MOOC users behaviour prediction

The goal is to predict if a user will finish the course or not based on the first 2 days of activity on the platform. We assume that user will finish course if he/she has successfully solved more than 40 practical problems.

We are given a data with user activity splitted in two datasets.

Then we have a data with first 2 days of activities for 6184 users.

In [1]:

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
```

In [2]:

```python
# loading traning data
events_data_train = pd.read_csv("/data/raw/event_data_train.zip")
submission_data_train = pd.read_csv("/data/raw/submissions_data_train.zip")
```

In [3]:

```python
# loading test data
events_data_test = pd.read_csv('/data/raw/event_data_test.csv')
submission_data_test = pd.read_csv('/data/raw/submissions_data_test.csv')
```

In [60]:

```python
events_data_test.head()
```

Out[60]:

	step_id	timestamp	action	user_id	date	day
0	30456	1526893787	viewed	24417	2018-05-21 09:09:47	2018-05-21
1	30456	1526893797	viewed	24417	2018-05-21 09:09:57	2018-05-21
2	30456	1526893954	viewed	24417	2018-05-21 09:12:34	2018-05-21
3	30456	1526895780	viewed	24417	2018-05-21 09:43:00	2018-05-21
4	30456	1526893787	discovered	24417	2018-05-21 09:09:47	2018-05-21

In [61]:

```python
submission_data_test.head()
```

Out[61]:

	step_id	timestamp	submission_status	user_id
0	31971	1526800961	wrong	24370
1	31971	1526800976	wrong	24370
2	31971	1526800993	wrong	24370
3	31971	1526801054	correct	24370
4	31972	1526800664	wrong	24370

1. Let's take only the first 2 day activities from the train dataset

In [6]:

```python
# define a 2-day threshold in seconds
```

```
learning_time_threshold = 2 * 24 * 60 * 60
```

In [7]:

```
# create an events df with timestamp of the first action made by user
events_user_min_timestamp = events_data_train.groupby('user_id') \
    .agg({'timestamp': 'min'}) \
    .rename(columns={'timestamp': 'min_timestamp'}) \
    .reset_index()
```

In [8]:

```
events_user_min_timestamp.head()
```

Out[8]:

	user_id	min_timestamp
0	1	1472827464
1	2	1514383364
2	3	1434358476
3	5	1466156809
4	7	1521634660

In [9]:

```
# merge this with train dataset
events_train_with_min_timestamp = pd.merge(events_data_train, events_user_min_timestamp,
                                           on='user_id', how='outer')
```

In [10]:

```
# Nothing lost
events_train_with_min_timestamp.user_id.nunique() == events_data_train.user_id.nunique()
```

Out[10]:

```
True
```

In [11]:

```
events_train_with_min_timestamp.head()
```

Out[11]:

	step_id	timestamp	action	user_id	min_timestamp
0	32815	1434340848	viewed	17632	1434340848
1	32815	1434340848	passed	17632	1434340848
2	32815	1434340848	discovered	17632	1434340848
3	32811	1434340895	discovered	17632	1434340848
4	32811	1434340895	viewed	17632	1434340848

In [12]:

```
# filter everything that is in 2-day interval
events_train_2days = events_train_with_min_timestamp.query('timestamp <= min_timestamp +
```

In [13]:

```
# we don't need a min_timestamp column, so drop it
events_train_2days = events_train_2days.drop('min_timestamp', axis=1)
```

In [14]:

```
# let's do the same with submission_train dataset
# create a submission df with timestamp of the first action made by user
submissions_user_min_timestamp = submission_data_train.groupby('user_id') \
    .agg({'timestamp': 'min'}) \
    .rename(columns={'timestamp': 'min_timestamp'}) \
    .reset_index()
```

In [15]:

```
submissions_user_min_timestamp.head()
```

Out[15]:

	user_id	min_timestamp
0	2	1514383420

1	3	1434358533
2	5	1499859650
3	8	1480603432
4	14	1436368601

In [16]:

```
submission_train_2days = submission_data_train.merge(submissions_user_min_timestamp, on=

submission_train_2days = submission_train_2days.query('timestamp <= min_timestamp + @lea
    .drop('min_timestamp', axis=1)
```

In [17]:

```
submission_train_2days.head()
```

Out[17]:

	step_id	timestamp	submission_status	user_id
0	31971	1434349275	correct	15853
1	31972	1434348300	correct	15853
4	31976	1434348123	wrong	15853
5	31976	1434348188	correct	15853
7	31977	1434347371	correct	15853

In [18]:

```
events_train_2days.head()
```

Out[18]:

	step_id	timestamp	action	user_id
0	32815	1434340848	viewed	17632
1	32815	1434340848	passed	17632
2	32815	1434340848	discovered	17632
3	32811	1434340895	discovered	17632
4	32811	1434340895	viewed	17632

2. Create base features

Base feautures are user actions and correct/wrong answers

In [19]:

```
# user actions
users_events_data = pd.pivot_table(data=events_train_2days,
                                   values='step_id',
                                   index='user_id',
                                   columns='action',
                                   aggfunc='count',
                                   fill_value=0) \
                                   .reset_index() \
                                   .rename_axis('', axis=1)
```

In [20]:

```
users_events_data.head()
```

Out[20]:

	user_id	discovered	passed	started_attempt	viewed
0	1	1	0	0	1
1	2	9	9	2	9

2	3	15	15	4	20
3	5	1	1	0	1
4	7	1	1	0	1

In [21]:

```
# correct/wrong answers
users_scores = pd.pivot_table(data=submission_train_2days,
                              values='step_id',
                              index='user_id',
                              columns='submission_status',
                              aggfunc='count',
                              fill_value=0) \
                              .reset_index() \
                              .rename_axis('', axis=1)

# add column with correct/wrong answers ratio
users_scores['correct_ratio'] = (users_scores.correct / (users_scores.correct + users_sc
```

In [22]:

```
users_scores.head()
```

Out[22]:

	user_id	correct	wrong	correct_ratio
0	2	2	0	1.0
1	3	4	4	0.5
2	5	2	2	0.5
3	8	9	21	0.3
4	14	0	1	0.0

In [23]:

```
# number of steps that user tried to pass
users_steps_tried = submission_train_2days.groupby('user_id', as_index=False) \
    .step_id.nunique() \
    .rename(columns={'step_id': 'steps_tried'})
```

In [24]:

```
# combine all together
users_data = pd.merge(users_events_data, users_scores,
                      on='user_id',
                      how='outer').fillna(0)
```

In [25]:

```
users_data = users_data.merge(users_steps_tried, how='outer').fillna(0)
```

In [26]:

```
users_data.head()
```

Out[26]:

	user_id	discovered	passed	started_attempt	viewed	correct	wrong	correct_ratio	steps_tried
0	1	1	0	0	1	0.0	0.0	0.0	0.0
1	2	9	9	2	9	2.0	0.0	1.0	2.0
2	3	15	15	4	20	4.0	4.0	0.5	4.0
3	5	1	1	0	1	2.0	2.0	0.5	2.0
4	7	1	1	0	1	0.0	0.0	0.0	0.0

In [27]:

```
# Nothing lost
users_data.user_id.nunique() == events_data_train.user_id.nunique()
```

Out[27]:

```
True
```

3. Calculate target variable

Target condition: if a user finishes 40 practical tasks, we conclude that he's going to finish the course.

In [28]:

```
# count all finished practical tatsks for each user
users_count_correct = submission_data_train[submission_data_train.submission_status == '
    .groupby('user_id', as_index=False).agg({'step_id': 'count'}) \
    .rename(columns={'step_id': 'corrects'})
```

In [29]:

```
users_count_correct.head()
```

Out[29]:

	user_id	corrects
0	2	2
1	3	29
2	5	2
3	8	9
4	16	77

In [30]:

```
# add a rule: if corrects are equal or more than 40, then we set user 'passed' the cours
users_count_correct['passed_course'] = (users_count_correct.corrects >= 40).astype('int'

users_target_feature = users_count_correct.drop(['corrects'], axis=1)

users_target_feature.head()
```

Out[30]:

	user_id	passed_course
0	2	0
1	3	0
2	5	0
3	8	0
4	16	1

4. Create time features

In [31]:

```
# add columns with dates
events_train_2days['date'] = pd.to_datetime(events_train_2days['timestamp'],
                                            unit='s')
events_train_2days['day'] = events_train_2days['date'].dt.date

# create a table with users first/last actions and number of unique days spend on the co
users_time_feature = events_train_2days.groupby('user_id') \
    .agg({'timestamp': ['min', 'max'], 'day': 'nunique'}) \
    .droplevel(level=0, axis=1) \
    .rename(columns={'nunique': 'days'}) \
    .reset_index()

# add column with a difference between first and last action = time user spent on the co
users_time_feature['hours'] = round((users_time_feature['max'] - users_time_feature['min

# drop 'min' and 'max' columns - we keep only time spent in hours
users_time_feature = users_time_feature.drop(['max', 'min'], axis=1)
```

In [32]:

```
users_time_feature.head()
```

Out[32]:

	user_id	days	hours
0	1	1	0.0
1	2	1	0.1
2	3	1	0.3
3	5	1	0.0
4	7	1	0.0

5. Combine all features and target variable

In [33]:

```
# merge with time feature
users_data = users_data.merge(users_time_feature, how='outer')

# add target variable
users_data = users_data.merge(users_target_feature, how='outer').fillna(0)

users_data.head()
```

Out[33]:

	user_id	discovered	passed	started_attempt	viewed	correct	wrong	correct_ratio	steps_tried	days	hours
0	1	1	0	0	1	0.0	0.0	0.0	0.0	1	0.0
1	2	9	9	2	9	2.0	0.0	1.0	2.0	1	0.1
2	3	15	15	4	20	4.0	4.0	0.5	4.0	1	0.3
3	5	1	1	0	1	2.0	2.0	0.5	2.0	1	0.0
4	7	1	1	0	1	0.0	0.0	0.0	0.0	1	0.0

6. Separate features from target and save them to X and y

In [34]:

```
# get X
X_train = users_data.drop(['passed_course'], axis=1)

# get y
y_train = users_data['passed_course'].map(int)
```

7. Create a test df with the same features as our train df

The test dfs we have already contains only the data about the first 2 days for each user. So, no need to cut it. The only thing we need is to combine all the features that we have in train dfs.

In [35]:

```
events_data_test.head()
```

Out[35]:

	step_id	timestamp	action	user_id
0	30456	1526893787	viewed	24417
1	30456	1526893797	viewed	24417
2	30456	1526893954	viewed	24417
3	30456	1526895780	viewed	24417
4	30456	1526893787	discovered	24417

In [36]:

```
submission_data_test.head()
```

Out[36]:

	step_id	timestamp	submission_status	user_id
0	31971	1526800961	wrong	24370
1	31971	1526800976	wrong	24370
2	31971	1526800993	wrong	24370
3	31971	1526801054	correct	24370
4	31972	1526800664	wrong	24370

In [37]:

```
# user actions
users_events_data_test = pd.pivot_table(data=events_data_test,
                                   values='step_id',
                                   index='user_id',
                                   columns='action',
                                   aggfunc='count',
                                   fill_value=0) \
                                   .reset_index() \
                                   .rename_axis('', axis=1)
```

In [38]:

```
# correct/wrong answers
users_scores_test = pd.pivot_table(data=submission_data_test,
                                   values='step_id',
                                   index='user_id',
                                   columns='submission_status',
                                   aggfunc='count',
                                   fill_value=0) \
                                   .reset_index() \
                                   .rename_axis('', axis=1)

# add column with correct/wrong answers ratio
users_scores_test['correct_ratio'] = (users_scores_test.correct / (users_scores_test.cor
```

In [39]:

```
# number of steps that user tried to pass
users_steps_tried_test = submission_data_test.groupby('user_id', as_index=False) \
    .step_id.nunique() \
    .rename(columns={'step_id': 'steps_tried'})
```

In [40]:

```
# combine all together
users_data_test = pd.merge(users_events_data_test, users_scores_test,
                      on='user_id',
                      how='outer').fillna(0)

users_data_test = users_data_test.merge(users_steps_tried_test, how='outer').fillna(0)
```

In [41]:

```
users_data_test.head()
```

Out[41]:

	user_id	discovered	passed	started_attempt	viewed	correct	wrong	correct_ratio	steps_tried
0	4	1	1	0	1	0.0	0.0	0.000000	0.0
1	6	1	1	0	1	0.0	0.0	0.000000	0.0
2	10	2	2	0	6	0.0	0.0	0.000000	0.0
3	12	11	9	4	14	1.0	0.0	1.000000	1.0
4	13	70	70	35	105	29.0	36.0	0.446154	29.0

In [42]:

```
# Nothing lost
```

```
users_data_test.user_id.nunique() == events_data_test.user_id.nunique()
```

Out[42]:

```
True
```

In [43]:

```
# time features
# add columns with dates
events_data_test['date'] = pd.to_datetime(events_data_test['timestamp'],
                                          unit='s')
events_data_test['day'] = events_data_test['date'].dt.date

# create a table with users first/last actions and number of unique days spend on the co
users_time_feature_test = events_data_test.groupby('user_id') \
    .agg({'timestamp': ['min', 'max'], 'day': 'nunique'}) \
    .droplevel(level=0, axis=1) \
    .rename(columns={'nunique': 'days'}) \
    .reset_index()

# add column with a difference between first and last action = time user spent on the co
users_time_feature_test['hours'] = round((users_time_feature_test['max'] - users_time_fe

# drop 'min' and 'max' columns - we keep only time spent in hours
users_time_feature_test = users_time_feature_test.drop(['max', 'min'], axis=1)
```

In [44]:

```
# combine base features with time features
users_data_test = users_data_test.merge(users_time_feature_test, how='outer')
```

In [45]:

```
# so this would be our X_test
X_test = users_data_test
```

8. Model Training

In [46]:

```
# finding the best parameters for a Random Forest model,
# training on our train data

X_tr, X_te, y_tr, y_te = train_test_split(X_train, y_train, test_size=0.20, random_state

params = {'n_estimators': range(20, 51, 3), 'max_depth': range(5, 15)}

clf = RandomForestClassifier()

grid_clf = GridSearchCV(clf, param_grid=params, cv=5, n_jobs=-1)
grid_clf.fit(X_tr, y_tr)

print(f'Best parameters: {grid_clf.best_params_}')

ypred_prob = grid_clf.predict_proba(X_te)
roc_score = roc_auc_score(y_te, ypred_prob[:,1])
score = grid_clf.score(X_te, y_te)

print(f'Accuracy on a training dataset: {score:.2f}')
print(f'Roc score: {roc_score}')
```

```
Best parameters: {'max_depth': 7, 'n_estimators': 35}
Accuracy on a training dataset: 0.91
Roc score: 0.8765632311346764
```

9. Run the model on our test data for predictions

In [59]:

```
# Run the model with best parameters
X_tr, X_te, y_tr, y_te = train_test_split(X_train, y_train, test_size=0.20, random_state
clf_best = RandomForestClassifier(max_depth=7, n_estimators=35, random_state=42)
clf_best.fit(X_tr, y_tr)
```

```
ypred_prob = clf_best.predict_proba(X_te)

roc_score = roc_auc_score(y_te, ypred_prob[:,1])
score = clf_best.score(X_te, y_te)

print(f'Accuracy on the train dataset: {score:.3f}')
print(f'Roc-auc score on the train dataset: {roc_score:.5f}')

feature_importances = clf_best.feature_importances_
feature_importances_df = pd.DataFrame({'features': list(X_train),
                                       'feature_importances': feature_importances}) \
                                        .sort_values('feature_importances', ascending=Fa

ypred_prob_final = clf_best.predict_proba(X_test)
```

```
Accuracy on the train dataset: 0.904
Roc-auc score on the train dataset: 0.87655
```

In [53]:

```
feature_importances_df
```

Out[53]:

	features	feature_importances
8	steps_tried	0.301864
5	correct	0.257177
2	passed	0.096773
3	started_attempt	0.082460
7	correct_ratio	0.054779
10	hours	0.051610
4	viewed	0.044872
1	discovered	0.036833
0	user_id	0.032811
6	wrong	0.030943
9	days	0.009879

10. Make predictions on the test dataset

In [54]:

```
result = X_test['user_id'].to_frame().sort_values('user_id')
result['is_gone'] = ypred_prob_final[:,1]
```

In [55]:

```
result.head()
```

Out[55]:

	user_id	is_gone
0	4	0.000038
1	6	0.000038
2	10	0.000038
3	12	0.072936
4	13	0.662574

In [56]:

```
#result.to_csv(f'my_predict_{roc_score:.5f}.csv', index=False)
print(f'Results saved in my_predict_{roc_score:.5f}.csv')
```

```
Results saved in my_predict_0.87655.csv
```

In [ ]:

```

```