1. **­­­Baseline prediction error percentage:** 37.7345

**for** l **in** readGz(**"train.json.gz"**):  
 data.append(l)  
  
trainData, validData = np.array\_split(data, 2)  
  
businessCount = defaultdict(int)  
visitedBusinesses = defaultdict(int)  
users = defaultdict(int)  
totalVisits = 0;  
  
**for** d **in** trainData:  
 user, business = d[**'userID'**], d[**'businessID'**]  
 businessCount[business] += 1  
 totalVisits += 1  
 visitedBusinesses[user + **'-'** + business] += 1  
 users[user] += 1  
  
mostPopular = [(businessCount[x], x) **for** x **in** businessCount]  
mostPopular.sort()  
mostPopular.reverse()  
  
return1 = set()  
count = 0  
**for** ic, i **in** mostPopular:  
 count += ic  
 return1.add(i)  
 **if** count > totalVisits / 2: **break**businessList = []  
usersList = []  
  
vbusinessCount = defaultdict(int)  
vvisitedBusinesses = defaultdict(int)  
vusers = defaultdict(int)  
vtotalVisits = 0;  
vnotvisited = defaultdict(int)  
  
**for** d **in** validData:  
 user, business = d[**'userID'**], d[**'businessID'**]  
 vbusinessCount[business] += 1  
 vtotalVisits += 1  
 vvisitedBusinesses[user + **'-'** + business] += 1  
 vusers[user] += 1  
  
**for** c **in** vbusinessCount:  
 businessList.append(c)  
**for** u **in** vusers:  
 usersList.append(u)  
  
**import** random  
  
**while** (len(vnotvisited) < 100000):  
 pair = random.choice(usersList) + **'-'** + random.choice(businessList)  
 **while** (pair **in** vvisitedBusinesses):  
 pair = random.choice(usersList) + **'-'** + random.choice(businessList)  
 vnotvisited[pair] += 1  
  
wrongPred = 0;  
correctPred = 0;  
  
  
**def** baselinePredict(userBusinessPair, visited):  
 **global** wrongPred  
 **global** correctPred  
 **for** p **in** userBusinessPair:  
 u, b = p.strip().split(**'-'**)  
 **if** ((b **in** return1 **and** visited) **or** (b **not in** return1 **and not** visited)):  
 correctPred += 1  
 **else**:  
 wrongPred += 1  
  
  
baselinePredict(vvisitedBusinesses, **True**)  
baselinePredict(vnotvisited, **False**)  
  
print(wrongPred / (wrongPred + correctPred) \* 100)

1. Using the same code as above, the values for different thresholds for labeling a business as popular. There is slight improvement in error% for a threshold of 55% but very small improvement. The threshold of 50% works because if approx. half or slightly more than half the number of users have together visited some places over other places, they must be popular.

|  |  |
| --- | --- |
| **Threshold** | **Error %** |
| 1 | 49.303999999999995 |
| 0.9 | 43.4765 |
| 0.7 | 38.5635 |
| 0.8 | 40.426 |
| 0.4 | 38.6875 |
| 0.3 | 40.4025 |
| 0.2 | 42.933 |
| 0.1 | 46.077 |
| 0.6 | 37.606 |
| 0.65 | 37.915 |
| 0.55 | 37.4985 |

1. **Building if user visited same category before then true:**

**Wrong** **predictions**: 69473

**Right** **predictions**: 130527

**Error rate**: 0.347365

**Code stub:**

visitedCategory = defaultdict(set)

bCategory = defaultdict(set)  
**for** d **in** trainData:  
 **for** c **in** d[**'categories'**]:  
 visitedCategory[d[**'userID'**]].add(c)**for** d **in** validData:  
 **for** c **in** d[**'categories'**]:  
 bCategory[d[**'businessID'**]].add(c)wPred = 0  
cPred = 0  
  
**def** categoryVisitedPred(ubPairs, visited):  
 **global** wPred  
 **global** cPred  
 **for** p **in** ubPairs:  
 u,b = p.strip().split(**'-'**)  
 match = False;  
 **for** c **in** bCategory[b]:  
 **for** uc **in** visitedCategory[u]:  
 **if**(c==uc):  
 match = True  
 **break  
 if**(match):  
 **break  
 if**((match **and** visited) **or** (**not** match **and not** visited)):  
 cPred += 1  
 **else**:  
 wPred += 1  
  
categoryVisitedPred(vvisitedBusinesses, True)  
categoryVisitedPred(vnotvisited, False)  
  
**print**(wPred)  
**print**(cPred)  
**print**(wPred/(wPred+cPred))

1. **Kaggle user name:** divyasp

**def** categoryVisitedPredFile(inputFile, outputFile):  
 predictions = open(outputFile, **'w'**)  
 **for** l **in** open(inputFile):  
 **if** l.startswith(**"userID"**):  
 predictions.write(l)  
 **continue** u,b = l.strip().split(**'-'**)  
 match = False;  
 **for** c **in** bCategory[b]:  
 **for** uc **in** visitedCategory[u]:  
 **if**(c==uc):  
 match = True  
 **break  
 if**(match):  
 **break  
 if**(match):  
 predictions.write(u + **'-'** + b + **",1\n"**)  
 **else**:  
 predictions.write(u + **'-'** + b + **",0\n"**)  
  
categoryVisitedPredFile (**"pairs\_Visit.txt"**, **"predictions\_Visit.txt"**)

5. The trivial model would be just returning the average of all the user ratings.

**Alpha =** 4.18703

**MSE for this model:** 0.7483437445

allRatings = []  
userRatings = defaultdict(dict)  
busRatings = defaultdict(dict)  
  
**for** d **in** trainData:  
 user, bus, rating = d[**'userID'**], d[**'businessID'**], d[**'rating'**]  
 allRatings.append(rating)  
 userRatings[user][bus] = rating  
 busRatings[bus][user] = rating  
  
globalAvg = sum(allRatings)/len(allRatings)  
  
**print**(globalAvg)

vallRatings = []  
vuserRatings = defaultdict(dict)  
vbusRatings = defaultdict(dict)  
  
**for** d **in** validData:  
 user, bus, rating = d[**'userID'**], d[**'businessID'**], d[**'rating'**]  
 vallRatings.append(rating)  
 vuserRatings[user][bus] = rating  
 vbusRatings[bus][user] = rating  
  
predRatings = [globalAvg **for** x **in** range(100000)]

**from** sklearn.metrics **import** mean\_squared\_error  
  
MSE\_alpha = mean\_squared\_error(vallRatings, predRatings)  
**print**(MSE\_alpha)

6. For the predictor of give form: **Alpha =** 4.224718130267249

Using lambda = 1,

**MSE for this model:** 0.645692830316

**Code stub:**

**def** alphaUpdate():  
 **global** alpha  
 **global** betaU  
 **global** betaI  
 total = 0;  
 count = 0  
 **for** u **in** userRatings:  
 **for** b **in** userRatings[u]:  
 **if**(u **in** betaU):  
 total -= betaU[u]  
 **if**(bus **in** betaI):  
 total -= betaI[b]  
 total += userRatings[u][b]  
 count += 1  
 alpha = total/count

**def** betaUUpdate(lambdaval):  
 **global** alpha  
 **global** betaU  
 **global** betaI  
 total = 0  
 count = lambdaval  
 **for** u **in** userRatings:  
 **for** b **in** userRatings[u]:  
 total -= alpha  
 **if**(bus **in** betaI):  
 total -= betaI[b]  
 total += userRatings[u][b]  
 count += 1  
 betaU[u] = total/count  
 total = 0  
 count = lambdaval

**def** betaIUpdate(lambdaval):  
 **global** alpha  
 **global** betaU  
 **global** betaI  
 total = 0  
 count = lambdaval  
 **for** b **in** busRatings:  
 **for** u **in** busRatings[b]:  
 total -= alpha  
 **if**(user **in** betaU):  
 total -= betaU[u]  
 total += busRatings[b][u]  
 count += 1  
 betaI[b] = total/count  
 total = 0  
 count = lambdaval

alpha = 0  
betaU = defaultdict(int)  
betaI = defaultdict(int)  
**while**(True):  
 oldAlpha = alpha  
 alphaUpdate()  
 betaUUpdate(1)  
 betaIUpdate(1)  
 **if**(abs(oldAlpha - alpha) <= 0.000001):  
 **break  
print**(alpha)

**def** predictRating():  
 **global** betaU  
 **global** betaI  
 **global** alpha  
 predRatings = []  
 **for** d **in** validData:  
 user, business = d[**'userID'**], d[**'businessID'**]  
 prediction = alpha  
 **if** user **in** betaU:  
 prediction += betaU[user]  
 **if** business **in** betaI:  
 prediction += betaI[business]  
 predRatings.append(prediction)  
  
 MSE\_alpha = mean\_squared\_error(vallRatings, predRatings)  
 **return** MSE\_alpha  
  
**print**(predictRating())

7. User U417838537 has least beta: -2.834463263979296

User U357799541 has max beta: 1.162953088597245

Business B241777680 has least beta: -2.2356078402713204

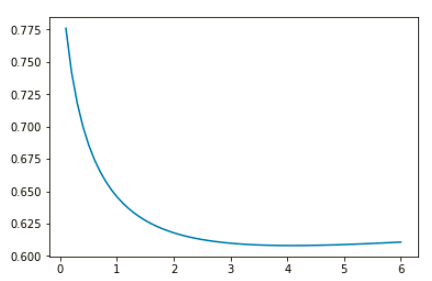
User B093985406 has max beta: 1.1683978344255426

minUser = min(betaU, key=betaU.get)  
**print**(**"User "**+str(minUser)+**" has least beta: "** +str(betaU[minUser]))  
maxUser = max(betaU, key=betaU.get)  
**print**(**"User "**+str(maxUser)+**" has max beta: "** +str(betaU[maxUser]))  
minBus = min(betaI, key=betaI.get)  
**print**(**"Business "**+str(minBus)+**" has least beta: "** +str(betaI[minBus]))  
maxBus = max(betaI, key=betaI.get)  
**print**(**"Business "**+str(maxBus)+**" has max beta: "** +str(betaI[maxBus]))

8. **Lambda for min MSE =** 4.1

**Minimum MSE** : 0.607994393749

Plot of MSE vs. Lambda value.



alpha = 0  
betaU = defaultdict(int)  
betaI = defaultdict(int)  
plotLambda = []  
plotMSE = []  
p = 0  
**for** i **in** range(0, 60):  
 p += 0.1  
 plotLambda.append(p)  
 **while**(True):  
 oldAlpha = alpha  
 alphaUpdate()  
 betaUUpdate(p)  
 betaIUpdate(p)  
 **if**(abs(oldAlpha - alpha) <= 0.000001):  
 **break** plotMSE.append(predictRating())  
 alpha = 0  
 betaU = defaultdict(int)  
 betaI = defaultdict(int)  
  
**print**(**"done"**)  
  
**import** matplotlib.pyplot **as** plt  
plt.plot(plotLambda, plotMSE)  
plt.show()

**print**(plotLambda[np.argmin(plotMSE)])  
**print**(plotMSE[np.argmin(plotMSE)])

**Using lambda = 4.1 on data to be tested:**

alpha = 0  
betaU = defaultdict(int)  
betaI = defaultdict(int)  
**while**(True):  
 oldAlpha = alpha  
 alphaUpdate()  
 betaUUpdate(4.1)  
 betaIUpdate(4.1)  
 **if**(abs(oldAlpha - alpha) <= 0.000001):  
 **break  
print**(predictRating())

**def** ratingPredFile(inputFile, outputFile):  
 **global** alpha  
 **global** betaU  
 **global** betaI  
 predictions = open(outputFile, **'w'**)  
 **for** l **in** open(inputFile):  
 **if** l.startswith(**"userID"**):  
 predictions.write(l)  
 **continue** user,business = l.strip().split(**'-'**)  
 prediction = alpha  
 **if** user **in** betaU:  
 prediction += betaU[user]  
 **if** business **in** betaI:  
 prediction += betaI[business]  
 predictions.write(user + **'-'** + business + **","** + str(prediction)+**"\n"**)  
  
ratingPredFile(**"pairs\_Rating.txt"**,**"predictions\_Rating.txt"**)