Recommendation Engine

Fateme Rahimi

July 26, 2020

Abstract

Online judges provide a platform where many users solve problems every day to improve their programming skills. The users can be beginners or experts in competitive programming. Some users might be good at solving specific category of problems (e.g. Greedy, Graph algorithms, Dynamic Programming etc.) while others may be beginners in the same. There can be patterns to everything, and the goal of the machine learning would be to identify these patterns and model user's behavior from these patterns.

1 Introduction

The goal of this project is to predict range of attempts a user will make tosolve a given problem given user and problem details. Finding these patternscan help the programming committee, as it will help them to suggest relevantproblems to solve and provide hints automatically on which users can get stuck.

2 Dataset and Features

We use data set that available in Analytics Vidhya web site that contained 3 training data :

- **train − submissions**:Contains 3 columns ('user_id', 'problem_id', 'attempts_range'). The variable 'attempts-range' denoted the range no. in which attempts the user made to get the solution.
- **user − data**, This is the file containing data of users.It contains the following features :
    1. user_id - unique ID assigned to each user
    2. submission_count - total number of user submissions
    3. problem_solved - total number of accepted user submissions
    4. contribution - user contribution to the judge
    5. country - location of user
    6. follower_count - amount of users who have this user in followers
    7. last_online_time_seconds - time when user was last seen online
    8. max_rating - maximum rating of user

9. rating - rating of user
10. rank - can be one of 'beginner' ,'intermediate' , 'advanced', 'expert'
11. registration_time_seconds - time when user was registered

- **problem** − **data**, This is the file containing data of the problems.It contains the following features :
    1. problem_id - unique ID assigned to each problem
    2. level_id - the difficulty level of the problem between 'A' to 'N'
    3. points - amount of points for the problem
    4. tags - problem tag(s) like greedy, graphs, DFS etc.

First with k-mean algorithm we cluster user data, then merged this labeled data with train submission and problem data, and with Random forest algorithm predict attempt range.

3 Import Libraries

[152]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import re
```

4 Import Data

[153]:
```
TrainData=pd.read_csv("train_submissions.csv")
ProblemData=pd.read_csv("problem_data.csv")
UserData=pd.read_csv("user_data.csv")
```

5 preproccesing

[154]:
```
TrainData.isnull().sum()
```

[154]:
```
user_id           0
problem_id        0
attempts_range    0
dtype: int64
```

[155]:
```
UserData.isnull().sum()
```

[155]:
```
user_id                      0
submission_count             0
```

```
problem_solved                   0
contribution                     0
country                       1153
follower_count                   0
last_online_time_seconds         0
max_rating                       0
rating                           0
rank                             0
registration_time_seconds        0
dtype: int64
```

[156]:
```python
ProblemData.isnull().sum()
```

[156]:
```
problem_id       0
level_type     133
points        3917
tags          3484
dtype: int64
```

5.1 Imputing miss value

Strategy for imputing the null values will be based on the ratio of occurence of the countries in the rest of the data. For example, India occured 25.6% and Bangladesh occured 13.6% and so on. We will use this ratio of all the countries to fill the missing data.

[157]:
```python
#Getting all the ratios
country_data = (UserData["country"].value_counts()/UserData["country"].count())
#imputing missing values
UserData["country"]= UserData["country"].fillna(pd.Series(np.random.
 ↪choice(country_data.index,p=country_data.values, size=len(UserData))))
```

[158]:
```python
level_type_data = (ProblemData["level_type"].value_counts()/
 ↪ProblemData["level_type"].count())
ProblemData["level_type"]= ProblemData["level_type"].fillna(pd.Series(np.random.
 ↪choice(level_type_data.index,p=level_type_data.values, size=len(ProblemData))))
ProblemData["tags"].fillna("other", inplace=True)
ProblemData.fillna(ProblemData.mean(), inplace=True)
```

5.2 Encoding categorical data

[159]:
```python
TrainData.dtypes
```

[159]:
```
user_id           object
problem_id        object
attempts_range     int64
dtype: object
```

[160]:
```
TrainData['user_id'] = TrainData['user_id'].apply(lambda x: re.search(r'\d+', x).
 ↪group()).astype(int)
TrainData['problem_id'] = TrainData['problem_id'].apply(lambda x: re.
 ↪search(r'\d+', x).group()).astype(int)
```

[161]:
```
UserData.dtypes
```

[161]:
```
user_id                       object
submission_count               int64
problem_solved                 int64
contribution                   int64
country                       object
follower_count                 int64
last_online_time_seconds       int64
max_rating                   float64
rating                       float64
rank                          object
registration_time_seconds      int64
dtype: object
```

[162]:
```
UserData['user_id'] = UserData['user_id'].apply(lambda x: re.search(r'\d+', x).
 ↪group()).astype(int)
UserData["rank"]=UserData["rank"].
 ↪replace(['beginner','intermediate','advanced','expert'],[1,2,3,4])
UserData['country'] = UserData['country'].astype('category')
UserData['country'] = UserData['country'].cat.codes
```

[163]:
```
ProblemData.dtypes
```

[163]:
```
problem_id     object
level_type     object
points        float64
tags           object
dtype: object
```

[164]:
```
ProblemData["level_type"]=ProblemData["level_type"].replace(['A'␣
 ↪,"B","C","D","E","F","G","H","I","J","K","L","M","N"],
                                                              [␣
 ↪1,2,3,4,5,6,7,8,9,10,11,12,13,14])
ProblemData['problem_id'] = ProblemData['problem_id'].apply(lambda x: re.
 ↪search(r'\d+', x).group()).astype(int)
ProblemData['tags'] = ProblemData['tags'].astype('category')
ProblemData['tags'] =ProblemData['tags'].cat.codes
```

5.3 Scaling some feature

[165]:
```
UserData["contribution"]=(UserData["contribution"]-UserData["contribution"].
↪min())/UserData["contribution"].max()
ProblemData['points']=(ProblemData['points']-ProblemData['points'].min())/
↪(ProblemData['points'].max()-ProblemData['points'].min())
```

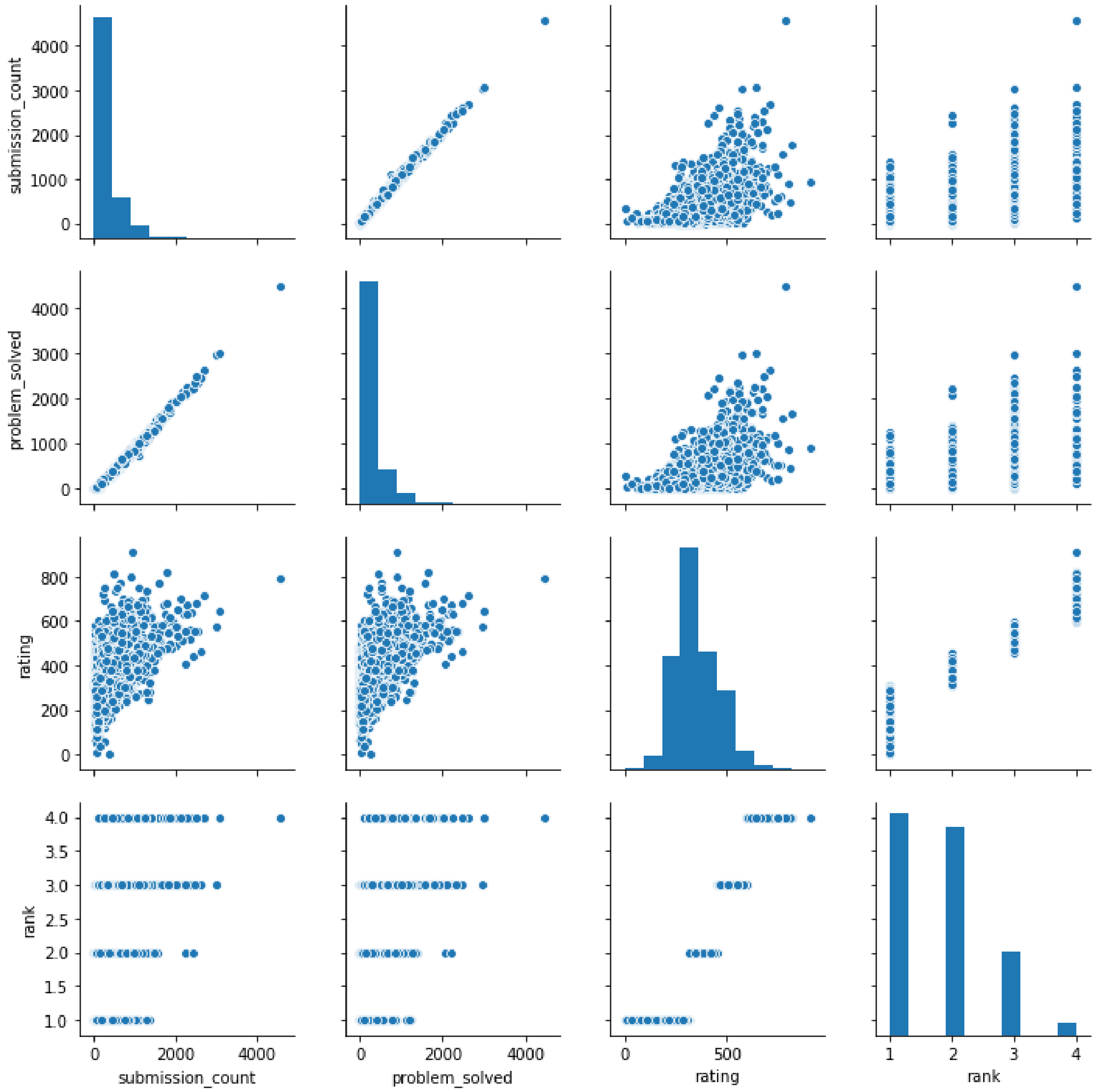
5.4 add new feature

We add the new feature as duration(days) that user is in the platform, and drop two related feature.

[166]:
```
UserData["duration"]=(UserData['registration_time_seconds']-UserData['last_online_time_seconds'
↪(24*3600)
UserData=UserData.
↪drop(["registration_time_seconds","last_online_time_seconds"],axis=1)
```

5.5 Plotting

[167]:
```
ax = sns.pairplot(UserData[["submission_count", "problem_solved", "rating",␣
↪"rank"]])
```



So what do the above pair plots tell us?

1.Higher number of submissions were made by fewer number of users

2.Higher count of problems were solved by fewer number of users

3.The rating of users is uniformly spread, and most are with a rating somewhere in the middle

4.Majority users are at an intermediate and beginner level, with very few experts

5.Rating is not directly proportional to the number of problems solved or submissions - this means that difficulty level should have played a part.

6 Multiclass Classification

6.1 Merging Files

First we have merged three files with user id and problem id:

[168]:
```
df = pd.merge(TrainData,UserData,how = 'left',on = "user_id")
df = pd.merge(df,ProblemData,how = 'left',on = "problem_id")
X=df.drop("attempts_range",axis=1)
y=df["attempts_range"]
```

6.2 Random Forest

[173]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
forest_clf1 = RandomForestClassifier(n_estimators = 100, criterion = 'entropy',␣
 ↪random_state = 84)
forest_clf1.fit(X, y)
```

[173]:
```
RandomForestClassifier(criterion='entropy', random_state=84)
```

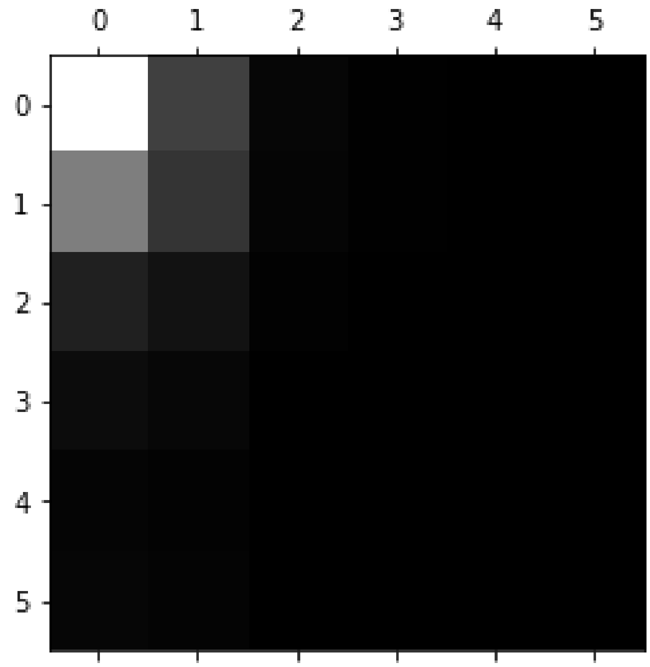
6.2.1 Validation

[174]:
```
cross_val_score(forest_clf1, X, y, cv=5, scoring="accuracy")
```

[174]:
```
array([0.50194791, 0.50548955, 0.50278502, 0.5029782 , 0.5060047 ])
```

6.2.2 Confusion matrix

[175]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
y_pred1 = cross_val_predict(forest_clf1, X, y, cv=3)
conf_mx1 = confusion_matrix(y, y_pred1)
plt.matshow(conf_mx1, cmap=plt.cm.gray)
plt.show()
```



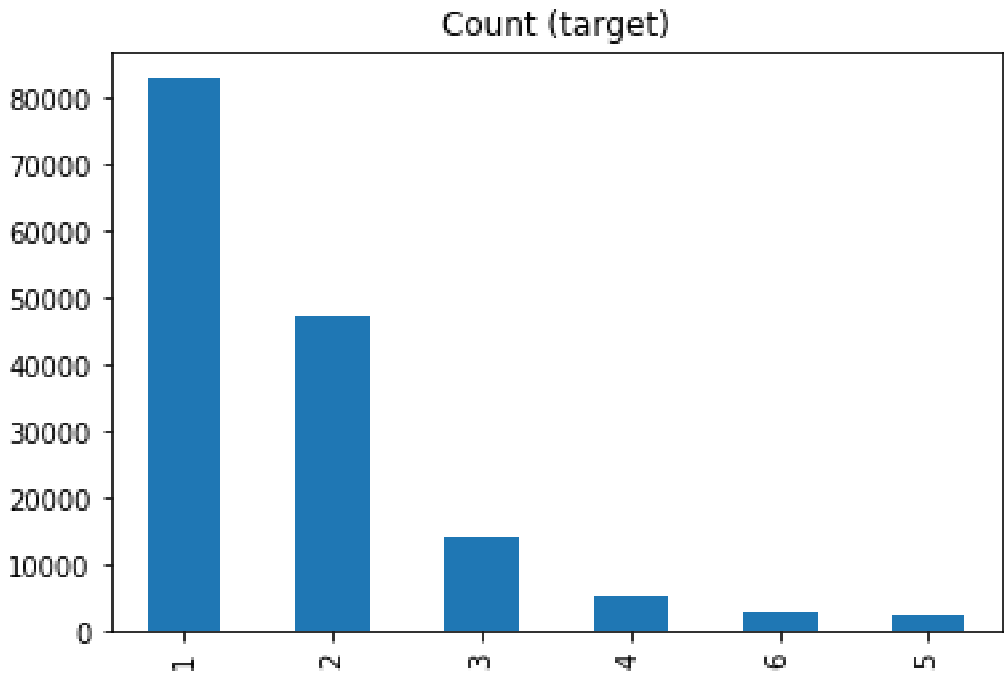
As you can see, the accuracy of the model is around 50%, and also considering the confusion matrix, it is clear that this classification can only correctly determine class 1 and to some extent class 2, to solve this problem and improve the model we use resampling.

7 Resampling

why we need resampling? First we calculate the number of samples in each class.

[176]:
```
target_count = df.attempts_range.value_counts()
target_count.plot(kind='bar', title='Count (target)')
```

[176]: <matplotlib.axes._subplots.AxesSubplot at 0x1354ae148>



As you can see, most of the samples are in Class 1, and we have an imbalanced classification problem that we can use resampling to balance that.

[177]:
```
from sklearn.utils import resample
```

[178]:
```
class6 = df[df.attempts_range==6]
class5 = df[df.attempts_range==5]
class4 = df[df.attempts_range==4]
class3 = df[df.attempts_range==3]
class2 = df[df.attempts_range==2]
class1 = df[df.attempts_range==1]
k=len(class2)
```

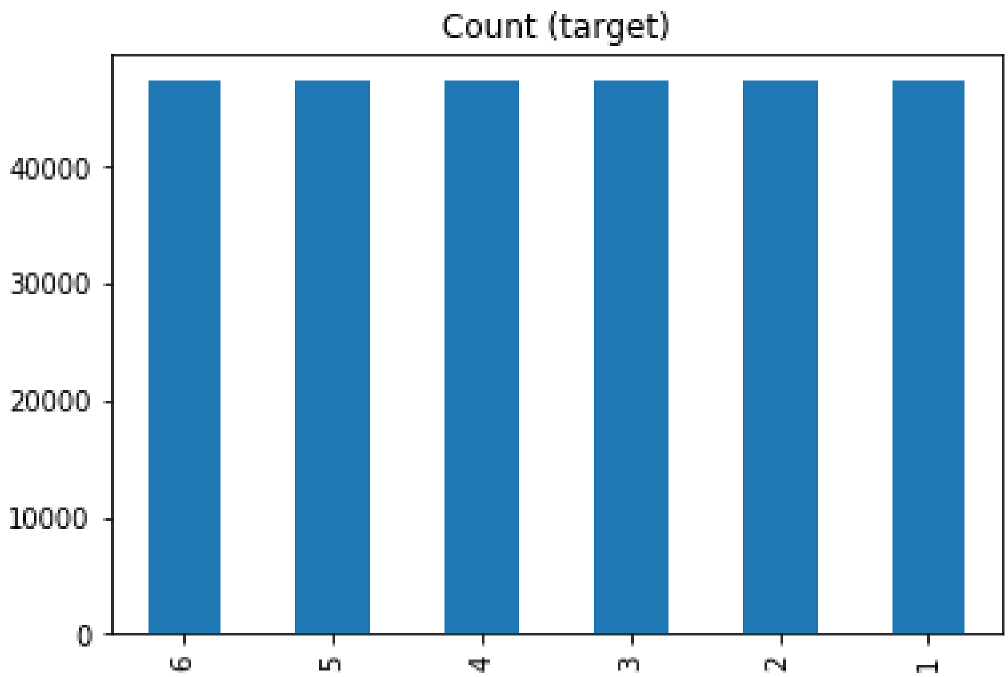
7.1 Oversampling Minority Class

We equalize the number of instances of all classes by undersampling the number of instances of class 1 and increasing the number of instances of the other classes by oversampling.

[179]:
```
class6_upsampled = resample(class6,replace=True,n_samples=k,random_state=27)
class5_upsampled = resample(class5,replace=True,n_samples=k,random_state=27)
class4_upsampled = resample(class4,replace=True,n_samples=k,random_state=27)
class3_upsampled = resample(class3,replace=True,n_samples=k,random_state=27)
class1_undersampled = resample(class1,replace=True,n_samples=k,random_state=27)
df2= pd.
 ↪concat([class2,class1_undersampled,class3_upsampled,class4_upsampled,class5_upsampled,class6_
```

[180]:
```python
target_count = df2.attempts_range.value_counts()
target_count.plot(kind='bar', title='Count (target)')
```

[180]: <matplotlib.axes._subplots.AxesSubplot at 0x134fbc9c8>



Now the number of each class is the equal.

7.2 Random Forest

[181]:
```python
X=df2.drop(['attempts_range'],axis=1)
y=df2["attempts_range"]
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(X, y)
```

[181]: RandomForestClassifier(random_state=42)

7.2.1 Validation

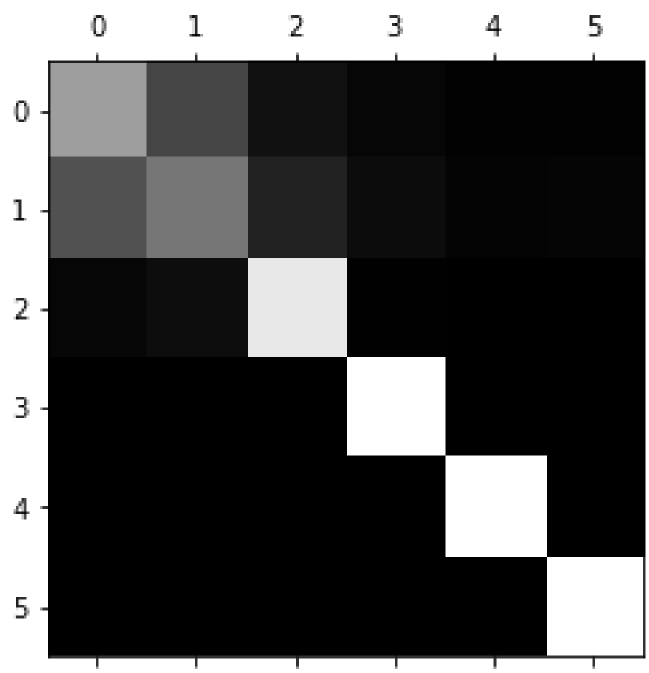
[182]:
```python
from sklearn.model_selection import cross_val_score
cross_val_score(forest_clf, X, y, cv=5, scoring="accuracy")
```

[182]: array([0.84705199, 0.84714004, 0.84719287, 0.84742181, 0.84611862])

7.2.2 Confusion Matrix

[183]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
y_pred = cross_val_predict(forest_clf, X, y, cv=3)
conf_mx = confusion_matrix(y, y_pred)
print(conf_mx)
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
```

```
[[29244 12768  3321  1135   402   450]
 [14991 21864  6340  2251   839  1035]
 [ 1269  2488 42940   329   117   177]
 [   30    81    27 47174     0     8]
 [    0     0     0     0 47320     0]
 [    0     0     0     0     0 47320]]
```



8 Prediction in test set

[184]:
```
TestData=pd.read_csv("test_submissions.csv")
TestData['user_id'] = TestData['user_id'].apply(lambda x: re.search(r'\d+', x).
 ↪group()).astype(int)
TestData['problem_id'] = TestData['problem_id'].apply(lambda x: re.
 ↪search(r'\d+', x).group()).astype(int)
```

[185]:
```
testdf = pd.merge(TestData,UserData,how = 'left',on = "user_id")
testdf = pd.merge(testdf,ProblemData,how = 'left',on = "problem_id")
testdf=testdf.drop("ID",axis=1)
testdf["attempts_range"]=forest_clf.predict(testdf)
```

9 Submission in analyticsvidhya

Finally, we uploaded the prediction file of the test set on the site and got a score of 0.41. The highest score on this site is 0.50, and it seems that this score is not bad.