**MOVIE RECOMMENDATION SYSTEM**

A PROJECT REPORT

***Submitted by***

BL.EN.U4AIE19007 APOORVA.M

BL.EN.U4AIE19041 TANUJ.M

BL.EN.U4AIE19068 AISHWARYA.V

***for the course***

***19AIE214-Big Data Analytics***

***Guided and Evaluated by***

***Mr.Saravanan S***

IN

ARTIFICIAL INTELLIGENCE ENGINEERING



AMRITA SCHOOL OF ENGINEERING, BANGALORE

**BANGALORE 560 035**

#### **TABLE OF CONTENTS**

1. **ABSTRACT………………………………………………...3**
2. **INTRODUCTION………………………………………….4**
3. **SYSTEM DESIGN…………………………………………12**
4. **IMPLEMENTATION……………………………………..15**
5. **RESULTS…………………………………………………..25**
6. **CONCLUSION & FUTURE ENHANCEMENTS………28**
7. **REFERENCES…………………………………………….28**

**ABSTRACT**

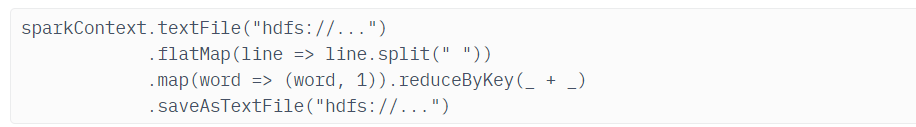
Movie Recommendation system recommends movies for its users to watch, based on their film preferences using collaborative filtering. Recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications. Data taken from [Group lens site](https://grouplens.org/datasets/movielens/1m/) and contain 1 million ratings from 6000 users on 4000 movies. tools we used Spark and Python - PySpark, ALS, Regression Evaluator. we have used Apache Spark ML, alternating least squares (ALS) for collaborative filtering for making recommendations. Alternating least square (ALS) matrix factorization basically take a large (or potentially huge) matrix and factor it into some smaller representation of the original matrix through alternating least squares. We end up with two or more lower dimensional matrices whose product equals the original one. ALS comes inbuilt in Apache Spark. The idea behind the project is to improve customer's experience through personalized recommendations based on prior user feedback.

**KeyWords**: Pyspark, Alternating least square, Matrix factorization, Collaberative Filtering, MovieRecommendation, Prediction,

**INTRODUCTION TO APACHE SPARK**

Spark is an Apache project advertised as “lightning-fast cluster computing”. It has a thriving open-source community and is the most active Apache project now.

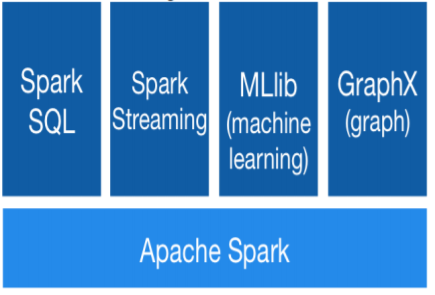
Spark provides a faster and more general data processing platform. Spark lets you run programs up to 100x faster in memory, or 10x faster on disk, than Hadoop. Last year, Spark took over Hadoop by completing the 100 TB Daytona Gray Sort contest 3x faster on one tenth the number of machines and it also became the fastest open-source engine for sorting a petabyte.

Spark also makes it possible to write code more quickly as you have over 80 high-level operators at your disposal. To demonstrate this, let’s have a look at the “Hello World!” of Bigdata: The Word Count example. Written in Java for MapReduce it has around 50 lines of code, whereas in Spark (and Scala) you can do it as simply as this:

Another important aspect when learning how to use Apache Spark is the interactive shell (REPL) which it provides out-of-the box. Using REPL, one can test the outcome of each line of code without first needing to code and execute the entire job. The path to working code is thus much shorter and ad-hoc data analysis is made possible.

Additional key features of Spark include:

Currently provides APIs in Scala, Java, and Python, with support for other languages (such as R) on the way. Integrates well with the Hadoop ecosystem and data sources (HDFS, Amazon S3, Hive, HBase, Cassandra, etc.) Can run on clusters managed by Hadoop YARN or Apache Mesos and can also run standalone.

The Spark core is complemented by a set of powerful, higher-level libraries which can be seamlessly used in the same application. These libraries currently include SparkSQL, Spark Streaming, MLlib (for machine learning), and GraphX, each of which is further detailed in this article. Additional Spark libraries and extensions are currently under development as well.

**Python Spark (pySpark)**

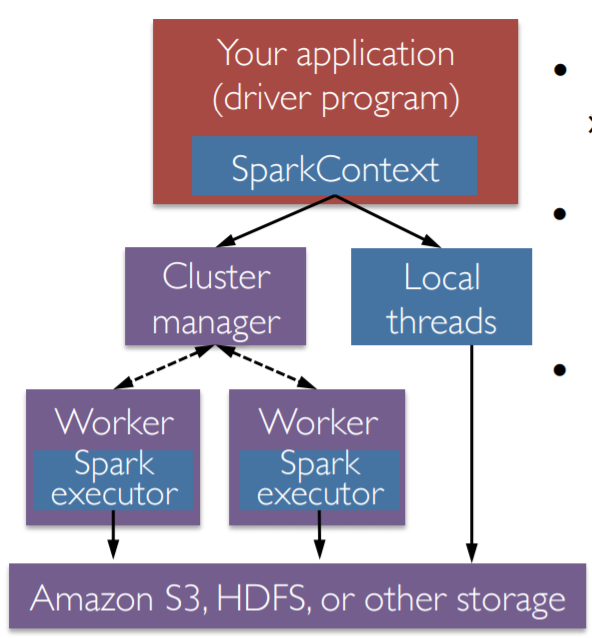
We are using the Python programming interface to Spark (pySpark) .It provides an easy-to-use programming abstraction and parallel runtime. RDDs are the key concept.

**Spark Driver and Workers:**

A Spark program is two programs: A driver program and a workers program

Worker programs run on cluster nodes or in local threads.RDDs are distributed

across workers.



**Spark Context**

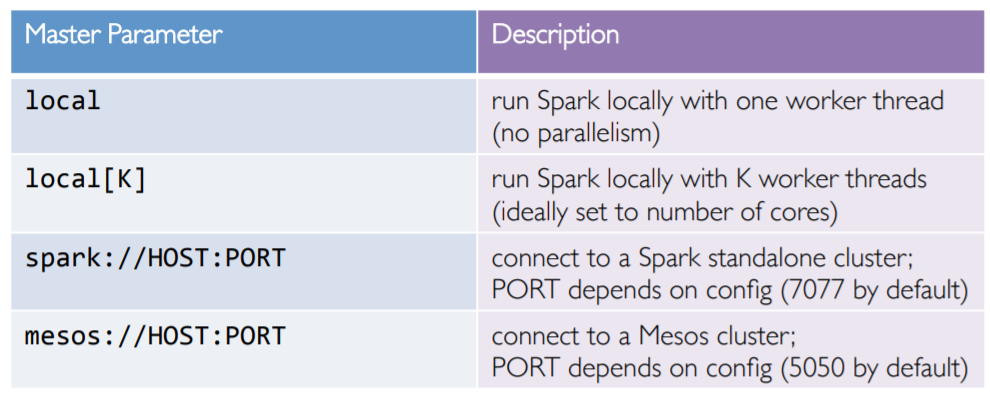
A Spark program first creates a SparkContext object

* » Tells Spark how and where to access a cluster
* » pySpark shell and Databricks Cloud automatically create the sc variable
* » iPython and programs must use a constructor to create a new SparkContext

• Use SparkContext to create RDDs

**Spark Essentials: Master**

The master parameter for a SparkContext determines which type and size of cluster to use.



**Resilient Distributed Datasets:**

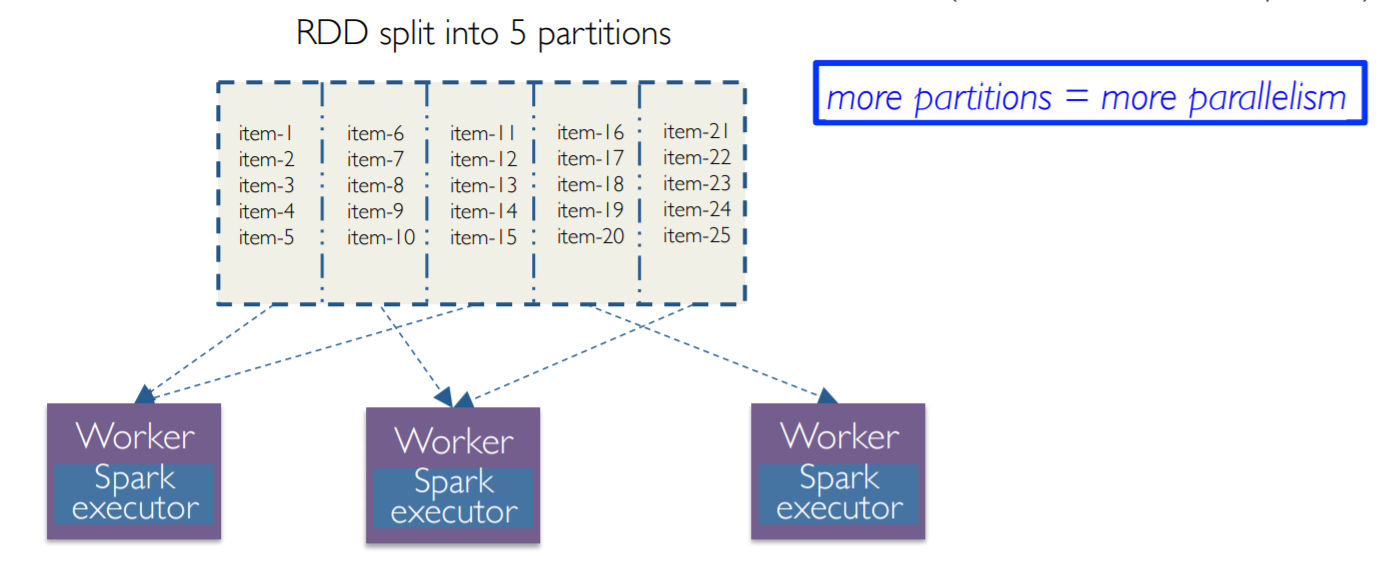
The primary abstraction in Spark

* » Immutable once constructed
* » Track lineage information to efficiently recompute lost data
* » Enable operations on collection of elements in parallel

You construct RDDs

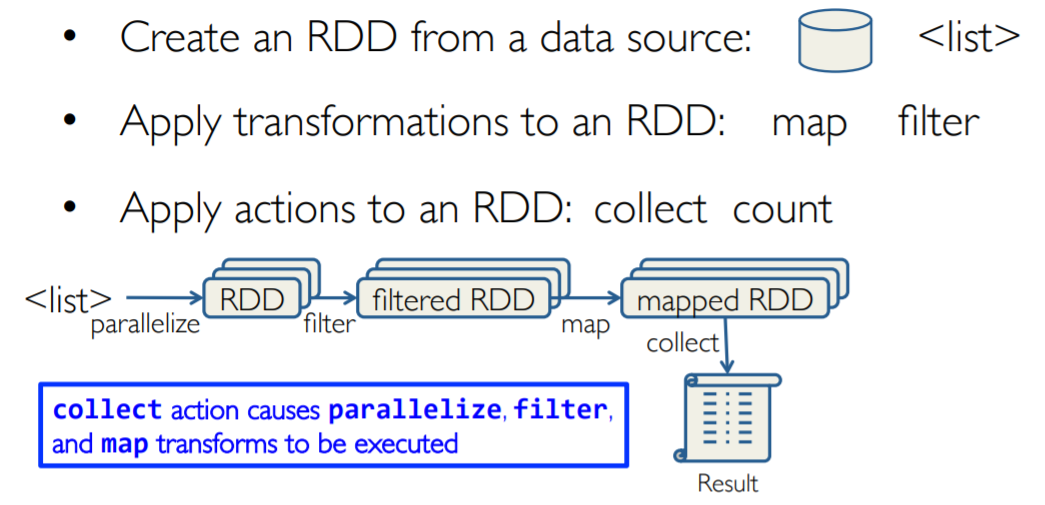
* » by parallelizing existing Python collections (lists)
* » by transforming an existing RDDs
* » from files in HDFS or any other storage system

Programmer specifies number of partitions for an RDD.



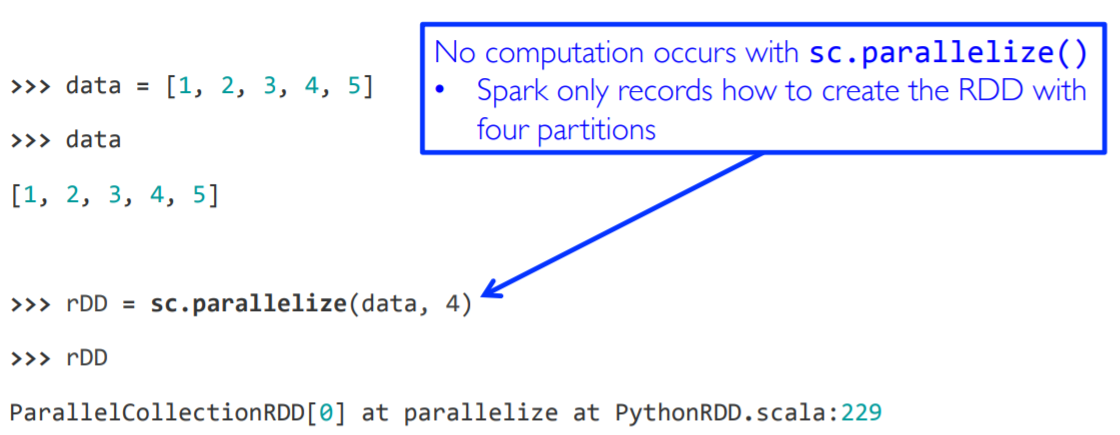
* Two types of operations: transformations and actions
* Transformations are lazy (not computed immediately)
* Transformed RDD is executed when action runs on it
* Persist (cache) RDDs in memory or disk

**Working with RDDs**

****

**Creating an RDD**

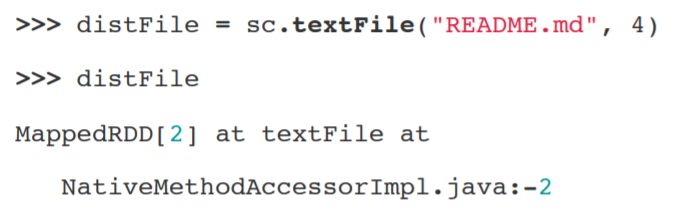
Create RDDs from Python collections (lists)

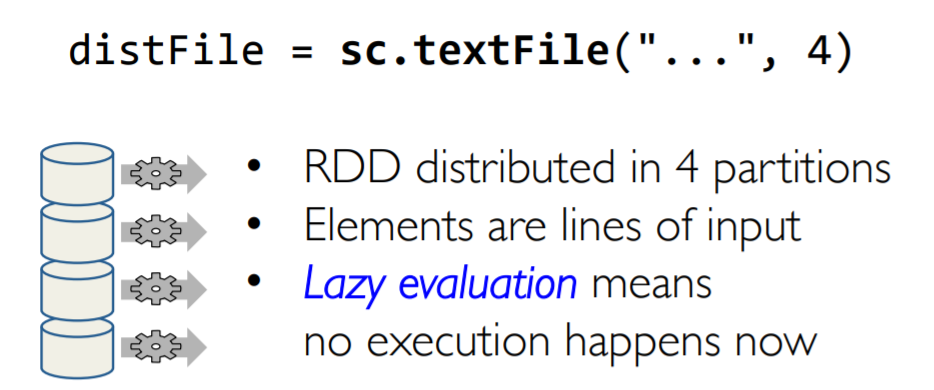


• From HDFS, text files, Hypertable, Amazon S3, Apache Hbase,

SequenceFiles, any other Hadoop InputFormat, and directory or

glob wildcard: /data/201404\*





##### **Content-based Filtering (CBF):**

The main idea behind CBF is to recommend items like the items previously liked by the user. For example, if the user has rated some items in the past, then these items are used for user-modeling where the user’s interests are quantified.

This can be achieved in two different ways:

* Predicting ratings using parametric models like regression or logistic regression for multiple ratings and binary ratings respectively based on the previous ratings.
* Similarity based techniques using distance measures to find similar items to the items liked by the user based on item features.

CB can be applied even when a strong user-base is not built, as it depends on the item’s meta data (features) therefore does not suffer from cold-start problem. However, this also makes it computationally intensive, as similarities between each user and all the items must be computed. Since the recommendations are based on the item similarity to the item that the user already knows about, it leaves no room for serendipity and causes over specialisation. CB also ignores popularity of an item and other user’s feedbacks.

**Collaborative Filtering**- This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like its content-based counterparts. This is the most sophisticated personalized recommendation that means it takes into account what user likes and not likes. The main example of this is Google Ads. Under the umbrella of Collaborative filtering, we have following kind of methods-

* **Memory Based -**It basically identifies the clusters of users in order to calculate theinteractions of one specific user to predict the interactions of other similar users. The second thought process will be identifying the items clusters rated by user A and predicting user’s interaction with item B. Memory based methods fail while dealing with large sparse matrices*.*
* **Model Based-**Methods basically revolves around ML and Data mining tools and techniques. The traditional Machine learning approach is used to train models and getting prediction out of it**.** One advantage of these methods is that they are able to recommend a larger number of items to a larger number of users, compared to other methods like memory-based. These methods work well with large sparse matrices as compared to the memory-based approach.

**Hybrid Systems -**Consolidated both types of information with the aim of avoiding problems that are generated when working with one kind of Recommender systems.

**SYSTEM DESIGN**

**DATASET**

These files contain 1,000,209 anonymous ratings of approximately 3,900 movies

made by 6,040 MovieLens users who joined MovieLens in 2000.

**RATINGS FILE DESCRIPTION**

All ratings are contained in the file "ratings.dat" and are in the

following format:

UserID :: MovieID :: Rating :: Timestamp

- UserIDs range between 1 and 6040

- MovieIDs range between 1 and 3952

- Ratings are made on a 5-star scale (whole-star ratings only)

- Timestamp is represented in seconds since the epoch as returned by time(2)

- Each user has at least 20 ratings

**USERS FILE DESCRIPTION**

User information is in the file "users.dat" and is in the following

format:

UserID :: Gender :: Age :: Occupation :: Zip-code

All demographic information is provided voluntarily by the users and is

not checked for accuracy. Only users who have provided some demographic

information are included in this data set.

- Gender is denoted by a "M" for male and "F" for female

- Age is chosen from the following ranges:

\* 1: "Under 18"

\* 18: "18-24"

\* 25: "25-34"

\* 35: "35-44"

\* 45: "45-49"

\* 50: "50-55"

\* 56: "56+"

- Occupation is chosen from the following choices:

\* 0: "other" or not specified

\* 1: "academic/educator"

\* 2: "artist"

\* 3: "clerical/admin"

\* 4: "college/grad student"

\* 5: "customer service"

\* 6: "doctor/health care"

\* 7: "executive/managerial"

\* 8: "farmer"

\* 9: "homemaker"

\* 10: "K-12 student"

\* 11: "lawyer"

\* 12: "programmer"

\* 13: "retired"

\* 14: "sales/marketing"

\* 15: "scientist"

\* 16: "self-employed"

\* 17: "technician/engineer"

\* 18: "tradesman/craftsman"

\* 19: "unemployed"

\* 20: "writer"

**MOVIES FILE DESCRIPTION**

Movie information is in the file "movies.dat" and is in the following

format:

MovieID::Title::Genres

- Titles are identical to titles provided by the IMDB (including

year of release)

- Genres are pipe-separated and are selected from the following genres:

\* Action

\* Adventure

\* Animation

\* Children's

\* Comedy

\* Crime

\* Documentary

\* Drama

\* Fantasy

\* Film-Noir

\* Horror

\* Musical

\* Mystery

\* Romance

\* Sci-Fi

\* Thriller

\* War

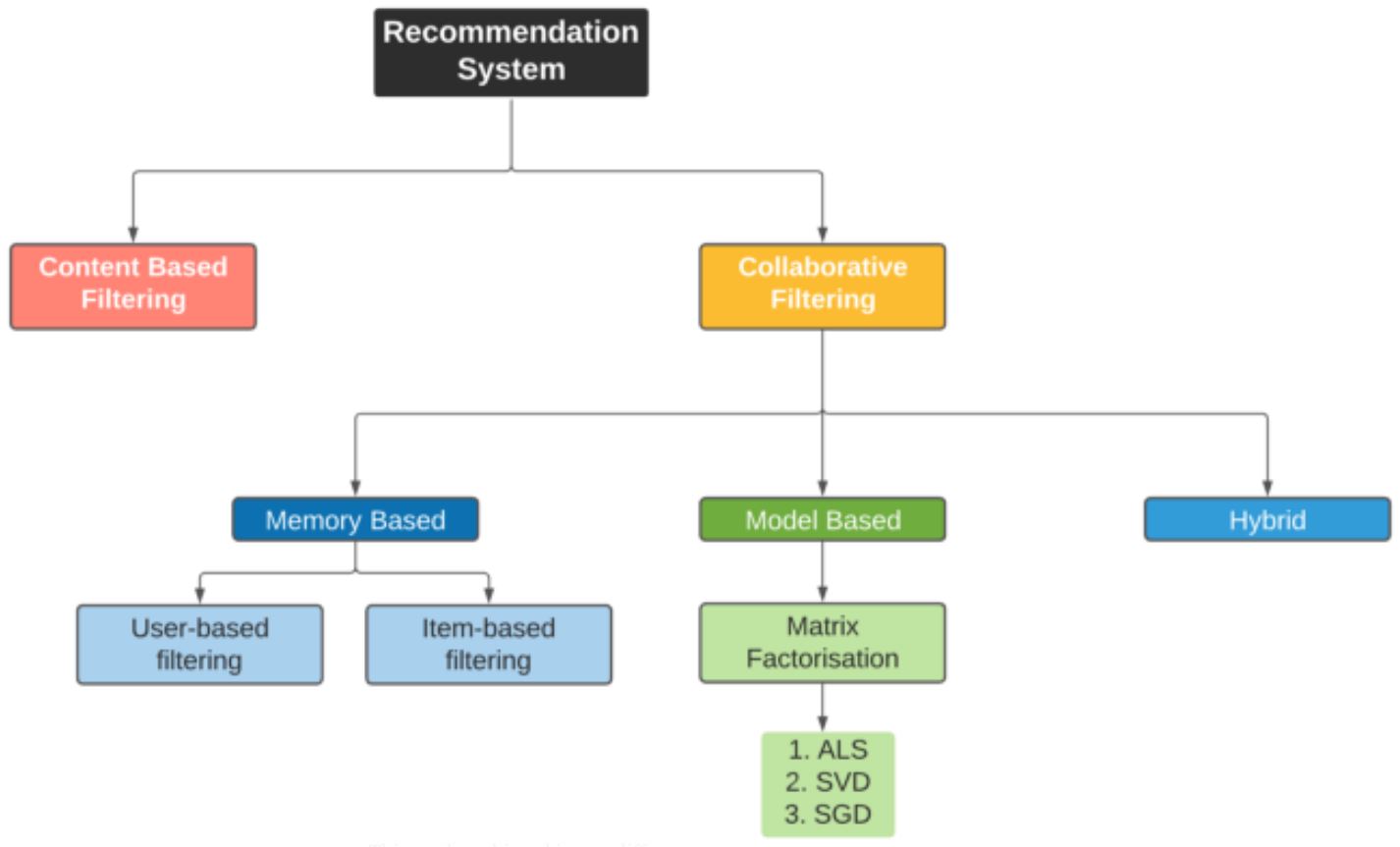
\* Western

- Some MovieIDs do not correspond to a movie due to accidental duplicate

entries and/or test entries

- Movies are mostly entered by hand, so errors and inconsistencies may exist

**FLOW CHART**

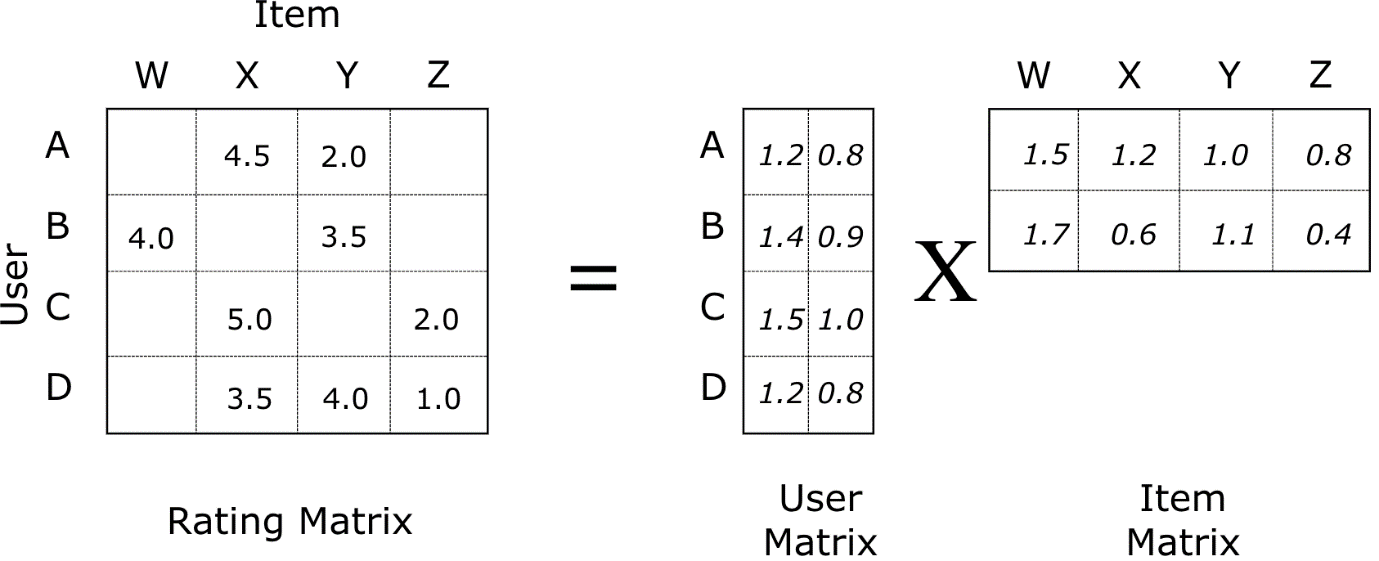


**1**

**IMPLEMENTATION**

# **Matrix Factorization**

In collaborative filtering, **matrix factorization** is the state-of-the-art solution for sparse data problem



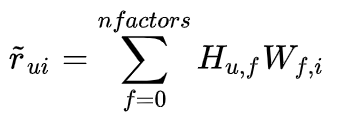
**Fig: Matrix Factorization of Movie Ratings Data**

What is matrix factorization? Matrix factorization is simply a family of mathematical operations for matrices in linear algebra. To be specific, a matrix factorization is a factorization of a matrix into a product of matrices. In the case of collaborative filtering, matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two **lower dimensionality rectangular matrices**. One matrix can be seen as the user matrix where rows represent users and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns represent items.

How does matrix factorization solve our problems?

1. Model learns to factorize rating matrix into user and movie representations, which allows model to predict better personalized movie ratings for users
2. With matrix factorization, less-known movies can have rich latent representations as much as popular movies have, which improves recommender’s ability to recommend less-known movies

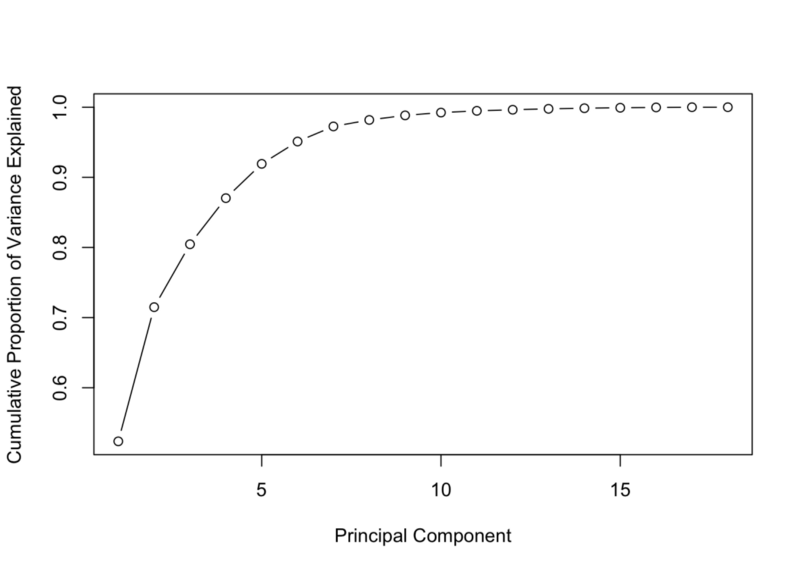
In the sparse user-item interaction matrix, the predicted rating user u will give item i is computed as:



where H is user matrix, W is item matrix

Rating of item i given by user u can be expressed as a dot product of the user latent vector and the item latent vector.

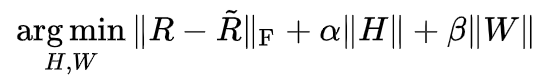
Notice in above formula, the number of **latent factors** can be tuned via cross-validation. **Latent factors** are the features in the lower dimension latent space projected from user-item interaction matrix. The idea behind matrix factorization is to use latent factors to represent user preferences or movie topics in a much lower dimension space. Matrix factorization is one of very effective **dimension reduction** techniques in machine learning.



**Fig: Variance Explained by Components In PCA**

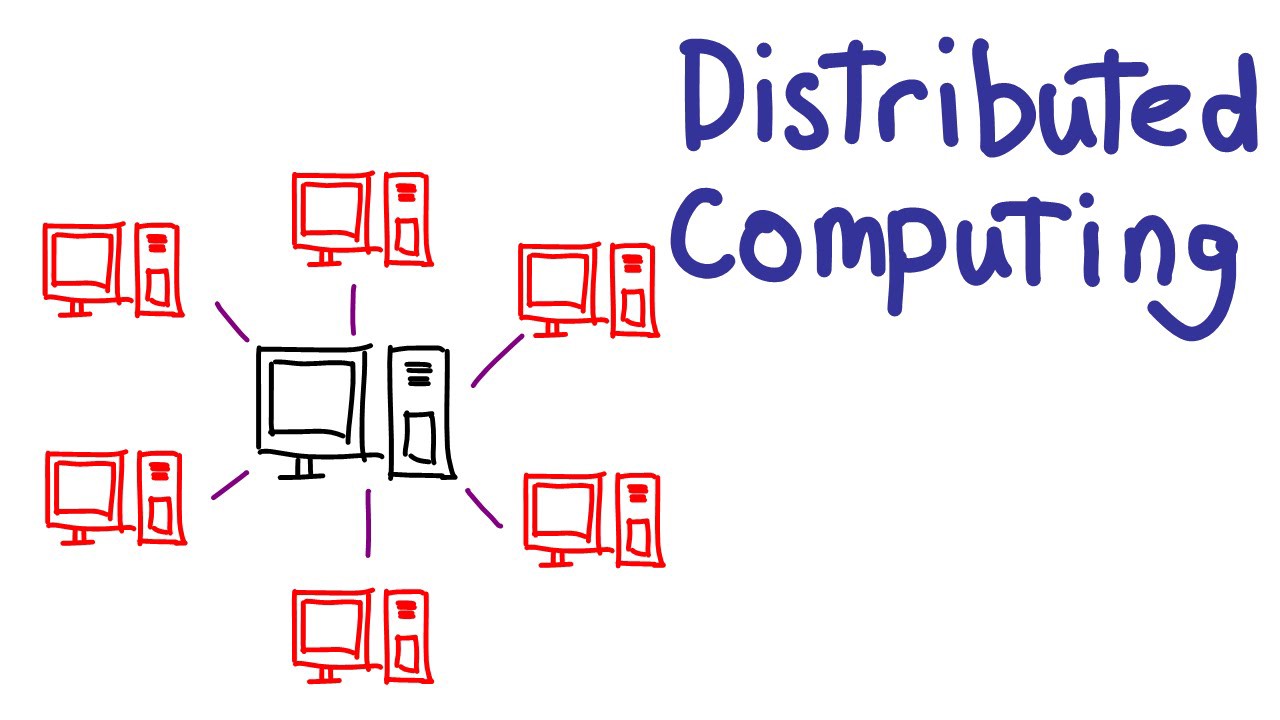
Very much like the concept of **components** in **PCA**, the number of latent factors determines the amount of abstract information that we want to store in a lower dimension space. A matrix factorization with one latent factor is equivalent to a most popularortop popular recommender (e.g. recommends the items with the most interactions without any personalization). Increasing the number of latent factors will improve personalization, until the number of factors becomes too high, at which point the model starts to overfit. A common strategy to avoid overfitting is to add **regularization terms** to the objective function.

The objective of matrix factorization is to minimize the error between true rating and predicted rating:



where H is user matrix, W is item matrix

Once we have an objective function, we just need a training routine (eg, gradient descent) to complete the implementation of a matrix factorization algorithm. This implementation is actually called **Funk SVD**. It is named after Simon Funk, who he shared his findings with the research community during Netflix prize challenge in 2006.



**Fig: Scaling Machine Learning Applications With Distributed Computing**

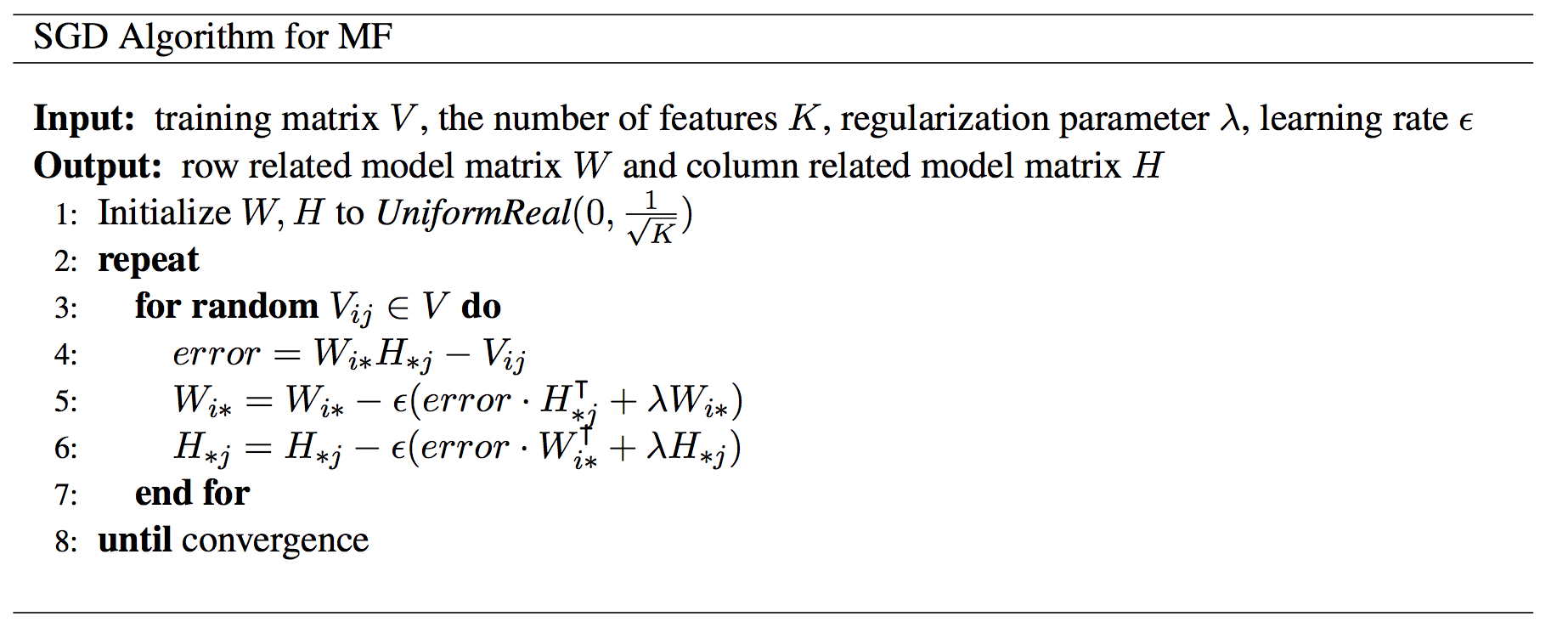
Although Funk SVD was very effective in matrix factorization with single machine during that time, it’s not **scalable** as the amount of data grows today. With terabytes or even petabytes of data, it’s impossible to load data with such size into a single machine. So we need a machine learning model (or framework) that can train on dataset spreading across from cluster of machines.

# **Alternating Least Square (ALS) with Spark ML**

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larger-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it’s simple and scales well to very large datasets.

Some high-level ideas behind ALS are:

* Its objective function is slightly different than Funk SVD: ALS uses **L2 regularization** while Funk uses **L1 regularization.**
* Its training routine is different: ALS minimizes **two loss functions alternatively**; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix.
* Its scalability: ALS runs its gradient descent in **parallel** across multiple partitions of the underlying training data from a cluster of machines



**Fig: Pseudocode for SGD In Matrix Factorization**

If you are interested in learning more about ALS, you can find more details in this paper: [Large-scale Parallel Collaborative Filtering for the Netflix Prize](https://endymecy.gitbooks.io/spark-ml-source-analysis/content/%E6%8E%A8%E8%8D%90/papers/Large-scale%20Parallel%20Collaborative%20Filtering%20the%20Netflix%20Prize.pdf)

Just like other machine learning algorithms, ALS has its own set of hyper-parameters. We probably want to tune its hyper-parameters via **hold-out validation** or **cross-validation**.

Most important hyper-params in Alternating Least Square (ALS):

* maxIter: the maximum number of iterations to run (defaults to 10)
* rank: the number of latent factors in the model (defaults to 10)
* regParam: the regularization parameter in ALS (defaults to 1.0)

Hyper-parameter tuning is a highly recurring task in many machine learning projects. We can code it up in a function to speed up the tuning iterations.

After tuning, we found the best choice of hyper-parameters: maxIter=10, regParam=0.05, rank=20

# **Implementing ALS Recommender System**

Now that we know we have a wonderful model for movie recommendation, the next question is: how do we take our wonderful model and productize it into a recommender system? Machine learning model productization is another big topic and I won’t get into details about it. In this post, I will show how to build a MVP (minimum viable product) version for ALS recommender.

To productize a model, we need to build a workflow around the model. Typical ML workflow roughly starts with data preparation via pre-defined set of ETL jobs, offline/online model training, then ingesting trained models to web services for production. In our case, we are going to build a very minimum version of movie recommender that just does the job. Our workflow is following:

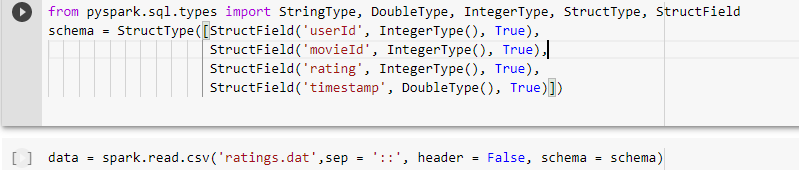
1. A new user inputs his/her favorite movies, then system create new user-movie interaction samples for the model.
2. System retrains ALS model on data with the new inputs.
3. System creates movie data for inference (in my case, I sample all movies from the data)
4. System makes rating predictions on all movies for that user
5. System outputs top N movie recommendations for that user based on the ranking of movie rating predictions.

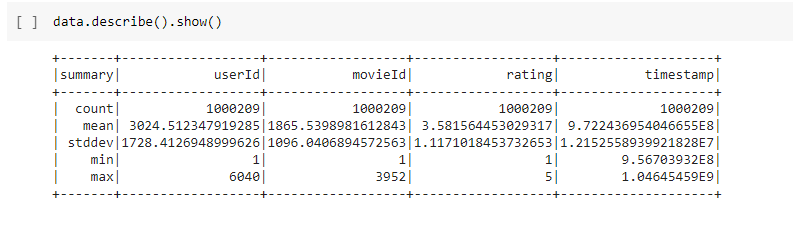
**CODE**

Importing required libraries:

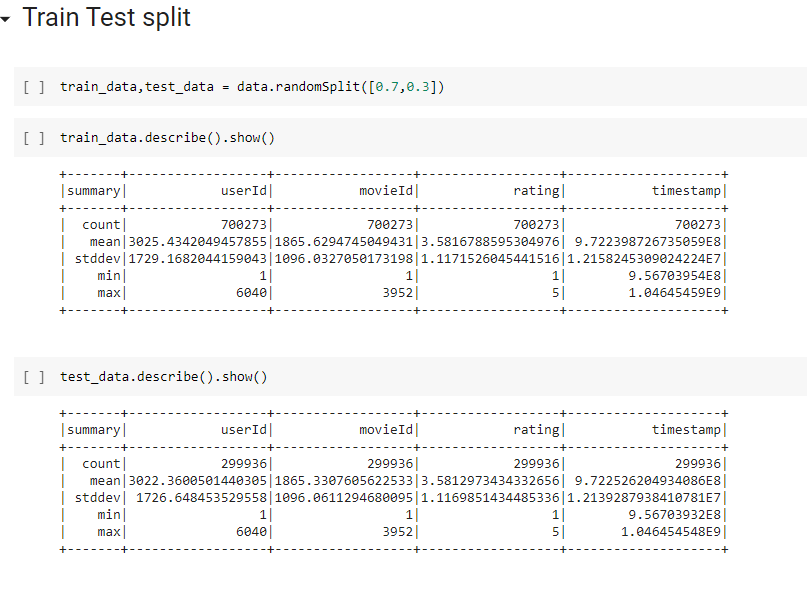
1. **SparkSession**: A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files. To create a SparkSession, use the following builder pattern.
2. **ALS:** ALS attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. X \* Yt = R. Typically these approximations are called 'factor' matrices.
3. **RegressionEvaluator:** RegressionEvaluator allows to measure the performance of ALS.
4. **f:** importing functions



****

****

Data is split into Training data and Test data in the ratio 7:3.

****

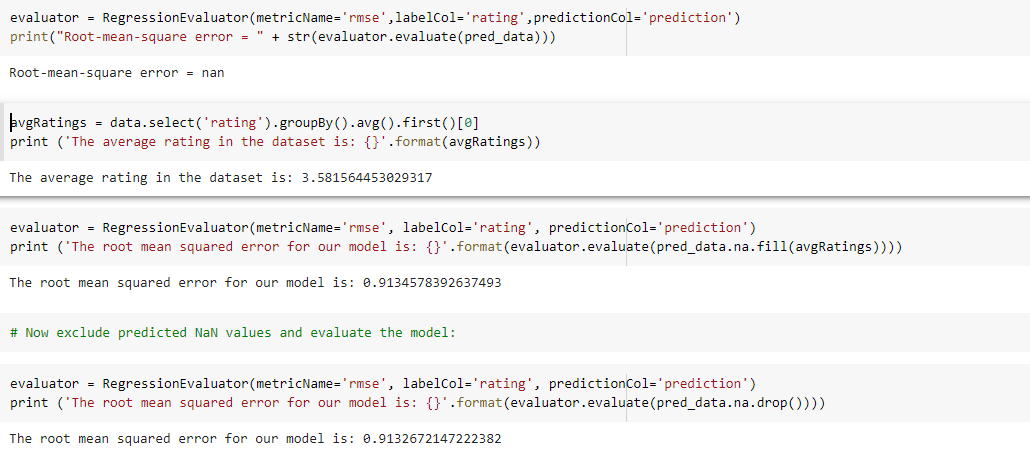
Alternating Least Squares(ALS) method is used to predict recommendations.

****

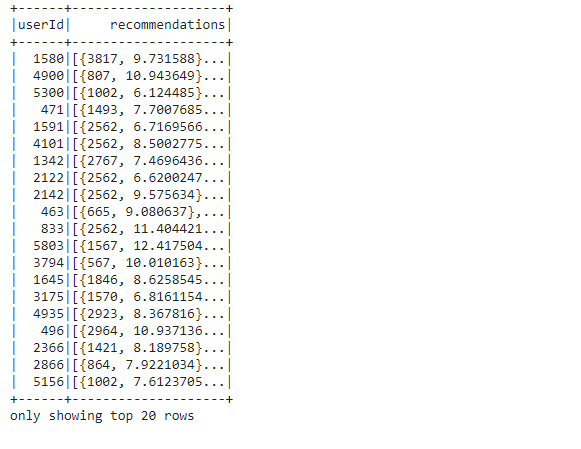
Evaluator for Regression, which expects input columns prediction, label and an optional weight column.

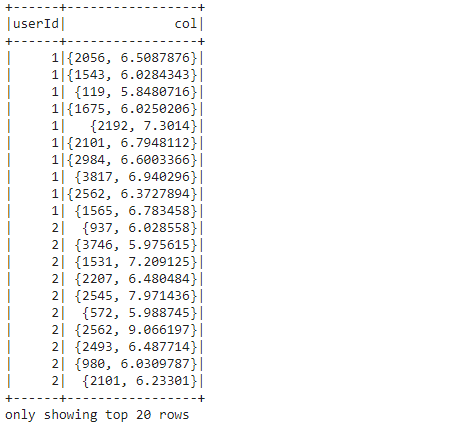
****

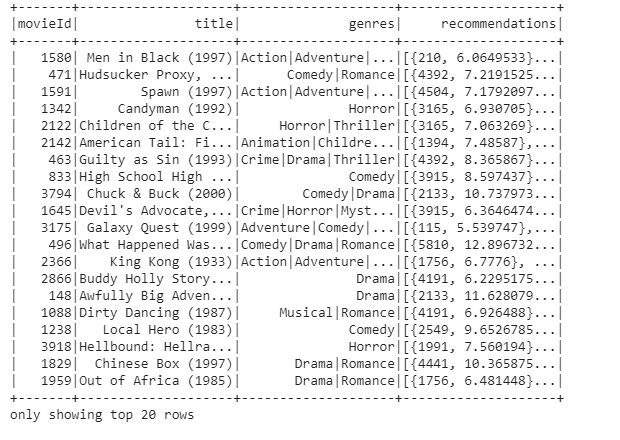
Calculating Root Mean Squared Error.

****

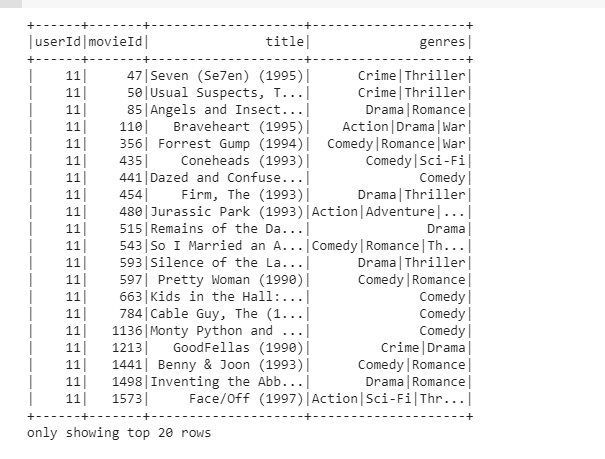
**RESULTS**

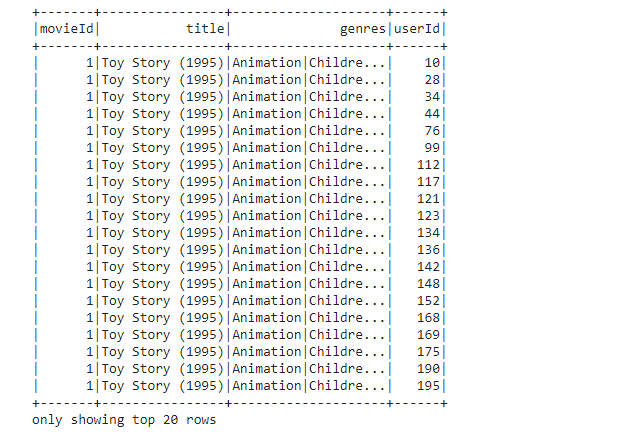
****

****

****Generate top 10 user recommendations for each movie

 Generate top 10 movie recommendations for a specified set of users

****

****Generate top 10 user recommendations for a specified set of movies

**CONCLUSION**

Movies Recommends system for its users to watch, based on their film preferences using collaborative filtering. Recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications.

**FUTURE ENHANCEMENTS**

In future we take the data of movies available in different websites and find out the movies available according to the OTT subscription they and we can recommend better movies by views count.

**REFERENCES**

1. [*https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679*](https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679)
2. [*https://medium.com/analytics-vidhya/crafting-recommendation-engine-in-pyspark-a7ca242ad40a*](https://medium.com/analytics-vidhya/crafting-recommendation-engine-in-pyspark-a7ca242ad40a)
3. [*https://medium.com/@patelneha1495/recommendation-system-in-python-using-als-algorithm-and-apache-spark-27aca08eaab3*](https://medium.com/@patelneha1495/recommendation-system-in-python-using-als-algorithm-and-apache-spark-27aca08eaab3)