**Enron Submission Free-Response Questions**

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.  As part of your answer, give some background on the dataset and how it can be used to answer the project question.  Were there any outliers in the data when you got it, and how did you handle those?  [relevant rubric items: “data exploration”, “outlier investigation”]

The source dataset is the enron emails corpus which contains emails from Enron and financial data by person. The objective of the project is to go through emails and financials of those persons and try to identify Person Of Interest (POI). A POI is someone who was charge, convicted or link (settlement or testification) to fraud. Going through this amount of data manually would be very long, problematic and prone to error. Using Machine Learning, we can read those data and try to infer behavior from it.

As we want to identify POI, we need to select the right email and financial features to get the best results possible. Going through the dataset, we can see that a TOTAL is present. This is an omission and is removed by hand. I also removed ‘THE TRAVEL AGENCY IN THE PARK’ which was a good indicator but is not a person. All persons were also check for emptiness conducting to the removal of LOCKHART EUGENE E.

In the end, 143 data points were used.

What features did you end up using in your POI identifier, and what selection process did you use to pick them?  Did you have to do any scaling?  Why or why not?  As part of the assignment, you should attempt to engineer your own feature that doesn’t come ready-made in the dataset--explain what feature you tried to make, and the rationale behind it.  (You do not necessarily have to use it in the final analysis, only engineer and test it.)  If you used an algorithm like a decision tree, please also give the feature importances of the features that you use.  [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]

I created the following new features, but did not end up using them:

* Exercised stock ratio = exerciced stock/total stock. I was thinking that fraudulent people would have exercised their stocks when knowing that everything was over
* Sent to POI ratio: message to POI/total messages sent. The idea here was that POI would send more messages to POI

As my chosen algorithm is RandomForectClassifier, features scaling is not necessary. During algorithm testing, I used PCA and RandomizedPCA to see if dimensionality reduction could help (it did not).

I first used all the variables except the totals (as they are just sum of other variables) and the to\_messages and from\_messages (I do not see how those are important for the classification). I then set up a pipeline using SelectKBest to feed my classifier. SelectKBest select the K best features from a list depending of their score.

Scores and features importance were the following:

|  |  |  |
| --- | --- | --- |
| Features | SelectKBest Score | RFC features importance |
| 'salary' | 18.28968404 | 0.10942848 |
| 'bonus' | 20.79225205 | 0.0730103 |
| 'deferral\_payments' | 0.22461127 | 0.00547413 |
| 'loan\_advances' | 7.18405566 | 0. |
| 'restricted\_stock\_deferred' | 0.06549965 | 0.00289266 |
| 'deferred\_income' | 11.45847658 | 0.03517199 |
| 'expenses' | 6.09417331 | 0.14233204 |
| 'exercised\_stock\_options' | 24.81507973 | 0.03311027 |
| 'other' | 4.18747751 | 0.17217012 |
| 'long\_term\_incentive' | 9.92218601 | 0.02205841 |
| 'restricted\_stock' | 9.21281062 | 0.08540041 |
| 'director\_fees' | 2.1263278 | 0. |
| 'from\_poi\_to\_this\_person' | 5.24344971 | 0.05503196 |
| 'from\_this\_person\_to\_poi' | 2.38261211 | 0.07362909 |
| 'shared\_receipt\_with\_poi' | 8.58942073 | 0.12608925 |
| 'deferral\_ratio' | 1.36043361 | 0.01560276 |
| 'deferred\_stock\_ratio' | 0.76477268 | 0. |
| 'sent\_to\_poi\_ratio' | 4.09465331 | 0.04859813 |

Seeing the 0 for features importance, I decided to remove those features and the very low ones (like restricted\_stok\_deferred) before further testing.

The final features used and the SelectKBest scores as well as the RFC features importance are listed below :

|  |  |  |
| --- | --- | --- |
| Features | SelectKBest Score | RFC Features Importance |
| 'salary' | 18.00373999 | 0.13683818 |
| 'bonus' | 20.52464518 | 0.08416222 |
| 'deferred\_income' | 11.32148678 | 0.02467169 |
| 'expenses' | 5.95454429 | 0.22153078 |
| 'exercised\_stock\_options' | 24.53272246 | 0.04152406 |
| 'other' | 4.1288734 | 0.14819085 |
| 'long\_term\_incentive' | 9.77210354 | 0.03645776 |
| 'restricted\_stock' | 9.07907666 | 0.06100561 |
| 'from\_poi\_to\_this\_person' | 5.14221919 | 0.02704881 |
| 'from\_this\_person\_to\_poi' | 2.33883611 | 0.06419041 |
| 'shared\_receipt\_with\_poi' | 8.43263542 | 0.08610173 |
| 'deferral\_ratio' | 1.38320315 | 0.00713841 |
| 'sent\_to\_poi\_ratio' | 4.02040499 | 0.06113949 |

What algorithm did you end up using?  What other one(s) did you try? [relevant rubric item: “pick an algorithm”]

I used a randomforestclassifier. NBClassifier, as the default one gave low scores, but still a baseline. I then tried a SVMClassifier, but ended up with some errors because of the scarcity of positive instances in the dataset.

DecisonTree gave good results but needed to be strengthen. So I tried AdaBoost and RandomForest. Adaboost gave better recall but RandomForest seemed more consistent and balanced between precision and recall so I kept it and worked on getting the best I could from it.

Hereafter are the different scores I obtained with the different classifiers I tried:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classifier | Parameters | Accuracy | Precision | Recall |
| GaussianNB() |  | 0.809 | 0.262 | 0.236 |
| DecisionTreeClassifier | criterion='entropy',  min\_samples\_split=4, | 0.810 | 0.265 | 0.238 |
| AdaBoostClassifier | n\_estimators=100 | 0.836 | 0.347 | 0.254 |
| GradientBoostingClassifier | min\_samples\_split=4 | 0.848 | 0.378 | 0.213 |
| Pipeline(steps=[('features', SelectKBest()), ('classifier', RandomForestClassifier())]) | k='all', score\_func=<function f\_classif at 0x0000000017BEB0B8>  bootstrap=True, class\_weight='subsample', criterion='entropy', max\_depth=2, max\_features='auto', max\_leaf\_nodes=None, min\_samples\_leaf=1, min\_samples\_split=4, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=1, oob\_score=False, random\_state=None, verbose=0, warm\_start=True | 0.851 | 0.470 | 0.898 |

What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well?  How did you tune the parameters of your particular algorithm?  (Some algorithms don’t have parameters that you need to tune--if this is the case for the one you picked, identify and briefly explain how you would have done it if you used, say, a decision tree classifier). [relevant rubric item: “tune the algorithm”]

Algorithms are mathematical formula. Tuning a parameter is basically modifying the formula. Parameters can greatly influence scores. For example without tuning, DecisionTree overfit easily and quickly, basically learning the whole dataset.

Using RandomForestClassifier, I first set up a GridSearchCV with several parameters to test through 5 stratified folds:

* K : selection of features to retain when applying SelectKBest. Tested values were [5, 7, 10, 'all'],
* N\_estimators: number of trees in the forest. Tested values were [10, 50, 100, 200]
* Min\_sample\_split: minimum number of element to split a node; less than 3 implies leaves with only 1 element.
* Criterion: measure of quality of a split. Tested values were [‘gini’, ‘entropy’]
* Class\_weight: weight of each class. Tested values were [‘auto’, ‘subsample’, ‘None’]
* Max\_depth: maximum of level in a tree. Tested values were [2, 4, 6]
* Warm\_start: use the previous call to fit to add new trees or not. Tested values were [False, True]

The dictionary used for the GridSearch was the following:

{ 'features\_\_k': [5, 7, 10, 'all'],

'classifier\_\_n\_estimators': [50, 100, 200],

'classifier\_\_min\_samples\_split': [4, 6, 8],

'classifier\_\_criterion': ['entropy', 'gini'],

'classifier\_\_class\_weight': ['subsample', 'auto', None],

'classifier\_\_max\_depth': [2, 4, 6],

'classifier\_\_warm\_start': [False, True]}

The best classifier found was with the following parameters:

{'classifier\_\_class\_weight': 'subsample',

'classifier\_\_criterion': 'entropy',

'classifier\_\_max\_depth': 2,

'classifier\_\_min\_samples\_split': 4,

'classifier\_\_n\_estimators': 100,

'classifier\_\_warm\_start': True,

'features\_\_k': 'all'}

What is validation, and what’s a classic mistake you can make if you do it wrong?  How did you validate your analysis?  [relevant rubric item: “validation strategy”]

Validation is the process to vet a classifier or an estimator generalization capacity. Validation needs to be done against data not used during training. Training data is already known and in the end, the model will be used with unknown data. Validation is testing our model against an unknown set that we take from the training.

Cross-validation is to split the dataset into training and testing sets so that we can test our model during training by simulating an unknown dataset. Basic strategy is to split the original dataset into a training dataset and a testing dataset. Several other ways exists that generate more sets of data from the original dataset.

I used a StratifiedKFolds as a cross validation strategy. I chose a K of 5. This generates 5 sets of sets on which to validate the algorithm on. Each time, a different set is used as the testing data, the others being used as training data. So each point is used both for training (several times) and testing (once).

Stratified folds mean that the percentage of poi and non-poi is preserved for each fold so that I do not end up with a set without any poi.

Give at least 2 evaluation metrics, and your average performance for each of them.  Explain an interpretation of your metrics that says something human-understandable about your algorithm’s performance. [relevant rubric item: “usage of evaluation metrics”]

Several metrics exist to validate the results:

* Accuracy is the number of correct classification (poi or not poi in our case) divided by the number of data points.
* Recall is the percentage of relevant results returned
* Precision is the revelancy of the results

A high recall means that most of the positive labels are returned. A high precision means that most of the results labeled as positive are actually positive. Both high recall and high precision is to be aimed for. But we can have high recall and low precision, meaning the algorithm labeled a lot of point as positive but few actually are (high false positive). Or low recall and high precision when a few positive are actually returned but we can be quite confident with the results (high false negative).

I was mainly concerned with accuracy and recall as I wanted to identify as many POI as possible, the idea being that the list would be given to someone to investigate them properly.

My results on those metrics are:

Accuracy: 0.88013 Precision: 0.52978 Recall: 0.89850

Accuracy of 0.88013 means that 88.01% of the classifications were correct.

Precision of 0.52978 means that 52.98% of the person classified as POI are actually POI.

Recall of 0.89850 means that 89.85% of all POI were identified as such.

In context, those results can be translated as :When the model identifies someone as a POI, there is a 52.98% chance that this person is a POI. But at the same time this allows the model to identify 89.50% of all POI. The classifier identifies most of the POIs, only 10% do not show up in the results. But when looking at our results, we need to be careful as someone labeled as POI has a 47.02% chance of being a non-POI.