"""

Model Description:

I have used a hybrid recommendation system. I have used the user.json, business.json, review\_train.json, checkin.json, photo.json, tip.json for the features of my model-based system. Initially, I tried using XGBoost, but despite tweaking, the best RMSE I could achieve was 0.9824, and it was pretty slow. So, I switched to CatBoost, which not only ran faster but also dropped my RMSE to 0.9791 after quite a lot of iterations.

Features:

user.json: review\_count, average\_stars, useful, funny, cool, fans, compliment\_hot, compliment\_more, compliment\_profile, compliment\_cute, compliment\_list, compliment\_note, compliment\_plain, compliment\_cool, compliment\_funny, compliment\_writer, compliment\_photos, yelping\_since

business.json: review\_count, stars, is\_open, RestaurantsPriceRange2, latitude, longitude

review\_train.json: useful, funny, cool, date

checkin.json: time values summed

tip.json: count, date

photo.json: count

RMSE:

0.979121575758248 (validation set)

0.977292987102194 (test set)

Error Distribution:

>=0 and <1: 102058

>=1 and <2: 33045

>=2 and <3: 6158

>=3 and <4: 780

>=4: 3

"""

from pyspark import SparkContext

import sys, time, json

import numpy as np

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import MinMaxScaler

from catboost import CatBoostRegressor

from datetime import datetime

# if the business is not rated by a user, it is given an average of all ratings by that user.

# if the user has never rated anything before, a default rating of 3.0 is assigned.

def missing(i, u):

if i not in busUser:

return userAverage[u]

if u not in userBus:

return 3.0

return None

# co-rated item to item collaborative filtering implementation of pearson correlation co-efficient w and predicted rating p.

def pearson(i, u):

missingData = missing(i, u)

if missingData is not None:

return missingData

wList = []

# pearson coefficient

for j in userBus[u]:

item = tuple(sorted((j, i)))

w = wDict.get(item, None)

if w is None:

common = busUser[i] & busUser[j]

if len(common) <= 1:

w = (5.0 - abs(businessAverage[i] - businessAverage[j])) / 5

else:

rui = []

ruj = []

for v in common:

rui.append(float(busUserRate[i][v]))

ruj.append(float(busUserRate[j][v]))

ri = sum(rui) / len(rui)

rj = sum(ruj) / len(ruj)

item1 = [x - ri for x in rui]

item2 = [x - rj for x in ruj]

numerator = sum([x \* y for x, y in zip(item1, item2)])

denominator = ((sum([i1 \*\* 2 for i1 in item1]) \*\* 0.5) \* (sum([i2 \*\* 2 for i2 in item2]) \*\* 0.5))

w = numerator / denominator if denominator != 0 else 0

wDict[item] = w

wList.append((w, float(busUserRate[j][u])))

neighbours = sorted(wList, key=lambda x: -x[0])[:10]

# predicted value

num = sum(w \* r for w, r in neighbours)

den = sum(abs(w) for w, \_ in neighbours)

if den == 0:

P = 3.0

else:

P = num / den

return P

# catboost model and features

def cat(yelpCSV, userCSV, businessCSV, reviewCSV, valCSV):

xVal = []

valList = []

xTrain = []

yTrain = []

for u, b, r in yelpCSV.collect():

userInfo = userCSV.get(u, (None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None))

userReviews, userStars, userUseful, userFunny, userCool, userFans, userCompHot, userCompMore, userCompProfile, userCompCute, userCompList, userCompNote, userCompPlain, userCompCool, userCompFunny, userCompWriter, userCompPhotos, userYelp = userInfo

businessInfo = businessCSV.get(b, (None, None, None, None, None))

bsnReviews, bsnStars, bsnOpen, bsnPrice, bsnLat, bsnLong = businessInfo

reviewInfo = reviewCSV.get(b, (None, None, None, None))

reviewUseful, reviewFunny, reviewCool, reviewDate = reviewInfo

checkins = checkinCSV.get(b, 0)

tips, tipsDate = tipCSV.get(b, (0, 0))

photos = photoCSV.get(b, 0)

xTrain.append([reviewUseful, reviewFunny, reviewCool, reviewDate, bsnReviews, bsnStars, bsnOpen, bsnPrice, bsnLat, bsnLong, userReviews, userStars, userUseful, userFunny, userCool, userFans, userCompHot, userCompMore, userCompProfile, userCompCute, userCompList, userCompNote, userCompPlain, userCompCool, userCompFunny, userCompWriter, userCompPhotos, userYelp, checkins, tips, tipsDate, photos])

yTrain.append(r)

for v in valCSV.collect():

u, b = v[0], v[1]

valList.append((u, b))

valUserInfo = userCSV.get(u, (None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None))

valUserReviews, valUserStars, valUserUseful, valUserFunny, valUserCool, valUserFans, valUserCompHot, valUserCompMore, valUserCompProfile, valUserCompCute, valUserCompList, valUserCompNote, valUserCompPlain, valUserCompCool, valUserCompFunny, valUserCompWriter, valUserCompPhotos, valUserYelp = valUserInfo

valBusinessInfo = businessCSV.get(b, (None, None, None, None, None))

valBsnReviews, valBsnStars, valBsnOpen, valBsnPrice, valBsnLat, valBsnLong = valBusinessInfo

valReviewInfo = reviewCSV.get(b, (None, None, None, None))

valReviewUseful, valReviewFunny, valReviewCool, valReviewDate = valReviewInfo

valCheckins = checkinCSV.get(b, 0)

valTips, valTipsDate = tipCSV.get(b, (0, 0))

valPhotos = photoCSV.get(b, 0)

xVal.append([valReviewUseful, valReviewFunny, valReviewCool, valReviewDate, valBsnReviews, valBsnStars, valBsnOpen, valBsnPrice, valBsnLat, valBsnLong, valUserReviews, valUserStars, valUserUseful, valUserFunny, valUserCool, valUserFans, valUserCompHot, valUserCompMore, valUserCompProfile, valUserCompCute, valUserCompList, valUserCompNote, valUserCompPlain, valUserCompCool, valUserCompFunny, valUserCompWriter, valUserCompPhotos, valUserYelp, valCheckins, valTips, valTipsDate, valPhotos])

xTrain = np.array(xTrain, dtype='float32')

yTrain = np.array(yTrain, dtype='float32')

xVal = np.array(xVal, dtype='float32')

catParams = {

'subsample': 0.8,

'learning\_rate': 0.04,

'l2\_leaf\_reg': 0.2,

'n\_estimators': 1000,

'depth':10,

'random\_state': 42

}

catModel = CatBoostRegressor(\*\*catParams)

scaler = MinMaxScaler()

xTrainScaled = scaler.fit\_transform(xTrain)

xValScaled = scaler.transform(xVal)

catModel.fit(xTrainScaled, yTrain)

yPred = catModel.predict(xValScaled)

return yPred, valList

if \_\_name\_\_ == '\_\_main\_\_':

sc = SparkContext(appName="competition")

sc.setLogLevel('WARN')

startTime = time.time()

compFolder = sys.argv[1]

compTest = sys.argv[2]

compOutput = sys.argv[3]

# main data

yelpCSV = sc.textFile(compFolder + "/yelp\_train.csv")

yelpHeader = yelpCSV.first()

yelpCSV = yelpCSV.filter(lambda h: h != yelpHeader).map(lambda r: r.split(","))

def noOfDays(ys):

yd = datetime.strptime(ys, '%Y-%m-%d')

cd = datetime.now()

return (cd - yd).days

# user.json

userCSV = sc.textFile(compFolder + '/user.json') \

.map(lambda u: json.loads(u)) \

.map(lambda r: (

r['user\_id'],

(

float(r['review\_count']), float(r['average\_stars']), float(r['useful']),

float(r['funny']), float(r['cool']), float(r['fans']),

float(r['compliment\_hot']), float(r['compliment\_more']), float(r['compliment\_profile']),

float(r['compliment\_cute']), float(r['compliment\_list']), float(r['compliment\_note']),

float(r['compliment\_plain']), float(r['compliment\_cool']), float(r['compliment\_funny']),

float(r['compliment\_writer']), float(r['compliment\_photos']), noOfDays(r['yelping\_since'])

)

)) \

.collectAsMap()

def prices(attributes):

if attributes and 'RestaurantsPriceRange2' in attributes:

return float(attributes['RestaurantsPriceRange2'])

else:

return 2.5

def latlong(business):

lat = float(business['latitude']) if business.get('latitude') is not None else 0.0

long = float(business['longitude']) if business.get('longitude') is not None else 0.0

return lat, long

# business.json

businessCSV = sc.textFile(compFolder + "/business.json") \

.map(lambda b: json.loads(b)) \

.map(lambda r: (

r['business\_id'],

(

float(r['review\_count']), float(r['stars']), float(r['is\_open']),

prices(r.get('attributes')), \*latlong(r)

)

)) \

.collectAsMap()

# review\_train.json

reviewCSV = sc.textFile(compFolder + "/review\_train.json") \

.map(lambda r: json.loads(r)) \

.map(lambda rt: (

rt['business\_id'],

(float(rt['useful']), float(rt['funny']), float(rt['cool']), noOfDays(rt['date']))

)) \

.groupByKey() \

.mapValues(lambda v: tuple(sum(x) / len(v) for x in zip(\*v))) \

.collectAsMap()

# checkin.json

checkinCSV = sc.textFile(compFolder + "/checkin.json") \

.map(lambda c: json.loads(c)) \

.map(lambda r: (r['business\_id'], sum(r['time'].values()))) \

.collectAsMap()

# tip.json

tipCSV = sc.textFile(compFolder + "/tip.json") \

.map(lambda t: json.loads(t)) \

.map(lambda r: (r['business\_id'], (1, noOfDays(r['date'])))) \

.reduceByKey(lambda a, b: (a[0] + b[0], a[1])) \

.collectAsMap()

# photo.json

photoCSV = sc.textFile(compFolder + "/photo.json") \

.map(lambda p: json.loads(p)) \

.map(lambda r: (r['business\_id'], 1)) \

.reduceByKey(lambda a, b: a + b) \

.collectAsMap()

# validation data

valCSV = sc.textFile(compTest)

valHeader = valCSV.first()

valCSV = valCSV.filter(lambda h: h != valHeader).map(lambda r: r.split(","))

wDict = {}

busUserRate = yelpCSV.map(lambda r: (r[0], (r[1], r[2]))).groupByKey().mapValues(lambda userRatings: {u: r for u, r in userRatings}).collectAsMap()

busUser = yelpCSV.map(lambda r: (r[0], r[1])).groupByKey().mapValues(set).collectAsMap()

userBus = yelpCSV.map(lambda r: (r[1], r[0])).groupByKey().mapValues(set).collectAsMap()

businessAverage = yelpCSV.map(lambda r: (r[0], float(r[2]))).groupByKey().mapValues(lambda ratings: sum(ratings) / len(ratings)).collectAsMap()

userAverage = yelpCSV.map(lambda r: (r[1], float(r[2]))).groupByKey().mapValues(lambda ratings: sum(ratings) / len(ratings)).collectAsMap()

itemBased = []

for row in valCSV.collect():

itemPredictions = pearson(row[0], row[1])

itemBased.append(itemPredictions)

modelBased, valList = cat(yelpCSV, userCSV, businessCSV, reviewCSV, valCSV)

outputHeader = "user\_id, business\_id, prediction\n"

alpha = 0.06

for i in range(len(modelBased)):

result = alpha \* itemBased[i] + (1 - alpha) \* modelBased[i]

outputHeader += valList[i][0] + "," + valList[i][1] + "," + str(result) + "\n"

with open(compOutput, "w") as f:

f.write(outputHeader)

duration = time.time() - startTime

print("Duration:", duration)

# rmse

outputFile = sc.textFile(compOutput).filter(lambda x: x != "user\_id, business\_id, prediction").map(lambda x: x.split(',')).map(lambda x: float(x[2])).collect()

valFile = sc.textFile(compTest).filter(lambda x: x != "user\_id,business\_id,stars").map(lambda x: x.split(',')).map(lambda x: float(x[2])).collect()

RMSE = mean\_squared\_error(valFile, outputFile)

errors = np.abs(np.array(valFile) - np.array(outputFile))

errorBins = [sum((errors < 1)), sum((1 <= errors) & (errors < 2)), sum((2 <= errors) & (errors < 3)),

sum((3 <= errors) & (errors < 4)), sum(errors >= 4)]

print("Error distribution:", errorBins)