Udacity - Intro to ML

Identifying persons of interest in Enron dataset

Anirudh Ramesh

Brief :

The goal of this project is to use ML algorithms from the python-[sklearn] library and identify Persons of Interest (POIs) in the given enron dataset - with *Precision* and *Recall* >= 0.3.

Understanding of Dataset and Questions :

The dataset had 146 datapoint with 21 features.

POIs : 18 ('poi' attribute set to 1).
Non-POIs : 146 ('poi' attribute set to 0).

If data about a feature was unavailable , it was denoted by 'NaN' string for any person in the dataset. The given features fit into two categories - email-related (metadata , ex. Total number of emails , number of emails to POI) or financial (bonus , salary , stock options etc.)

Removing outliers :

There were few outliers I noticed in the dataset - 'TOTAL' - which is probably a spreadsheet quirk overlooked during munging; After that , I was looking into the datapoint which had more number of missing ('NaN') features - particularly , greater than 80% missing). (refer - 'outlier_identification.py')

(Name , Missing features %) # ('WODRASKA JOHN', 85.0) , # ('WHALEY DAVID A', 90.0) , # ('CLINE KENNETH W', 85.0) ,	(Name , Missing features %) **# ('LOCKHART EUGENE E', 100.0) ,** **# ('THE TRAVEL AGENCY IN THE PARK', 90.0) ,** # ('SCRIMSHAW MATTHEW', 85.0) ,

# ('WAKEHAM JOHN', 85.0) , # ('WROBEL BRUCE', 90.0) , # ('GILLIS JOHN', 85.0) ,	# ('SAVAGE FRANK', 85.0) , # ('GRAMM WENDY L', 90.0)

I removed 'THE TRAVEL AGENCY IN THE PARK' since I don't see how this might correspond to a 'person' and 'LOCKHART EUGENE E' since this datapoint had all features missing! [Experiments → outlier_identification.py]

I was worried if few of these points might skew the results but I decided against removing other data points in the list ,since I later found out that keeping them provided better results anyway (Also, due to the size of the dataset!)

Optimize Feature Selection/Engineering :

For this project , I decided to use 3 algorithms - which I found were especially well-suited for binary classifications ,after trying a few which produced less than spectacular results. I ended up using :

1. ExtraTreesClassifier
2. LogisticRegression.
3. Linear-SVM.

I decided to proceed with the problem at hand iteratively.

Iteration 1 :

For the initial iteration, I decided to use the given set of features extracted from the e-mails. The results were very much below the threshold of **0.3 precision and recall.** [Experiments -> Given set of features folder].

Iteration 2 :

I created three new features :

1. ratio_to_poi
2. ratio_from_poi
3. ratio_shared_receipt

I tried to classify using these three features but the results were very poor i.e the features by themselves weren't really helpful. Therefore, I decided for my second iteration , I'll try and classify the dataset by using these three features + given set of features [Experiments → 1.email_ratios+given_data]. The idea was that people who are *POIs* might have higher ratio_to_poi and ratio_from_poi since they have communicate more with people who were also in on the fraud attempt.

Iteration 3 :

I created log of features that are related to finances and tried to classify using log(finances) + email-features from the given set(Ignoring financial features from the given set)
Logarithmic transformation should give a good insight on how well the financial aspect of *POIs* increase exponentially(since POIs have higher deferred income, stock options etc.). [Experiments → 2.log_financial_data+emails].

Iteration 4:

I decided to use all the features I had created thus far and used :
log(financial features) + ratios from iteration 2 + given set of features. (both financial and emails) to see if this would improve precision and recall.[Experiments → 3.log_financial+email_ratios+emails].

Iteration 5:

I normalized the financial features using financial features/salary. The idea is to get an insight as to how features vary for a person getting a salary 'X', between POIs and non-POIs. [Experiments → 4.normalized_finance +emails].

Iteration 6 :

I decided to use the normalized financial features + given set of features. , since using only normalized features resulted in poor precision and recall[Experiments → 5.normalized_finance + emails_ratio + emails].

Iteration 7 :

I used normalized financial features + given set of features + ratios of email data (that I had created in iteration 2). [Experiments → 6.normalized_finances+emails_ratio + emails + finances].

Final iteration :

I found that the logarithm transformation produced good results but the recall was pretty low. When I started using normalized features in iteration 5 through 7, I found they had very good recall but very bad precision - one of the reason was only 2 of the normalised features had higher probability score. Therefore, I decided I will use two of these normalized features (‘Bonus’ and ‘deferred_income’) along with log(financial features) + given set of features + email-ratios (from iteration 2). This produced better results. [Experiments → final_iteration].

For selecting features I tried Tree-based features selection as well Select-K-Best method. I found that Select-K-Best was giving me better Accuracy and f1-score overall , therefore I decided to use SelectKBest in all the final algorithms.

I did not use parameter tuning for feature scaling - because the default values of parameters being used were :

1. Copy = [False , True]
2. Range = (0,1) which were the values I wanted to test. So I left it untouched.

Pick and Tune an Algorithm :

For this project, I decided to use 3 algorithms.

1. When using **ExtraTreesClassifier**, I tuned the following parameters :

* n_estimators - [5 , 10 , 15]
* criterion = ['gini' , 'entropy']
* min_sample_split = [2 , 3 , 4 , 5 , 10].

2. For logistic regression :

* max_iter = [100, 200 , 300 , 500 , 1000 , 10000]
* penalty = ['l1' , 'l2'] (liblinear uses both l1 and l2 , liblinear with dual uses only l2 and newton-cg , sags , lbfgs use only l2).
* solver = ['liblinear' , 'newton-cg' , 'sag', 'lbfgs'].

* fit_intercept : [True , False]

3. For linear-SVM :

* loss = ['hinge','squared_hinge']
* max_iter = [100,200,500,1000,10000]
* tolerance = [1e-2 , 1e-4 , 1e-6 , 1e-8 , 1e-10],
* multi_class = ['ovr' , 'crammer_singer'] .

I ran these all the three algorithm with and without [Principle component analysis](#) - with all the variation of the parameter tuning.

Parameter tuning in machine learning is really useful to optimize the performance. A given algorithm may perform very much different(for a given dataset) by changing the 'hyper-parameters'. For ex. We can set the learning rate of the machine learning algorithm, and set penalty for overshooting/under-reaching the optimum value, set a tolerance value etc.

By parameter tuning I get to have the best possible metrics(which I validate) to judge the performance of the algorithm.

I used sklearn-pipeline and sklearn-gridsearchcv for this purpose. I ended up using the best value from the parameter tuning across all the three algorithm in the final submission (which I verified using validation).

Validate and Evaluate:

For this project, the metrics for validations were Precision and Recall.
Precision is the 'positive predict' i.e given that a datapoint is +ve , does the algorithm predict it to be so. Recall is the 'sensitivity' i.e given that the algorithm says a datapoint is +ve, what is the chance that it actually is.
I required a precision and recall of >= 0.3.
For this purpose, I split the data using Stratified-Shuffle-Split - since the number of POIs is fewer compared to Non-POIs and I wanted a even distribution of POIs and Non-POIs in training and testing dataset to get a good classification (Instead of Stratified split or simple test-train split , which splits at random boundaries). I used 30% of the data for testing and rest for training.

---

* All the result from the experiment are performed for **split value of 5,**(n_split = 5), therefore there might be small variations in the final result value depending on the number of n_splits used in tester.py or any other such files.

* Results in 'Enron Dataset - ML results.pdf'.

* The 'best' results from various test are commented in the file under the corresponding directories at the very bottom.

Addition mini-project : Text-based classification :

I downloaded the body of the emails from the Enron dataset and performed text-based classification on the emails using Count-Vectorizer as well as TfIdf transformer. I got an accuracy of 50% when the dataset had equal amount of POIs and Non-POIs. When the ratio of POIs to Non-POIs was 1:3 or close , I got 25% accuracy. Anything beyond 30 points in dataset failed to converge and produced 0% accuracy. This is because the sampling of data I use has very skewed distribution of POIs and Non-POIs.

Refer : get_poi_names.py

* I did not use tolerance parameter for logistics regression as it often resulted in convergence error.
*All the values of precision and recall are for sampling of **FIVE** in stratified shuffle split. There might be small variations in value.