Exam BDA AA 2020 - Alex Panchot - M20190546

May 30, 2020

Alex Panchot M20190546

1 Six Degrees of Kevin Bacon

Introduction - Six Degrees of Kevin Bacon is a game based on the "six degrees of separation" concept, which posits that any two people on Earth are six or fewer acquaintance links apart. Movie buffs challenge each other to find the shortest path between an arbitrary actor and prolific actor Kevin Bacon. It rests on the assumption that anyone involved in the film industry can be linked through their film roles to Bacon within six steps. The analysis of social networks can be a computationally intensive task, especially when dealing with large volumes of data. It is also a challenging problem to devise a correct methodology to infer an informative social network structure. Here, we will analyze a social network of actors and actresses that co-participated in movies. We will do some simple descriptive analysis, and in the end try to relate an actor/actress's position in the social network with the success of the movies in which they participate.

####Rules & Notes - Please take your time to read the following points: 2. The submission deadline will be set for the 30th of May at 23:59h. 3. It is acceptable that you **discuss** with your colleagues different approaches to solve each step of the problem set, but the assignment is individual. That is, you are responsible for writing your own code, and analyzing the results. Clear cases of cheating will be penalized with 0 points in this assignment. 4. After review of your submission files, and before a mark is attributed, you might be called to orally defend your submission. 5. You will be scored first and foremost by the number of correct answers, secondly by the logic used in trying to approach each step of the problem set. 1. You can add as many cells as you like to answer the questions. 1. It is also important you clearly indicate what your final answer to each question is when you are using multiple cells (for example you can use `print("My final anwser is:")` before your answer or use cell comments). 11. Consider skipping questions that you are stuck on, and get back to them later. 12. Expect computations to take a few minutes to finish in some of the steps. 1. It is recommended you read the whole assignment before starting. 1. You can make use of caching or persisting your RDDs or Dataframes, this may speed up performance. You do not need to cache every dataframe, but usually you want to do this at least once after the data has been imported. 1. If you have trouble with graphframes in databricks (specifically the import statement) you need to make sure the graphframes package is installed on the cluster you are running. If you click home on the left, then click on the graphframes library which you loaded in Lab 11 you can install the package on your cluster (check the graphframes checkbox and click install) 1. Be careful, you must not 'Publish' this notebook in databricks. 13. **IMPORTANT** It is expected you have developed skills beyond writing SQL queries. Any question where you directly write a SQL query (by for example creating a temporary view and then using spark.sql to pass the query) will receive a 25% penalty. Using the spark syntax (for

example dataframe.select("*").where("conditions")) is acceptable and does not incur this penalty. 13. **Questions** – Any questions about this assignment should be posted in the Forum@Moodle. Questions by e-mail will not be answered. The lab will run at the normal time. During this period you can ask any questions you have about the exam (we can't provide you the actual answers of course, but there may be helpful tips if you are stuck on any of the steps). As such, it is probably useful to attempt the assignment before the scheduled lab. 14. **Delivery** - To fulfil this activity you will have to upload the following materials to Moodle: - An exported IPython notebook. From the menu at the top, select 'File', then 'Export', then 'IPython Notebook', to download the notebook. The notebook should be solved (have results displayed), but should contain all necessary code so that when the notebook is run in databricks it should also replicate these results. This means that all data downloading and processing should be done in this notebook. It is also important you clearly indicate where your final answer to each question is when you are using multiple cells (for example you can us `print("My final answer is:")` before your answer or use cell comments). - A PDF version of your code and answers. There are a couple of ways you can do this. You can convert the downloaded IPython Notebook to pdf (check out nbconvert if you have Jupyter notebook), or you can just copy your code and answers into a word file and save as pdf, or finally you can take screenshots of each page of the notebook and put them into a word file and save it as pdf. It is important that all code and aswers are visible in this pdf. - You will also need to provide a signed statement of authorship, which is available on Moodle.

####Data Sources and Description We will use data from IMDB. You can download raw datafiles from https://datasets.imdbws.com. Note that the files are tab delimited (.tsv) You can find a description of the each datafile in https://www.imdb.com/interfaces/

[2]:
```
from pyspark.sql.types import *
from pyspark.sql import Row
```

1.1 Questions

1.1.1 Data loading and preperation

Review the file descriptions and load the necessary data onto your databricks cluser and into spark dataframes or rdds. You will need to use shell commands to download the data, unzip the data, load the data into spark. Note that the data might require parsing and preprocessing to be ready for the questions below.

Hints You can use `gunzip` to unzip the .tz files. The data files will then be tab seperated (.tsv), which you can load into a dataframe using the tab seperated option instead of the comma seperated option we have typically used in class: `.option("sep","\t")`

[4]:
```
%sh wget https://datasets.imdbws.com/name.basics.tsv.gz
```

[5]:
```
%sh gunzip name.basics.tsv.gz
```

[6]:
```
namebasicsDF = spark.read.option("sep","\t").csv('file:/databricks/driver/name.
↪basics.tsv',header=True, inferSchema = True)
```

[7]:
```
%sh wget https://datasets.imdbws.com/title.basics.tsv.gz
```

[8]:
```
%sh gunzip title.basics.tsv.gz
```

[9]:
```
titlebasicsDF = spark.read.option("sep","\t").csv('file:/databricks/driver/
↪title.basics.tsv',header=True, inferSchema = True)
```

[10]:
```
%sh wget https://datasets.imdbws.com/title.principals.tsv.gz
```

[11]:
```
%sh gunzip title.principals.tsv.gz
```

[12]:
```
titleprincipalsDF = spark.read.option("sep","\t").csv('file:/databricks/driver/
↪title.principals.tsv',header=True, inferSchema = True)
```

[13]:
```
%sh wget https://datasets.imdbws.com/title.ratings.tsv.gz
```

[14]:
```
%sh gunzip title.ratings.tsv.gz
```

[15]:
```
titleratingsDF = spark.read.option("sep","\t").csv('file:/databricks/driver/
↪title.ratings.tsv',header=True, inferSchema = True)
```

1.1.2 Network Inference, Let's build a network

In the following questions you will look to summarise the data and build a network. We want to examine a network that abstracts how actors and actress are related through their co-participation in movies. To that end perform the following steps:

Q1 Create a DataFrame that combines the information on each of the titles (i.e., movies, tv-shows, etc …) and the information on the participants in those movies (i.e., actors, directors, etc …), make sure the actual names of the movies and participants are included. It may be worth reviewing the following questions to see how this dataframe will be used.

How many rows does yoru dataframe have?

[17]:
```
DF_1=titleprincipalsDF.join(titlebasicsDF, "tconst", "inner")
DF_2=DF_1.join(titleratingsDF, "tconst", "inner")
DF_3=DF_2.join(namebasicsDF, "nconst", "inner")
DF=DF_3.drop("characters").drop("originalTitle").drop("runtimeMinutes").
↪drop("primaryProfession").drop("knownForTitles").drop("job")
```

[18]:
```
DF.count()
```

1.1.3 Q1 Final

[19]:
```
print("My final answer for Q1 is: 8689095") # AS OF 29/5/2020
```

** Q2 ** Create a new DataFrame based on the previous step, with the following removed: 1. Any participant that is not an actor or actress (as measured by the category column); 1. All adult movies; 1. All dead actors or actresses; 1. All actors or actresses born before 1920 or with no date of birth listed; 1. All titles that are not of the type movie.

How many rows does your dataframe have?

[21]:
```
DFQ2=DF.where((DF['category']=='actor') | (DF['category']=='actress')).
 ↪where(DF['isAdult']==0).drop("isAdult").where(DF['deathYear']=='\\N').
 ↪where((DF['birthYear']!='null') | (DF['birthYear']>'1920')).
 ↪where(DF['titleType']=='movie').drop("titleType")

display(DFQ2)
```

[22]:
```
DFQ2.count()
```

1.1.4 Q2 Final

[23]:
```
print("My final answer for Q2 is: 682129") # AS OF 29/5/2020
```

Q3 Convert the above Dataframe to an RDD (you can use `.rdd` to convert a dataframe to and RDD of row objects). Use map and reduce to create a paired RDD which counts how many movies each actor / actress appears in.

Display names of the top 10 actors/actresses according to the number of movies in which they appeared. Be careful to deal with different actors / actresses with the same name, these could be different people.

[25]:
```
rddQ3_1=DFQ2.rdd
DFQ3=rddQ3_1.map(lambda x: str(x[0])+"_"+str(x[10])).map(lambda x: (x,1)).
 ↪reduceByKey(lambda x,y: x+y).map(lambda x: (x[1],x[0])).
 ↪sortByKey(ascending=False).map(lambda x: (x[1],x[0])).toDF().
 ↪withColumnRenamed("_1", "id").withColumnRenamed("_2", "movie_appearances")
DFQ3.show()
```

1.1.5 Q3 Final

[2]:
```
print("My final answer for Q3 is: \n (nm0103977_Brahmanandam, 425),\n␣
 ↪(nm0482320_Mohanlal, 325),\n (nm0007123_Mammootty, 317),\n (nm0149822_Mithun␣
 ↪Chakraborty, 298),\n (nm0035067_Cüneyt Arkin, 286),\n (nm0004429_Dharmendra,␣
 ↪240),\n (nm0415549_Jagathi Sreekumar, 220),\n (nm0595934_Mohan Babu, 217),\n␣
 ↪(nm0007106_Shakti Kapoor, 198),\n (nm0352032_Kamal Haasan, 193") # AS OF 29/
 ↪5/2020
```

```
My final answer for Q3 is:
 (nm0103977_Brahmanandam, 425),
 (nm0482320_Mohanlal, 325),
 (nm0007123_Mammootty, 317),
 (nm0149822_Mithun Chakraborty, 298),
 (nm0035067_Cüneyt Arkin, 286),
 (nm0004429_Dharmendra, 240),
 (nm0415549_Jagathi Sreekumar, 220),
 (nm0595934_Mohan Babu, 217),
```

```
(nm0007106_Shakti Kapoor, 198),
(nm0352032_Kamal Haasan, 193
```

Q4 Start with the dataframe from **Q2**. Generate a DataFrame that lists all links of your network. Here we shall consider that a link connects a pair of actors/actresses if they participated in at least one movie together (actors / actresses should be represented by their unique ID's). For every link we then need anytime a pair of actors were together in a movie as a link in each direction (A -> B and B -> A). However links should be distinct we do not need duplicates when two actors worked together in several movies.

[28]:
```
from pyspark.sql.functions import concat, col,lit
```

[29]:
```
actorsA=DFQ2.select('nconst','tconst','primaryName').distinct().
 ↪withColumnRenamed("nconst", "src").withColumnRenamed("primaryName", "Name_A")
actsA=actorsA.withColumn('src',concat(col("src"),lit("_"),col("Name_A"))).
 ↪drop("Name_A")
actorsB=DFQ2.select('nconst','tconst','primaryName').distinct().
 ↪withColumnRenamed("nconst", "dst").withColumnRenamed("primaryName", "Name_B")
actsB=actorsB.withColumn('dst',concat(col("dst"),lit("_"),col("Name_B"))).
 ↪drop("Name_B")
act=actsA.join(actsB,'tconst','inner')
act=act.where(act.src!=act.dst)
DFQ4=act.select('src','dst').distinct()
```

1.1.6 Q4 Final

[30]:
```
print("My final answer for Q4 is below in datafame DFQ4:") # AS OF 29/5/2020
```

[31]:
```
DFQ4.show()
```

Q5 Compute the page rank of each actor. This can be done using GraphFrames or by using RDDs and the iterative implementation of the PageRank algorithm. Do not take more than 5 iterations and use reset probility = 0.1.

List the top 10 actors / actresses by pagerank.

[33]:
```
import graphframes.graphframe as gfm
```

[34]:
```
actors=DFQ2.select('nconst','primaryName').distinct().
 ↪withColumnRenamed("nconst", "id")
actors=actors.withColumn('id',concat(col("id"),lit("_"),col("primaryName"))).
 ↪drop("primaryName")
```

[35]:
```
actors.show()
```

[36]:
```
DFQ4.show()
```

[37]:
```
actorgraph = gfm.GraphFrame(actors, DFQ4)
```

[38]:
```
pageRanks = actorgraph.pageRank(resetProbability=0.1, maxIter = 5)
```

1.1.7 Q5 Final

[39]:
```
print("My final answer for Q5 is below in dataframe pageRanks.vertices") # AS␣
↪OF 29/5/2020
```

[40]:
```
pageRanks.vertices.orderBy('pagerank',ascending=False).limit(10).show()
```

Q6: Create an RDD with the number of outDegrees for each actor. Display the top 10 by outDegrees.

[42]:
```
rddoutdegrees=actorgraph.outDegrees.rdd
```

1.1.8 Q6 Final

[44]:
```
rddoutdegrees.top(10, key=lambda x: x[1])
```

[1]:
```
print("My final answer for Q6 is: \n [Row(id='nm0103977_Brahmanandam',␣
↪outDegree=539), \n Row(id='nm0000616_Eric Roberts', outDegree=509), \n␣
↪Row(id='nm0149822_Mithun Chakraborty', outDegree=490), \n␣
↪Row(id='nm0007123_Mammootty', outDegree=409), \n Row(id='nm0695177_Prakash␣
↪Raj', outDegree=390), \n Row(id='nm0007106_Shakti Kapoor', outDegree=376),␣
↪\n Row(id='nm0945189_Simon Yam', outDegree=371), \n␣
↪Row(id='nm0482320_Mohanlal', outDegree=371), \n Row(id='nm0621937_Nassar',␣
↪outDegree=363), \n Row(id='nm0451600_Anupam Kher', outDegree=360)]") # AS OF␣
↪29/5/2020
```

```
My final answer for Q6 is:
 [Row(id='nm0103977_Brahmanandam', outDegree=539),
 Row(id='nm0000616_Eric Roberts', outDegree=509),
 Row(id='nm0149822_Mithun Chakraborty', outDegree=490),
 Row(id='nm0007123_Mammootty', outDegree=409),
 Row(id='nm0695177_Prakash Raj', outDegree=390),
 Row(id='nm0007106_Shakti Kapoor', outDegree=376),
 Row(id='nm0945189_Simon Yam', outDegree=371),
 Row(id='nm0482320_Mohanlal', outDegree=371),
 Row(id='nm0621937_Nassar', outDegree=363),
 Row(id='nm0451600_Anupam Kher', outDegree=360)]
```

1.1.9 Let's play Kevin's own game

Q7 Start with the graphframe / dataframe you developed in the previous section. Using Spark GraphFrame and/or Spark Core library perform the following steps:

1. Identify the id of Kevin Bacon, there are two actors named 'Kevin Bacon', we will use the one with the highest degree, that is, the one that participated in most titles;

2. Estimate the shortest path between every actor/actress in the database and Kevin Bacon, keep a dataframe with a column that includes the number of steps to Kevin Bacon as you will need it later (this will require a little processing to get from the graphframes output);
3. Summarise the data, that is, count the number of actors at each number of degress from kevin bacon (you will need to deal with actors unconnected to kevin bacon, if not connected to Kevin Bacon given these actors / actresses a score of 20). You could use the display() barchart functionality of databricks to easily display the distribution of the data.

Note: The solution time on this step can be ~15 minutes

[46]:
```
print("Part 1")
```

[47]:
```
DFQ2.where(DFQ2.primaryName == "Kevin Bacon").show()
```

[48]:
```
print("kevin bacon is: nm0000102")
```

[49]:
```
print("Part 2")
```

[50]:
```
DFQ7 = actorgraph.shortestPaths(landmarks=["nm0000102_Kevin Bacon"])
```

[51]:
```
from pyspark.sql.functions import map_values

DFQ7_2=DFQ7.select('id',map_values("distances").alias("distances"))
```

[52]:
```
DFQ7rdd=DFQ7_2.rdd
```

[53]:
```
def func(x):
  a,b=x
  if len(b)==0:
    return (a,20)
  else:
    return (a,b[0])

DFQ7rdd2=DFQ7rdd.map(func)
```

[54]:
```
print("Part 3")
```

[55]:
```
DFQ7rdd3=DFQ7rdd2.map(lambda x: (x[1],1)).reduceByKey(lambda x,y: x+y).
↪map(lambda x: (int(x[0]),int(x[1])))
```

[56]:
```
DFQ7_ids=DFQ7rdd2.toDF().withColumnRenamed("_1", "id").withColumnRenamed("_2",␣
↪"Distance")
```

[57]:
```
DFQ7_3=DFQ7rdd3.toDF().withColumnRenamed("_1", "Distance").
↪withColumnRenamed("_2", "Count")
```

1.1.10 Q7 Final

[58]:
```
print("My final answer for Q7 is below in dataframe DFQ7_3") # AS OF 29/5/2020
```

[59]:
```
display(DFQ7_3)
```

1.1.11 Exploring the data with RDD's

Using RDDs and (not dataframes) answer the following questions (if you loaded your data into spark in a dataframe you can convert to an RDD of rows easily using `.rdd`) :

Hint: paired RDD's will be useful.

Q8 Movies can have multiple genres. Considering only titles of the type 'movie' what is the combination of genres that is the most popluar (as measured by number of reviews)?

[61]:
```
DF8rdd=DF.where(DF.titleType=='movie').rdd
DF8rdd.take(10)
```

[62]:
```
rdd8_1=DF8rdd.map(lambda x: (x[1],x[9],x[11])).distinct().map(lambda x:␣
↪(x[1],x[2]))
```

[63]:
```
rdd8_2=rdd8_1.reduceByKey(lambda x,y: x+y)
```

[64]:
```
rdd8_2.map(lambda x: (x[1],x[0])).sortByKey(ascending=False).map(lambda x:␣
↪(x[1],x[0])).take(10)
```

1.1.12 Q8 Final

[3]:
```
print("My final answer for Q8 is: ('Action,Adventure,Sci-Fi', 43434839)") # AS␣
↪OF 29/5/2020
```

```
My final answer for Q8 is: ('Action,Adventure,Sci-Fi', 43434839)
```

Q9 Movies can have multiple genres. Considering only titles of the type 'movie', and movies with more than 500 ratings, what is the combination of genres that has the highest **average movie rating** (you can average the movie rating for each movie in that genre combination).

[67]:
```
rdd9_1=DF8rdd.map(lambda x: (x[1],x[9],x[10],x[11])).distinct().map(lambda x:␣
↪(x[1],x[2],x[3])).filter(lambda x: x[2]>500).mapValues(lambda x: (x, 1))
```

[68]:
```
rdd9_2=rdd9_1.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).mapValues(lambda x:
↪ x[0]/x[1]).map(lambda x: (x[1],x[0])).sortByKey(ascending=False).map(lambda␣
↪x: (x[1],x[0]))
```

[69]:
```
rdd9_2.take(10)
```

1.1.13 Q9 Final

[4]:
```
print("[My final answer for Q9 is: ('Music,Musical', 8.5)") # AS OF 29/5/2020
```

```
[My final answer for Q9 is: ('Music,Musical', 8.5)
```

Q10 Movies can have multiple genres. What is **the individual genre** which is the most popular as meaured by number of votes. Votes for multiple genres count towards each genre listed.

Hint: Think about the wordcount exercise we have done with RDDs.

[72]:
```
rdd10_1=DF8rdd.map(lambda x: (x[1],x[9],x[11])).distinct().map(lambda x:␣
↪(x[1],x[2]))
```

[73]:
```
rdd10_2 = rdd10_1.flatMap(lambda x: ((v, x[1]) for v in x[0].split(','))).
↪reduceByKey(lambda x,y: x+y).map(lambda x: (x[1],x[0])).
↪sortByKey(ascending=False).map(lambda x: (x[1],x[0]))
```

[74]:
```
rdd10_2.take(10)
```

1.1.14 Q10 Final

[5]:
```
print("My final answer for Q10 is: ('Drama', 405749355)") # AS OF 29/5/2020
```

```
My final answer for Q10 is: ('Drama', 405749355)
```

1.2 Engineering the perfect cast

We have created a number of potential features for predicting the rating of a movie based on its cast. Use sparkML to build a simple linear model to predict the rating of a movie based on the following features:

1. The total number of movies in which the actors / actresses in the current movie have acted (based on Q3)
2. The average pagerank of the cast in each movie (based on Q5)
3. The average outDegree of the cast in each movie (based on Q6)
4. The average value for for the cast of degrees of Kevin Bacon (based on Q7).

If you were unable to generate any of these features as you could not answer the previous questions, just skip that particular feature.

You will need to create a dataframe with the required features and label. Use a pipeline to create the vectors required by sparkML and apply the model. Remember to split your dataset, leave 30% of the data for testing, when splitting your data use the option `seed=0`.

Q11 Provide the coefficients of the regression and the accuracy of your model on the test dataset according to RSME.

[77]:
```
DFQ11=DFQ2.withColumn('id',concat(col("nconst"),lit("_"),col("primaryName"))).
↪select('id','tconst','averageRating')
```

[78]:
```
DFQ3.show()
```

[79]:
```
Q5=pageRanks.vertices.orderBy('pagerank',ascending=False)
```

[80]:
```
Q5.show()
```

[81]:
```
Q6=rddoutdegrees.toDF().withColumnRenamed("_1", "id").withColumnRenamed("_2",␣
↪"outdegrees")
```

[82]:
```
Q6.show()
```

[83]:
```
DFQ7_ids.show()
```

[84]:
```
DFQ11_1=DFQ11.join(DFQ3, "id", "inner")
DFQ11_2=DFQ11_1.join(Q5, "id", "inner")
DFQ11_3=DFQ11_2.join(Q6, "id", "inner")
DFQ11_4=DFQ11_3.join(DFQ7_ids, "id", "inner")
```

[85]:
```
display(DFQ11_4)
```

[86]:
```
DFQ11_4_1=DFQ11_4.select('tconst','pagerank').rdd.mapValues(lambda x: (x, 1))
#DFQ11_4_1.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).mapValues(lambda x:␣
↪x[0]/x[1]).take(10)
DFQ11_4_1_1=DFQ11_4_1.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).
↪mapValues(lambda x: x[0]/x[1]).toDF().withColumnRenamed("_1", "tconst").
↪withColumnRenamed("_2", "pagerank")
#DFQ11_4_1_1.show()
```

[87]:
```
DFQ11_4_2=DFQ11_4.select('tconst','outDegree').rdd.mapValues(lambda x: (x, 1))
#DFQ11_4_2.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).mapValues(lambda x:␣
↪x[0]/x[1]).take(10)
DFQ11_4_2_1=DFQ11_4_2.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).
↪mapValues(lambda x: x[0]/x[1]).toDF().withColumnRenamed("_1", "tconst").
↪withColumnRenamed("_2", "outDegree")
#DFQ11_4_2_1.show()
```

[88]:
```
DFQ11_4_3=DFQ11_4.select('tconst','Distance').rdd.mapValues(lambda x: (x, 1))
#DFQ11_4_3.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).mapValues(lambda x:␣
↪x[0]/x[1]).take(10)
DFQ11_4_3_1=DFQ11_4_3.reduceByKey(lambda x,y: (x[0]+y[0],x[1]+y[1])).
↪mapValues(lambda x: x[0]/x[1]).toDF().withColumnRenamed("_1", "tconst").
↪withColumnRenamed("_2", "Distance")
#DFQ11_4_3_1.show()
```

[89]:
```
DFQ11_4_4=DFQ11_4.select('tconst','movie_appearances').rdd
#DFQ11_4_4.reduceByKey(lambda x,y: x+y).take(10)
```

```
DFQ11_4_4_1=DFQ11_4_4.reduceByKey(lambda x,y: x+y).toDF().
 ↪withColumnRenamed("_1", "tconst").withColumnRenamed("_2", "movie_appear")
#DFQ11_4_4_1.show()
```

[90]:
```
DFQ11_5_1=DFQ11_4.select('tconst','averageRating')
DFQ11_5_2=DFQ11_5_1.join(DFQ11_4_1_1, "tconst", "inner")
DFQ11_5_3=DFQ11_5_2.join(DFQ11_4_2_1, "tconst", "inner")
DFQ11_5_4=DFQ11_5_3.join(DFQ11_4_3_1, "tconst", "inner")
DFQ11_5=DFQ11_5_4.join(DFQ11_4_4_1, "tconst", "inner")
DFQ11_6=DFQ11_5.rdd.distinct().toDF().withColumnRenamed("_1", "tconst").
 ↪withColumnRenamed("_2", "averageRating").withColumnRenamed("_3", "pagerank").
 ↪withColumnRenamed("_4", "outDegree").withColumnRenamed("_5", "Distance").
 ↪withColumnRenamed("_6", "movie_appear")
```

[91]:
```
DFQ11_6.show()
```

[92]:
```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

def transData(data):
  # Combine columns to a dense vector (excluding the last column)
  dataFeaturesRDD = data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]])

  # Convert the RDD back to a DataFrame, labelling the columns
  featuresDF =  dataFeaturesRDD.toDF(['features','label'])

  return featuresDF
```

[93]:
```
DFQ11_7=transData( DFQ11_6.
 ↪select('pagerank','outDegree','Distance','movie_appear','averageRating') )
```

[94]:
```
display(DFQ11_7)
```

[95]:
```
(trainingData12, testData12) = DFQ11_7.randomSplit([0.7, 0.3],seed=0)
```

[96]:
```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression

# Define LinearRegression algorithm
lr = LinearRegression()
```

[97]:
```
# Fit 2 models, using different regularization parameters
model12 = lr.fit(trainingData12, {lr.regParam:0.0})
```

[98]:
```
# Print the fitted model parameters
print(">>>> Model intercept: %r, coefficient: %r" % (model12.intercept, model12.
 ↪coefficients))
```

[99]:
```
predictions12_train = model12.transform(trainingData12)
predictions12_test = model12.transform(testData12)
```

[100]:
```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="rmse")
```

[101]:
```
RMSE12_train = evaluator.evaluate(predictions12_train)
RMSE12_test = evaluator.evaluate(predictions12_test)
```

[102]:
```
print("Model: Root Mean Squared Train Error = " + str(RMSE12_train))
print("Model: Root Mean Squared Test Error = " + str(RMSE12_test))
```

1.2.1 Q11 Final

[6]:
```
print("My final answer for Q11 is: \n Model intercept: 5.851793963763911,␣
↪coefficient: DenseVector([-0.0648, 0.0089, 0.0229, -0.0019]) \n Root Mean␣
↪Squared Test Error = 1.3232355120146824") # AS OF 29/5/2020
```

```
My final answer for Q11 is:
 Model intercept: 5.851793963763911, coefficient: DenseVector([-0.0648, 0.0089,
0.0229, -0.0019])
 Root Mean Squared Test Error = 1.3232355120146824
```

Q12 What score would your model predict for the 1997 movie Titanic and how does this compare to it's actual score.

[105]:
```
DFQ2.where(DFQ2.primaryTitle == "Titanic").show()
```

[106]:
```
DFQ11_titanic=transData( DFQ11_6.where(DFQ11_6.tconst == "tt0120338").
↪select('pagerank','outDegree','Distance','movie_appear','averageRating') )
```

[107]:
```
display(DFQ11_titanic)
```

[108]:
```
predictions12=model12.transform(DFQ11_titanic)
```

[109]:
```
display(predictions12)
```

1.2.2 Q12 Final

[7]:
```
print("My final answer for Q12 is: \n Predicted: 5.962043925599998 \n Actual: 7.
↪8") # AS OF 29/5/2020
```

```
My final answer for Q12 is:
 Predicted: 5.962043925599998
 Actual: 7.8
```

Q13 Create dummy variables for each of the top 10 movie genres from **Q10**. These variable should have a value of 1 if the movie was rated with that genre and 0 otherwise. For example the 1997

movie Titanic should have a 1 in the dummy variable column for Romance, and a 1 in the dummy variable column for Drama, and 0's in all the other dummy variable columns.

If you were unable to answer Q10 you can just select 10 different genres and construct the same data.

Note: Question 10 uses the number of votes per genre and not the average votes per genre.

Does adding these variables to the regression improve your results? What is the new RMSE and predicted rating for the 1997 movie Titanic.

[112]:
```
DFQ13=DFQ11_6.join(DFQ2.select('tconst','genres'),'tconst','inner')
```

[113]:
```
DFQ13.show()
```

[114]:
```
def q13s(x):
  ls=[]
  for i in range(len(x)):
    ls.append(x[i])
  xl=x[-1].split(',')
 ␣
 ↪genre=['Drama','Action','Comedy','Adventure','Crime','Thriller','Sci-Fi','Romance','Fantasy
  for i in range(10):
    for j in xl:
      if j==genre[i]:
        ls.append(1)
        break
    else:
      ls.append(0)

  return ls
```

[115]:
```
DFQ13rdd = DFQ13.rdd.distinct().map(lambda x: (q13s(x)))
```

[116]:
```
DFQ13_1=DFQ13rdd.
 ↪toDF(['tconst','averageRating','pagerank','outDegree','Distance','movie_appear','genres','D
```

[117]:
```
DFQ13_1.show()
```

[118]:
```
DFQ13_2=transData( DFQ13_1.
 ↪select('pagerank','outDegree','Distance','movie_appear','Drama','Action','Comedy','Adventur
 ↪)
```

[119]:
```
display(DFQ13_2)
```

[120]:
```
(trainingData13, testData13) = DFQ13_2.randomSplit([0.7, 0.3],seed=0)
```

[121]:
```
# Define LinearRegression algorithm
lr = LinearRegression()

# Fit 2 models, using different regularization parameters
model13 = lr.fit(trainingData13, {lr.regParam:0.0})
```

[122]:
```
# Print the fitted model parameters
print(">>>> Model intercept: %r, coefficient: %r" % (model13.intercept, model13.
 ↪coefficients))
```

[123]:
```
predictions13_train = model13.transform(trainingData13)
predictions13_test = model13.transform(testData13)
```

[124]:
```
evaluator = RegressionEvaluator(metricName="rmse")
RMSE13_train = evaluator.evaluate(predictions13_train)
RMSE13_test = evaluator.evaluate(predictions13_test)
```

[125]:
```
print("Model: Root Mean Squared Train Error = " + str(RMSE13_train))
print("Model: Root Mean Squared Test Error = " + str(RMSE13_test))
```

[126]:
```
DFQ13_2=transData( DFQ13_1.where(DFQ13_1.tconst == "tt0120338").
 ↪select('pagerank','outDegree','Distance','movie_appear','Drama','Action','Comedy','Adventur
 ↪)
```

[127]:
```
predictions13=model13.transform(DFQ13_2)
```

[128]:
```
display(predictions13)
```

1.2.3 Q13 Final

[8]:
```
print("My final answer for Q13 is: YES, adding these variables does lower the␣
 ↪RMSE value slightly. In addition, the 'Titanic' prediction is closer to the␣
 ↪actual rating. \n Predicted: 6.3058172447665 \n RMSE: 1.2801885237607695") #␣
 ↪AS OF 29/5/2020
```

```
My final answer for Q13 is: YES, adding these variables does lower the RMSE
value slightly. In addition, the 'Titanic' prediction is closer to the actual
rating.
 Predicted: 6.3058172447665
 RMSE: 1.2801885237607695
```

Q14 Improve your model by testing different machine learning algorithms, using hyperparameter tuning on these algorithms, changing the included features. Be careful not to cheat and use test data in the training of your model.

Note: We are not testing your knowledge of different algorithms, we are just testing that you can apply the different tools in the spark toolkit and can compare between them.

What is the RMSE of you final model and what rating does it predict for the 1997 movie Titanic.

[131]:
```
DFQ13_1 = spark.read.option("sep",",").csv('dbfs:/FileStore/df/DFQ13_1.
↪csv',header=True, inferSchema = True) # to load dataframe from a file␣
↪instead of doing everything from scratch
```

[132]:
```
DFQ13_1.show()
```

[133]:
```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

def transData(data):
  # Combine columns to a dense vector (excluding the last column)
  dataFeaturesRDD = data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]])

  # Convert the RDD back to a DataFrame, labelling the columns
  featuresDF =  dataFeaturesRDD.toDF(['features','label'])

  return featuresDF
```

[134]:
```
DFQ14=transData( DFQ13_1.
↪select('pagerank','outDegree','Distance','movie_appear','Drama','Action','Comedy','Adventur
↪)
```

[135]:
```
(trainingData14, testData14) = DFQ14.randomSplit([0.9, 0.1],seed=0)
```

[136]:
```
from pyspark.ml.regression import GBTRegressor, LinearRegression
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="rmse")
```

[137]:
```
lr = LinearRegression()
gbt= GBTRegressor()
dt = DecisionTreeRegressor()
```

[138]:
```
paramGrid = ParamGridBuilder() \
    .addGrid(lr.maxIter, [50, 100]) \
    .addGrid(lr.regParam, [0.0, 0.1]) \
    .addGrid(lr.fitIntercept, [True, False]) \
    .build()

crossval = CrossValidator(estimator=lr,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(metricName="rmse"),
                          numFolds=3)  # use 3+ folds in practice
```

```
# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(trainingData14)
```

[139]:
```
prediction14_train = cvModel.transform(trainingData14)
prediction14_test = cvModel.transform(testData14)
```

[140]:
```
evaluator = RegressionEvaluator(metricName="rmse")
RMSE14_train = evaluator.evaluate(prediction14_train)
RMSE14_test = evaluator.evaluate(prediction14_test)
```

[141]:
```
print("Model: Root Mean Squared Train Error = " + str(RMSE14_train))
print("Model: Root Mean Squared Test Error = " + str(RMSE14_test))
```

[142]:
```
```

[143]:
```
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [5, 10]) \
    .addGrid(dt.maxBins, [16, 32]) \
    .build()

crossval2 = CrossValidator(estimator=dt,
                           estimatorParamMaps=paramGrid,
                           evaluator=RegressionEvaluator(metricName="rmse"),
                           numFolds=3)  # use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel2 = crossval2.fit(trainingData14)
```

[144]:
```
prediction14_train2 = cvModel2.transform(trainingData14)
prediction14_test2 = cvModel2.transform(testData14)
```

[145]:
```
evaluator = RegressionEvaluator(metricName="rmse")
RMSE14_train2 = evaluator.evaluate(prediction14_train2)
RMSE14_test2 = evaluator.evaluate(prediction14_test2)
```

[146]:
```
print("Model2: Root Mean Squared Train Error = " + str(RMSE14_train2))
print("Model2: Root Mean Squared Test Error = " + str(RMSE14_test2))
```

[147]:
```
import numpy as np
```

[148]:
```
cvModel2.getEstimatorParamMaps()[ np.argmax(cvModel.avgMetrics) ]
```

[149]:
```
```

[150]:
```
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml import Pipeline
```

[151]:
```
scaler=MinMaxScaler(inputCol="features", outputCol="scaledfeatures")
pipeline = Pipeline(stages=[scaler, dt])
```

[152]:
```
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [5,7, 10,12,15]) \
    .addGrid(dt.maxBins, [16,24, 32,40,48,54,64]) \
    .build()

crossval3 = CrossValidator(estimator=pipeline,
                           estimatorParamMaps=paramGrid,
                           evaluator=RegressionEvaluator(metricName="rmse"),
                           numFolds=3)  # use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel3 = crossval3.fit(trainingData14)
```

[153]:
```
prediction14_train3 = cvModel3.transform(trainingData14)
prediction14_test3 = cvModel3.transform(testData14)
```

[154]:
```
evaluator = RegressionEvaluator(metricName="rmse")
RMSE14_train3 = evaluator.evaluate(prediction14_train3)
RMSE14_test3 = evaluator.evaluate(prediction14_test3)
```

[155]:
```
print("Model3: Root Mean Squared Train Error = " + str(RMSE14_train3))
print("Model3: Root Mean Squared Test Error = " + str(RMSE14_test3))
```

[156]:
```
cvModel3.getEstimatorParamMaps()[ np.argmax(cvModel.avgMetrics) ] # best params␣
↪for DT
```

[157]:
```
print("Best params for DT: 'maxDepth':5 ; 'maxBins':40")
```

[158]:
```
DFQ14_1=transData( DFQ13_1.select('pageRank',
                                  'outDegree',
                                  'Distance',
                                  'movie_appear',
                                 ␣
↪'Drama','Action','Comedy','Adventure','Crime','Thriller','Sci-Fi','Romance','Fantasy','Myst
↪)
```

[159]:
```
(trainingData14_1, testData14_1) = DFQ14_1.randomSplit([0.9, 0.1],seed=0)
```

[160]:
```
scaler=MinMaxScaler(inputCol="features", outputCol="scaledfeatures")
pipeline = Pipeline(stages=[scaler, dt])

# Fit 2 models, using different regularization parameters
model_final = pipeline.fit(trainingData14_1, {dt.maxDepth:5, dt.maxBins:40})
```

[161]:
```
predictions14_1_test = model_final.transform(testData14_1)
```

[162]:
```
RMSE14_1_test = evaluator.evaluate(predictions14_1_test)
```

[163]:
```
print("Model: Root Mean Squared Test Error = " + str(RMSE14_1_test))
```

[164]:
```
print("Without 'pagerank':rmse=1.2899 ; 'outDegree': 1.2825 ; 'Distance': 1.
↪27834 ; 'movie_appear': 1.28142 \n without u")
```

[165]:
```

```

[166]:
```
DFQ14_2=transData( DFQ13_1.where(DFQ13_1.tconst == "tt0120338").
↪select('pagerank','outDegree','Distance','movie_appear','Drama','Action','Comedy','Adventur
↪)
```

[167]:
```
predictions14_2=model_final.transform(DFQ14_2)
```

[168]:
```
display(predictions14_2)
```

1.2.4 Q14 Final

[9]:
```
print("My final answer for Q14 is: Test RMSE = 1.276828459374738 ; Titanic␣
↪prediction = 6.270848489975485 \n The RMSE is slightly better than the␣
↪previous model in Q13 but the prediction is worse. This is possibly due to␣
↪overfitting.")
```

```
My final answer for Q14 is: Test RMSE = 1.276828459374738 ; Titanic prediction =
6.270848489975485
 The RMSE is slightly better than the previous model in Q13 but the prediction
is worse. This is possibly due to overfitting.
```

[170]:
```

```