Karthik Rajasethupathy
Udacity DAND Project 5

Identifying POI's in Enron Dataset with Machine Learning!

Responses

1. 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 goal of this project is to predict whether or not an enron employee is a person of interest (POI) based on a set of financial/email data. The data consists of 145 points, and each point has 20 features and 1 outcome

Dataset
Data Points: 144 (after removing 2 outliers, see Outliers section below)
Dimensions: 21 (20 features, 1 outcome)
Missing Values: Every feature has missing values (see *Features* section below for more details)

Outcome
The outcome is POI. It is a binary attribute that is distributed as follows:

- True: 18 (12.5%)
- False: 126 (87.5%)

It's worth noting that the outcome is very imbalanced, and that should affect how we interpret our results later.

Features
For the remaining features, we have the following

Feature	Count	Missing Values (Count)	Missing Values (%)
salary	144	51	0.35

deferral_payments	144	107	0.74
total_payments	144	21	0.14
loan_advances	145	142	0.98
bonus	144	64	0.44
restricted_stock_deferred	144	128	0.88
deferred_income	144	97	0.67
total_stock_value	144	20	0.14
expenses	144	51	0.35
exercised_stock_options	144	44	0.30
other	144	53	0.37
long_term_incentive	144	80	0.55
restricted_stock	144	36	0.25
director_fees	144	129	0.89
to_messages	144	59	0.41
from_poi_to_this_person	144	59	0.41
from_messages	144	59	0.41
from_this_person_to_poi	144	59	0.41
shared_receipt_with_poi	144	59	0.41
email_address	144	34	.24

Some features, such as loan_advances, restricted_stock_deffered, and director_fees have over 80% of the values missing. This, by itself, does not imply that these values are not informative. For example, director_fees may only be available to “non-employee directors”, according to [1]. So, it would make sense for this value to be “NaN” for most employees. On the other hand, I would expect every employee to have an email_address, and to have to_messages or from_messages. For these features, the missing value may indicate an issue with data collection/availability. Currently, the *featureFormat* function handles string “NaN” values by imputing a value of 0 whenever this appears. I will not exclude any features just based on their missingness. We can use some feature selection techniques later on to see if any of these features are informative after imputing a value of 0 for the missing values.

Outliers

Originally, there were 146 data points. I removed 2 outliers:

- “TOTAL”: It is an outlier because it formed from aggregating other individuals data. So, we should not include this in the training data for our model.
- "THE TRAVEL AGENCY IN THE PARK”: This is clearly not an individual, but perhaps some business that Enron cooperated with it for arranging travel.

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 does not 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.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]

Feature Engineering

I engineered 3 features based on this intuition:

1. Employees that exercised more of their stock options may have been more likely to have been an “insider”, and been aware that the Enron stock would soon plummet. These individuals would likely be POI’s. So, rather use the absolute amount of exercised stock options, I’m looking for a feature that describes the ratio of exercised stock options to the total available for a given employee.
    a. total_stock_exercised_ratio* = exercised_stock_options/total_stock_value
2. Employees that had a higher ratio of their to/from messages with POI’s may be more likely to be POI’s themselves.
    a. from_messages_to_poi_ratio* = from_this_person_to_poi/from_messages
    b. to_messages_from_poi_ratio* = from_poi_to_this_person/to_messages

*In each of the created features, if either the numerator or denominator are “NaN” then the value of the created feature will be “NaN”.

Feature Selection

The model was constructed using a Pipeline. First, univariate feature selection method SelectKBest (trying values for k = 3, 5, 10, 15, and ‘all’) was applied, followed by the construction of a Decision Tree Classifier. Parameters were optimized using GridSearchCV. Given that our goal is to achieve at least .3 precision and recall, I set GridSearchCV was to choose the model with the highest f1 score (which weights precision/recall equally).

The model that achieved the best average f1 score on the cross validated data used:

- ‘All’ for the SelectKBest
- Gini impurity for the Decision Tree split criterion and ‘auto’ for max_features (works out to 4 features).

Below, I have made a table that lists the feature importances (using the gini impurity metric, and selectkbest score) for each feature, ordered by the gini impurity (descending). It is cool to note that one of the three features that I created, from_messages_to_poi_ratio, has the highest gini impurity score. One thing that's really interesting is that the total_stock_value feature, although it has the highest score for SelectKBest, seems to have the 2nd lowest gini impurity score.

Feature Scores

feature	Gini Impurity Score	Select K Best Score
from_messages_to_poi_ratio	0.19	16.41
bonus	0.17	20.79
salary	0.13	18.29
other	0.13	4.19
expenses	0.09	6.09
exercised_stock_options	0.08	24.82
from_messages	0.07	0.17
to_messages	0.06	1.65
shared_receipt_with_poi	0.05	8.59
from_poi_to_this_person	0.03	5.24
to_messages_from_poi_ratio	0.00	3.13
total_stock_exercised_ratio	0.00	0.04
from_this_person_to_poi	0.00	2.38
director_fees	0.00	2.13
restricted_stock	0.00	9.21
long_term_incentive	0.00	9.92
total_stock_value	0.00	24.18
deferred_income	0.00	11.46

Feature Scaling

I tried various scaling methods (RobustScaler, MinMaxScaler, StandardScaler). But, they were inconsequential because of the classifiers that I ultimately used (LogisticRegression and DecisionTreeClassifier). As we learned in the feature scaling part of the class, for logistic regression and decision trees, feature scaling can perhaps impact the rate of convergence for the underlying optimization algorithm, but does not necessarily improve the model. In logistic

regression, each feature has it's own coefficient to offset its scale. Decision trees divide a space into different regions. In both cases, it is not necessary to scale the features to achieve a better result. Had I tried a classifier such as SVM, or maybe a clustering based classifier such as KNN, it's likely that I would need to do some feature scaling first.

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]

As mentioned above, I used a DecisionTreeClassifier. I also tried a Logistic Regression. I found that the performance of the Decision Tree was better than that of the Logistic Regression in terms of the F1 score, precision and recall. Methods were tested against each other using the same Pipeline:

- First, SelectKBest is applied for Feature Selection
- Then, the classifier (Decision Tree or Logistic Regression) is constructed
    - For both classifiers, GridSearchCV was used for HyperParameter Optimization.
    - I discuss in detail what parameters were tried for the Decision Tree in the tuning section below.
    - For the Logistic Regression, I tuned the following parameters:
        - Penalty: l1 or l2
        - Regularization Strength: .01, .1, 1, 10
        - Fit Intercept: True, False

Based on the best model selected by GridSearchCV for each of these algorithms, I show the performance of the respective models on the test data (tester.py):

Algorithm (Only tuned parameters shown)	Precision	Recall	F1 Score
LogisticRegression (C=10, fit_intercept=True, penalty='l1', solver='liblinear')	0.40575	0.38100	0.39299
DecisionTreeClassifier (criterion='gini', max_depth=None, max_features='auto', splitter='best'))]	0.41294	0.39250	0.40246

The DecisionTreeClassifier performance slightly better than the LogisticRegression. The difference is very small and I acknowledge that a slight change in the tuned parameters or selected features could result in the Logistic Regression performing better.

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 do not 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 for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: "tune the algorithm"]

An algorithm, such as a Decision Tree, has many knobs that we can play with. Before we even build the tree, we can give different training data to the tree by picking different subsets of features and/or instances. Once the tree has the data, at each node of a Decision Trees, we choose which attribute to split on according to some criterion. Two popular choices are the gini impurity metric and shannon's information entropy metric. We can specify that the tree have a minimum support of X instances at the leaf level; is 1 instance enough? Do we need at least 10 instances to make a leaf? There are other criteria like this such as the depth of the tree, the max number of leaves, the minimum samples needed for a split, etc. Tuning the parameters of an algorithm essentially means that we pick the parameters of the Decision Tree that maximize some objective function. For our case, I would like to pick the model with the highest F1-score. So, I want to try all the different kinds of combinations of parameters for the tree, and pick the combination that yields the highest F1 score.

So, I am lucky to have access to Hyperparameter Optimization (HPO) function like GridSearchCV that allows to specify multiple choices for each parameter I want to tune, and will "find" me the combination of parameters that yield the best model under a specified objective function (f1 score).

Specifically, I used GridSearchCV to optimize my pipeline for k for SelectKBest (where k= 3,5,10,15,'all'), as well as 3 parameters for the decision tree: splitter ('best' or 'random'), criterion ('gini' or 'entropy') and max_features ('auto','sqrt','log2',None): this is a total of 80 different Decision Trees! I elaborate more in the following section, but the GridSearchCV was also passed a StratifiedShuffleSpilt parameter for cross validation.

The best tuned model returned the following choices for the optimized parameters:

- K = 'all' [SelectKBest]
- Criterion = 'gini' [DecisionTreeClassifier]
- Splitter = 'best' [DecisionTreeClassifier]
- Max_Features = 'auto' [DecisionTreeClassifier]

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 a critical part of the machine learning process in which a trained model is applied to data that it has *not* been trained on. There are many reasons to use validation. Many times, we may have built a model that performs very well on training data, but poorly on validation data. This model is called overfit. A more robust estimate of the models true ability is obtained from validation. Another thing that we should keep in mind when doing validation is class imbalance. For example, in the Enron case, the training data is 87.5% non-POI and 12.5% POI. We need to use a type of validation that maintains this distribution of labels in the validation data. Otherwise, our results on the validation may not be reflective of how the model will perform in practice.

I validated my model using the StratifiedShuffleSplit function with 100 splits, with train size at 90% and the test size at 10%. StratifiedShuffleSplit returns splits that preserve the class imbalance. However, unlike K fold cross validation, these splits may have overlapping instances, which implies that the splits are not disjoint sets (as is in the case of K fold cross validation). This technique is preferred to K fold cross validation for the Enron dataset because it is quite small.

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"]

The model is evaluated based on the F1 score, precision and recall. The F- score, is a class of metrics that compute a weighted average of precision and recall for a given Beta. For higher values of Beta (>1), recall is weighted more than precision. Alternatively, for lower values of Beta (<1), precision is weighted higher than recall. Since, we want to achieve at least .3 precision and recall, I set the GridSearchCV algorithm to pick the model with the highest F1 score (equally weights precision/recall).

My best model attained the following values for the evaluation metrics on the tester.py script:

Precision: 0.41294
Recall: 0.39250
F1: 0.40246

So, our classification task is to classify each individual as POI/Not POI. Given the metrics above, I can say the following :

- Precision: 41% of the time, when the model classifies someone as a POI, they are actually a POI.
- Recall: 39% of the all the actual POI's were classified as POI's by the model

If I was a data scientist working for the federal government when they investigated Enron, and we had access to this great dataset, I may chosen to pick an algorithm optimized a F - score with a Beta >1. This would favor choosing a model with a higher recall. My reason for selecting a model according to this criteria is that I wouldn't want to wrongly miss a POI. From my

perspective, it is okay and understandable if there are lots of false positives (incorrectly identifying an innocent employee as a POI) and many people at Enron are under suspicion of illegal activities. However, it would be more unforgivable to the public if **not all** the POI's are held accountable for their actions. Therefore, I would choose a model that favored recall. To go further, if the company is small enough, it would probably be best to simply investigate everyone and use the model as an assist -- something that can help us allocate resources to investigate people that are "more" interest than others. We can use the probability returned by a classifier (in the case of logistic regression, or the percentage of correctly classified instances at the leaf node of a decision tree) to help make this judgement.

References

[1] enron61702insiderpay.pdf
*http://scikit-learn.org/