Market-basket Analysis in IMDb Dataset

Algorithms for Massive Datasets
Università degli Studi di Milano
Prof. Dario Malchiodi

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Peesapati Venkata Sai Vamsi(933987)
Venkatasaivamsi.peesapati@studenti.unimi.it

Revan tokathi(924464)
revan.tokathi@studenti.unimi.it

Contents

We declare that this material, which will be now submitted for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.

1 THE DATASET

One of the main issues in data science is frequent item sets, especially when dealing with larger amounts of data. This is because it is complicated for human examinations, thus a great effort is needed on handling the large amounts of data.

In this assignment, we have used IMDb dataset that is found in kaggle. Each dataset is contained in a gzipped, tab-separated-values (TSV). The first line in each file contains headers that describe what is in each column. A '\N' is used to denote that a particular field is missing or null for that title/name. The available datasets are

- *title.akas.tsv.gz* : This dataset contains the information regarding the titles. The attributes are,

  titleid : unique identifier of the title.

  ordering : a number to uniquely identify rows for a given title.

  title : the localised title.

  region : the region for this version of the title.

  language : the language of the title.

  types : enumerated set of attributes for this alternative title.

  attributes : additional terms to describe the alternative title.

  isOriginalTitle : 0 - not original title, 1 - original title.

- *title.basic.tsv.gz* : This dataset contains the information for titles.

  tconst : unique identifier of title.

  titleType : the type/format of the title.

  primaryTitle : the popular title used at point of release.

  originalTitle : original title.

  isAdult : 0 - Adult, 1 - not Adult.

  startYear : release year of the title.

  endYear : TV series end year.

  runtimeMinutes : primary runtime of title.

  genres : includes three genres

- *title.principals.tsv.gz* : This dataset contains the principal cast and crew for titles.

  ordering : a number to uniquely identify rows for a given title.

  nconst : unique identifier of person/name.

  category : category of job that person was in.

  job : the specific job title, if applicable.

  characters : the name of the character played.

- *title.ratings.tsv.gz* : This dataset contains the IMDb ratings for the titles.

  tconst : unique identifier of the title.

  averageRatings : weighted average of all user ratings.

  numVotes : number of votes the titles has received.

- *name.basics.tsv.gz* : contains the information for names.

  nconst : unique identifier of name/person.

  primaryName : name by which the person is credited often.

  birthYear : year of birth.

  deathYear : year of death, if applicable.

  primaryProfession : top 3 professions of the person.

  knownforTitles : titles the person is known for.

2 DATA ORGANIZATION

Data organisation is a method used to classify and organise the datasets to make them more useful. In this assignment, we have used the Spark DataFrame API for loading the data, preprocessing and building the detector of similar items. Spark DataFrame is a dataset organized into named columns, conceptually equivalent to a table in a relational database or a data frame used in by programs such as R or Python. Spark SQL is a module for structured data processing, that provides information about the structure of both the data and the computation that is being performed, including the ability to join RDDs, represent a collection of items distributed across many compute nodes that can be manipulated in parallel, and SQL tables. There are several ways to interact with Spark SQL, including SQL, and Dataset API, and when computing a result, the same execution engine is used, independent of which API/language is used to express the computation. This unification means that developers can easily switch back and forth between different APIs based on which provides the most natural way to express a given transformation, meaning that when running SQL from within another programming language, like it will be done on this project, the results will be returned in the form of a Dataset or DataFrame. We have also used python in spark architecture to implement priori algorithm.

3 DATA PRE-PROCESSING

Data cleaning is the first step and also one of the most important in any data science project. The main ideology of this assignment is to implement a system finding frequent itemsets, by considering movies as baskets and actors as items. From the above datasets, taking *title.akas.tsv.gz*, *title.principals.tsv.gz*, *name.basics.tsv.gz datasets* as, it is important to determine the actors based on the movies. For starting the pre-processing, we fetch the datasets and filter the records to actors and actresses from the dataset *title.principals.tsv.gz*. To get the names of the actors we join both *title.principals.tsv.gz*, *name.basics.tsv.gz*. We split the movies based on primary names of the actor using split() function and explode the movies using explode() function which returns a new row for

each element, so that we can use it for joins. As we have obtained the exploded data, we join the exploded data to *title.akas.tsv.gz* dataset to get the titles of the movies. As the data was reduced to the requirements of the project.

4 Implementation of the algorithms

In order to achieve the frequent itemsets, we consider FP-Growth Algorithm and Apriori Algorithm.

4.1 FP-Growth Algorithm

"FP" stands for frequent pattern. Given a dataset of transactions, the first step of FP-growth is to calculate item frequencies and identify frequent items. Different from Apriori algorithm, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly, which are usually expensive to generate. After the second step, the frequent itemsets can be extracted from the FP-tree.

FP-growth implementation takes the following (hyper-)parameters:

- minSupport: the minimum support for an itemset to be identified as frequent. For example, if an item appears 3 out of 5 transactions, it has a support of 3/5=0.6.
- minConfidence: minimum confidence for generating Association Rule. Confidence is an indication of how often an association rule has been found to be true. For example, if in the transactions itemset X appears 4 times, X and Y co-occur only 2 times, the confidence for the rule X =¿ Y is then 2/4 = 0.5.
- numPartitions: the number of partitions used to distribute the work. By default the param is not set, and number of partitions of the input dataset is used.

The FPGrowthModel provides:

- freqItemsets: frequent itemsets in the format of DataFrame.
- associationRules: association rules generated with confidence above minConfidence.
- transform: For each transaction in itemsCol, the transform method will compare its items against the antecedents of each association rule. If the record contains all the antecedents of a specific association rule, the rule will be considered as applicable and its consequents will be added to the prediction result.

4.2 Apriori Algorithm

The A-Priori Algorithm is designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather than one pass.

The First Pass of A-Priori : In the first pass, we create two tables. The first table, if necessary, translates item names into integers from 1 to n. The other table is an array of counts the ith array element counts the occurrences of the item numbered i. Initially, the counts for all the items are 0. As we read baskets, we look at each item in the basket and translate its name into an integer. We use that integer to index into the array of counts, and we add 1 to the integer found there.

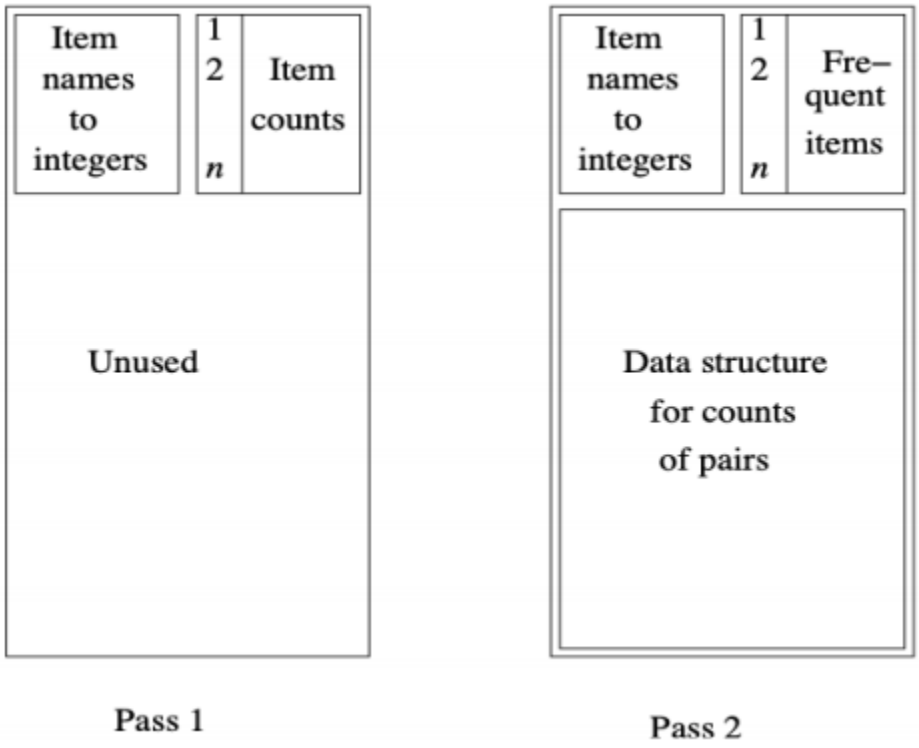
Between the Passes of A-Priori : After the first pass, we examine the counts of the items to determine which of them are frequent as singletons. We set the threshold s sufficiently high that we do not get too many frequent sets; a typical s would be 1% of the baskets. For the second pass of A-Priori, we create a new numbering from 1 to m for just the frequent items. This table is an array indexed 1 to n, and the entry for i is either 0, if item i is not frequent, or a unique integer in the range 1 to m if item i is frequent. We shall refer to this table as the frequent-items table.

The Second Pass of A-Priori : During the second pass, we count all the pairs that consist of two frequent items. Consider that a pair cannot be frequent unless both its members are frequent. Thus, we miss no frequent pairs.

The mechanics of the second pass are as follows.

- For each basket, look in the frequent-items table to see which of its items are frequent.
- In a double loop, generate all pairs of frequent items in that basket.
- For each such pair, add one to its count in the data structure used to store counts.

Finally, at the end of the second pass, examine the structure of counts to determine which pairs are frequent.



The same technique used for finding frequent pairs without counting all pairs lets us find larger frequent itemsets without an exhaustive count of all sets. In the A-Priori Algorithm, one pass is

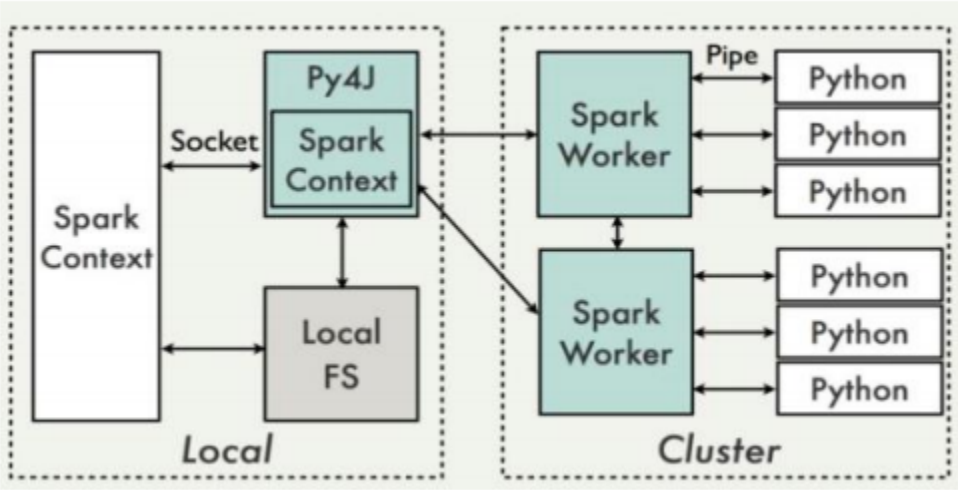
taken for each set-size k. If no frequent itemsets of a certain size are found, then monotonicity tells us there can be no larger frequent itemsets, so we can stop.

The pattern of moving from one size k to the next size k + 1 can be sum- marized as follows. For each size k, there are two sets of itemsets,

- Ck is the set of candidate itemsets of size k – the itemsets that we must count in order to determine whether they are in fact frequent.
- Lk is the set of truly frequent itemsets of size k

5 Scalability

Spark can load data into memory on the worker nodes, many distributed computa- tions, even ones that process terabytes of data across dozens of machines, can run in a few seconds. This makes the sort of iterative, ad hoc, and exploratory analysis commonly done in shells a good fit for Spark. Spark provides both Python and Scala shells that have been augmented to support connecting to a cluster. Our detector is fast and efficient and scales with the data size. This is because of the usage of Spark, which uses Regular expressions and Spark DataFrame based API, and the default datatype is "String", which occupies relatively very less memory. Apache Spark is a unified analytics engine for large-scale data processing. It achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine. Spark's Catalyst op- timizer converts user inputs into a Directed Acyclic Graph (DAG) of transformations before distributing the optimized tasks across a network of computers for in-memory processing. Hence, the reason, the data cleaning was done in seconds.



A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. Represents an im- mutable, partitioned collection of elements that can be operated on in parallel. This class contains the basic operations available on all RDDs, such as map, filter, and persist. RDD transformation that is used to apply the transformation function (lambda) on every element of RDD/DataFrame and returns a new RDD.

6 Description of experiments

To fetch movies as baskets and actors as items by grouping by movie title. As the list of baskets is obtained we run FP-Growth algorithm to get frequent itemsets, by taking min support as 0.0001 and minconfidence as 0.001 as it yields better results. And hence deduce the association rules.

antecedent	consequent	confidence	lift	support
[Stephen Hunter, William Kircher, Sylvester McCoy, Dean O'Gorman, Graham McTavish, Ken Stott, Aidan Turner, Adam Brown]	[Peter Hambleton]	1.0	8218.576923076922	1.1231590
[Stephen Hunter, William Kircher, Sylvester McCoy, Dean O'Gorman, Graham McTavish, Ken Stott, Aidan Turner, Adam Brown]	[Richard Armitage]	1.0	7368.379310344827	1.1231590
[Stephen Hunter, William Kircher, Sylvester McCoy, Dean O'Gorman, Graham McTavish, Ken Stott, Aidan Turner, Adam Brown]	[Dean Knowsley]	1.0	7368.379310344827	1.1231590
[Stephen Hunter, William Kircher, Sylvester McCoy, Dean O'Gorman, Graham McTavish, Ken Stott, Aidan Turner, Adam Brown]	[Martin Freeman]	1.0	7914.185185185185	1.1231590
[Stephen Hunter, William Kircher, Sylvester McCoy, Dean O'Gorman, Graham McTavish, Ken Stott, Aidan Turner, Adam Brown]	[John Callen]	1.0	8412.716535433072	1.1231590
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Rachael Lillis]	1.0	6636.11801242236	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Eric Stuart]	1.0	6595.154320987654	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Veronica Taylor]	1.0	6002.331460674158	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Ayako Shiraishi]	1.0	5479.051282051282	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Madeleine Blaustein]	1.0	5263.128078817735	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Inuko Inuyama]	1.0	5870.412087912087	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Chinami Nishimura]	1.0	6248.040935672514	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Ikue Ôtani]	1.0	6248.040935672514	1.3197119
[Michael Haigney, John Laeffler, Rica Matsumoto, Yûji Ueda, Mayumi Izuka]	[Rodger Parsons]	1.0	7419.548611111111	1.3197119
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[Sylvester McCoy]	1.0	8346.9921875	1.1231590
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[William Kircher]	1.0	8686.300813008813	1.1231590
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[Peter Hambleton]	1.0	8218.576923076922	1.1231590
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[John Callen]	1.0	8412.716535433072	1.1231590
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[Dean O'Gorman]	1.0	7973.246268656716	1.1231590
[Stephen Hunter, Martin Freeman, Graham McTavish, Richard Armitage, Aidan Turner, Adam Brown]	[Ken Stott]	1.0	7122.766666666666	1.1231590

To implement the apriori algorithm, we parallelise the data so that the collections are copied to form a distributed dataset that can be operated on in parallel. We run supportrdd to get the first item support tuples. These support values are achieved by considering each item separately. Inorder to decide which itemsets will stay in the support tables, we need to define a minimum support value. If any support values in itemsets are less than the min support value, we should remove that item. To get more consistent result we set min support was 2, that is, if any item or items occurs in a transaction less than 2 times we neglect it. We also creat a baseRdd which represents our first support values for every single item. This will be updated with upcoming support vales, we also define supportRDD that shows only items without support values. Now, we create an algorithm that generates combinations of items and find patterns to remove them. As the support value we have set earlier is 2 that means the first support table is going to be created will have all itemsets which has 2 items. Inorder to create combinations of items, we use the cartesian function and distinct method to get a unique itemsset. We then filter the process that control each item and obtain the frequencies using reduceByKey method and filter them according to min support value. We add the final support table to baseRDD and update the supportRdd. Finally, we calculate the confidence values and filter data according to confidence.

7 Results and Discussion

FP-Growth algorithm provides the confidence for very low value of support while, Apriori algorithm could not provide confidence for low values. Because the dataset we have ,have very low similarity features in basket.

References

[1] J. Leskovec, A. Rajaraman, and J. D. Ullman, Mining of massive datasets. Cam- bridge University Press, 3 ed., 2014.

[2] Kaggle, "Dataset from IMDb to make a recommendation system". Retrieved from Kaggle, https://www.kaggle.com/ashirwadsangwan/imdb-dataset.

[3] Apache.org, "RDD Programming Guide". Retrieved from Apache.org, https://spark.apache.org/docs/latest/rdd-programming-guide.html.

[4] Apache.org, "RDD Programming Guide". Retrieved from Apache.org, https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html.

[5] Towards Data Science,"Big Data Market Basket Analysis". Retrieved from Towards Data Science, https://towardsdatascience.com/big-data-market-basket-analysis-with-apriori-algorithmon-spark-9ab094b5ac2c.