Market Basket Analysis and Movie Recommendation System with Imdb Dataset

for

Algorithms for massive datasets

by

Marticola No: 942091

Amanpreet singh

Università degli Studi di Milano

TABLE OF CONTENTS

Abstract

1 Project-Market Basket Analysis **1**

1.1 Introduction 1
1.2 Implementation 1
1.3 Pyspark Tuning 2
1.4 Data Preprocessing 3
1.5 Algorithm and Implementation 5
1.6 Solution scales up with data size 8
1.7 Results 8
1.8 References and Citations 9

Chapter 1

Project-Market Basket Analysis

1.1 Introduction

This project is made to finding the optimal way for doing a Market Basket analysis and using the analysis for recommendation system while working with a Massive Dataset. I used IMDB dataset,published on Kaggle, under IMDb non-commercial licensing. Analysis is done considering movies as baskets and actors as items.

1.2 Implementation

First I manipulated the Big data using Pyspark to create baskets,then I created NGRAMS of 2 for cast for each Movie. After creating Ngrams,in order to find out which cast and crew are frequently together in movies,
For the first step, I used bigrams and counted the frequency of bigrams. A bigram is a sequence of two adjacent elements from a string of tokens. In this case, a bigram will be the names of two actors that are in a movie one after another.
For example, if 'a', 'b', 'c' are together in one movie, the bigrams will be
'a b', 'b c'.
in another movie 'b', 'c', 'd' are together, then the bigrams will be
'b c', 'c d'.

After getting all bigrams, I counted how many times each bigram appear.The results of bigram frequency will be cleaned into a python dictionary. The Top layer key is the first actor code, and first layer value is another dictionary. The second layer keys are all second actor in the different bigrams associated with actor in key value, and second layer values are the frequency of each first-actor-second-actor bigram.
So,the final dictionary will be:
{ 'a': {'b': 1 }, 'c': {'d': 2 } , {'d': {'e': 1}}

For the second step, we will give recommendation based on the bigram frequency. then recommendation is given based on the bigram frequency according to the frequency dictionary created before in Market Basket Analysis.
First need to sort the frequencies for each bigram in descending order.
For example, { 'a'{ 'c': 5, 'e':5, 'f': 7, 'g': 1}}, it will be sorted as
'a'+'f': 7
'a'+'e': 5
'a'+'c': 5
'a'+'g': 1
Then,it starts from the ones with highest frequency. and in some case if frequency of biagrams are the same then it will depend on the number of recommendations requested or if there are not enough combinations for recommendations as requested then it will start with one step further recommendation of one with highest frequency, that is to say, it will first see what actors are together with selected actor. If still more positions left, we will move to the actor with most frequency with the requested actor and then show the that actor's recommendations,

1.3 Pyspark Tuning

Executor memory =I assigned more memory to executors to keep the algorithm fast and avoiding pending tasks

Driver Memory= I assigned 2gb Ram to drivers,for storing temporary variables for spark as I dont used too many collects or broadcasts for this algorithm

Shuffle partitons=I kept number of partitons to 3,so there is no long ques for pending tasks and slow the speed,but in case of Scalability ,these could be increased

Apache Arrow= I enabled Apache Arrow to accelerate analytic workload for NGRAMS

1.4 Data Preprocessing

Ratings Dataframe has all the movies and their respective Ratings tconst=Unique id for each movie

```
ratings.show()
```

```
+---------+-------------+--------+
|   tconst|averageRating|numVotes|
+---------+-------------+--------+
|tt0000001|          5.6|    1550|
|tt0000002|          6.1|     186|
|tt0000003|          6.5|    1207|
|tt0000004|          6.2|     113|
|tt0000005|          6.1|    1934|
|tt0000006|          5.2|     102|
|tt0000007|          5.5|     615|
|tt0000008|          5.4|    1668|
|tt0000009|          5.4|      81|
|tt0000010|          6.9|    5549|
|tt0000011|          5.2|     236|
|tt0000012|          7.4|    9440|
|tt0000013|          5.7|    1447|
|tt0000014|          7.1|    4115|
|tt0000015|          6.1|     742|
|tt0000016|          5.9|    1088|
|tt0000017|          4.7|     214|
|tt0000018|          5.4|     447|
|tt0000019|          5.5|      17|
|tt0000020|          5.0|     249|
+---------+-------------+--------+
only showing top 20 rows
```

Castcrew1 Dataframe has all the cast and crews that worked in a particular movie nconts=Uniqueid for each cast and crew

```
castcrew12.show()
```

```
+---------+---------+---------+---------+---------+
|   nconst|     mov1|     mov2|     mov3|     mov4|
+---------+---------+---------+---------+---------+
|nm0000001|tt0050419|tt0053137|tt0043044|tt0072308|
|nm0000002|tt0117057|tt0037382|tt0071877|tt0038355|
|nm0000003|tt0049189|tt0059956|tt0054452|tt0057345|
|nm0000004|tt0078723|tt0080455|tt0072562|tt0077975|
|nm0000005|tt0050986|tt0083922|tt0069467|tt0050976|
|nm0000006|tt0077711|tt0038787|tt0036855|tt0038109|
|nm0000007|tt0033870|tt0037382|tt0034583|tt0043265|
|nm0000008|tt0078788|tt0070849|tt0068646|tt0047296|
|nm0000009|tt0059749|tt0061184|tt0057877|tt0087803|
|nm0000010|tt0031867|tt0029870|tt0035575|tt0042041|
|nm0000011|tt0035896|tt0027996|tt0034167|tt0044706|
|nm0000012|tt0031210|tt0056687|tt0035140|tt0042192|
|nm0000013|tt0053172|tt0048317|tt0060463|tt0055100|
|nm0000014|tt0031381|tt0041452|tt0040806|tt0029843|
|nm0000015|tt0048545|tt0048028|tt0044245|tt0049261|
|nm0000016|tt0091763|tt0096320|tt0057345|tt0069946|
|nm0000017|tt0051201|tt0055031|tt0021156|tt0052311|
|nm0000018|tt0052365|tt0080736|tt0054331|tt0043338|
|nm0000019|tt0056801|tt0050783|tt0071129|tt0053779|
|nm0000020|tt0064116|tt0082846|tt0050083|tt0032551|
+---------+---------+---------+---------+---------+
only showing top 20 rows
```

Castcrew12 dataframe is a manipulation of castcrew1 dataframe to split all the movies of different actors in different columns.This dataframe has duplicate nconst with multiple combinations of different movies of each actor because the orignal data has duplication and different movies are listed under duplicated nconst so all of these movies needs to be bundled together


```
castcrew12.show()
```

```
+---------+---------+---------+---------+---------+
|   nconst|     mov1|     mov2|     mov3|     mov4|
+---------+---------+---------+---------+---------+
|nm0000001|tt0050419|tt0053137|tt0043044|tt0072308|
|nm0000002|tt0117057|tt0037382|tt0071877|tt0038355|
|nm0000003|tt0049189|tt0059956|tt0054452|tt0057345|
|nm0000004|tt0078723|tt0080455|tt0072562|tt0077975|
|nm0000005|tt0050986|tt0083922|tt0069467|tt0050976|
|nm0000006|tt0077711|tt0038787|tt0036855|tt0038109|
|nm0000007|tt0033870|tt0037382|tt0034583|tt0043265|
|nm0000008|tt0078788|tt0070849|tt0068646|tt0047296|
|nm0000009|tt0059749|tt0061184|tt0057877|tt0087803|
|nm0000010|tt0031867|tt0029870|tt0035575|tt0042041|
|nm0000011|tt0035896|tt0027996|tt0034167|tt0044706|
|nm0000012|tt0031210|tt0056687|tt0035140|tt0042192|
|nm0000013|tt0053172|tt0048317|tt0060463|tt0055100|
|nm0000014|tt0031381|tt0041452|tt0040806|tt0029843|
|nm0000015|tt0048545|tt0048028|tt0044245|tt0049261|
|nm0000016|tt0091763|tt0096320|tt0057345|tt0069946|
|nm0000017|tt0051201|tt0055031|tt0021156|tt0052311|
|nm0000018|tt0052365|tt0080736|tt0054331|tt0043338|
|nm0000019|tt0056801|tt0050783|tt0071129|tt0053779|
|nm0000020|tt0064116|tt0082846|tt0050083|tt0032551|
+---------+---------+---------+---------+---------+
only showing top 20 rows
```

Data Dataframe=This dataframe is created to create baskets as Movie(tconst) as a basket and all cast and crew in that particular movie as items(nconst),and also bundling all the duplicated nconst and grouping their all the movies and then creating pivot and Left join to make baskets.

```
data.show(truncate=False)

+---------+-------------+--------+------------------------------------------------------------------
|tconst   |averageRating|numVotes|castcrew
+---------+-------------+--------+------------------------------------------------------------------
|tt0000003|6.5          |1207    |[nm0721526, nm5442194, nm5442200, nm5442215, nm5442293, nm1335271]
|tt0000012|7.4          |9440    |[nm9735580, nm0525908, nm0525910, nm2880396]
|tt0000014|7.1          |4115    |[nm0166380, nm0244989, nm0525910]
|tt0000018|5.4          |447     |[nm3692071]
|tt0000029|5.9          |2639    |[nm2350838, nm0525900]
|tt0000121|4.6          |43      |[nm0609678, nm0780534, nm5718242, nm0832461]
|tt0000165|5.2          |76      |[nm0024876, nm0083196, nm0556371, nm0563758, nm0278321]
|tt0000174|4.9          |98      |[nm1024447, nm0471818]
|tt0000215|4.4          |90      |[nm0471818]
|tt0000247|4.9          |371     |[nm0784327, nm2156608, nm2259742, nm2261015, nm2263402, nm5858730, nm6010696, nm6023385, nm6023386, nm6114857,
|tt0000319|5.9          |99      |[nm1272063, nm1272675, nm0793094]
|tt0000335|6.2          |39      |[nm0675260, nm1010955, nm1011210, nm1012612, nm1012621, nm0095714, nm0675239, nm0675140]
|tt0000420|6.3          |2176    |[nm1539007, nm0926498, nm1012587, nm0164281, nm0378408, nm1539443]
|tt0000488|6.9          |632     |[nm0794919, nm1096358, nm0298300, nm0422465, nm0259860]
|tt0000499|7.6          |2889    |[nm8706003, nm0257866, nm0617588]
|tt0000546|6.7          |1566    |[nm0106151, nm0567363]
|tt0000583|5.5          |20      |[nm0022608, nm0470250, nm0647719, nm5216764, nm5216822, nm0676941, nm0526168]
|tt0000584|5.0          |46      |[nm0305591, nm0400103, nm0191133]
|tt0000602|4.8          |146     |[nm0630737, nm4572288]
|tt0000609|4.5          |11      |[nm0143332, nm0143333, nm0892614]
+---------+-------------+--------+------------------------------------------------------------------
only showing top 20 rows
```

1.5 Algorithm and Implementation

1.5.1 NGRAMS

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (n 1)–order Markov model.
Two benefits of n-gram models (and algorithms that use them) are simplicity and scalability – with larger n, a model can store more context with a well-understood space–time tradeoff, enabling small experiments to scale up efficiently.
I used bigrams,A bigram is an n-gram for n=2.

```
ngramDataFrame.show(truncate=False)

+------------------------------------------------------------------------------------------------
|bigrams
+------------------------------------------------------------------------------------------------
|[nm0721526 nm5442194, nm5442194 nm5442200, nm5442200 nm5442215, nm5442215 nm5442293, nm5442293 nm1335271]
|[nm9735580 nm0525908, nm0525908 nm0525910, nm0525910 nm2880396]
|[nm0166380 nm0244989, nm0244989 nm0525910]
|[]
|[nm2350838 nm0525900]
|[nm0609678 nm0780534, nm0780534 nm5718242, nm5718242 nm0832461]
|[nm0024876 nm0083196, nm0083196 nm0556371, nm0556371 nm0563758, nm0563758 nm0278321]
|[nm1024447 nm0471818]
|[]
|[nm0784327 nm2156608, nm2156608 nm2259742, nm2259742 nm2261015, nm2261015 nm2263402, nm2263402 nm5858730, nm5858730 nm6010696, nm6010696 nm60233
|[nm1272063 nm1272675, nm1272675 nm0793094]
|[nm0675260 nm1010955, nm1010955 nm1011210, nm1011210 nm1012612, nm1012612 nm1012621, nm1012621 nm0095714, nm0095714 nm0675239, nm0675239 nm06751
|[nm1539007 nm0926498, nm0926498 nm1012587, nm1012587 nm0164281, nm0164281 nm0378408, nm0378408 nm1539443]
|[nm0794919 nm1096358, nm1096358 nm0298300, nm0298300 nm0422465, nm0422465 nm0259860]
|[nm8706003 nm0257866, nm0257866 nm0617588]
|[nm0106151 nm0567363]
|[nm0022608 nm0470250, nm0470250 nm0647719, nm0647719 nm5216764, nm5216764 nm5216822, nm5216822 nm0676941, nm0676941 nm0526168]
|[nm0305591 nm0400103, nm0400103 nm0191133]
|[nm0630737 nm4572288]
|[nm0143332 nm0143333, nm0143333 nm0892614]
+------------------------------------------------------------------------------------------------
```

1.5.2 Nested Dictionary for Frequency Mining

After getting all bigrams, I counted how many times each bigram appear.The results of bigram frequency will be cleaned into a python dictionary. The Top layer key is the first actor code, and first layer value is another dictionary. The second layer keys are all

second actor in the different bigrams associated with actor in key value, and second layer values are the frequency of each first-actor-second-actor bigram.

```
table
```

```
'nm0573806': {'nm0574726': 1},
'nm1288013': {'nm0203559': 1},
'nm0265466': {'nm0562838': 1},
'nm0562838': {'nm0360617': 1},
'nm0394743': {'nm0480104': 1, 'nm1707869': 1},
'nm1707869': {'nm5459192': 1},
'nm5459192': {'nm0367542': 1},
'nm0081940': {'nm0460021': 1},
'nm0460021': {'nm0481193': 1},
'nm0481193': {'nm0886389': 1},
'nm0886389': {'nm0886582': 1, 'nm0887223': 1},
'nm0887223': {'nm0297274': 1},
'nm0297274': {'nm0297210': 1},
'nm0297210': {'nm0458607': 1, 'nm0486247': 1, 'nm1271528': 1},
'nm0458607': {'nm0543029': 1, 'nm0888028': 1},
'nm0888028': {'nm0576791': 1},
'nm0313906': {'nm0285633': 1, 'nm0612958': 1},
'nm0612958': {'nm2699442': 1},
'nm2699442': {'nm0240538': 1},
'nm0240538': {'nm0368875': 1, 'nm0601067': 1, 'nm0932806': 1},
'nm0043776': {'nm1137520': 1},
'nm1137520': {'nm0940485': 1},
'nm0725845': {'nm0790977': 1},
'nm0055865': {'nm0068931': 1, 'nm0171281': 1},
'nm0068931': {'nm0234089': 1},
'nm0234089': {'nm0832955': 1},
'nm0832955': {'nm0048466': 1, 'nm0135053': 1},
'nm0135053': {'nm0307863': 1, 'nm0444404': 1},
'nm0444404': {'nm0163549': 1},
'nm0163549': {'nm0201254': 1, 'nm0240647': 1, 'nm0308551': 1, 'nm0613860': 1},
```

1.5.3 Baskets and Frequency

With this piece of code in image below and selecting the range,for example here it is showing the bigrams who repeated more than 3 times in whole dataFrame

```
for firstWord in table:
    for secondWord in table[firstWord]:
        if table[firstWord][secondWord] > 3:
            print(firstWord, " + ", secondWord, ": ", table[firstWord][secondWord])
```

```
nm1410831  +  nm1410832 :  4
nm1141563  +  nm1141616 :  4
nm0268716  +  nm0268717 :  4
nm3186243  +  nm3186250 :  4
nm2684503  +  nm2686766 :  4
nm0244505  +  nm0244523 :  4
nm1721924  +  nm1724662 :  4
nm0994831  +  nm0994966 :  4
nm0970179  +  nm0970239 :  4
nm0968911  +  nm0968932 :  4
nm1288519  +  nm1288938 :  4
nm2369182  +  nm2375001 :  4
nm4845589  +  nm4845617 :  4
nm0402867  +  nm0402970 :  4
nm0817915  +  nm0818139 :  4
nm4672282  +  nm4672646 :  4
nm2340048  +  nm2340183 :  4
nm1172359  +  nm1172360 :  4
nm1068656  +  nm1070798 :  4
nm2306904  +  nm2309639 :  4
nm0709120  +  nm0761694 :  4
nm5008801  +  nm5008865 :  4
nm3075069  +  nm3076505 :  4
nm1649516  +  nm1651852 :  4
nm2654523  +  nm2654526 :  4
nm2663953  +  nm2664666 :  4
nm1179503  +  nm1179582 :  4
nm1318843  +  nm1319757 :  4
nm2213735  +  nm2332106 :  4
nm1312722  +  nm1312843 :  4
nm3022868  +  nm3023013 :  4
```

1.5.4 Recommendation System

For example, if in a movie these three are together 'a', 'b', 'c' bigrams will be
'a b', 'b c'.
in another movie 'b', 'c', 'd', then the bigrams will be
'b c', 'c d'.
then recommendation is given based on the bigram frequency according to the frequency dictionory created before in Market Basket Analysis first need to sort the frequencies for each bigram in decending order.
For example, { 'a'{ 'c': 5, 'e':5, 'f': 7, 'g': 1}}, it will be sorted as
'a'+'f': 7
'a'+'e': 5
'a'+'c': 5
'a'+'g': 1
Actors will be recommended from the ones with highest frequency
In image below ,finding the most frequent(Top 5) cast/crew with unique id (nconst) nm1468859 ,Name Naomi Klein (Filmotrographer)

```
xxx=getRecommend("nm1468859",5)
castcrew1.filter(castcrew1.nconst.isin(xxx)).select('primaryName').collect()
```

```
[Row(primaryName='Avi Lewis'),
 Row(primaryName='Anoop Singh'),
 Row(primaryName='Sean Devlin'),
 Row(primaryName='Alex Kelly'),
 Row(primaryName='Arnold Harberger')]
```

then looking for all the movies in the Movies Dataframe ,in which those recommended cats/crew have worked and suggesting those movies

```
movierecommendations=castcrew12.filter(castcrew12.nconst.isin(xxx))
movierecommendations=movierecommendations.select("mov1",'mov2','mov3','mov4').rdd.flatMap(lambda x: x)
movierecommendations1=movies.filter(movies.tconst.isin((movierecommendations).collect()))
movierecommendations1.select("primaryTitle").rdd.flatMap(lambda x: x).collect()
```

```
['Commanding Heights: The Battle for the World Economy',
 'The Take',
 'Zach & Avery of Fergus',
 "Fire the Director: The Making of 'The Take'",
 'The Shock Doctrine',
 'Nothing Rhymes with Ngapartji',
 'This Changes Everything',
 'Queen of the Desert',
 'Chicago Boys',
 'The Party',
 'Whoa Canada',
 'Island of the Hungry Ghosts',
 'When the Storm Fades']
```

1.6 Solution scales up with data size

Algorithm used Pyspark for all the heavy weightlifting to create a final dataset,and to create Ngrams plus to find movies and actors recommended by the Recommendation System and Pyspark is fine tuned to handle heavy tasks efficiently.

Rather than using computationally expensive algorithms like Apriori Algorithm and creating Association Matrix es,which will involve multiple Cartesian joins and will be impossible to maintain scalability of the data,I researched for other efficient methods,and implemented NGRAMS with Frequency Mining method ,taking consideration of scalability as Ngrams simplicity and scalability – with larger n, a model can store more context with a well-understood space–time trade off, enabling small experiments to scale up efficiently.

1.7 Results

Testing how James dean ,the famous 50s actor ,got recommended with which people,and in the images below you can see that results are good

```
xxx=getRecommend("nm0000015",5)
castcrew1.filter(castcrew1.nconst.isin(xxx)).select('primaryName').collect()
```

```
[Row(primaryName='Ernest Haller'),
 Row(primaryName='Ted D. McCord'),
 Row(primaryName='Mary Anderson'),
 Row(primaryName='Don Appell'),
 Row(primaryName='Virginia Brissac')]
```

```
movierecommendations=castcrew12.filter(castcrew12.nconst.isin(xxx))
movierecommendations=movierecommendations.select("mov1",'mov2','mov3','mov4').rdd.flatMap(lambda x: x)
movierecommendations1=movies.filter(movies.tconst.isin((movierecommendations).collect()))
movierecommendations1.select("primaryTitle").rdd.flatMap(lambda x: x).collect()
```

```
['Gone with the Wind',
 'The Little Foxes',
 'Henry and Dizzy',
 'Lifeboat',
 'Wilson',
 'Mildred Pierce',
 'The Scarlet Clue',
 'Johnny Belinda',
 'The Treasure of the Sierra Madre',
 'The Vaughn Monroe Show',
 'Campbell Summer Soundstage',
 'Executive Suite',
 'East of Eden',
 'Rebel Without a Cause',
 'The Vic Damone Show',
 'What Ever Happened to Baby Jane?',
 'The Sound of Music',
 'Stop! Look! and Laugh!']
```

1.8 References and Citations

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S. (2012). Scalable k-means++. Proceedings of the VLDB Endowment, 5(7), 622-633.

Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System.XGBoost: A Scalable Tree Boosting System, 785-794.

Collins, M. J. (1996, June). A new statistical parser based on bigram lexical dependencies. In Proceedings of the 34th annual meeting on Association for Computational Linguistics (pp. 184-191). Association for Computational Linguistics.

Friedman JH (2001). "Greedy function approximation: a gradient boosting machine." Annals of Statistics, pp. 1189–1232.

Ke, G., Meng, Q. (2017). GBDT with GOSS and EFB LightGBM.

Machine Learning Library Guide. (2017, December 20). Retrieved from: https://spark.apache.org/docs/2.1.1/ml-guide.html

Owies, M., Qendeel, N., Al Barri, S. (2017). Market Basket Analysis.