PROJECT

Identify Fraud from Enron Email

A part of the Data Analyst Nanodegree Program

PROJECT REVIEW | CODE REVIEW 1 | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

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.**

Optimize Feature Selection/Engineering

✓ **At least one new feature is implemented. Justification for that feature is provided in the written response, and the effect of that feature on the final algorithm performance is tested. The student is not required to include their new feature in their final feature set.**

Note that in regard to the use of POI to/from emails in engineering new features, there is a possible data leakage between training and test set i.e. when the classifier is indeed used to predict a person's POI-ness, we obviously do not know beforehand if this person is a POI or not, but in training and test we already know them as shown by existence of `from_poi_to_this_person` and `from_this_person_to_poi`. This thread has a really interesting discussion on this.

Given the limitations of our dataset (low number of data points, low proportion of POIs), we may accept these features from a practical standpoint, but please be aware of this potential issue when you try to use machine learning on future data.

✓ **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.**

Good job with feature selection step. From line 230 in your code:

```
k_best.fit(features, labels)
```

Looks like feature selection in this project was performed on the entire dataset. Although this is not a requirement to pass this specification (as long as the end scores are > 0.3), this is not a good idea since this way there is a data leakage from test data, i.e. feature selector is not supposed to know anything about test data.

This is optional, but a better approach would be to combine some sort of cross-validation with the selectKBest process (a stratified shuffle split might be best). To do this, you would need to **set up the cross-validation split** of the data set, and perform feature selection on the **training set** of each fold. In general, the selectKBest algorithm gets an F-score for each feature and chooses the k features with the best F-scores. When combined with cross-validation, an F score will be produced for each feature and fold. The best features can then be considered those with the **best average F-scores over all folds in the cross validation**. See the code below for a concrete example:

```
# For python 0.17 and older
# splits = StratifiedShuffleSplit(labels, n_iter = 1000, random_state=42)

splits = StratifiedShuffleSplit(n_splits = 1000, random_state=42)

k = 5 # Change this to number of features to use.

# We will include all the features into variable best_features, then group by their
# occurrences.
features_scores = {}
for i_train, i_test in splits:
    features_train, features_test = [features[i] for i in i_train], [features[i] for i in i_test]
    labels_train, labels_test = [labels[i] for i in i_train], [labels[i] for i in i_test]

    # fit selector to training set
    selector = SelectKBest(k = k)
    selector.fit(features_train, labels_train)

    # Get scores of each feature:
    sel_features = np.array(features_list[1:])[selector.get_support()]
    sel_list = []
    for i in range(len(sel_features)):
        sel_list.append([sel_features[i], selector.scores_[i],selector.pvalues_[i]])

    # Fill to feature_scores dictionary
    for feat, score, pvalue in sel_list:
        if feat not in features_scores:
            features_scores[feat] = {'scores': [], 'pvalues': []}
        features_scores[feat]['scores'].append(score)
        features_scores[feat]['pvalues'].append(pvalue)

# Average scores and pvalues of each feature
features_scores_l = [] # tuple of feature name, avg scores, avg pvalues
for feat in features_scores:
    features_scores_l.append((
        feat,
        np.mean(features_scores[feat]['scores']),
        np.mean(features_scores[feat]['pvalues'])
    ))

# Sort by scores and display
import operator
sorted_feature_scores = sorted(features_scores_l, key=operator.itemgetter(1), reverse=True)
sorted_feature_scores_str = ["{}, {}, {}".format(x[0], x[1], x[2]) for x in sorted_feature_scores]

print "feature, score, pvalue"
pprint(sorted_feature_scores_str)
```

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

The final dataset exported to `my_dataset.pkl` which will be tested with tester.py's `test_classifier` does not have scaling applied to it. This passes spec since final algorithm does not require scaling, but I suggest to adjust the code anyway to make it more robust i.e. scaling applied even when final algorithm requires scaling.

We can, of course, recombine the scaled features into a data dictionary which will then be dumped into my_dataset.pkl pickle file, but there is a much easier and recommended method to do this with python + sklearn.

For this method we will need to use Pipeline. I know this technique may look a little complicated at first, but understanding this will greatly benefit your work as a Machine Learning specialist.

Basically, a Pipeline object is a custom-made classifier consisting of any set of preprocessing tools and a single estimator (i.e. a single machine learning algorithm). That way, we can pass in our dataset into it, of any kind, and it will preprocess the dataset before finally running an estimator on it.

Here is an example of how to set it up in this project

```
steps = [
    ('scaler', sklearn.preprocessing.MinMaxScaler()),
    ('classifier', YourChosenClassifier(random_state=random))
]

clf = Pipeline(steps)
```

Pick and Tune an Algorithm

✓ **At least 2 different algorithms are attempted and their performance is compared, with the more performant one used in the final analysis.**

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

Importance of tuning explained, good work.

✓ **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.**

Correct explanation of validation and its importance included in the write-up. good work.

✓ **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.**

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

After waiting for awhile for the code to finish running, I got the following result:

```
Accuracy: 0.86614       Precision: 0.54280      Recall: 0.39950 F1: 0.46025     F2: 0.42177
```

Good job on reaching high scores on this project, but notice this:

Due to how the base code was structured, if you pick the right few features then feature_format.py will remove a substantial number of observations and make the overall problem much easier for the classifier. Looks like you have found the right features here.

This can be confirmed by seeing how many data are being processed in `test_classifier` function inside `tester.py` script e.g. by adding this code between line 27 and 28 in tester.py:

```
print "Number of observations: {}".format(len(features))
```

Or in your `poi_id.py` file, you can examine number of data that was outputted from code

```
data = featureFormat(my_dataset, features_list, sort_keys = True)
```

You'll see that there are only 131 observations in this project, which means 13 observations got removed out of total 144 observations. This was mainly caused by missing data in Enron dataset, e.g. several employees have empty inbox / outbox, as well as in shared receipt.

You don't need to do anything here, just be aware that there is this kind of issue too on machine learning projects.

DOWNLOAD PROJECT

1 CODE REVIEW COMMENTS

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH