Analysis of Medline Data

To be run on Google Colab.

The following cell can take long, about 3 min. Only execute it once per session.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.3.2/spark-3.3.2-bin-hadoop2.tgz
!tar xf spark-3.3.2-bin-hadoop2.tgz
```

```
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.2-bin-hadoop2"

import findspark
findspark.init("spark-3.3.2-bin-hadoop2")# SPARK_HOME

import csv

import pyspark
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf

sc = SparkContext.getOrCreate()
spark = SparkSession.builder.getOrCreate()
```

```
!wget http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
!unzip ml-latest-small.zip
!mv ml-latest-small/ratings.csv .
!mv ml-latest-small/movies.csv .
linesRDD = sc.textFile("ratings.csv")

linesRDD = linesRDD.map(lambda line: line.split("|"))

#print(linesRDD.collect())

firstLine = linesRDD.first()
```

```
linesRDD = linesRDD.filter(lambda x:firstLine != x)
#print(linesRDD.collect())

#smcnt = rdd.map(lambda x: (x,1)).reduce(lambda t,r: (t[0]+r[0],t[1]+r[1]))

#print(smcnt)





#topics = medline_lists.flatMap(lambda topiclist: topiclist)
#print(topics.take(5))
```

```
--2024-03-12 23:37:31--  http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
Resolving files.grouplens.org (files.grouplens.org)... 128.101.65.152
Connecting to files.grouplens.org (files.grouplens.org)|128.101.65.152|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 978202 (955K) [application/zip]
Saving to: ‘ml-latest-small.zip.12’

ml-latest-small.zip 100%[===================>] 955.28K  --.-KB/s    in 0.1s

2024-03-12 23:37:31 (7.89 MB/s) - ‘ml-latest-small.zip.12’ saved [978202/978202]

Archive:  ml-latest-small.zip
replace ml-latest-small/links.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: ml-latest-small/links.csv
replace ml-latest-small/tags.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: ml-latest-small/tags.csv
  inflating: ml-latest-small/ratings.csv
replace ml-latest-small/README.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: ml-latest-small/README.txt
  inflating: ml-latest-small/movies.csv
```

```
new_rdd = linesRDD.map(lambda x: x[0].split(","))
#print(new_rdd.take(10))

#user id = u1
#movie id = m1
#rating = 4
#timestamp = 964982703

new_rdd = new_rdd.map(lambda x: {'user':'user'+x[0], 'movieid':'movieid'+x[1], 'rating':float(x[2]), 'timestamp':float(x[3])})
#print(new_rdd.take(5))

#user rdd

user_rdd = new_rdd.map(lambda x:(x['user'],x['rating']))
#print(user_rdd.take(5))

seqFunc = (lambda x,y: (x[0] + y, x[1] + 1))
combFunc = (lambda x,y: (x[0] + y[0], x[1] + y[1]))
user_rdd = user_rdd.aggregateByKey((0, 0), seqFunc,combFunc)
#user_rdd.collect()

user_rdd = user_rdd.map(lambda x: (x[0], x[1][0] / x[1][1]))
#user_rdd.collect()

#movie rdd

movie_rdd = new_rdd.map(lambda x:(x['movieid'],x['rating']))
#print(movie_rdd.take(5))

seqFunc = (lambda x,y: (x[0] + y, x[1] + 1))
combFunc = (lambda x,y: (x[0] + y[0], x[1] + y[1]))
movie_rdd_agg = movie_rdd.aggregateByKey((0, 0), seqFunc,combFunc)
#print(movie_rdd.collect())

movie_rdd_avg = movie_rdd_agg.map(lambda x: (x[0], x[1][0] / x[1][1]))
#movie_rdd.collect()

user_movie_rdd = user_rdd.union(movie_rdd_avg) #im just going to output them seperately


print(user_rdd.take(5))
print(movie_rdd_avg.take(5))
```

```
[('user1', 4.366379310344827), ('user2', 3.9482758620689653), ('user4', 3.5555555555555554), ('user5', 3.6363636363636362), ('user6', 3.493630
[('movieid1', 3.9209302325581397), ('movieid3', 3.2596153846153846), ('movieid6', 3.946078431372549), ('movieid47', 3.9753694581280787), ('mov
```

```
#now we do movies.csv

datafile = open('movies.csv', 'r')
myreader = csv.reader(datafile)

movieGenres = {}
for row in myreader:
  if row != ['movieId','title','genres']:
    movieGenres[row[0]] = row[2].split('|')
#print(movieGenres)

m_rdd = sc.parallelize([('movieid'+key,value) for key, value in movieGenres.items()])

#print(m_rdd.take(5))

#print(user_movie_rdd.take(5))
#print(movie_rdd.take(5))


joined_rdd = movie_rdd.join(m_rdd)
#print(joined_rdd.take(5))

only_1 = joined_rdd.filter(lambda x: x[0] == 'movieid1')
#print(only_1.collect())

tuple_rdd = joined_rdd.flatMap(lambda x: [(genre,x[1][0]) for genre in x[1][1]])
#print(tuple_rdd.take(5))

seqFunc = (lambda x,y: (x[0] + y, x[1] + 1))
combFunc = (lambda x,y: (x[0] + y[0], x[1] + y[1]))
tuple_rdd = tuple_rdd.aggregateByKey((0, 0), seqFunc,combFunc)
#print(tuple_rdd.take(5))


tuple_rdd = tuple_rdd.map(lambda x: (x[0], x[1][0] / x[1][1]))
print(tuple_rdd.collect())
```

```
[('Fantasy', 3.4910005070136894), ('Thriller', 3.4937055799183425), ('Romance', 3.5065107040388437), ('Western', 3.583937823834197), ('Sci-Fi'
```