Project P5: Identify Fraud from Enron Email

Peter Eisenschmidt

February 24, 2017

1 Introduction

The goal of this project is to identify potential persons of interest (POI) based on their financial and email data. Machine learning is used here as this allows building a model which determines whether or not a person is a POI based on numerous input features. The clear advantage of machine learning here is that multiple features can be combined whereas this would not be possible in a classical approach.

The problem here in this project is a supervised classification problem: supervised because in the available Enron dataset we know who is a POI and who is not and classification because the output is not continuous (either POI=1 or not POI=0, but no intermediate values).

2 Data Understanding and Exploration

2.1 Dataset Description

The Enron data set contains total number of 146 data points. From these 146 entries, 18 are labelled as POI. 21 different features (financial and email) are included in the data set. However, not all features are available for all persons, so a few features have missing values.

The next step is to check if all 146 entries are valid entries and if all features can be used for this project.

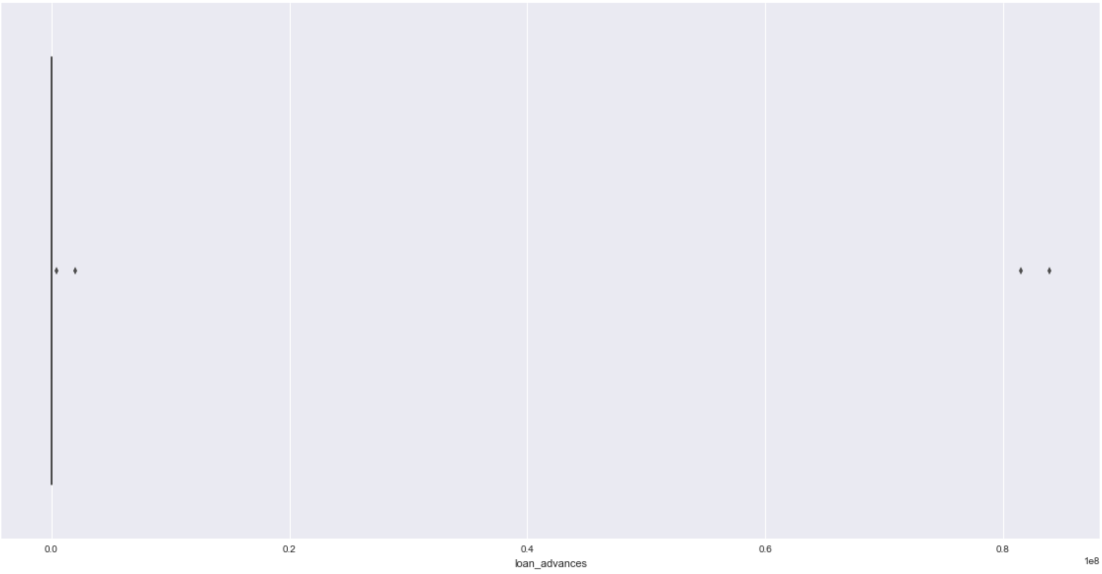
2.2 Outlier Removal

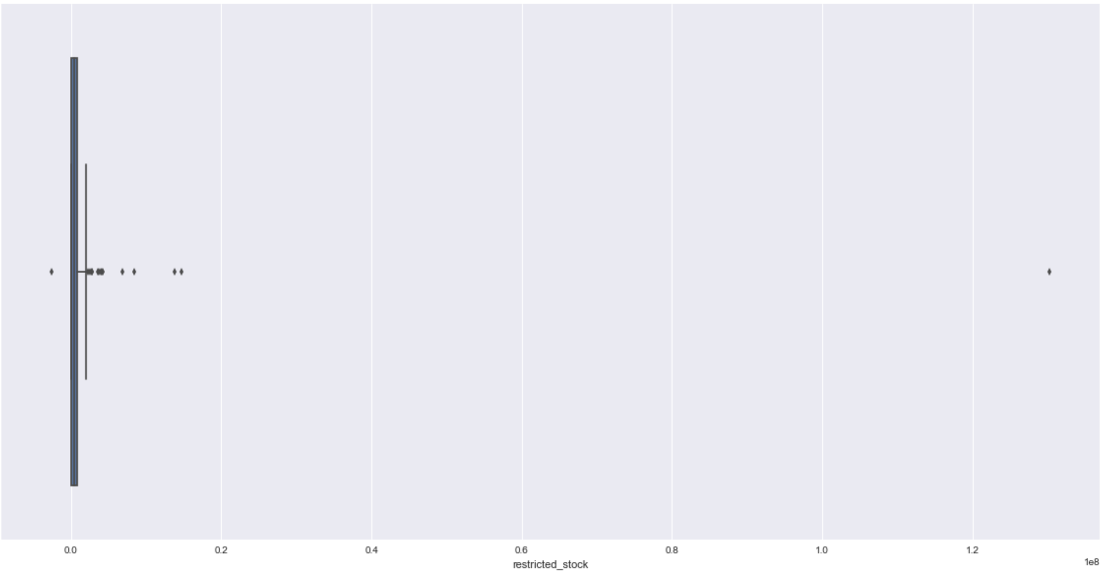
The first step is to detect any outliers and to see if they need to be removed from the dataset. The TOTAL entry is removed, as it details the sum over all persons included in the dataset. Furthermore, the entry THE TRAVEL AGENCY IN THE PARK is removed, as this is not a person but a corporation.

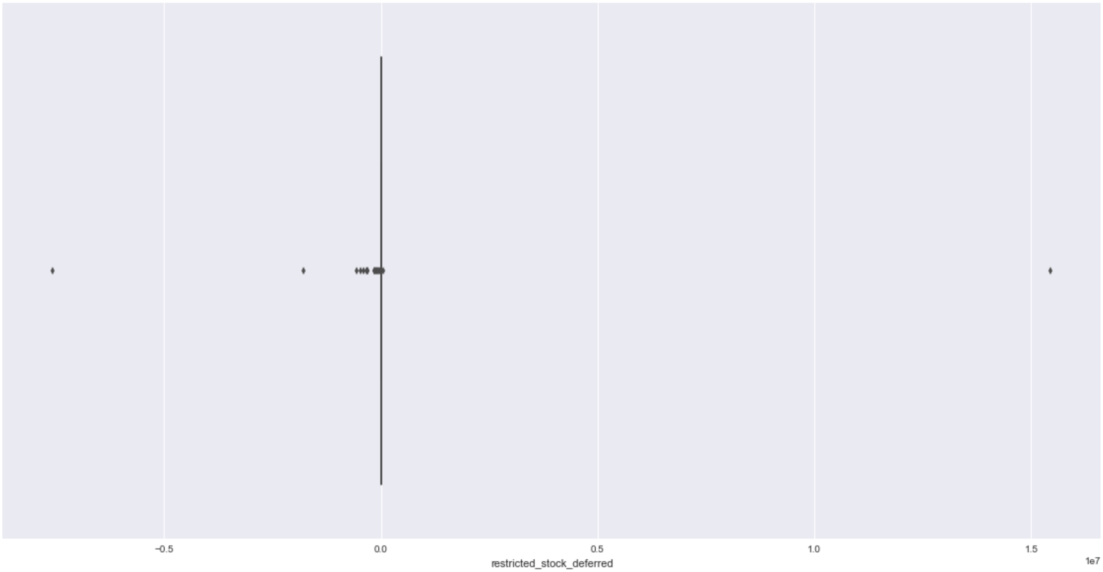
In order to detect further entries that need to be excluded, boxplots are created for all the features, as this allows detecting outliers visually.

Three parameters show outliers that require further investigation:

loan_advances	Only few data points are available for this parameter. One extreme point (81,525,000) sticks out. As this data point belongs to Ken Lay, it should be considered as valid.
restricted_stock_deferred	Only few data points available, most of them negative. The positive value requires further checking.
restricted_stock	One negative value whereas the remaining data points are positive.







The entries where restricted_stock_deferred restricted_stock show outliers belong to BHATNAGAR SANJAY and BELFER ROBERT. These entries are then removed from the dataset.

The parameters total_payments and total_stockvalue are the sum of all payments (salary, bonus, etc) and stock values (exercised stock option, restricted stock option, and restricted stock option deferred) respectively. So, either all other values are excluded and only the sums are included or vice versa.

2.3 First Feature Selection

Lastly, it can be seen that a lot of parameters contain zeros (i.e. NaNs in the original dictionary). As this may influence the performance of the algorithm, all features which contain more than 75% of zeros are removed.

The retained parameters are:
`['poi', 'salary', 'deferral_payments', 'bonus', 'deferred_income', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'to_messages', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']`

2.4 Feature Creation

The dataset contains information about the numbers of emails a person sent and received ('to_messages' and 'from_messages'). Furthermore, the parameters 'from_poi_to_this_person' and 'from_this_person_to_poi' indicate the numbers of emails exchanged with a POI.

The features that are created are the fractions of POI-related emails and total emails. This is done because it emphasizes the importance of POI-related emails. For example, if someone sent 8 emails to a POI and sent only 10 emails in total, this must be weighed higher against someone who sent 15 emails to a POI but has a total of 1000 emails.

The new features are **'frac_to_poi'** and **'frac_from_poi'**.

3 Algorithm Selection and Tuning

3.1 Assessment of Several Algorithms

As a first step, 5 algorithms are selected and run with their default parameters:

1. Gaussian Naives Bayes
2. K Nearest Neighbors
3. Decision Tree Classifier
4. Random Forest Classifier
5. Support Vector Classifier

The dataset is split into a training and a test dataset using `train_test_split`; the size of the test set is .2.

The results on the selected test set are:

```
Step 1:  0 GaussianNB
Precision:  0.75
Recall:     0.6
F1:         0.666666666667
done...
Step 1:  1 KNearestNeighbors
Precision:  0.0
Recall:     0.0
F1:         0.0
done...
Step 1:  2 Decision Tree
Precision:  1.0
Recall:     0.2
F1:         0.333333333333
done...
Step 1:  3 Random Forest
Precision:  0.0
Recall:     0.0
F1:         0.0
done...
Step 1:  4 Support Vector Classifier
```

```
Precision:  0.0
Recall:     0.0
F1:         0.0
done...
```

The results evaluated with tester.py are:

```
GaussianNB(priors=None)
	Accuracy: 0.82571	Precision: 0.34637	Recall: 0.24800	F1: 0.28904	F2: 0.26293
	Total predictions: 14000	True positives:  496	False positives:  936	False negatives: 1504	True negatives: 11064

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
	Accuracy: 0.88079	Precision: 0.82515	Recall: 0.21000	F1: 0.33479	F2: 0.24680
	Total predictions: 14000	True positives:  420	False positives:   89	False negatives: 1580	True negatives: 11911

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best')
	Accuracy: 0.81121	Precision: 0.32247	Recall: 0.29200	F1: 0.30648	F2: 0.29763
	Total predictions: 14000	True positives:  584	False positives: 1227	False negatives: 1416	True negatives: 10773

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=42,
            verbose=0, warm_start=False)
	Accuracy: 0.84736	Precision: 0.38487	Recall: 0.11450	F1: 0.17649	F2: 0.13322
	Total predictions: 14000	True positives:  229	False positives:  366	False negatives: 1771	True negatives: 11634

Got a divide by zero when trying out: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
Precision or recall may be undefined due to a lack of true positive predicitons.
```

It can be seen that most classifiers do not perform well on the test set; only the Gaussian Naives Bayes and the Decision Tree yield results. Whereas on the test set the GaussianNB has precision and recall greater than .3 (required value), using tester.py the performance is not as good, albeit close. The Decision Tree is already close using tester.py but has only a recall of .2 on the test set. Interestingly, the K Nearest Neighbor algorithm has values of 0 for recall and precision on the test set, but achieves significantly better results with tester.py.

One important thing to note is that when the algorithms are trained with their default parameters, the scoring parameter that is used to obtain the best fit is accuracy. However, in this case, accuracy is not the best metric to assess classifier performance, as there are only limited numbers of POIs in the dataset. Therefore, if all 18 POIs were incorrectly classified as Non-POIs, the accuracy would still be 0.8732 (based on 142 entries in the dataset, as 4 were removed initially). This will be addressed in the next step.

3.2 First Tuning of All Algorithms

As the performance was generally not good for the selected five algorithms, the next step is to use `GridSearchCV` to make a first tuning of the algorithms. Furthermore, MinMax scaling is applied to all algorithms except the Decision Tree and the Random Forest.

As mentioned in the previous section, accuracy is not a good metric for this data set. Precision and recall on the other hand are better metrics in this case. What this means for this project is as follows:

Recall Number of True Positives (TP) divided by the sum of True Positives and False Negatives (FN). A low recall indicates that the number of False Negatives is too high, i.e. a lot of employees are predicted not to be a POI whereas in reality they were

Precision Number of True Positives (TP) divided by the sum of True Positives and False Positives (FP). A low precision indicates that the number of False Positive is too high, i.e. a lot of employed are predicted to be a POI whereas in reality they were not

In both cases the number of True Positives may also be too low, i.e. the number of correctly identified POIs

It is arguably which one is more important but I would tend to achieve a better recall than precision, as it seems better to falsely identify someone as POI (and to exonerate them later) than to miss a potential POI. On the other hand, optimizing the algorithm for a good recall value may lead to a low precision, i.e. too many employees are falsely identified as POIs. Therefore, the scoring parameter that is used in `GridSearchCV` is F1. F1 is defined as

$$F1 = 2\frac{precision \times recall}{precision + recall} \tag{1}$$

`GridSearchCV` automatically applies cross-validation when fitting the algorithm to the training data. In this step, `StratifiedShuffleSplit` is used with 30 splits and a test size of .2.

The results on the selected test set are:

```
Step 2:  0 GaussianNB
Best score:  0.088177008177
GaussianNB(priors=None)
Precision:  0.75
Recall:     0.6
F1:         0.666666666667
done...

Step 2:  1 KNearestNeighbors
Best score:  0.08
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
Precision:  0.0
Recall:     0.0
```

```
F1:         0.0
done...

Step 2:  2 Decision Tree
Best score:  0.333095238095
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=6,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best')
Precision:  1.0
Recall:     0.2
F1:         0.333333333333
done...

Step 2:  3 Random Forest
Best score:  0.176931216931
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=5, n_jobs=1, oob_score=False, random_state=42,
            verbose=0, warm_start=False)
Precision:  0.0
Recall:     0.0
F1:         0.0
done...

Step 2:  4 Support Vector Classifier
Best score:  0.24246031746
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=1, kernel='sigmoid',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
Precision:  0.666666666667
Recall:     0.4
F1:         0.5
done...
```

The results evaluated with tester.py are:

```
GaussianNB(priors=None)
	Accuracy: 0.82571	Precision: 0.34637	Recall: 0.24800	F1: 0.28904	F2: 0.26293
	Total predictions: 14000	True positives:  496	False positives:  936	False negatives: 1504	True negatives: 11064

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

```
        Accuracy: 0.86086       Precision: 0.53533      Recall: 0.19700 F1: 0.28801     F2: 0.22550
        Total predictions: 14000        True positives:  394    False positives:  342   False negatives: 1606   True negatives: 11658

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=6,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best')
        Accuracy: 0.85000       Precision: 0.46318      Recall: 0.31450 F1: 0.37463     F2: 0.33608
        Total predictions: 14000        True positives:  629    False positives:  729   False negatives: 1371   True negatives: 11271

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=5, n_jobs=1, oob_score=False, random_state=42,
            verbose=0, warm_start=False)
        Accuracy: 0.83379       Precision: 0.34903      Recall: 0.18900 F1: 0.24522     F2: 0.20808
        Total predictions: 14000        True positives:  378    False positives:  705   False negatives: 1622   True negatives: 11295

Got a divide by zero when trying out: SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=1, kernel='sigmoid',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
Precision or recall may be undefined due to a lack of true positive predicitons.
```

This shows the importance of parameter tuning. In the previous step, the entire training data is used and the classifier is fit to that. However, the performance on the test set was bad in most cases. This is most likely due to over-fitting, which then means that it does not generalize well to the new data. Parameter tuning allows controlling over-fitting so that performance improves for the new, unknown test data.

Here, with only a few parameters that are tuned, it can be seen that, especially for the Decision Tree and Random Forest, the performance with tester.py increased.

Furthermore, it shows that scaling is important for classifiers such as Support Vector Machines, as it maximizes the Euclidean distance between two clusters. Previously, there were no results on the test set but now it achieves a precision and recall of .667 and .4 respectively. The fact that it fails with tester.py shows that more tuning is necessary though.

3.3 Algorithm Selection

For the final algorithm the decision tree is selected, as this algoritm gives the best results with tester.py. The next step is to further tune this algorithm using `GridSearchCV`; the parameters that are tuned are:

- criterion
- max_features
- max_depth
- min_samples_split
- min_samples_leaf
- max_leaf_nodes

- splitter
- class_weight
- presort

The number of splits of `StratifiedShuffleSplit` is changed to 60 (in the previous section it was 30).

The results on the selected test set are:

```
Step 3a: Decision Tree Optimization
Best score:  0.465944148444
Precision:  0.5
Recall:     0.8
F1:         0.615384615385
```

The results evaluated with tester.py are:

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
            max_depth=1, max_features=15, max_leaf_nodes=10,
            min_impurity_split=1e-07, min_samples_leaf=2,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random')
        Accuracy: 0.79257       Precision: 0.37895      Recall: 0.70750 F1: 0.49355     F2: 0.60295
        Total predictions: 14000        True positives: 1415    False positives: 2319   False negatives:  585   True negatives: 9681
```

It can be seen that now both the test set and tester.py yield results greater than the required values of .3.

The importance of the features is checked with `feature_importances_` attribute of the decision tree:

```
Feature Importance:
                   salary : 0.00000
         deferral_payments : 0.00000
                    bonus : 0.00000
          deferred_income : 0.00000
                 expenses : 0.20266
   exercised_stock_options : 0.09872
                    other : 0.00000
       long_term_incentive : 0.00000
          restricted_stock : 0.00000
               to_messages : 0.00000
   from_poi_to_this_person : 0.00000
             from_messages : 0.00000
   from_this_person_to_poi : 0.00000
   shared_receipt_with_poi : 0.00000
             frac_from_poi : 0.69862
               frac_to_poi : 0.00000
```

4 Feature Selection

In the previous section it has been shown that 'expenses', 'exercised_stock_options', and the newly created feature 'frac_from_poi' have the highest importance in the decision tree. So, the decision tree is trained again with only these features.

In the Grid Search, the parameter max_features now needs to be adapted as the previous Grid Search returned a value of 15, which is greater than the number of features now.

The results on the selected test set are:

```
Step 3b: Decision Tree Optimization
['poi', 'expenses', 'exercised_stock_options', 'frac_from_poi']
Best score:  0.48737993488
Precision:  0.25
Recall:     0.333333333333
F1:         0.285714285714
```

The results evaluated with tester.py are:

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
            max_depth=3, max_features=2, max_leaf_nodes=10,
            min_impurity_split=1e-07, min_samples_leaf=3,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random')
        Accuracy: 0.83929       Precision: 0.42656      Recall: 0.36300 F1: 0.39222     F2: 0.37415
        Total predictions: 14000        True positives:  726    False positives:  976   False negatives: 1274   True negatives: 11024
```

The results are not as good as when using all features; however, the results with tester.py are greater than .3. So for the final validation, it has been decided to keep all initial features in order to obtain the best performance.

5 Final Algorithm and Validation

Before doing the final validation, a check is done without the two newly created features. The classifier is run with all features except the two new ones (and the ones with more than 75% zeros). Please note that this requires adapting the parameter max_features to 14.

The results are the following with the test set:

```
Precision:  0.277777777778
Recall:     1.0
F1:         0.434782608696
```

tester.py yields the following:

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
            max_depth=1, max_features=14, max_leaf_nodes=10,
            min_impurity_split=1e-07, min_samples_leaf=2,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random')
        Accuracy: 0.65886       Precision: 0.26346      Recall: 0.77300 F1: 0.39298     F2: 0.55740
        Total predictions: 14000        True positives: 1546    False positives: 4322   False negatives:  454   True negatives: 7678
```

Both on the test set and tester.py the performance is worse; precision is below the required value of .3. This shows how important these new features are.

The previous sections have highlighted the importance of validation. In this case, the available data was split into a training and a test set. The test set was determined by the random state set in `train_test_split` (here fixed to 42 in order to get reproducable results for this project).

The algorithm was tuned on the training data; here, `GridSearchCV` applied cross-validation using `StratifiedShuffleSplit` in order to avoid over-fitting.

Then, at each step, the classifier performance was evaluated with the previously unseen test set. It is important that the test data is not used for model training, as this could (and most likely will) result in over-fitting. The second validation step is done with tester.py, which simulates new, future data. Again, it is important that the data used for validation is not used in the training, in order to ensure that the classifier generalizes well to new data.

So, the final algorithm has the following performance:

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
            max_depth=1, max_features=15, max_leaf_nodes=10,
            min_impurity_split=1e-07, min_samples_leaf=2,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random')
        Accuracy: 0.79257       Precision: 0.37895      Recall: 0.70750 F1: 0.49355     F2: 0.60295
        Total predictions: 14000        True positives: 1415    False positives: 2319   False negatives:  585   True negatives: 9681
```