay quickly towards either 0 or 1.

3.3 Selection

The choice of feature selection process was challenging. I considered univariate filtering methods (e.g. t-tests), which rank the relevance of each variable in isolation, as well as multivariate filtering methods (e.g. Decision Trees, Wrappers) which try to take into account the interaction between variables.

Wrapper methods methods, instead of selecting features as a separate preprocessing step, are applied as part of hyperparameter tuning on a given model.

The wrapper method that I consider (detailed in cited paper in references) is called Best First Search, and is a semi-greedy search method of forward feature selection guided by the performance of the model as estimated by cross-validation.

I expected different selection processes to be naturally compatible with different models.

There is a subtle point here about how feature selection methods are applied. If we use the entire dataset to select features, then we are directly optimizing over some of our test data and our generalization estimates will be optimistic.

For our purposes, this will be ignored since the tester evaluates the model on the entire dataset. However, if we wanted to be more rigorous (and had a more substantial dataset) we would perform the feature selection step within the cross-validation loop.

4 Evaluation Metrics

Raw accuracy provides little information in binary classification where there is a class imbalance. More fine-grained metrics are required.

Precision measures the proportion of correctly identified POI's to the total number of predicted POI's. (i.e. how likely is a person to be a POI if the model says he/she is.)

Recall measures the proportion correctly identified POI's to the total number of true POI's. (i.e. how likely is the model to correctly identify a true POI.) This is the same as the positive accuracy rate. (TPR)

Finally, the negative accuracy rate (TNR) is the proportion of correctly identified non-POI's to the total number of non-POI's.

For POI identification, a false negative might be somewhat more costly than a false positive, so we may want to optimize for recall over precision. However, predicting POI regardless of the data gives a recall of 1.0 so we need to also keep precision in mind.

5 Parameter Tuning

Model parameters refer to the quantities (such as mean and standard deviations for GNB) which represent the information learned by the model. These are computed on the training data by the model's learning algorithm.

Some models have "hyper"-parameters which can control the modelling capacity, complexity, or the speed of convergence during the training phase. The most common use of hyper-parameters is to reduce the potential for the model to overfit its parameters to the training data with the goal of improving the model's generalization to unseen test data.

One big mistake with hyper-parameter tuning is to fit the hyper-parameters to the same data used for training, which has the opposite effect of increasing overfitting, and reducing generalization on the test data.

A subtler mistake is to fit the hyper-parameters to the test data used for evaluation, which has the effect of indirectly overfitting the hyperparameters to the test set and causing us to overestimate the model's performance on really unseen test data.

6 Validation Strategy

I used cross-validation to estimate each model's performance. Because of the limited size of the dataset and the imbalanced distribution of POI status, sampling with replacement was used to produce a large number of datasets and stratified sampling was used to split these into training and validation sets. This functionality is combined into the StratifiedShuffleSplit class in sklearn.

Estimates of any metric can then be averaged over these datasets. This has the advantage of also giving an estimate of variance, so that comparison of hyper-parameter settings or different models is more robust.

This method addresses the big mistake mentioned above, but not the subtler one of overfitting to the test data. Ideally we would like to evaluate our models on completely held out data, however due to the scarcity of data in this case, the evaluation (in tester.py) uses the same dataset on which both training and validation are performed. This means our performance estimates will be optimistic.

7 Model Selection

7.1 Gaussian Naive Bayes (GNB)

The Gaussian Naive Bayes classifer models each feature's class-conditional distribution as an independent Normal with parameters (mean, std) estimated from the training data. The model is ill-suited for handling highly correlated features, so it is not expected to do very well on this task. It is used here as a baseline model. It has no hyperparameters.

7.1.1 Evaluation

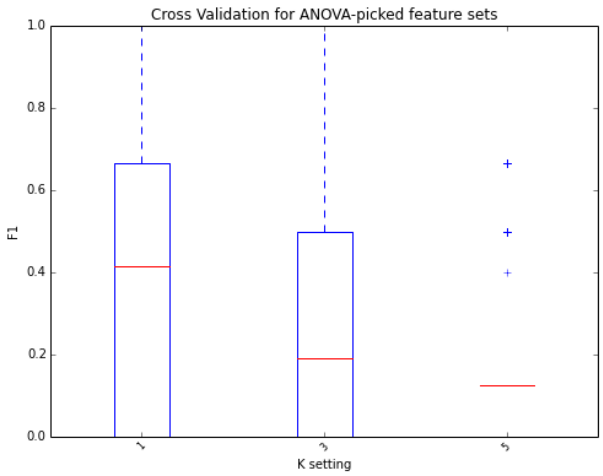
Table 1: Final evaluation (using tester.py) of each of the best obtained feature sets.

	# Features Selected	**Accuracy**	**Precision**	**Recall (TPR)**	**TNR**	**F1**
ANOVA	24 (ALL)	0.67967	0.23692	0.63150	0.68708	0.34456
BFS	8	0.86213	0.48113	0.43350	0.92808	0.45608

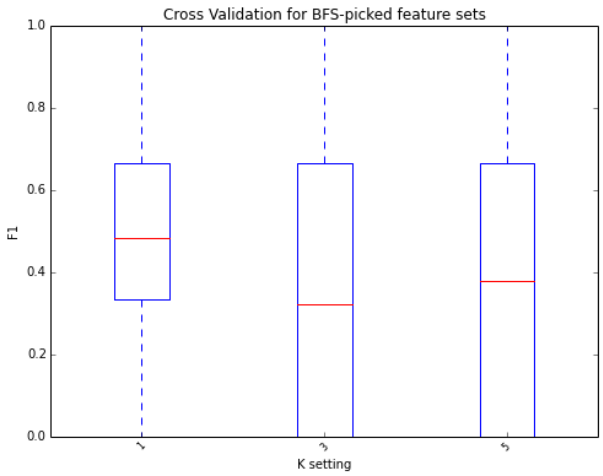
7.2 K-nearest neighbours (KNN)

The K-nearest neighbours model classifies data points based on the majority class for its closest neighbors in the feature space. The only hyperparameter that it has is the setting for k which controls the number of training data points used to estimate the class of for a test case.

7.2.1 Hyperparameter Tuning



(a) Different settings of K on ANOVA-picked feature sets.



(b) Different settings of K on BFS-picked feature sets.

Figure 1: Boxplots of F1 validation scores

Figure 1a shows the effect of setting K when selecting features using ANOVA. For each setting of K, the box shows the validation scores for the best subset obtained (as measured by mean F1). The complete results are in table 6 (which is in the appendix for the sake of brevity).

Likewise, figure 1b shows the effect of setting K when selecting features using the best first search wrapper approach using only 3-fold cv to estimate performance. The table 7 also shows the length of the obtained feature sets.

7.2.2 Evaluation

The results from hyperparameter tuning suggest the best setting of K to be 1, so only the single closest neighbor is used to predict a test example. Running the BFS wrapper again on this model with a more expensive search (StratifiedShuffleSplit with 300 iterations) produces an even better result. This is shown in table 2

Table 2: Final evalutation (using tester.py) of each of the best feature/hyperparameter combinations. (K=1)

	# Features Selected	Accuracy	Precision	Recall (TPR)	TNR	F1
ANOVA	24 (ALL)	0.88727	0.63060	0.37300	0.96638	0.46874
BFS	10	0.87627	0.53896	0.49800	0.93446	0.51767
BFS (more CV)	13	0.90093	0.64294	0.57800	0.95062	0.60874

7.3 Decision Tree Classifier (DTree)

Decision trees try to divide the feature space into class homogenous regions using by constructing a binary tree from the training data.

Decision trees have a built-in feature selection mechanism as part of their learning algorithm which uses information gain to guide a greedy selection of features.

Various hyperparameters control the size of the tree model (maximum depth, maximum features to use, etc.), as well as the misclassification cost per-class.

7.3.1 Hyperparameter Tuning

Table 3 shows the performance of the best hyperparameter settings when constraining the built-in feature selection of Decision Trees to only considering different numbers of features.

We see that performance generally increases with the number of features we allow. We also see that the best performing models all use a 5:1 cost ratio for misclassifying true POI's and they all have a maximum tree depth of 2, so are heavily regularized.

Table 3: Cross validation results for the best hyperparameter settings using different numbers of features.

Num Features	Class Weight	Max Depth	Mean F1	F1 Std.
1	{0: 1, 1: 5}	2	0.311785	0.24295
3	{0: 1, 1: 5}	2	0.377921	0.243593
5	{0: 1, 1: 5}	2	0.389283	0.24805
10	{0: 1, 1: 5}	2	0.39299	0.26053
15	{0: 1, 1: 5}	2	0.40983	0.253382
20	{0: 1, 1: 5}	2	0.415531	0.236952
24	{0: 1, 1: 5}	2	0.394014	0.21629

Table 9 in the appendix shows the performance of all of those hyperparameter settings under BFS feature selection wrapper. Interestingly, the best results are again using a 5:1 ratio and a maximum depth of 2, but the F1 score is significantly higher due to the removal of all but 2 features by BFS! The resulting tree can easily be drawn as in figure 2, and it might make for a useful rule-of-thumb for investigators.

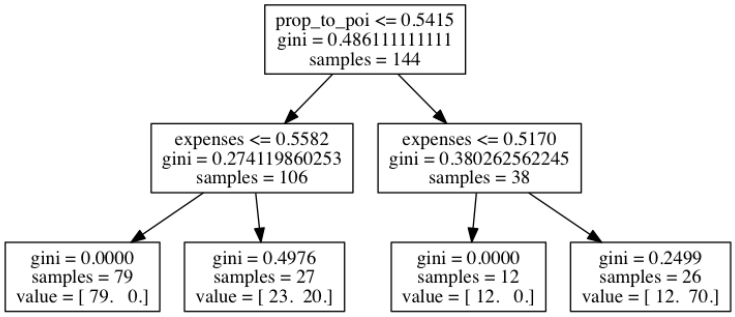


Figure 2: Visualization of the best decision tree model.

7.4 Evaluation

Table 4: Final evalutation (using tester.py) of each of the best feature/hyperparameter combinations (Max Depth = 2, Class Weight = 5:1).

	# Features Selected	**Accuracy**	**Precision**	**Recall (TPR)**	**TNR**	**F1**
Built-In	20	0.77133	0.31591	0.61350	0.79562	0.41706
BFS	2	0.86400	0.49283	0.68700	0.89123	0.51767

8 Conclusion

The KNN model with 13 features (chosen by BFS) and K=1 turns out to produce the best F1-score and best overall accuracy. With more specific domain requirements, we may be able to optimize for different metrics, but in this case, the KNN model produces the best overall classifier. This is the model which i have used in poi_id.py.

9 Appendix

9.1 GNB

Table 5: Full cross validation results for GNB feature selection using ANOVA.

Num Features	Mean F1	F1 Std.
1	0.303233	0.333306
3	0.334438	0.317345
5	0.297906	0.306209
10	0.320381	0.312564
15	0.285421	0.304204
20	0.354962	0.232852
24	0.368971	0.210408

9.2 KNN

Table 6: Full cross validation results for KNN hyperparameter tuning using ANOVA feature selection.

Num Features	K	Mean F1	F1 Std.
1	1	0.23478	0.264004
1	3	0.172467	0.281428
1	5	0.0684667	0.196715
3	1	0.353157	0.310312
3	3	0.348371	0.352423
3	5	0.300967	0.340049
5	3	0.162757	0.254256
5	1	0.122878	0.219711
5	5	0.0700524	0.196802
10	1	0.169362	0.253606
10	3	0.0656667	0.194334
10	5	0.0395667	0.156383
15	1	0.221795	0.287486
15	3	0.0539667	0.172922
15	5	0.00596667	0.0603781
20	1	0.207407	0.282396
20	5	0.1002	0.232966
20	3	0.0723333	0.199324
24	1	0.416257	0.342748
24	3	0.192438	0.293641
24	5	0.1244	0.257587

Table 7: Full cross validation results for KNN hyperparameter tuning using BFS feature selection.

K	Length of best subset	Mean F1	Std. F1
1	10	0.484879	0.314204
3	9	0.3224	0.332239
5	5	0.380233	0.350437

9.3 DTree

Table 8: Full cross validation results for Decision tree hyperparameter tuning and built in feature selection.

Max Features	Misclassification Costs	Max Depth	Mean Validation Score
1	{0: 1, 1: 1}	2	0.0781286
3	{0: 1, 1: 1}	2	0.148558
5	{0: 1, 1: 1}	2	0.186271
10	{0: 1, 1: 1}	2	0.184786
15	{0: 1, 1: 1}	2	0.168728
20	{0: 1, 1: 1}	2	0.13948
24	{0: 1, 1: 1}	2	0.11984
1	{0: 1, 1: 1}	8	0.257061
3	{0: 1, 1: 1}	8	0.280619
5	{0: 1, 1: 1}	8	0.263014
10	{0: 1, 1: 1}	8	0.256777
15	{0: 1, 1: 1}	8	0.230883
20	{0: 1, 1: 1}	8	0.208077
24	{0: 1, 1: 1}	8	0.204052
1	{0: 1, 1: 1}	16	0.274094
3	{0: 1, 1: 1}	16	0.270268
5	{0: 1, 1: 1}	16	0.276008
10	{0: 1, 1: 1}	16	0.2457
15	{0: 1, 1: 1}	16	0.240406
20	{0: 1, 1: 1}	16	0.206792
24	{0: 1, 1: 1}	16	0.204898
1	{0: 1, 1: 2}	2	0.175957
3	{0: 1, 1: 2}	2	0.281634
5	{0: 1, 1: 2}	2	0.294213
10	{0: 1, 1: 2}	2	0.340164
15	{0: 1, 1: 2}	2	0.347209
20	{0: 1, 1: 2}	2	0.35941
24	{0: 1, 1: 2}	2	0.353802
1	{0: 1, 1: 2}	8	0.256578
3	{0: 1, 1: 2}	8	0.287594
5	{0: 1, 1: 2}	8	0.278933
10	{0: 1, 1: 2}	8	0.287838
15	{0: 1, 1: 2}	8	0.280044
20	{0: 1, 1: 2}	8	0.297617
24	{0: 1, 1: 2}	8	0.308575
1	{0: 1, 1: 2}	16	0.270637
3	{0: 1, 1: 2}	16	0.278479
5	{0: 1, 1: 2}	16	0.26311
10	{0: 1, 1: 2}	16	0.269942
15	{0: 1, 1: 2}	16	0.272898
20	{0: 1, 1: 2}	16	0.29645
24	{0: 1, 1: 2}	16	0.308316
1	{0: 1, 1: 5}	2	0.311785
3	{0: 1, 1: 5}	2	0.377921
5	{0: 1, 1: 5}	2	0.389283
10	{0: 1, 1: 5}	2	0.39299
15	{0: 1, 1: 5}	2	0.40983
20	{0: 1, 1: 5}	2	0.415531
24	{0: 1, 1: 5}	2	0.394014
1	{0: 1, 1: 5}	8	0.281879
3	{0: 1, 1: 5}	8	0.288259
5	{0: 1, 1: 5}	8	0.3013
10	{0: 1, 1: 5}	8	0.310978
15	{0: 1, 1: 5}	8	0.334608
20	{0: 1, 1: 5}	8	0.335956
24	{0: 1, 1: 5}	8	0.348049
1	{0: 1, 1: 5}	16	0.259851
3	{0: 1, 1: 5}	16	0.261362

Table 9: Full cross validation results for Decision tree hyperparameter tuning with BFS feature selection.

Misclassification Costs	Max Depth	Length of best Subset	Mean F1	Std. F1
{0: 1, 1: 1}	2	2	0.276833	0.295714
{0: 1, 1: 1}	8	6	0.29461	0.263833
{0: 1, 1: 1}	16	4	0.367721	0.299026
{0: 1, 1: 2}	2	2	0.474983	0.277678
{0: 1, 1: 2}	8	9	0.401393	0.307837
{0: 1, 1: 2}	16	12	0.352874	0.277889
{0: 1, 1: 5}	2	2	0.490283	0.246631
{0: 1, 1: 5}	8	7	0.481477	0.29019
{0: 1, 1: 5}	16	7	0.448617	0.313566