UDACITY

PROJECT

Identify Fraud from Enron Email

A part of the Data Analyst Nanodegree Program

PROJECT REVIEW

CODE REVIEW 2

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Hi there!

Great work on this resubmission that meets all the required criteria!
Before going ahead, I strongly encourage taking a look at the suggestions provided below.
The idea behind these comments is to keep on improving the report and code so it looks everytime better in your portfolio!

Keep up the solid work and good luck on the next project!

Quality of Code

Code reflects the description in the answers to questions in the writeup. i.e. code performs the functions documented in the writeup and the writeup clearly specifies the final analysis strategy.

poi_id.py can be run to export the dataset, list of features and algorithm, so that the final algorithm can be checked easily using tester.py.

Understanding the Dataset and Question

Student response addresses the most important characteristics of the dataset and uses these characteristics to inform their analysis. Important characteristics include:

- **total number of data points**
- **allocation across classes (POI/non-POI)**
- **number of features used**
- **are there features with many missing values? etc.**

Student response identifies outlier(s) in the financial data, and explains how they are removed or otherwise handled.

GREAT WORK

Nice work on correctly finding and removing the `TOTAL`, `THE TRAVEL AGENCY IN THE PARK` and `LOCKHART EUGENE E` outliers!

SUGGESTIONS

- In practice, we do have a few more "optional" outliers in the dataset. Even though this is not mandatory, I strongly encourage putting some effort on finding them so you can practice more your skills on outlier handling. ☺
  My suggestion here would be using Tukey's Inter Quartile Range method to easily find these outliers! If you consider removing more outliers, please note that they should be removed only from **training** data, and not from **test** data!

- Also, try to spend some extra time to see if we can't find any more "absurd" values, such as negative salaries and things like that. If you find any, we could then compare these values to the ones available in the provided PDF. This way, we will be able to solve these last discrepancies.

Optimize Feature Selection/Engineering

At least one new feature is implemented. Justification for that feature is provided in the written response. The effect of that feature on final algorithm performance is tested or its strength is compared to other features in feature selection. The student is not required to include their new feature in their final feature set.

Univariate or recursive feature selection is deployed, or features are selected by hand (different combinations of features are attempted, and the performance is documented for each one). Features that are selected are reported and the number of features selected is justified. For an algorithm that supports getting the feature importances (e.g. decision tree) or feature scores (e.g. SelectKBest), those are documented as well.

If algorithm calls for scaled features, feature scaling is deployed.

Pick and Tune an Algorithm

At least two different algorithms are attempted and their performance is compared, with the best performing one used in the final analysis.

Response addresses what it means to perform parameter tuning and why it is important.

At least one important parameter tuned with at least 3 settings investigated systematically, or any of the following are true:

- **GridSearchCV used for parameter tuning**
- **Several parameters tuned**
- **Parameter tuning incorporated into algorithm selection (i.e. parameters tuned for more than one algorithm, and best algorithm-tune combination selected for final analysis).**

Validate and Evaluate

At least two appropriate metrics are used to evaluate algorithm performance (e.g. precision and recall), and the student articulates what those metrics measure in context of the project task.

Response addresses what validation is and why it is important.

Performance of the final algorithm selected is assessed by splitting the data into training and testing sets or through the use of cross validation, noting the specific type of validation performed.

SUGGESTION

Great work on implementing `KFold`!
In this project, we are dealing with a small and imbalanced dataset.
As seen throughout the lectures, working with smaller datasets is hard and in order to make validation models robust, we often go with **k-fold cross-validation**, which is what we do when we use a shuffle split.
But well, in this project (and so many others we have in our day to day work), a stratified shuffle split is of choice. When dealing with small imbalanced datasets, it is possible that some folds contain almost none (or even none!) instances of the minority class. The idea behind stratification is to keep the percentage of the target class as close as possible to the one we have in the complete dataset. Please take a look here and here for more.
Luckily, to make our validation stratified, all we need to do is replace `KFold` with `StratifiedKFold`!

When tester.py is used to evaluate performance, precision and recall are both at least 0.3.

OUTPUT

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=2,
            min_samples_split=9, min_weight_fraction_leaf=0.0,
            presort=False, random_state=123, splitter='best')
    Accuracy: 0.85873    Precision: 0.45563    Recall: 0.30550    F1: 0.36576    F2: 0.32705
    Total predictions: 15000    True positives:  611    False positives:  730    False negatives: 1389    True negatives: 12270
```

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

RETURN TO PATH

Student FAQ