**MovieLens Dataset Analytics with Apache Spark**

1. **Introduction:**

In this work, Apache Spark together with Python programming language is used to develop a console-based application that processes a large dataset from MovieLens. Python Spark (PySpark), Python driver has APIs to load, transform, evaluate and store large dataset in a fault-tolerant way. PySpark supports building data processing projects in Python and comes with MLlib, distributed machine learning framework for advanced applications as recommender engines.

1. **Datasets Structures:**

The small dataset was first used to run the application and eventually the large dataset. Each of the datasets has CSV files for movies, tags, ratings. The following describes the structure of the entities:

1. Movies: movieId (int)

title (string)

genres (string)

Instances:

1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy

29,"City of Lost Children, The (Cité des enfants perdus, La) (1995)",Adventure|Drama|Fantasy|Mystery|Sci-Fi

Titles that have “,” inline are enclosed in double quotes “ ”. The year of publication of movies is enclosed in brackets appended to the title. While the genres are delimited by vertical bars ( | ).

1. Ratings: userId (int)

movieId (int)

rating (float)

timestamp (long)

Instances:

537,1199,5.0,879502559

537,1200,2.0,879502608

The rating is between 0 and 5 in step of 0.5 with 5 as the highest rating.

1. Tags: userId (int)

movieId (int)

tag (string)

timestamp (long)

Instances:

68,8623,Steve Martin,1249808497

73,107999,action,1430799184

1. **System Components:**

The application is divided into three components: datasets loading, datasets analysis, and clustering and recommendation engine.

1. Datasets loading: The datasets are loaded from CSV files through a Spark CSV adapter module (<https://github.com/databricks/spark-csv>). This adapter is capable of reading a CSV file, parsing and automatically inferring the schema. The result is saved as a Spark Data Frame which can be cached in the memory for faster accessing. Data Frame (DF), a distributed collection of data grouped into named columns, provides an API that is faster and more robust than the earlier Resilient Distributed Dataset (RDD).
2. Datasets Analysis: Different queries can be made on the dataset to seek details about movies, users’ ratings and tagging. The datasets are manipulated and transformed in a number of ways: filtering, sorting, columns selection etc. Finally, the results of the actions are displayed on the console in a very friendly tabular form.
3. Clustering and Recommendation engine: Movie tastes of an individual user depend on a number of factors such as the type of movies (described by genres), year of release, reviews (tags and ratings). Therefore, recommending movies for a user, based on his/her tastes, can be achieved through: the traits of the user’s previously watched movies (item-based and less accurate) or the traits of the other users that share the same taste as the user. The latter is Collaborative Filtering and it’s deployed using the Alternating Least Squares (ALS) algorithm of the Spark ML library.
4. **System Features:**

The application implemented the following features summarized in the following table:

Table 1: List of features

|  |  |  |
| --- | --- | --- |
| Section | S/N | Features |
| Part 1 | 1 | Read dataset using Spark CSV adapter |
| 2 | Storing dataset in Data Frame format |
| 3 | Searching user by ID or a list of users |
| 4 | Searching movie by ID/title |
| 5 | Searching genre or list of genres |
| 6 | Searching movie by year |
| 7 | Listing the top rated movies |
| 8 | Listing the top watched movies |
| Part 2 | 9 | Searching genre favorite of user(s) |
| 10 | Comparing the movie tastes of two users |
| Part 3 | 11 | Clustering users by movie taste |
| 12 | Visualization and interaction of dataset |
| 13 | Recommending movies for a user |

1. Read dataset using Spark CSV adapter - Data in CSV file format are read and parsed into distributed data collection with the schema inferred. The small datasets from MovieLens are saved in the user directory with path (~/movie-rec-data/ OR user-computer-path/movie-rec-data/).
2. Storing dataset - Each of the entities (tags, ratings and movies) classes’ constructors handles the dataset reading into individual dataframes. Timestamp is dropped in both the ratings and tags dataframes. Dataframe API is one of the two latest APIs from Apache Spark and it is preferable considering its speed and clean structures. The datasets were initialized stored as RDD and the performance evaluations recorded longer latency.
3. Searching User – a user or a list of users can be searched by ID or a list of IDs. This application returns the total number of movies watched and the total number of genre by the user or each of the list has watched. First, the user’s activities in rating and/or tagging the movies are filtered. This is achieved by combining (union) all the movies IDs he/she rated and those that were tagged, then finding the distinct to avoid repetition of movies. All the unique genres from the lists of movies genre are obtained by joining movies IDs with the movies Dataframe.
4. Searching movie – If the movie title is used in querying, the application firstly filters the movies Dataframe with the movie title to get the ID. Then the ratings Dataframe is filtered with the ID to collect all the ratings by different watchers. The number of users that have watched the movie is recorded, and then the average rating is calculated.
5. Searching genre – All movies in the genre or list of genres are obtained by firstly, filtering movies Dataframe for the ones that have the genre(s). The details of the movies are returned and displayed in a table.
6. Searching movie by year – The year of release of each movie is enclosed in a bracket within the movie title. Using Regular Expression to draw a pattern and matched with the pattern extraction of the Spark API. All movies that are produced in the year are returned and presented in a tabular form.
7. Listing the top rated movies – Top movies are collected according to the highest average rating. The average ratings of all the movies are computed, sorted in descending order, and the first number of movies required are selected.
8. Listing the top watched movies- This is according to the number of users who have watched the movies. This is accomplished by aggregating all the tags and ratings, selecting the distinct watches and sorting in descending order, then the top number of movies with the highest number of watches are selected.
9. Searching user(s)’ favorite – This is obtained according to the number of movies watched by the user(s). All the movies are filtered and the genres present are counted. The genre with the highest number of counts is considered the user’s favorite. This is reported as a percentage of the total genres he watches to normalize for comparison with other users.
10. Comparing the movie tastes of two users – To compare the movie tastes of two or more users, the percentages of the genres (which describe the types of movies) of each of the user are calculated. By using percentage, it standardizes the metrics, which are reported with composite bar charts providing an intuitive visual comparison.
11. Clustering users by movie taste – Apache Spark provides Machine Learning library for advanced operations other than manipulation of large data. With these tools, deeper insights can be obtained from big data. Clustering is one of the techniques of unsupervised machine learning. In this application, the K-Means library is deployed with a value (k=4) is considered after comparing different number of clusters and the corresponding Within-Square-Sum Error (WSSE). The model fits the dataset to obtain the cluster centers which group the datapoints.

First, each user’s movie tastes are represented the features as the fraction of each of the movie genres. This normalized metrics are then converted into vector to be fed as the training set. After ten arbitrarily iterations, the model is evaluated with the an acceptable WWSE error.

1. Visualization and interaction of dataset – Principal Component Analysis reduction technique is employed in converting the multi-dimensional features (20 different genres) into two components which are then plotted in a 2-D figure. While clustering has been done, the representation of each of the clusters are exposed with related datapoints around the clusters centers. MatPlot Lib is used in the plotting.
2. Recommending movies for a user – Using Collaborative Filtering Technique, movies can be recommended for a user based on the past ratings of the users that share similar traits as the user. This technique takes in train datapoints with the user’s past rated items included and then prediction is made to recommend the top rated movies that the user will like to watch.

User’s preferences for a new item is predicted by a machine learning method that first train on available data to learn to compute the user’s current preference. It is assumed that the properties of the movies are represented by a set of features (genres). User preference for a movie can be expressed in terms of the features.

1. **Problems:**

Some challenges were encountered during the development and evaluation of this application.

1. Setting up Apache Spark – After installing all the dependencies (Java Runtime, Scala, Hadoop etc.), to get the Spark running in the console or through Jupyter Notebook was a bit easier as the Spark Context is always initiated. However, running Spark in an IDE such as Spyder was a bit tricky, the Spark doesn’t automatically load. Therefore, a walk-around was to run some scripts saved in startup profiles of Interactive Python (IPython) console using a helper module (findspark) that loads the Spark on to the environment.
2. Computer resources – due to low computing resources, the distributed power of the Apache Spark could not be fully explored. After a lot of caching, it still always took a considerable amount of time to compute some complex queries. The training time is especially longer in the machine learning parts. Generally strategic caching to and especially saving the trained models on the memory helped.
3. Data cleaning – the first observable encounter was having a comma-separated data having commas in between the columns. This was taken care by the CSV Spark helper module (<https://github.com/databricks/spark-csv>)
4. Data Storage – This application was first built using the RDD API as most of the materials, documentations and tutorials that are readily available used this format. However, low performance was evaluated due to the long time and costly computations. The application was redeveloped using Dataframe of the latest APIs (the second is Dataset). Although, the materials and tutorials are available but not sufficient, therefore, deeper searches into the Spark latest documentations alongside questions on StackOverFlow and DataBrick helped.
5. Data normalization – Because all users do not watch the same number of movies and genres, it was therefore necessary to normalize the number of movies by finding the percentage of each of the genres counts. Also, some genres that are not preferred by some users yielded null upon joining the dataframes of user Ids and genres, and thus must be defaulted to zero.
6. **Summary:**

Working with Apache Spark and Python provides great insights into large data handling in a fault-tolerant (resilient) distributed methods. It exposes me to the unlimited possibilities that can be achieved through big data analytics and the machine learning techniques. Overall, I am really acquitted with the Apache Spark driver for Python (PySpark) and substantial parts of its components. Also, I used external libraries as MatPlot and numpy and as well create user defined functions (UDF) in Spark.

1. **Figures:**

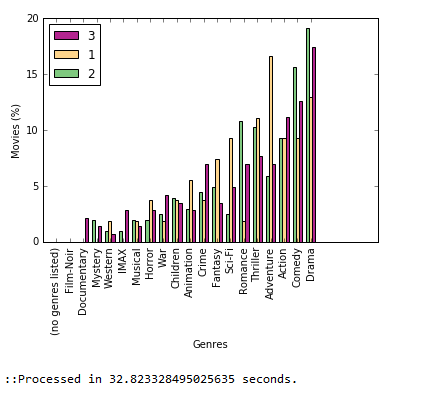


Fig. 1 Users Movie Tastes Comparison

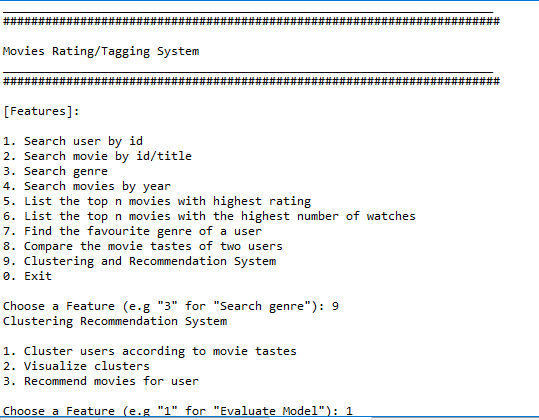


Fig. 2 Application Console Menu