Final Project - Module 5

1. Goal of the Project:

The dataset is a pre-selected dataset consisting of financial and email information on a larger group of ENRON employees. From these people a known subgroup were convicted for fraud - they are in this project referred to as Person of Interest (POI). It is the goal of the project to use the features given about the ENRON employees, how well we can predict if an employee is an POI or not. To answer this question a dataset of financial features and a dataset of email content is provided. Furthermore, a dataset with known POI (persons of interests) is given as a test set.

Start data

- Number of People in dataset: 146
- Number of POIs in dataset: 18
- Number of features: 20

Due to the existence of the POI list, this dataset is a labeled dataset, and we can therefore make use of supervised learning techniques in making an adequate look into who POIs are.

The reason for using a machine learning (ML) algorithm to do this job, is that ML is suited for making prediction on datasets where no clear demarcation line exists. Besides the label as POI, no other feature contains this information on who is POI or who is not. It is therefore something we must deduct from all the different data sources.

The number of POIs are only a small part of the full dataset, which makes the dataset unbalanced. With an unbalanced data set the often-used metric of Accuracy is not very useful, since we can have a high accuracy without finding any POIs; therefore other metrics are needed – in this project Recall and Precision. The unbalanced dataset also gives issues with splitting into test and training sets, since we can easily split the dataset into uneven sets – this is countered by using StratifiedShuffleSplit which averages on multiple test and training sets.

Furthermore, the full dataset is relatively small - only 146 persons. This is an issue for getting a good result from our ML algorithm since we have very few data points to calculate our result. This means that we need to make the features we pick have as much significance as possible since even small noise can distort our result, furthermore it becomes important to wean out any outliers, since just a single one can have a large impact on the end result. Finally issues on test versus training sets from the small dataset is also countered by using the StratifiedShuffleSplit.

2. Preparing Datasets

First, I ran through the different financial features to see if anything showed up that looked interesting, to find outliers (and decide upon keeping or removing), and in general clean the data.

Features

A function was setup for checking how many NaN values where present, and I created an overview of how high a percentage had values for All compared to just the POIs. With these numbers I looked for features that had few values for both Non-POIs and POIs. Based on these numbers I removed three features ('director_fees', 'loan_advances', 'restricted_stock_deferred) - but after running the GridSearchCV I found it caused a slight decrease in both precision and recall, and I therefore did not remove them for the final run.

I also looked at a correlation calculation for each feature against the others, and based on this I removed three features that had a high correlation with other features ("other", "to_messages", and "restricted_stock"). The reason for removing similar features is to avoid that they skew the result with unecessary datapoints.

New features

I also created and added a few new features. I selected the numbers of emails from and to POIs and calculated the fractions for how many emails from and to POIs and also the fraction of all messages to and from - wondering if POIs displayed an increased rate of receiving or sending emails in general.

The three new features increased the metrics. Especially the two features: 'fraction_to_poi' and 'fraction_to_vs_from' combined:

Before new features

- Accuracy: 0.67860
- Precision: 0.25465
- Recall: 0.31500
- F1: 0.28163

After adding the two features 'fraction_to_poi' and 'fraction_to_vs_from':

- Accuracy: 0.66670
- **Precision: 0.32746**
- **Recall: 0.63350**
- **F1: 0.43191**

With these changes (NB: the numbers achieved above was not possible in the final run due to manipulation with datapoints), the new features have proved to be a huge improvement for the dataset.

Outliers

Then I tried to look at individual people in the dataset, and checked three features: 1) partly manually looking for outliers by plotting and spotting high values, 2) checking NaN values for selected features (in particular Salary), and 3) finally checking if the name was just a one word string.

Based on the results I contemplated removing several persons – but due to the low number of datapoints, I kept everyone except non-persons ('TOTAL', 'THE TRAVEL AGENCY IN THE PARK').

I also did not apply any regression to remove outliers over a certain threshold, since the POI data often was extreme and therefore could happen to be taken out.

After sorting out people and features:

- Number of persons: 144
- Number of POIs: 18
- Number of features: 18

Final list of features:

['poi', 'salary', 'deferral_payments', 'total_payments', 'bonus', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'long_term_incentive', 'restricted_stock', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi', 'fraction_to_poi', 'fraction_to_vs_from', 'fraction_from_poi']

3. Final GridSearchCV / Pipeline

For selecting the features two steps was used in the gridsearch: 1) SelectKBest and 2) PCA

SelectKBest is used to select the best suited algorithms - according to the Sklearn documentation, it uses f_classic algorithm (based on a ANOVA F-values) to compare the features and then selecting the top K number of features.

In the gridsearch different ranges was tested from 4-19, before a final of 6 different features was selected:

The final features selected by SelectKBest is scored as follows:

- 'bonus', '30.729'
- 'salary', '15.859'
- 'fraction_from_poi', '15.838'
- 'fraction_to_vs_from', '10.723'
- 'total_stock_value', '10.634'
- 'exercised_stock_options', '9.680'
- 'total_payments', '8.959'
- 'deferred_income', '8.792'
- 'from_poi_to_this_person', '8.058'
- 'restricted_stock', '7.555'

PCA - principal component analysis - is similarly working with features, but looks at the features to see if they can be clustered and thereby provide a better fit. The PCA in my GridSearchCV pointed out that the features could be combined into 2 principal components.

Scaler

The data points where all send through a scaler to make the distances between values between different features have the same scale. The scaling is important to avoid that features with a scale of very high data points does not affect features with a scale of low data points.

Not all classifiers would need a scaled input, for example would a scaled input not be reflected in the output of a Naïve-Bayes algorithm, while other like SVC or KNN will give very different results with or without scaling. It should be noted, that in cases where the features are related measurements it can be beneficial to avoid a scaling.

The StandardScaler() was selected from SKLEARN since it had an improved performance compared with MinMaxScaler() for the SVM classifier.

Classifier Selection

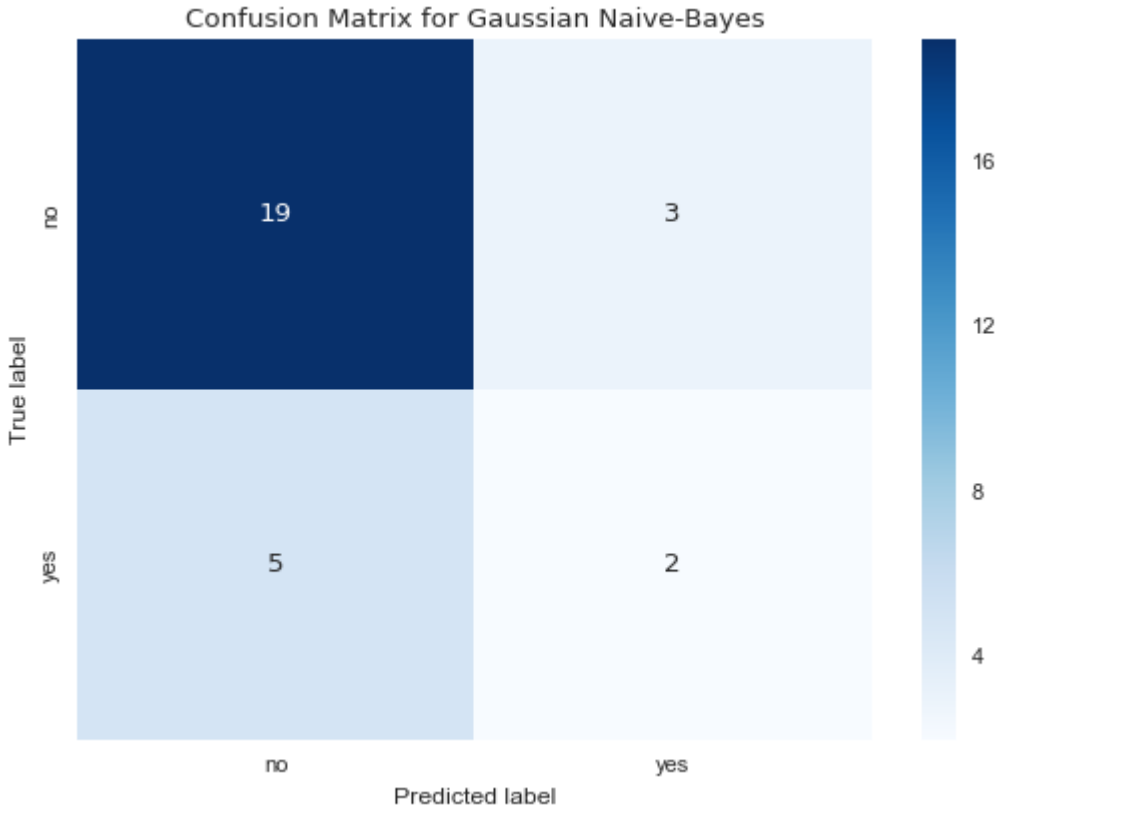
A range of classifiers was tested with different parameters and setups (with and without scalers, SelectKBest, PCA etc.). It took quite a while to get the GridSearchCV to run with a usable output and also get the tester.py to run correctly.

Before I started the GridSearchCV I ran some quick test runs with different classifiers which I scored on accuracy. The ones I tested ran from .65 - .75 in accuracy, where the SVC scored the highest:

Accuracy for classifiers:

- Decision Tree 0.655172413793
- **Support Vector Machine 0.758620689655**
- Gaussian Naive-Bayes 0.724137931034
- Random Forest 0.689655172414
- AdaBoost 0.724137931034
- GradientBoosting 0.724137931034

The accuracy is not a good metric for this dataset, since it is unbalanced (see under section "Metrics"), so it was only used as a preliminary test of the classifiers. A test with a confusion matrix to see how they fared in getting the POIs predicted was further more done, example here for Gaussian Naïve-Bayes, which had a low score on true positives:



All the classifiers fared equal or worse than GaussianNB in the confusion matrix – so I brought all classifiers into the GridSearchCV to tweak the parameters and get a proper validation.

Pipeline

For the pipeline in GridSearchCV I initially tested several classifiers in my pipeline; where I tweaked the parameters, and changed feature-set and data points to guide the precision and recall upward. I found that consistently the SV classifier ran with the highest scores, and it therefore became the one selected:

- Naïve-Bayes: Precision: 0.35041 Recall: 0.19150
- DecisionTree: Precision: 0.32196 Recall: 0.22650
- KNearestNeighbour: Precision: 0.23293 Recall: 0.22350
- AdaBoose: Precision: 0.27549 Recall: 0.23100
- RandomForest: Precision: 0.15789 Recall: 0.00450
- **SVC** **Precision: 0.34345** **Recall: 0.59400**

For the SVC I iterated several times back and forth between adding features, datapoints and then the setup of the pipeline parameters.

The precision and recall changed after deciding not to exclude so many datapoints from my dataset.

Parameter tuning

The parameters I found the greatest benefit of adjusting for the SVC was as follows:

- Kernel ‘poly’ – gave a higher precision than any of the others.
- 'class_weight' - which when set to value 'balanced' gave a boost to both recall and precision; it is a parameter which tones down more often occurring classes.
- ‘degree’ – ended up through gridsearch to be 2
- ‘C’ – which is giving a penalty for -and the gridsearch picked the value ‘3’
- ‘max_iter – which is the number of iterations that the classifier runs through -with the chosen value of ‘-1’ there is no limit

Adjusting the parameters was made easy with gridsearch, though I can see that some guides suggest that too much parameter fiddling can give an algorithm that is overfitted, so it has to be done with moderation.

Errors

I spend some time going back and forth with a problem that I could get higher results by handpicking parameters instead of letting the gridsearch do it – but I found out that the issue was due to me looking at the F1 score just for the POIs, and not the average F1 score.

Validation

Validation is needed to make sure our model is generalizable and is not overfitted for the data; which means that the model is over-adjusted to predict the training data. The validation of the data is done through splitting all data into a training and a test set. This also prevents having a mixed set of data when testing the algorithm.

I used the Sklearn model.selection module which handles the split, where you provide the percentage of the data that should be in the test part. I selected a partition of 20 percent due to good results in the final classifier.

For the final GridSearchCV pipeline I furthermore used the StratifiedShuffleSplit which also acts as a cross validation by being able to iterate through several random splits of the data over and over, and then providing the average from all these iterations.

The StratifiedShuffleSplit is of good use for our little dataset, since we avoid biases due to a bad initial split and also avoid the overfitting.

Metrics

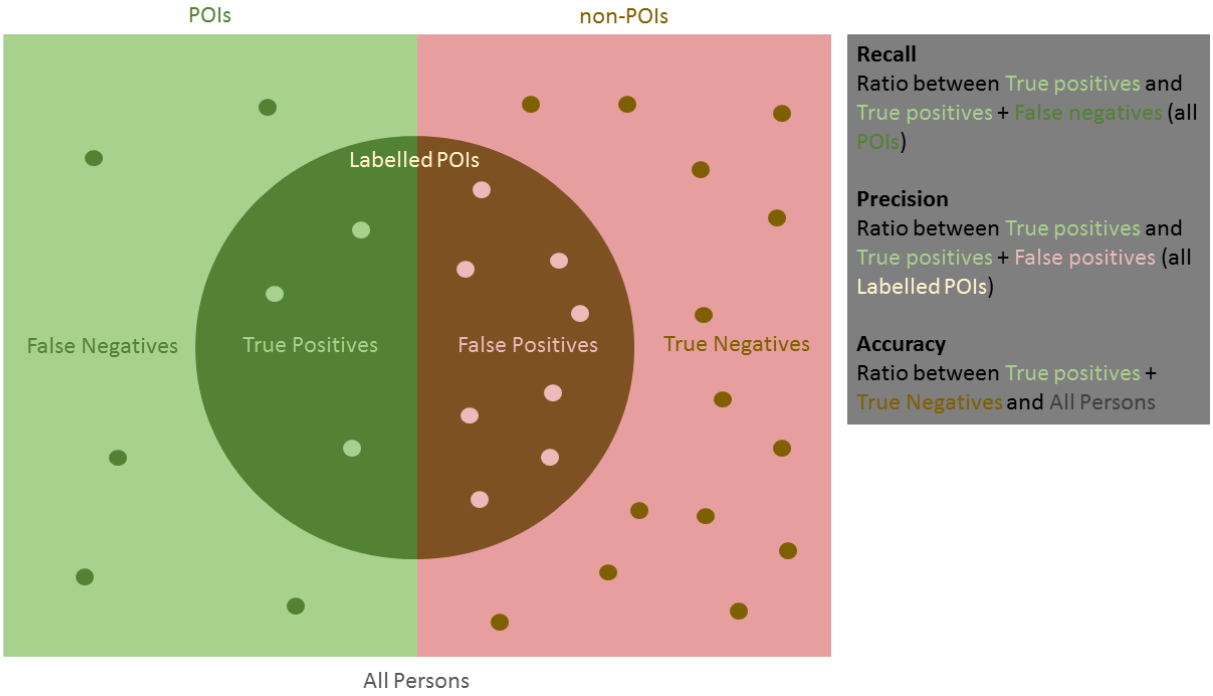
To measure the result I used the built in function from library SKLEARN metrics to see the precision, recall and f1 score. These scores I used as guidance, and then I finally also ran the tester.py to be sure the project could pass the 0.3 F1 score.

The final scores where in my last run:

Precision: 0.33636 **Recall: 0.40800** **F1: 0.36873**

The precision is the ratio of correctly labeled POIs compared to the full amount of POIs labeled by the algorithm, while Recall is the ratio of how many POIs was labeled POIs out of all the POIs existing in the dataset.

Since the Precision was just above the threshold it means that my setup has quite a high uncertainty on the data points it picks to be POIs - almost 70 percent are wrongly picked, while the Recall is a bit higher.



This means my algorithm would pick many innocent people (or at least if we follow the provided POI list).

The F1 display a balanced combination of Recall and Precision.