CSE258_HW3_lu

November 13, 2019

1 Task 1 : Read Prediction

1.1 1: Evaluate the performance (accuracy) of the baseline model on the validation set you have built

[79]:
```
import gzip
from collections import defaultdict
import random

def readGz(path):
  for l in gzip.open(path, 'rt'):
    yield eval(l)

def readCSV(path):
  f = gzip.open(path, 'rt')
  f.readline()
  for l in f:
    yield l.strip().split(',')

### Rating baseline: compute averages for each user, or return the global␣
↪average if we've never seen the user before
allRatings = []
allReview = []
userRatings = defaultdict(list)
allBooks = []

for user,book,r in readCSV("train_Interactions.csv.gz"):
  r = int(r)
  allRatings.append(r)
  userRatings[user].append(r)
  allReview.append([user,book,r])
  if book not in allBooks: allBooks.append(book)

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
  userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
```

1.1.1 Separate our data into Train and Validation

[80]:
```
reviewTrain = allReview[:190000]
reviewVal = allReview[190000:]
```

1.1.2 Modify the baseline so it depends on the training set we have

[81]:
```
### Would-read baseline: just rank which books are popular and which are not,␣
↪and return '1' if a book is among the top-ranked
bookCount = defaultdict(int)
totalRead = 0

for user,book,_ in reviewTrain:
  bookCount[book] += 1
  totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
for ic, i in mostPopular:
  count += ic
  return1.add(i)
  if count > totalRead/2: break
```

1.1.3 Prepare a negative-entry for our validation set

[82]:
```
## build the dictionary where we could find the read book and hasnt read book
userRatingsTrain = defaultdict(set)
for (user,book,r) in reviewTrain:
    userRatingsTrain[user].add(book)
print(len(userRatingsTrain))
print(userRatingsTrain['u79354815'])

userRatingsVal = defaultdict(set)
for (user,book,r) in reviewVal:
    userRatingsVal[user].add(book)
print(len(reviewVal))
print(len(userRatingsVal))
```

```
11357
{'b50552757', 'b56923147', 'b14275065', 'b66134745', 'b03661054', 'b10241989',
'b19841474', 'b73351796', 'b37649900', 'b78541130', 'b28542052'}
10000
6288
```

[83]:
```python
# build a list called readPredictVal to use as our validation set to test our␣
↪read baseline
readPredictVal = []
for (user,book,r) in reviewVal:
    h = random.choice([k for k in allBooks if k not in userRatings[user]])
    readPredictVal.append((user,book,r))
    readPredictVal.append((user,h,-1))
print(len(readPredictVal))
```

```
20000
```

1.1.4 Test the accuracy on our validation set with random book users haven't read

[84]:
```python
predictions = open("predictions_Read.txt", 'w')
tpredict = 0
totalpredict = 0
for (u,b,r) in readPredictVal:
    totalpredict += 1
    if r >= 0:
        if b in return1:
            predictions.write(u + '-' + b + ",1\n")
            tpredict += 1
        else:
            predictions.write(u + '-' + b + ",0\n")
    else:
        if b in return1:
            predictions.write(u + '-' + b + ",1\n")
        else:
            predictions.write(u + '-' + b + ",0\n")
            tpredict += 1
predictions.close()

accVal = tpredict/totalpredict
print('Accuracy of read prediction on the validation set',accVal)
print('Total books predicted',totalpredict)
```

```
Accuracy of read prediction on the validation set 0.64405
Total books predicted 20000
```

1.2 2: See if you can find a better threshold and report its performance on your validatin set

1.2.1 Modify our baseline model to become a function that we could modify the percentage of popular standard

[85]:
```
# input: the percentage of popular standard percentage
# output: the book set that are the most p% popular
def wouldRead(p):
    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalRead*(p): break
    return return1
```

1.2.2 Define a function for predicting accuracy calculate

[86]:
```
# input: the percentage of popular standard percentage
# output: the book set that are the most p% popular
def accCalculator(p):
    bookset = wouldRead(p)
    tpredict = 0
    totalpredict = 0
    for (u,b,r) in readPredictVal:
        totalpredict += 1
        if r > 0:
            if b in bookset:
                tpredict += 1
        else:
            if b not in bookset:
                tpredict += 1
    accVal = tpredict/totalpredict
    return accVal
accCalculator(0.5)
```

[86]:
```
0.6492
```

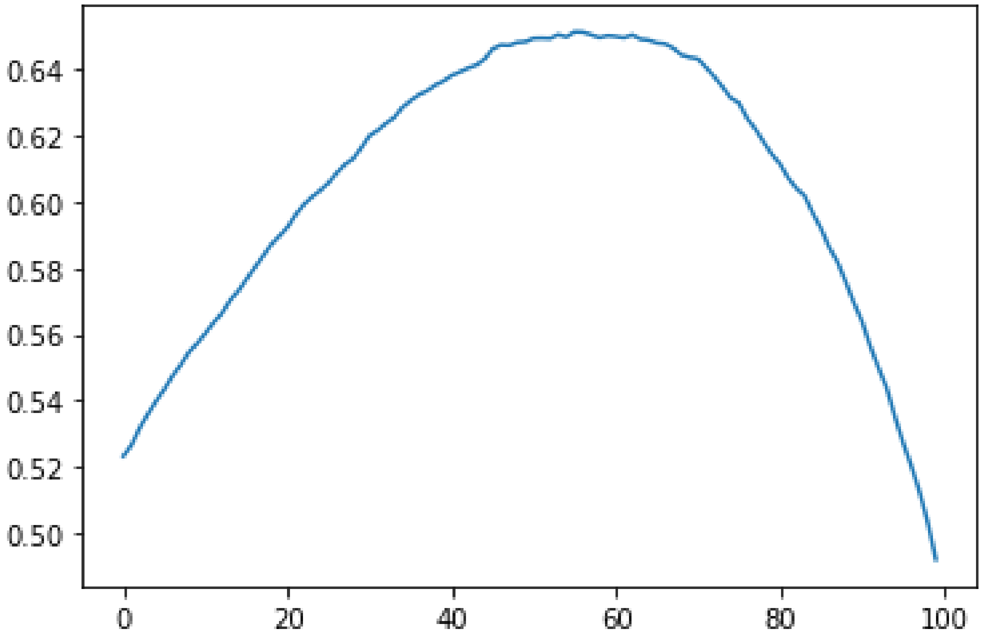
1.2.3 Plot a graph for our predicting accuracy result

[87]:
```
plodata = []
for i in range(100):
    plodata.append(accCalculator(i*0.01))

from matplotlib import pyplot as plt
plt.plot(plodata)
```

```
plt.show
print('most popular of',plodata.index(max(plodata)),'percent, with an accuracy␣
↪of',max(plodata))
```

```
most popular of 55 percent, with an accuracy of 0.65115
```



1.3 3: Use Jaccard similarity to conduct a read prediction

[88]:
```
### build our dictionary for each book in the training set
bookReadbyTrain = defaultdict(set)
for (user,book,r) in reviewTrain:
    bookReadbyTrain[book].add(user)
```

1.3.1 Define a function for calculate the similartiy over two books

[89]:
```
def jaccardSim(b1,b2):
    usersforb1 = bookReadbyTrain[b1]
    usersforb2 = bookReadbyTrain[b2]
    if len(usersforb1) == 0 and len(usersforb2) == 0:
        return 0
    else:
        numer = len(set(usersforb1).intersection(set(usersforb2)))
        denom = len(set(usersforb1).union(set(usersforb2)))
```

```
        return numer/denom
```

1.3.2 Define a function for predict the read status for a single pair of (user , book)

[91]:
```
def simJaccard(user,book):
    if user not in userRatingsTrain:
        return 0
    if book not in bookReadbyTrain:
        return 0
    simmax = 0
    for b in userRatingsTrain[user]:
        if b == book: continue
        simmax = max(jaccardSim(b,book),simmax)
    return simmax
```

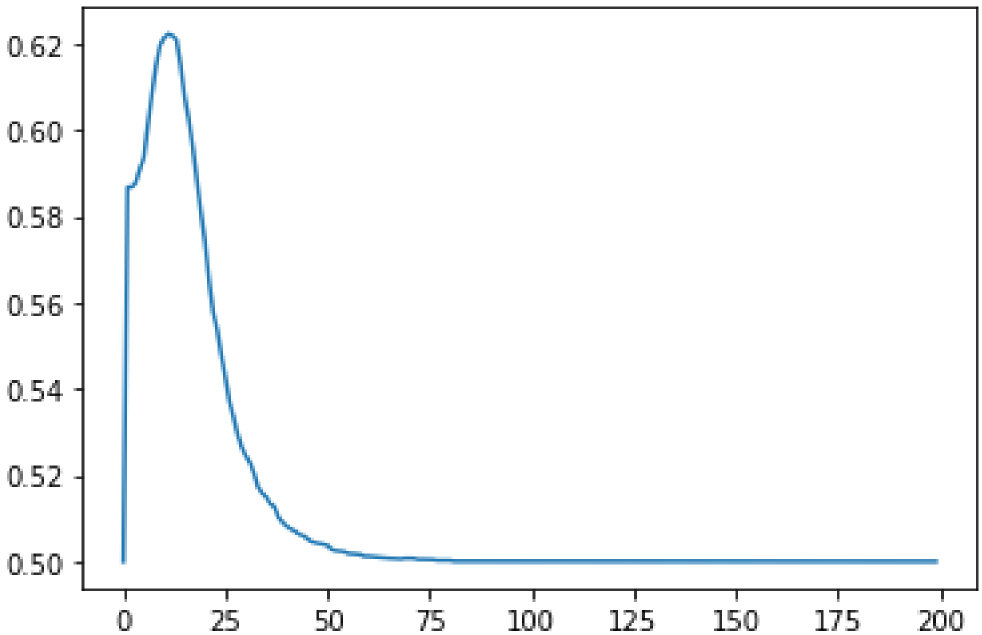
[92]:
```
totalpred = len(readPredictVal)
simmax = []
readPredictVal1 = []
for u, b, r in readPredictVal:
    simmax.append(simJaccard(u,b))
    if r >= 0:
        readPredictVal1.append(1)
    else:
        readPredictVal1.append(0)
```

1.3.3 Plot a graph for our prediction result with different threshold from 0 to 99 with Jaccard sim on validation set

after I tried with p from 0 to 100%, I noticed that when the threshold goes upper than 15%, the Jaccard Similarity stays the same. So we are only going to try in a more specific way

[93]:
```
predjaccard = []
acc = []
for i in range(200):
    predjaccard = [1 if simmax[j] >= i*0.001 else 0 for j in range(len(simmax))]
    acc.append(sum([i == j for i,j in zip(predjaccard,readPredictVal1)])/
 ↪len(readPredictVal1))
plt.plot(acc)
plt.show
print('max similarity of',
      0.001*acc.index(max(acc)),
      'percent similarity, with an accuracy of',max(acc))
```

```
max similarity of 0.011 percent similarity, with an accuracy of 0.62225
```



1.4 4. Incorporate two of our features to become a better model

[94]:
```
import numpy as np
```

[95]:
```
from sklearn import linear_model
```

[96]:
```
model = linear_model.LogisticRegression()
```

1.4.1 Use our validation set to select the best parameters for the classifier based on previous two models

[420]:
```
## data set we are going to use:
import warnings
warnings.filterwarnings('ignore')
def dataPrep(pop,per):
    predPop = [1 if b in wouldRead(pop) else 0 for u,b,r in readPredictVal]
    predjaccard = [1 if simmax[j] >= per else 0 for j in range(len(simmax))]
    return (predPop,predjaccard)

y = readPredictVal1
colOf1 = [1]*len(readPredictVal1)
accIncorModel = []
for pop in range(53,57):
    for per in range(9,12):
        #print(pop,per)
        predPop = dataPrep(0.01*pop,0.001*per)[0]
```

```
        predjaccard = dataPrep(0.01*pop,0.001*per)[1]
        X = np.column_stack((colOf1,predPop,predjaccard))
        model.fit(X, y)
        predictions = model.predict(X)
        correctPredictions = predictions == y
        accIncorModel.append((pop,per,sum(correctPredictions)/
 ↪len(correctPredictions)))
```

[421]:
```
print(accIncorModel)
```

```
[(53, 9, 0.6482), (53, 10, 0.64635), (53, 11, 0.64635), (54, 9, 0.64955), (54,
10, 0.64655), (54, 11, 0.64655), (55, 9, 0.65265), (55, 10, 0.6487), (55, 11,
0.6487), (56, 9, 0.6537), (56, 10, 0.649), (56, 11, 0.649)]
```

[422]:
```
accTn = [k for (p,q,k) in accIncorModel]
indexMax = accTn.index(max(accTn))
print('the parameters we choose',
      accIncorModel[indexMax],
      'with an accuracy of',max(accTn))
```

```
the parameters we choose (56, 9, 0.6537) with an accuracy of 0.6537
```

I tried several ways to incorporate two models, including have their union/intersection, with several paramaters input. This is after modifed since I tried different way to predict and try them on Kaggle

[431]:
```
predictionsLu = open("predictions_Read_Lu.txt", 'w')
for l in open("pairs_Read.txt"):
    if l.startswith("userID"):
        #header
        predictionsLu.write(l)
        continue
    u,b = l.strip().split('-')
    poppred = 1 if b in wouldRead(0.56) else 0
    Japred = 1 if simJaccard(u,b) >= 0.011 else 0
    out = str(1 if Japred == 1 else 0)
    predictionsLu.write(u + '-' + b + ","+ out +"\n")
predictionsLu.close()
```

[425]:
```
print('My Kaggle team name for this rating contest is PSG.LGD.LuNova with my␣
 ↪name Lu0321')
```

```
My Kaggle team name for this rating contest is PSG.LGD.LuNova with my name
Lu0321
```

2 Task 2 : Rating prediction

2.1 9. Fit a predictor of given form and use a regularization parameter of = 1. Report the MSE on the validation set (1 mark).

2.1.1 Prepare our data

[418]:
```
# as we used before, the traingin set contains 1-190,000 reviews, with the␣
↪validation set contains 190,001-200,000
reviewTrain
reviewVal
print(len(reviewTrain))
print(len(reviewVal))
```

```
190000
10000
```

[366]:
```
Usersreadi = defaultdict(set)
ItemRatebyu = defaultdict(set)

Usersreadi_count = defaultdict(list)
ItemRatebyu_count = defaultdict(list)

for u,b,r in reviewTrain:
    Usersreadi[b].add(u)
    ItemRatebyu[u].add(b)
book_list = list(Usersreadi.keys())
user_list = list(ItemRatebyu.keys())
```

[367]:
```
UsersreadiAvg = defaultdict(list)
ItemRatebyuAvg = defaultdict(list)

rateAfterReadi = defaultdict(list)
ratebyUser = defaultdict(list)

for u,b,r in reviewTrain:
    rateAfterReadi[b].append(r)
    ratebyUser[u].append(r)

for b in rateAfterReadi:
    UsersreadiAvg[b] = mean(rateAfterReadi[b])

for u in ratebyUser:
    ItemRatebyuAvg[u] = mean(ratebyUser[u])

globalAvgTrain = mean([r for u,b,r in reviewTrain])
```

[368]:
```
globalAvgTrain
```

[368]: 3.897121052631579

[369]:
```
# set up two dictionaries for count the amount of books for each user and each␣
↪book
Usersreadi_count = defaultdict(list)
ItemRatebyu_count = defaultdict(list)
```

[370]:
```
for b in Usersreadi:
    Usersreadi_count[b] = len(Usersreadi[b])
for u in ItemRatebyu:
    ItemRatebyu_count[u] = len(ItemRatebyu[u])
```

[371]:
```
print(len(Usersreadi))
print(len(ItemRatebyu))
```

```
7169
11357
```

[372]:
```
# set up our staring dictionary for betas
beta_i = defaultdict(float)
beta_u = defaultdict(float)
```

[373]:
```
for b in Usersreadi:
    beta_i[b] = 2
for u in ItemRatebyu:
    beta_u[u] = 2
print(len(beta_i))
print(len(beta_u))
```

```
7169
11357
```

[374]:
```
TrainFrame = pd.DataFrame(reviewTrain,columns=['user','book','rate'])
s = 0
s_ItemRatebyu = defaultdict(int)
s_Usersreadi = defaultdict(int)
for i in range(len(TrainFrame)):
    r = TrainFrame.iloc[i]
    if r[1] not in s_Usersreadi:
        s_Usersreadi[r[1]] = int(r[2])
    else:
        s_Usersreadi[r[1]] = s_Usersreadi[r[1]] + int(r[2])
    if r[0] not in s_ItemRatebyu:
        s_ItemRatebyu[r[0]] = int(r[2])
    else:
        s_ItemRatebyu[r[0]] = s_ItemRatebyu[r[0]] + int(r[2])
    s = s + int(r[2])
```

[375]:
```
def ratePredicting(a,betai,betau,lamda,times):
    for k in range(times):

        temp = 0
        for i in ItemRatebyu:
            temp = temp + betau[i]*ItemRatebyu_count[i]
        temp1 = 0
        for i in Usersreadi:
            temp1 = temp1 + betai[i]*Usersreadi_count[i]
        a = (total - temp - temp1)/len(TrainFrame)

        for i in ItemRatebyu:
            temp2 = 0
            for j in list(ItemRatebyu[i]):
                temp2 = temp2 + betai[j]
            betau[i] = (total_ItemRatebyu[i] - a*ItemRatebyu_count[i] - temp2)/
↪(lamda + ItemRatebyu_count[i])

        for i in Usersreadi:
            temp3 = 0
            for j in list(Usersreadi[i]):
                temp3 = temp3 + betau[j]
            betai[i] = (total_Usersreadi[i] - a*Usersreadi_count[i] - temp3)/
↪(lamda + Usersreadi_count[i])

    return a, betai, betau
```

[376]:
```
a,beta_i,beta_u = ratePredicting(2,beta_i,beta_u,1,400)
```

[378]:
```
a
```

[378]: 3.804997206562257

[379]:
```
beta_i['b14275065']
```

[379]: 0.07982214232076082

[380]:
```
beta_u['u79354815']
```

[380]: 0.007200183261631539

[381]:
```
def predRating(user,item):
    if item in beta_i and user in beta_u:
        y = a + beta_i[item] + beta_u[user]
    elif item not in beta_i and user in beta_u:
        y = ItemRatebyuAvg[user]
    elif item in beta_i and user not in beta_u:
        y = UsersreadiAvg[item]
    else:
        y = globalAvgTrain
    return y
```

[382]:
```
xVal = [[u,b] for u,b,r in reviewVal]
```

[383]:
```
yVal = [r for u,b,r in reviewVal]
```

[384]:
```
preR = [predRating(u,b) for u,b in xVal]
```

[385]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[386]:
```
MSE(preR,yVal)
```

[386]:
```
1.115954571083296
```

[387]:
```
print('MSE on the Validation set is: ',MSE(preR,yVal))
```

```
MSE on the Validation set is:  1.115954571083296
```

2.2 10. Report the user and book IDs that have the largest and smallest values of (1 mark).

for books

[388]:
```
key_list = list(beta_i.keys())
val_list = list(beta_i.values())
```

[389]:
```
max(val_list)
```

[389]:
```
1.4291517056125491
```

[390]:
```
booknamemx = key_list[val_list.index(max(val_list))]
```

[391]:
```
print('max of beta_i attained by book', booknamemx,'with max␣
↪beta_i',max(val_list))
```

```
max of beta_i attained by book b19925500 with max beta_i 1.4291517056125491
```

[392]:
```
booknamemx1 = key_list[val_list.index(min(val_list))]
```

[393]:
```
print('min of beta_i attained by book', booknamemx1,'with min␣
↪beta_i',min(val_list))
```

```
min of beta_i attained by book b84091840 with min beta_i -1.7548572177626496
```

for users

[394]:
```
key_listu = list(beta_u.keys())
val_listu = list(beta_u.values())
```

[395]:
```
max(val_listu)
```

[395]:
```
1.3234209748117745
```

[396]:
```
username = key_listu[val_listu.index(max(val_listu))]
```

[397]:
```
print('max of beta_u  attained by', username,'with max beta_u',max(val_listu))
```

```
max of beta_u  attained by u32162993 with max beta_u 1.3234209748117745
```

[398]:
```
username1 = key_listu[val_listu.index(min(val_listu))]
```

[399]:
```
print('min of beta_u  attained by', username1,'with min beta_u',min(val_listu))
```

```
min of beta_u  attained by u48313610 with min beta_u -3.746716329066186
```

2.3 11. Find a better value of using your validation set. Report the value you chose, its MSE, and upload your solution to Kaggle by running it on the test data (1 mark).

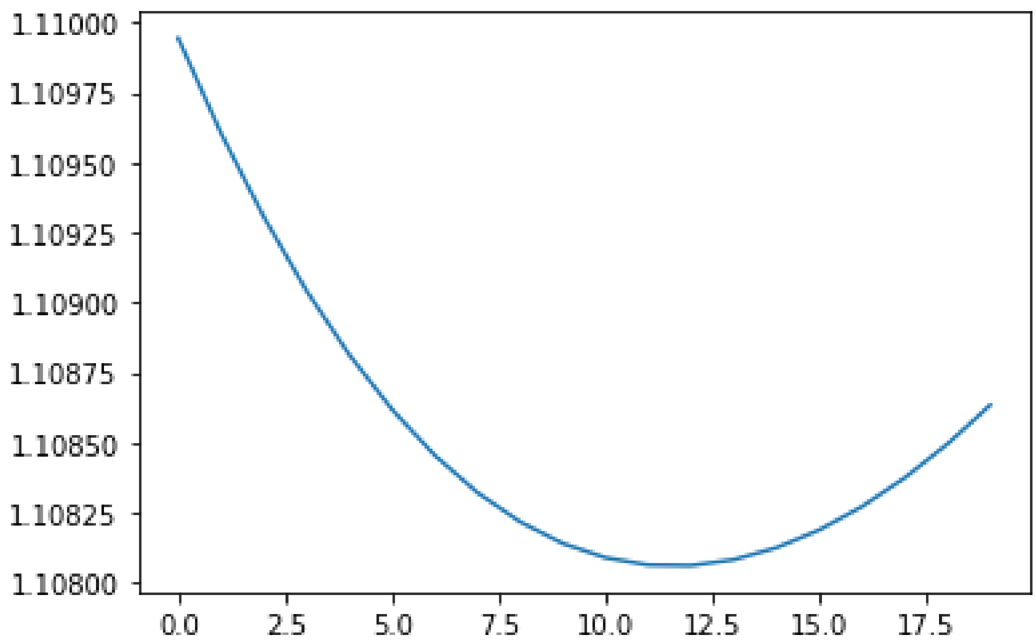
[413]:
```
MSElist = []
for lamda in range(20,40):
    a,beta_i,beta_u = ratePredicting(2,beta_i,beta_u,lamda*0.1,400)
    preR = [predRating(u,b) for u,b in xVal]
    MSElist.append(MSE(preR,yVal))
```

[414]:
```
MSElist
```

[414]:
```
[1.1099433236415106,
 1.1096049794606406,
 1.1093051163327652,
 1.1090418676304843,
 1.1088134767042521,
 1.1086182886118003,
 1.108454742633723,
 1.1083213654823691,
 1.1082167651247945,
 1.1081396251512856,
 1.1080886996304216,
 1.108062808399123,
 1.1080608327427375,
 1.1080817114258992,
 1.1081244370395802,
 1.1081880526336327,
 1.108271648607965,
 1.1083743598383722,
 1.1084953630155325,
 1.1086338741782735]
```

[419]:
```
from matplotlib import pyplot as plt
plt.plot(MSElist)
plt.show
print('min MSE attained at',
      0.1*(20+MSElist.index(min(MSElist))),
      'with a minimum MSE of',min(MSElist))
```

```
min MSE attained at 3.2 with a minimum MSE of 1.1080608327427375
```



[416]:
```
a,beta_i,beta_u = ratePredicting(2,beta_i,beta_u,3.2,400)
```

[417]:
```
predictionsR = open("predictions_Rating_Lu.txt", 'w')
for l in open("pairs_Rating.txt"):
    if l.startswith("userID"):
    #header
          predictionsR.write(l)
          continue
    u,b = l.strip().split('-')
    preR = predRating(u,b)
    predictionsR.write(u + '-' + b + ',' + str(preR) + '\n')
predictionsR.close()
```

[ ]:
```
print('My Kaggle team name for this rating contest is PSG.LGD.LuNova with my␣
 ↪name Lu0321')
```