BDLE : Graphes en DataFrame

Construction du graphe

```
#fonction utilisée par la suite pour calculer les poids des arcs
#df: dataframe, source: nom de la colonne contenant les noeuds source des arcs
# poids: poids initial avant normalisation, n: nombre maximum d'arcs à garder pc
from pyspark.sql.functions import row_number, sum
from pyspark.sql import Window

def calcul_poids(df, source, poids, n):

    window = Window.partitionBy(source).orderBy(col(poids).desc())

    filterDF = df.withColumn("row_number", row_number().over(window)) \
        .filter(col("row_number") <= n) \
        .drop(col("row_number"))

    tmpDF = filterDF.groupBy(col(source)).agg(sum(col(poids)).alias("sum_" + poi

    finalDF = filterDF.join(tmpDF, source, "inner") \
        .withColumn("norm_" + poids, col(poids) / col("sum_" + poids)) \
        .cache()
    return finalDF
```

Construction des liens pondérés entre utilisateurs et chansons

- Construire un DataFrame userTrack à partir de data pour stocker les arcs entre utilisateurs et chansons. Pour chaque utilisateur (userId) on ajoute un arc vers une chanson (trackId) avec un poids égal au nombre total de fois que l'utilisateur à écouté la chanson. Utiliser la fonction calcul_poids pour garder pour chaque utilisateur les 100 chansons avec le poids le plus élevé et normaliser les poids des arcs gardés.
- Affichage du résultat: garder uniquement les arcs qui ont les 5 plus grandes valeurs possibles des poids (utilisez la fonction rank() et la fenêtre over(window)). Afficher 20 lignes du résultat, en triant le résultat par ordre décroissant des poids, ensuite par ordre croissant des userId et des artistId.

```
1
```

```
#calculer le poids final en utilisant calcul_poids
userTrack = calcul_poids(userTrack,"userId","count",100).select("userId","trackI

window = Window.orderBy(col("norm_count").desc())

userTrackList = userTrack.withColumn("position", rank().over(window)).where("pos

for val in userTrackList:
   print("%s %s %s" % val)

userTrack.count()#résultat: 210675
```

Construction des liens pondérés entre utilisateurs et artistes

- Construire un DataFrame userArtist à partir de data pour stocker les arcs entre utilisateurs et artistes. Pour chaque utilisateur (userId) on ajoute un arc vers un artiste (artistId) avec un poids égal au nombre total de fois que l'utilisateur à écouté des chansons de cet artiste. Utiliser la fonction calcul_poids pour garder pour chaque utilisateur au plus 100 artistes avec le poids le plus élevé et normaliser les poids des arcs gardés.
- Affichage du résultat: garder uniquement les arcs qui ont les 5 plus grandes valeurs possibles des poids (utilisez la fonction rank() et la fenêtre over(window)). Afficher 20 lignes du résultat, en triant le résultat par ordre décroissant des poids, ensuite par ordre croissant des userId et des artistId.

```
#poids=nombre de fois qu'un utilisateurs a écouté des morceaux de cet artiste
#regrouper les données par userId et artistId
userArtist  = data.groupBy("userId","artistId").count()

#calculer le poids final en utilisant calcul_poids
userArtist = calcul_poids(userArtist,"userId","count",100).select("userId","arti

window = Window.orderBy(col("norm_count").desc())

userArtistList = userArtist.withColumn("position", rank().over(window)).where("p
 # à compléter ....

for val in userArtistList:
   print("%s %s %s" % val)

```

- Affichage du résultat: garder uniquement les arcs qui ont les 5 plus grandes valeurs possibles des poids (utilisez la fonction rank() et la fenêtre over(window)). Afficher 20 lignes du résultat, en triant le résultat par ordre décroissant des poids, ensuite par ordre croissant des artistId et des trackId.*

```
# poids de l'arc: nombre de fois qu'un track de l'artiste a été écouté par tous
artistTrack =  data.groupBy("artistId","trackId").count()

#calculer le poids final en utilisant calcul_poids
artistTrack =  calcul_poids(artistTrack,"artistId","count",100).select("artistId

window = Window.orderBy(col("norm_count").desc())

artistTrackList = artistTrack.withColumn("position", rank().over(window)).\
where("position < 5").select("artistId","trackId","norm_count").orderBy("artistI
     # à compléter ....

for val in artistTrackList:
   print("%s %s %s" % val)

artistTrack.count() #résultat: 35408
```

Construction des liens pondérés entre chansons

- Construire un DataFrame trackTrack à partir de data pour stocker les arcs entre les chansons. Le poids total d'un arc entre trackId1 et trackId2 est le nombre total d'utilisateurs qui ont écouté à la fois trackId1 et trackId2 dans un laps de temps de 10 minutes (à noter que le graphe est non orienté, trackTrack contient à la fois une entrée pour (trackId1, trackId2) et une entrée pour (trackId2, trackId1)).
  Utiliser la fonction calcul_poids pour garder pour chaque chanson au plus 100 chansons avec le poids le plus élevé et normaliser les poids gardés.

```
 6
 7 #pour chaque couple de trackId le nombre d'utilisateurs qui les ont écoutés ense
 8
 9
10 #calculer le poids final en utilisant calcul_poids
11 track1=data.withColumnRenamed('trackId',"trackId1").\
12 withColumnRenamed("timestamp","timestamp1").withColumnRenamed("userId","userId1'
13
14 trackTrack =  data.join(track1,( (data.userId == track1.userId1) &(data.trackId
15 trackTrack=calcul_poids(trackTrack,"trackId","count",100).select('trackId',"trac
16
17 """window = Window.orderBy(col("norm_count").desc())
18
19 trackTrackList = trackTrack.withColumn("position", rank().over(window))\
20     # à compléter ....
21
22 for val in trackTrackList:
23    print("%s %s %s" % val)"""
24
25 trackTrack.count() #résultat: 136257
```

```
136257
```

Construction du graphe final

Construire un DataFrame graphe pour stocker tous les noeuds et tous les liens calculés précédemment. Le dataframe contiendra une colonne 'source' (identifiant du noeud source), une colonne 'destination' et une colonne 'poids'. Les colonnes 'source' et 'dest' contiennent à la fois des identifiants utilisateur, chanson et artiste. La colonne 'poids' contient les poids des arcs

recommendation à chaque iteration du calcul.

x[i] = alpha * u[i] + (1-alpha)* sum(xant[j]*poids[j][i])

- poids[j][i] : poids de l'arc entre j à i
- u[i]: valeur de personalisation, u[10]=1 et u[i]=0 si i !=10
- xant[j] : valeur du score du noeud j à l'itération précédente (x0[10]=1 et x0[i]=0 si i

On considère alpha=0.15 et on effectue le calcul pour 5 itérations (maxiter=5)

Calcul du vecteur de recommendation x

```
import pandas as pd
from pyspark.sql.functions import when

user = 10
d=0.85
alpha=(1-d)
maxiter = 5
# Construire le vecteur d'importance initial
x0  = spark.createDataFrame(pd.DataFrame([(user,1)], columns=["id","rank"]))

```

```
# Définir une fonction qui prend comme argument une liste d'utilisateurs triés
# par ordre croissant (users) et retourne une liste de couples ordonnés d'utilis

from pyspark.sql.functions import udf
from pyspark.sql.types import *

def parse_string(users):
    results=[]
    for  u in users[:-1]:
      for v in users[1:]:
        if ( u < v ):
          results.append(str(u)+"_"+str(v))
    return results

parse_string_udf = udf(parse_string, ArrayType(StringType()))
```

```
# Implémentation de l'algorithme de calcul de triangles

```