Enron person of interest identification, rev.2, simplified but cleaner

Krasin Georgiev

1. Introduction

The objective of this project is to build a person of interest identifier (POI) based on financial and email data from ENRON corporation. For the purpose general data exploration and machine learning techniques will be used. Acceptable performance is precision > 0.3 and recall > 0.3 on unseen data. Most of the information in this report is based on Udacity Introduction to Machine Learning course materials [1].

2. Background

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives. A list of persons involved in the fraud case (individuals who were indicted, reached a settlement, or plea deal with the government, or testified in exchange for prosecution immunity - POI) is available.

3. Data Description and Exploration

3.1. Data and features

Input information consists of financial information tables and full text emails. Financial features are imported directly. Email features are generated from the emails. Finally we have labels for POI

- Financial features, numeric:

['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees']

- Email features
    - numeric:

['to_messages', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']

    - Email features, text:

['email_address']

- POI label, factor:

[‘poi’]

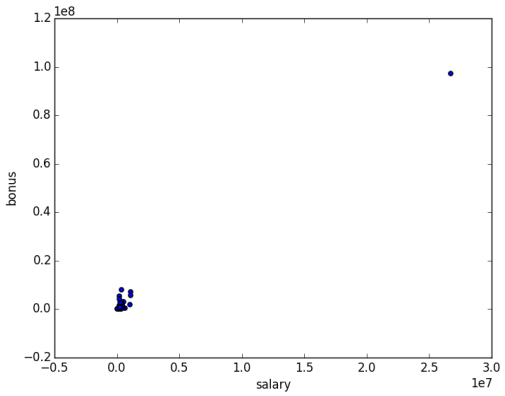
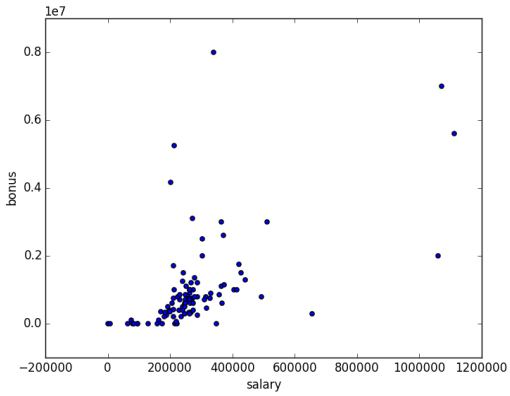
Therefore there are 19 initial numeric features and one label.

3.2. Basic statistics

Data were initially explored using summary statistics, scatter plots and a boxplot. The main ones are:

- total number of data points: total data points: 145
- number of POI: 18 (12%)

An obvious outlier is discovered. It is visible on the scatter plot of salary and bonus features - all data points are concentrated in a region far away from the outlier point.

Initial data lot	Plot after outlier removal
	

3.3. Outliers removal

From the raw data tables we can see that our data contain the 'Total' sum of all records. It is removed with the following line of code:

```
data_dict.pop( 'TOTAL', 0 )
```

Manual review of all record labels revealed a record which is not likely to be person - 'THE TRAVEL AGENCY IN THE PARK', which is also removed from the dataset with similar code.

4. Feature Creation and Selection

4.1. Assessment of available features

Classification using "salary" as the only one feature was used initially to achieve bug free code. Then classification with all starting features was performed with Random forest classifier.

Features	Precision	Recall	F1	Dataset
"salary"	**0.240**	**0.200**	**0.218**	**full, shuffeled, cv**
all	**0.31**	**0.14**	**0.19**	**full, shuffeled, cv**

Using all available features does not lead to improvement of the model and the performance is below the required level. Therefore smart selection of features and adding new features is needed.

4.2. Adding new features

New features were calculated based on email contacts relationships:

- email available, 'maildata' - accounts whether there is email information for the person under consideration.
- ratio of emails to POI and all send emails, 'to_ratio' - could happen to be more representative for the level of communication with POI than the absolute number of its own.
- ratio of emails from POI and all received emails, 'from_ratio' - the same logic for creation as for 'to_ratio'
- sum of above ratios, 'comm_ratio' - could be useful to merge both features if they have equivalent value for the classification
- communication with POI, topoi>60 and frpoi>60, 'comm' - this is non-linear feature in which we put a threshold of minimum number of exchanged emails above which we consider that something happens. Some experiments could be done with the thresholds.
- communication with POI, topoi>0 and frpoi>30, 'comm2' - the same as above with different thresholds
- total emails to/from POI, 'comm_sum' - we combine two features by summation. The same logic as for 'comm_ratio'
- max of emails to/from POI, 'comm_max' - we combine two features by taking only the maximum value
- min of emails to/from POI, 'comm_min' - we combine two features by taking only the minimum value

Also, transforming all features by taking logarithm was tested. No significant changes were recorded.

Features	Precision	Recall	F1	Dataset
all + created	**0.34**	**0.16**	**0.22**	**full, shuffeled, cv**

4.3. Feature selection

New approach in this second revision - systematically testing and recording top k features performance.

It is important to add 'poi' as first feature, otherwise the code will break.

The code used to select features are named ***features_*.py***.

4.3.1. Univariate feature selection

Univariate feature selection [2] - the effect of every single feature testing using SelectKBest with f_classif. Naive Bayes classifier then is used to estimate the performance of the best *k* features, for different values of *k* from 1 to 15.

Scores using f_classif, train set and full dataset

K	Score	SelectKBest, 50 % train set	Score	SelectKBest, fullset
1	0	bonus	0.000002	exercised_stock_options
2	0.000011	deferred_income	0.000002	total_stock_value
3	0.000052	salary	0.000011	bonus
4	0.000892	from_ratio	0.000035	salary
5	0.000989	long_term_incentive	0.000084	from_ratio

6	0.001113	total_payments	0.000113	comm_ratio
7	0.001305	exercised_stock_options	0.000922	deferred_income
8	0.001364	total_stock_value	0.001994	long_term_incentive
9	0.002724	comm_ratio	0.002863	restricted_stock
10	0.003587	shared_receipt_with_poi	0.003589	total_payments
11	0.004268	loan_advances	0.003946	shared_receipt_with_poi
12	0.009945	other	0.006089	comm_max
13	0.012084	restricted_stock	0.008232	loan_advances
14	0.0247	from_poi_to_this_person	0.014758	expenses
15	0.106774	comm_max	0.023514	from_poi_to_this_person

Classifier performance for increasing number of features (SelectKBest, k=1:14) (Naive Bayes classifier)

Subset						Full set					
k	Accur	Prec.	Recall	F1	F2	k	Accuracy	Prec	Recall	F1	F2
1	0.7882	0.58076	0.169	0.2618	0.1969	1	0.83418	0.60329	0.257	0.3604	0.2903
2	0.8195	0.60703	0.2765	0.3799	0.3103	2	0.84069	0.46889	0.2675	0.3407	0.2926
3	0.8333	0.5861	0.2825	0.3812	0.3151	3	0.843	0.48581	0.351	0.4076	0.3716
4	0.8379	0.52458	0.2935	0.3764	0.3219	4	0.84677	0.50312	0.323	0.3934	0.3479
5	0.8343	0.50418	0.332	0.4004	0.3563	5	0.84879	0.45558	0.3	0.3618	0.3220
6	0.838	0.39176	0.2425	0.2996	0.2625	6	0.84429	0.43894	0.3235	0.3725	0.3415
7	0.841	0.37443	0.287	0.3249	0.3011	7	0.84421	0.44195	0.3445	0.3872	0.3604
8	0.8437	0.39075	0.3085	0.3448	0.3221	8	0.84036	0.42822	0.3505	0.3855	0.3637
9	0.8384	0.37026	0.3025	0.3330	0.3140	9	0.83814	0.42017	0.35	0.3819	0.3621
10	0.8342	0.35685	0.3035	0.3280	0.3129	10	0.836	0.3636	0.3065	0.3326	0.3164
11	0.7756	0.24265	0.322	0.2768	0.3022	11	0.83087	0.34683	0.304	0.3240	0.3117
12	0.7733	0.23783	0.3175	0.2720	0.2976	12	0.82447	0.32956	0.306	0.3173	0.3104
13	0.7651	0.22688	0.3165	0.2643	0.2933	13	0.76927	0.2358	0.326	0.2737	0.3028
14	0.766	0.22822	0.317	0.2654	0.2941	14	0.77013	0.23692	0.326	0.2744	0.3032

best case, full dataset: k = 3, **['poi', 'exercised_stock_options', 'total_stock_value', 'bonus']**

best case 2, full dataset: k = 7, ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'from_ratio', 'comm_ratio', 'deferred_income']

best case, train set: k = 5, ['poi', 'salary', 'bonus', 'deferred_income', 'long_term_incentive', 'from_ratio']

4.3.2. Select k most important for Tree classifier

Similar procedure could be applied using Random Forests classifier. First features are sorted by importance. Then the performance of first k features is evaluated. Due to random nature of Random Forests if the experiment is run multiple times, different importance lists are generated. Example top features are as follow:

Selected, k=1: ['poi', 'bonus']

Accuracy: 0.75167 Precision: 0.4173 Recall: 0.2965F1: 0.34668 F2: 0.31472

Total predictions: 9000, True positives: 593, False positives: 828, False negatives: 1407, True negatives: 6172

4.3.3. Final list of features

The best performance is achieved with:

features_list = ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus']

5. Pick and tune an algorithm

5.1. Try a variety of classifiers

Random forests, Logistic Regression, Gaussian Naive Bayes and SVM were tested. Feature scaling was performed for logistic regression and SVM to check its influence on the results. Pipelines were used in order to do multi-stage operations, e.g. feature scaling first, applying classifier than [3].

The best set of features from section 4.3.3 was used:

features_list = ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus']

The script used to test different classifiers are provide final solution is in ***poi_classifier.py***.

Classifier	Precision	Recall	F1
RandomForestClassifier	0.590	0.302	0.399
GaussianNB	**0.486**	**0.351**	**0.408**
LogisticRegression	0.023	0.033	0.027
LogisticRegression, with feature scaling	0.635	0.210	0.316
SVC, no feature scaling	error	error	error

SVC(kernel="rbf", C=100), with scaling	0.161	0.052	0.078
SVC(kernel="linear", C=100), with scaling	0.639	0.111	0.189

We get best performance with Gaussian Naive Bayes algorithm, one of the fastest and simplest algorithms.

5.2. Tune the classifier

Some classifiers depend on multiple parameters which need to be set in advance. While default parameters are provided in sklearn implementation, they could not be the one that give the best performance for the specific dataset.

Because there are no parameters to tune for Gaussian Naive Bayes classifier we will try to improve the performance of the Random Forests. The idea is to test the model with different settings and select the one with best performance, i.e highest score. We select F1 as optimization metric, because it better reflect precision and recall than accuracy for skewed data.

The code used to tune classifier parameters is in ***poi_classifier.py***. It should be uncommented.

Classifier	Precision	Recall	F1
RandomForestClassifier() - no tuning	0.590	0.302	0.399
RandomForestClassifier(n_estimators=10, max_features=2, min_samples_split=8) {'n_estimators': [10, 100, 200], 'max_features': [None, 1, 2], 'min_samples_split': [2, 4, 8]}	0.576	0.335	0.423
RandomForestClassifier(n_estimators=50, max_features=2, min_samples_split=8) (manually adjusted)	0.613	0.344	0.440

From the table it is clear that adjusting model parameters leads to model improvement.

6. Test and evaluate the identifier

Validation is technique for assessing how the results of a statistical analysis will generalize to an independent data set.

To evaluate performance we use cross-validation routine provided by Udacity in test.py file. Precision and recall are used consistently through all analysis to evaluate algorithm performance. The precision can be interpreted as the likelihood that a person who is identified as a POI is actually a true POI; the fact that this is 0.49 means that using this identifier to flag POI's would result in over 50% of the positive flags being false alarms. Recall measures how likely it is that, given that there's a POI in the test set, this identifier would flag him or her; 35% of the time it would catch that person, and 65% of the time it wouldn't [5].

In case of classifier parameter tuning it is recommended to separate part of the dataset for checking the performance of models with different parameters so the best could be selected without seeing the test set [7]. We should have three subsets - train, validate and test set. The validation set is used to check model performance with different parameters. The test set is used to check the final model performance.

7. Summary and Conclusions

The main features of the model identified were [‘poi’, 'exercised_stock_options', 'total_stock_value', 'bonus']. This includes three financial features and no one reflecting email communication. Similar performance is provided by ‘'exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'from_ratio', 'comm_ratio', 'deferred_income'but with more features.

Selected classifier is Gaussian Naive Bayes with precision of 0.486 and recall 0.351. The other tested models couldn’t achieve better result even after some parameter tuning. Therefore the model could not be used on its own to make predictions. It will miss nearly 65 % of the targets. And the identified one will be true in only 50% of all cases..

Major limitation of the dataset is that there are only 18 cases of POI. “There were 35 people who were POIs in “real life”, but for various reasons, half of those are not present in this dataset--by the “more data is better than a finely-tuned algorithm” maxim of machine learning, having a larger dataset is likely the single biggest way to improve the performance of this analysis” [5].

Model could be improved to some extent if additional features are created based on more advanced processing of email corpus available. Another option is to do more extensive parameter tuning for Random Forest or SVM models, which is computationally intensive task and could be time consuming. In any case, overfitting is very likely.

Using validation subset will provide more accurate assessment of model performance but not so optimistic.

References

1. Udacity, Final Project Sample Report, https://www.udacity.com/wiki/ud120/finalproject
2. Scikit-learn documentation, Feature Selection, http://scikit-learn.org/stable/modules/feature_selection.html
3. Scikit-learn documentation, Pipeline and FeatureUnion: combining estimators, http://scikit-learn.org/stable/modules/pipeline.html
4. Grid Search: Searching for estimator parameters, http://scikit-learn.org/stable/modules/grid_search.html
5. Udacity, Enron POI Report Example.
6. Alex Brazhenko, Udacity Nanodegree. Project 4. Identifying Fraud from Enron Email, https://github.com/alexbra/ud-nd-project4/blob/master/poi_id.py
7. Scikit-learn documentation, Cross-validation: evaluating estimator performance, http://scikit-learn.org/stable/modules/cross_validation.html

Code snippets

```
################################################################################
# Classifier parameter tuning
# Grid search with pipeline

parameters = {'SVC__C':[10, 100, 10000, 1000000], 'SVC__gamma':[0, 1, 2, 4, 8,
100]}
parameters = {'svc__C':[100, 100000, 10000000], 'svc__gamma':[0, 2, 10]}

SVM = Pipeline([('scale', preprocessing.StandardScaler()), ('svc', SVC())])
clf = grid_search.GridSearchCV(SVM, parameters, scoring='f1')

test_classifier(clf, my_dataset, features_list)
print clf.best_estimator_

################################################################################
## Classifiers tested

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50, max_features=None)

from sklearn import linear_model
clf = linear_model.LogisticRegression()

from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()

from sklearn.svm import SVC
clf = SVC(kernel="rbf", C=10000)
clf = SVC(kernel="linear", C=10000)

from sklearn.pipeline import Pipeline, make_pipeline
from sklearn import preprocessing
clf = make_pipeline(preprocessing.StandardScaler(),
linear_model.LogisticRegression()) # big effect of normalization
```