Fraud detector model training

In [1]:
```
import sys
import types
import pickle

import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import sklearn.metrics as metrics
from sklearn.metrics import roc_auc_score, confusion_matrix, precision_reca
from sklearn.metrics import confusion_matrix, f1_score
import matplotlib.pyplot as plt
```

In [2]:
```
transactions = pd.read_csv('creditcard.csv')
```

In [3]:
```
transactions.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

In [4]:
```
number_of_rows = transactions.shape[0]
transactions.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
fraud_cases = transactions[(transactions.Class==1)]
fraud_cases.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.067794
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.399147
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.248778
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.496358

5 rows × 31 columns

In [6]:
```
number_of_fraud_cases = fraud_cases.shape[0]
fraud_cases.shape
```

Out[6]:
```
(492, 31)
```

Important Note

The classes are highly imbalanced!

In [7]:
```
percentage_of_fraud_transactions = number_of_fraud_cases / number_of_rows *
print('Percentage of fraud transactions', percentage_of_fraud_transactions,
```

```
Percentage of fraud transactions 0.1727485630620034 %
```

In [8]:
```
X = transactions.drop('Class', 1)
y = transactions['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

Simple random forest classifier

In [9]:
```
%%time
rf_model = RandomForestClassifier(
    n_estimators=1,
    criterion='gini',
    max_depth=7,
    min_samples_split=2,
    min_samples_leaf=5,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=16,
    random_state=None,
    verbose=100,
    warm_start=False,
    class_weight=None).fit(X_train, y_train)
```

```
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent
workers.
building tree 1 of 1
[Parallel(n_jobs=16)]: Done   1 tasks      | elapsed:    0.6s
[Parallel(n_jobs=16)]: Done   1 out of   1 | elapsed:    0.6s finished
CPU times: user 661 ms, sys: 15.3 ms, total: 676 ms
Wall time: 765 ms
```

In [10]:
```
pred_train = rf_model.predict(X_train)
pred_test = rf_model.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

In [11]:
```
confusion_matrix(y_train, pred_train)
```

Out[11]:
```
array([[227414,     39],
       [   102,    290]])
```

In [12]:
```
precision, recall, _ = precision_recall_curve(y_test, pred_test)
print('Precision: ', precision[1])
print('Recall: ', recall[1])
print('AUC', metrics.auc(precision, recall))
```

```
Precision:  0.8444444444444444
Recall:  0.76
AUC 0.8006773326467157
```

Improve random forest model using grid search

In [13]:
```
rf_model = RandomForestClassifier(
    n_estimators=1,
    criterion='gini',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=5,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=16,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None)
```

In [14]:
```
param_grid = {
    'n_estimators': [6, 7, 8, 9, 10, 20, 30, 50],
    'max_depth': [7, 8, 9, 10, 11]
}
```

In [15]:
```
clf = GridSearchCV(
    rf_model,
    param_grid,
    n_jobs=16,
    cv=3,
    scoring='recall'
)
```

In [16]:
```
%%time
clf.fit(X_train, y_train)
```

```
CPU times: user 9.36 s, sys: 464 ms, total: 9.82 s
Wall time: 4min 21s
```

Out[16]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weigh
t=None,
                                              criterion='gini', max_depth
=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=5,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.
0,
                                              n_estimators=1, n_jobs=16,
                                              oob_score=False,
                                              random_state=None, verbose=
0,
                                              warm_start=False),
             iid='warn', n_jobs=16,
             param_grid={'max_depth': [7, 8, 9, 10, 11],
                         'n_estimators': [6, 7, 8, 9, 10, 20, 30, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring='recall', verbose=0)
```

In [17]:
```
rf_best = clf.best_estimator_
pred_train = rf_best.predict(X_train)
pred_test = rf_best.predict(X_test)
confusion_matrix(y_train, pred_train)
```

Out[17]:
```
array([[227443,     10],
       [    87,    305]])
```

In [18]:
```
confusion_matrix(y_test, pred_test)
```

Out[18]:
```
array([[56855,     7],
       [   22,    78]])
```

In [19]:
```
precision, recall, _ = precision_recall_curve(y_train, pred_train)
print('Precision: ', precision[1])
print('Recall: ', recall[1])
```

```
Precision:  0.9682539682539683
Recall:  0.7780612244897959
```

In [20]:
```
precision, recall, _ = precision_recall_curve(y_test, pred_test)
print('Precision: ', precision[1])
print('Recall: ', recall[1])
```

```
Precision:  0.9176470588235294
Recall:  0.78
```

In [21]:
```
print('AUC', metrics.auc(precision, recall))
print('The general score for rf model on a test set:', rf_best.score(X_test
print('F1 score for the rf model on a test set: ', f1_score(y_test, pred_te
```

```
AUC 0.8472610842729003
The general score for rf model on a test set: 0.9994908886626171
F1 score for the rf model on a test set:  0.8432432432432432
```

Let's try gradient boosting

In [22]:
```
from catboost import CatBoostClassifier
from catboost import Pool
from catboost import MetricVisualizer
```

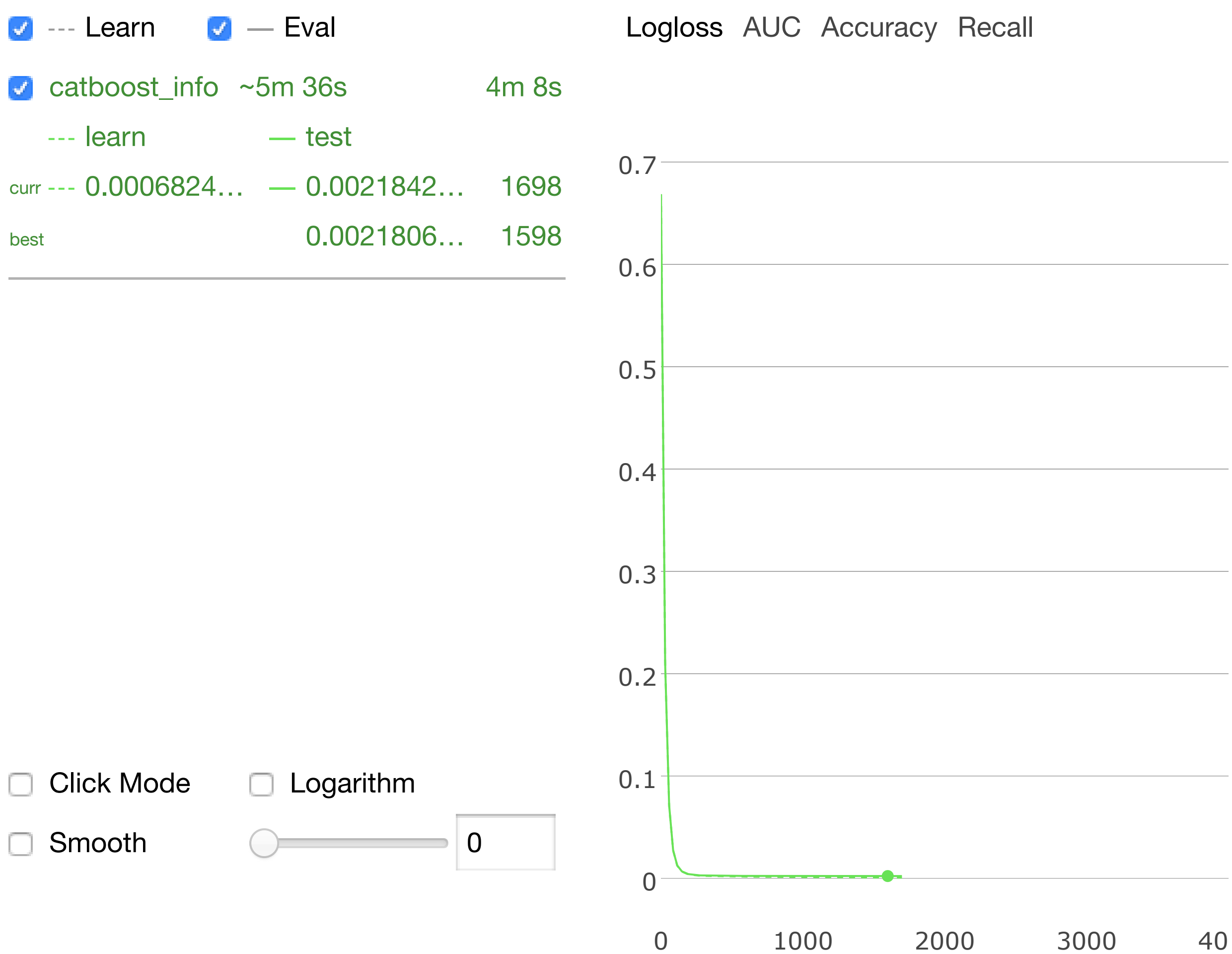
In [23]:
```
# create train_pool object
train_pool = Pool(
    data=X_train,
    label=y_train
)

# create validation_pool object
validation_pool = Pool(
    data=X_test,
    label=y_test
)
```

In [24]:
```python
# we create the object of CatBoostClassifier class
cbs = CatBoostClassifier(iterations=4000,
                         learning_rate=0.007,
#                          eval_metric='Recall',
                         custom_loss=['AUC', 'Accuracy', 'Recall'],
                         max_depth=10,
                         early_stopping_rounds=100)

# we are passing categorical features as parameters here
# verbose = 10 outputs only each 10th tree

cbs.fit(
    train_pool,
    eval_set=validation_pool,
    verbose=False,
    plot=True
);
```



In [25]:
```python
pred_test_cbs = cbs.predict(X_test)
confusion_matrix(y_test, pred_test_cbs)
```

Out[25]:
```
array([[56858,     4],
       [   18,    82]])
```

In [26]:
```
precision_cbs, recall_cbs, _ = precision_recall_curve(y_test, pred_test_cbs
print('AUC', metrics.auc(precision_cbs, recall_cbs))
print('Precision: ', precision_cbs[1])
print('Recall: ', recall_cbs[1])
print('The accuracy for cbs model on a test set:', cbs.score(X_test, y_test
print('F1 score for the cbs model on a test set: ', f1_score(y_test, pred_t
```

```
AUC 0.885146629780931
Precision:  0.9534883720930233
Recall:  0.82
The accuracy for cbs model on a test set: 0.9996137776061234
F1 score for the cbs model on a test set:  0.8817204301075269
```

Note: the best result for max_depth=6 model was AUC 0.879. So increasing the depth of the trees actually improves the classifier.

Note: in the models above we have dealt with unbalanced dataset in a pretty dummy way, so we really have to revisit the model creation

Side note: Comprehensive Kernel on unbalanced datasets oversampling. [https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets (https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets)](https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets)

Kernels [https://www.kaggle.com/mlg-ulb/creditcardfraud/kernels (https://www.kaggle.com/mlg-ulb/creditcardfraud/kernels)](https://www.kaggle.com/mlg-ulb/creditcardfraud/kernels)

Save both models

First, will use ordinary pickling to serialize the models.

In [27]:
```
filename = 'random_forest_model.sav'
pickle.dump(rf_best, open(filename, 'wb'))
```

In [28]:
```
filename = 'catboost_model.sav'
pickle.dump(cbs, open(filename, 'wb'))
```

In [29]:
```
import os
def print_file_size(filename):
    statinfo = os.stat(filename)
    print(filename, ':', statinfo.st_size/1000, 'Kb')

print_file_size('random_forest_model.sav')
print_file_size('catboost_model.sav')
```

```
random_forest_model.sav : 51.101 Kb
catboost_model.sav : 26796.498 Kb
```

Load both models

First of all we load random forest model.

In [56]:
```
%%time
loaded_rf_model = pickle.load(open('random_forest_model.sav', 'rb'))
```

```
CPU times: user 1.06 ms, sys: 2.72 ms, total: 3.78 ms
Wall time: 3.45 ms
```

In [57]:
```
%%time
predictions = loaded_rf_model.predict(X_test)
print('First 5 predicted classes:',
      predictions[:5], '\n')
prediction_probabilities = loaded_rf_model.predict_proba(X_test)
print('First 5 predicted probabilities:\n',
      prediction_probabilities[:5], '\n')
print('Out of', len(predictions), 'predictions',
      sum(predictions), 'are fraud\n')
```

```
First 5 predicted classes: [0 0 0 0 0]

First 5 predicted probabilities:
 [[9.99820548e-01 1.79451635e-04]
 [9.99766690e-01 2.33309961e-04]
 [9.99366122e-01 6.33877563e-04]
 [9.99820548e-01 1.79451635e-04]
 [9.99820548e-01 1.79451635e-04]]

Out of 56962 predictions 85 are fraud

CPU times: user 69.3 ms, sys: 13.2 ms, total: 82.5 ms
Wall time: 232 ms
```

Load catboost model.

In [60]:
```
%%time
loaded_cbs_model = pickle.load(open('catboost_model.sav', 'rb'))
```

```
CPU times: user 28.5 ms, sys: 43.3 ms, total: 71.7 ms
Wall time: 70.9 ms
```

In [62]:
```
%%time
predictions = loaded_cbs_model.predict(X_test)
print('First 5 predicted classes:',
      predictions[:5], '\n')
prediction_probabilities = loaded_cbs_model.predict_proba(X_test)
print('First 5 predicted probabilities:\n',
      prediction_probabilities[:5], '\n')
print('Out of', len(predictions), 'predictions',
      sum(predictions), 'are fraud\n')
```

```
First 5 predicted classes: [0. 0. 0. 0. 0.]

First 5 predicted probabilities:
 [[9.99979466e-01 2.05340649e-05]
 [9.99988859e-01 1.11409006e-05]
 [9.99815615e-01 1.84384576e-04]
 [9.99889708e-01 1.10291780e-04]
 [9.99894420e-01 1.05579926e-04]]

Out of 56962 predictions 86.0 are fraud

CPU times: user 687 ms, sys: 3.92 ms, total: 691 ms
Wall time: 149 ms
```

Preliminary conclusion: the heavier catboost model takes londer to load (70 ms comparing to 3 ms forest model)but suprusingly makes predictions faster then the random forest model.