MovieLens 25M dataset: Unveiling Cinematic Insights with Spark Data Analysis and Machine Learning

Part I Preparation

1. Overview

Our primary objective is to conduct a comparative analysis of the performance in big data analysis and machine learning tasks between a Hadoop cluster and a local environment. In pursuit, we employ an open-sourced MovieLens 25M dataset, sourced from the Grouplens, which offers an extensive and rich collection of movie ratings from 162541 users between January 9 1995 and November 21 2019.

The dataset is structured into 5 key files. This includes ratings.csv, a central repository with 25 million ratings detailing user and movie ID and timestamp. The movies.csv and links.csv provides foundational movie information. The tags.csv, genome-scores.csv and genome-tags.csv, offering tag relevance scores and movie descriptions.

	Feature	Description of the feature
1	userID	Unique identifier for users
2	title	Title of the movie
3	genres	Genres associated with the movie
4	rating	Rating of range 0.5-5 with 0.5 interval
5	timestamp	Time when the rating was given
6	imdbId	IMDb identifier
7	tmdbId	TMDb identifier
8	relevance	Relevance score for

Table 1. All features in MovieLens 25M dataset.

2. Workflow

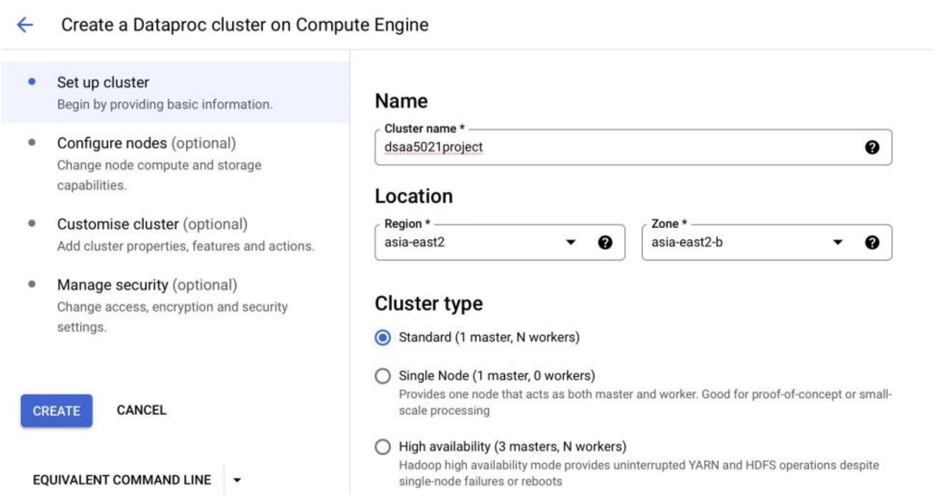
This section details the environmental setup and framework for dataset analysis and ML construct for rating predictions. We install Hadoop, Spark and PySpark in local driver.

Stage 1. Data analysis and movie recommendation using Spark	Stage 2. Rating prediction using ML
1) Create RDD	1) Train-test data split
2) Import, read movie and rating data	2) Predict using ALS
3) Read features and labels	3) Feature engineering
4) Count average ratings	4) Predict using regression model
5) Join database using ‘movieID’	5) Adjusting the hyperparameters
6) Data selection for recommendations	6) Compare two ways

Table 2. Framework for MovieLens 25M dataset.

3. Setup enivornment

We will use four different experimental environments in our future tasks. We built a local environment by using Winutils to enable Hadoop and Spark to run on PCs. In addition, we also created a Spark local environment under the MacOs system, a Yarn environment formed by building a 4-node cluster on Google Cloud's Dataproc, and a local environment by cutting off the communication between nodes.



Create a Dataproc cluster on Compute Engine

- Set up cluster
  Begin by providing basic information.
- Configure nodes (optional)
  Change node compute and storage capabilities.
- Customise cluster (optional)
  Add cluster properties, features and actions.
- Manage security (optional)
  Change access, encryption and security settings.

Name

Cluster name *: dsaa5021project

Location

Region *: asia-east2

Zone *: asia-east2-b

Cluster type

● Standard (1 master, N workers)

○ Single Node (1 master, 0 workers)
Provides one node that acts as both master and worker. Good for proof-of-concept or small-scale processing

○ High availability (3 masters, N workers)
Hadoop high availability mode provides uninterrupted YARN and HDFS operations despite single-node failures or reboots

Part II Basic Data Manipulation & Simple Recommendation

1. Count the number of ratings

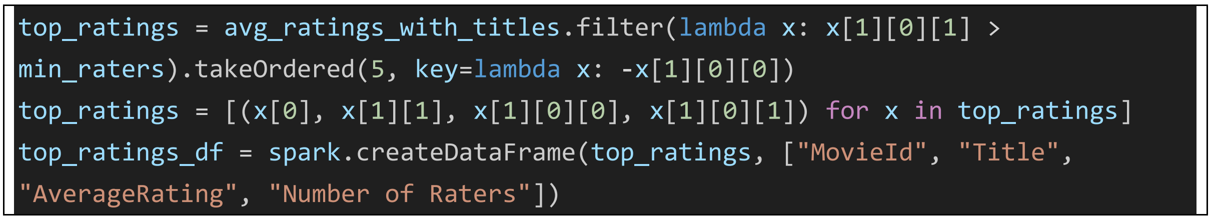
In this section we just used the Ratings.csv and read in into lines by SparkContext. The key is remove the header line or the result would be wrong. After former two steps we parsed the lines and created an RDD, and then easily count the number. Finally we got the result like this:

Number of ratings: 25000095

2. Recommend 5 highest rating movies

Read the file in the same way as first section, and we will see the movieId in column 1 while the rating in column 2. Map the movieId and rating as key-value pairs and thus we can use ***groupByKey()*** to calculate the average ratings. Now we get the rating for each MovieId without title, so the next step is to introduce another file which includes the movie title and create its RDD. Before proceeding to the next step, we first filter out the rows with too few ratings, and here we set the threshold to 3000. Then we join these two RDDs and order by average rating. The last thing is taking top 5 rows to create a DataFrame as the ouput.

```
avg_ratings_with_titles = movie_ratings_count.join(movie_titles)
min_raters = 3000  # Set the minimum number of raters
```



```
top_ratings = avg_ratings_with_titles.filter(lambda x: x[1][0][1] >
min_raters).takeOrdered(5, key=lambda x: -x[1][0][0])
top_ratings = [(x[0], x[1][1], x[1][0][0], x[1][0][1]) for x in top_ratings]
top_ratings_df = spark.createDataFrame(top_ratings, ["MovieId", "Title",
"AverageRating", "Number of Raters"])
```

Figure 1. Code snippet of performing a join operation between RDDs containing average ratings and movie titles, followed by extracting top 5 movies using thresholds. The selected data is mapped and converted into a DataFrame.

Result is as follow:

MovieId	Title	Average Rating	Number of Raters
318	Shawshank Redemption	4.41	81482
858	Godfather	4.32	52498
50	Usual Suspects	4.28	55366
1221	Godfather: Part II	4.26	34188
2019	Seven Samurai	4.25	13367

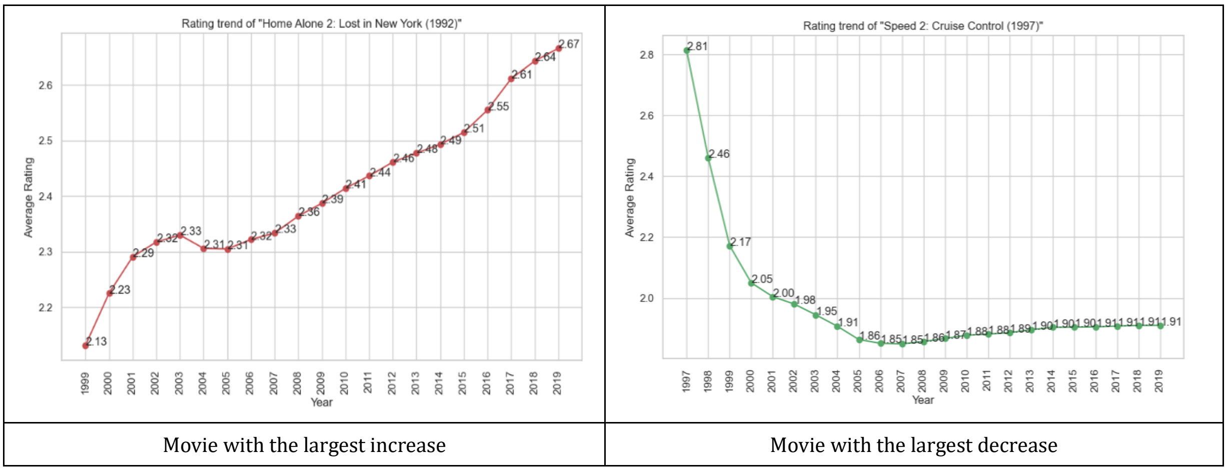
3. Data analysis & visualization

3.1. Movie Rating analysis

3.1.1 Movies with the biggest changes in rating trends

The total ratings and number of ratings per year for each film were calculated, and based on this data the cumulative average ratings for each film from the beginning to each year were calculated.

Next, the two films with the most significant increases and decreases in ratings over time were identified, and the trends in ratings over time for these two films were plotted using matplotlib. Finally, the names of these two films were obtained from another dataset and displayed (Figure 2).



Movie with the largest increase

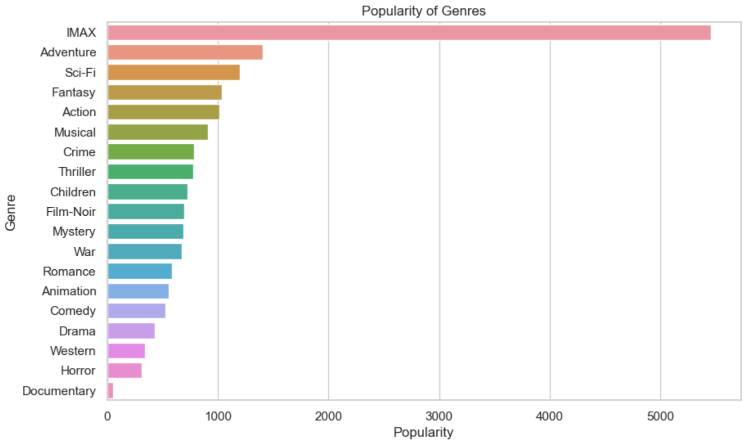
Movie with the largest decrease

Figure 2. Movies with the most drastic rating change - ‘Home Alone 2 :Lost in New York’ with an rating increase 2.13 to 22.67 (left),

and ‘Speed 2: Cruise Control’ with a rating decrease 2.1 to 1.91 (right).

3.1.2 Movies with the most popular genres

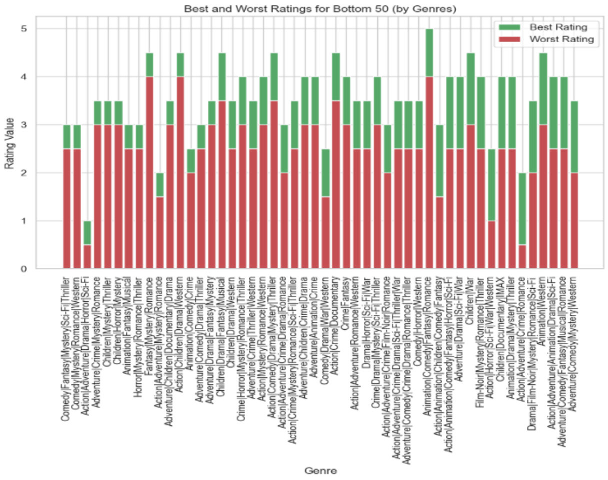
We counted the number of ratings for each film and split and counted the genres. The total number of ratings for each genre was then calculated and combined with the number of genres to calculate the popularity of each genre. Finally, this data was plotted as a horizontal bar graph.

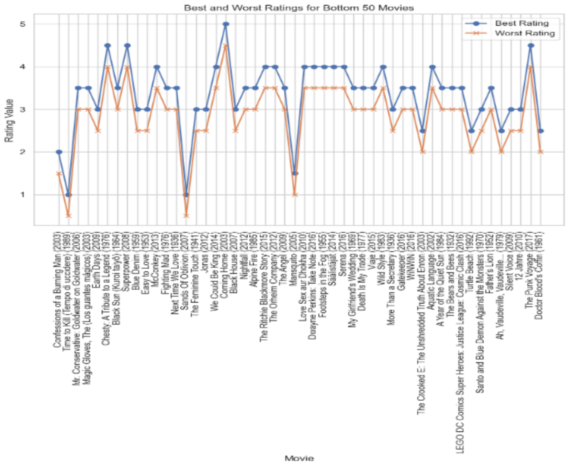


3.1.3 Films with the smallest rating gap (by subject)

We merged the ratings data with the film data (including genre) and then calculated the highest and lowest ratings for each film genre. Next, the rating variance for each genre was calculated and the genres with rating variances greater than 0 were filtered out.

Finally, the code identified the 50 genres with the smallest rating variance and plotted a bar chart to visualize the highest and lowest ratings for these genres, as well as the individual genres with the smallest rating variance.

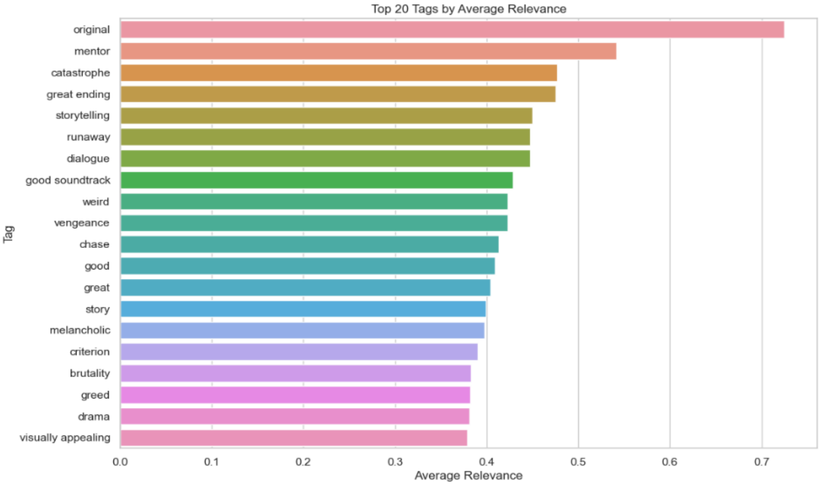




3.2. Genre and Tag analysis

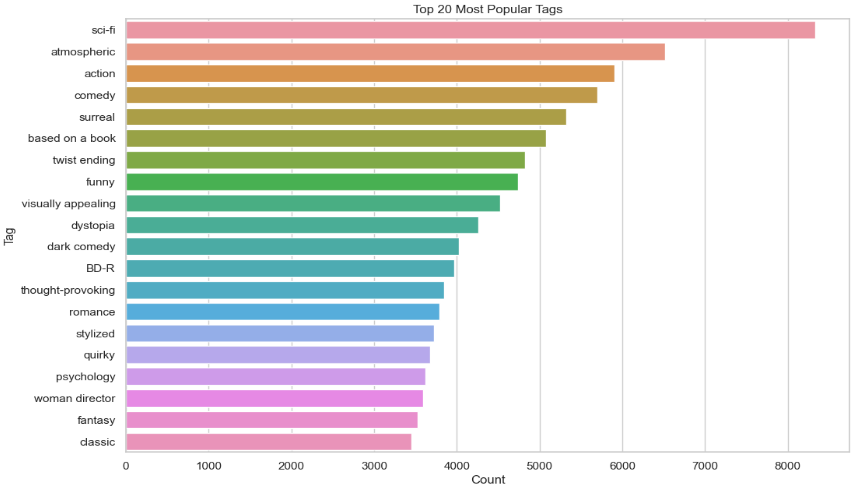
3.2.4. Top 20 Tags by Average Relevance

We use the Movie Tags and Tags Relevance Scores dataset. The average relevance score for each tag was first calculated, and then these scores were merged with the tag name dataset to obtain the name of each tag. Subsequently, the results were converted to a DataFrame and a bar chart was used to show the top 20 tags with the highest average relevance scores. This visualization shows the average importance or relevance of different tags in the film dataset.



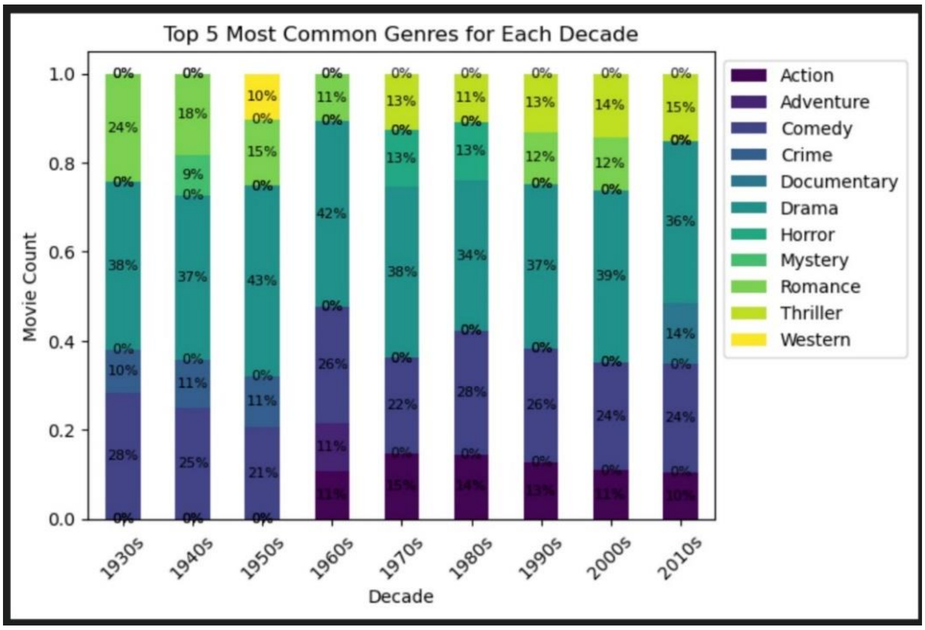
3.2.5. Top 20 Most Popular Tags

We grouped and counted each tag and calculated the tag with the highest number of occurrences. The resulting data was then converted into a DataFrame.Finally, a bar chart was drawn to show the top 20 most frequent tags, presenting the popularity of these tags in the dataset. The focus was on understanding and visualising the frequency of use of film tags.



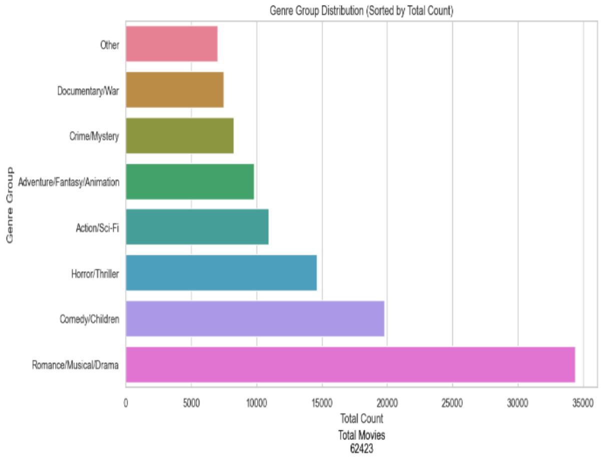
3.2.6. Top 5 Most Popular Film Titles of Every Era

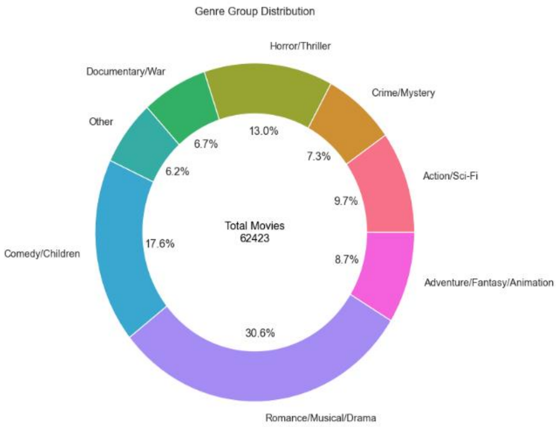
We extracted the years from the film titles and limited the data to between 1930 and 2022, then sorted the years into the corresponding decades (e.g. 1930s, 1940s) and split the film genres. The code then counted the top five film genres with the most occurrences in each decade and used Spark's window function to calculate the rankings for each decade. Using stacked bar charts, we show the five most common film genres and their proportions in each decade. A good indication of the evolutionary trend of popular film genres in different decades.



3.2.7. Genre Group Distribution (Sorted by Total Count)

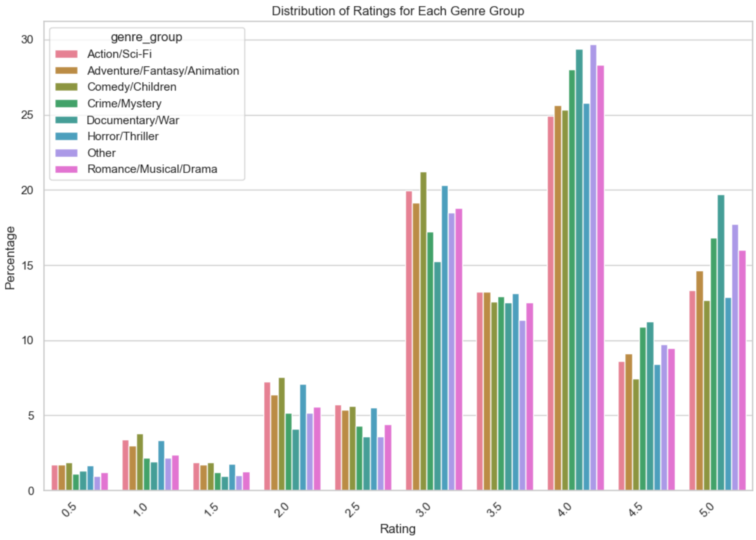
We first counted the number of unique film genres and counted the frequency of occurrence of each genre. Genres were then further analysed by grouping some genres into broader categories such as 'Crime/Mystery' or 'Romance/Musical/Drama'. We also filtered out film genres other than 'IMAX' and '(no genres listed)'. It then converted the resulting data into a Pandas DataFrame, and the graph also included text annotations for the total number of films. We used bar charts and pie charts to show the distribution of the different genre groups, with the total number of films labelled to provide a comparative perspective. This analysis reveals how representative the different film genre groups are in the dataset.





3.2.8. Distribution of Ratings for Each Genre Group

We combined film rating data and film information to filter out valid ratings, then sorted them by film genre and grouped certain niche genres into broader genre groups. The ratings were then binned using the Bucketizer and the frequency of occurrence of different rating intervals within each genre group was calculated. Bar charts were used to show the percentage distribution of different rating intervals within each genre group to visualise the differences in ratings across genre groups. The distributional characteristics of the ratings for different film genre groups were revealed.



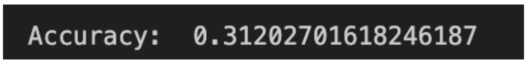
Part III Rating Prediction

1. Prediction by ALS

First we choose one matrix factorization algorithm to predict the rating score based on the rating data file only. the *ALS (Alternating Least Squares)* model is used for recommendation systems. It is a type of collaborative filtering algorithm that predicts user ratings for items based on historical data. So it is very suitable for our task.

The ALS model in PySpark is easily implemented using MLlib, which solves for the ratings matrix using a least squares approach. The algorithm alternates between updating the user and item factors until convergence. The resulting factors are then used to predict ratings for new user-item pairs.

All the things we need to do is loading the data(only movieId, userId and rating), splitting them and using them to train the model. We adjusting the hyperparameters using grid search and here's the result:

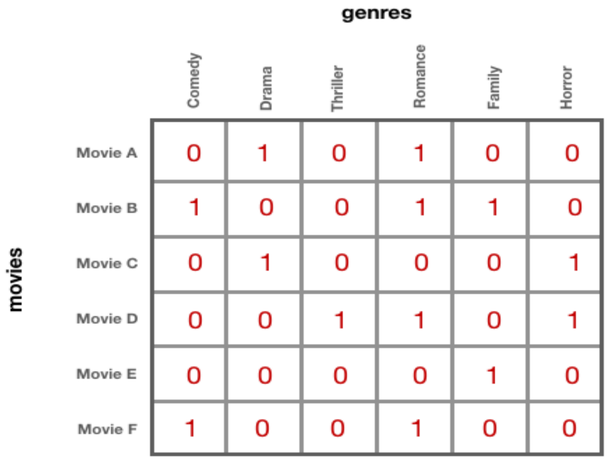


```
Accuracy:  0.31202701618246187
```

2. Prediction by Linear Regression and Random Forest Regression

2.1 Feature engineering

Since there are few valuable data types in the dataset, we need to do feature engineering. After considering the characteristics of the data and the model we want to use, we decided to create or convert these types of data: genres of each movie, average rating of each movie and each user, mode rating of each movie and each user.



genres

movies

	Comedy	Drama	Thriller	Romance	Family	Horror
Movie A	0	1	0	1	0	0
Movie B	1	0	0	1	1	0
Movie C	0	1	0	0	0	1
Movie D	0	0	1	1	0	1
Movie E	0	0	0	0	1	0
Movie F	1	0	0	1	0	0

In order to process numerical data rather than the film categorical data, we converted the film types to a single hot coded form and merged it with the ratings dataset. Next, we used "VectorAssembler" to create feature vectors to prepare the data for the machine learning model. In this way, we will get training vectors with 23 dimensional.

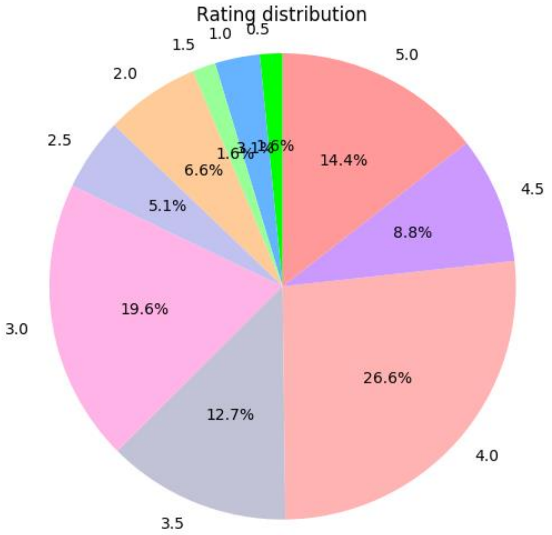
2.2 Choose model

In addition to ALS, we also need to find a suitable model for rating prediction. Due to the simplicity of the variables, we may assume that there is a linear relationship between the score and the variable we define. Therefore, a regression model is a good choice. Here, we use two regression models, and after comparing their performance, we will make a trade-off.

Based on this, two models were constructed: a random forest regression model and a linear regression model, but only the random forest model was trained and applied to the test data.

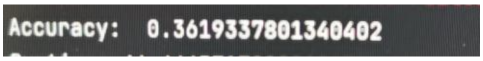
2.3 Adjusting weights

If only processed data is used to train the selected model, the effect is even worse than using ALS. But it is also difficult to further improve accuracy by adjusting hyperparameters or cleaning data when there are no flaws with the data and model.



So we use some statistical methods to adjust the weight of each predicted rating to approximate the final result rating according to the rating distribution of the training set. For example, in above figure we can see that the count of 5.0 and 4.0 are much higher than 4.5, we will approximate the predicted scores that would have been approximately 4.5 to 5.0 or 4.0.

By Adjusting weights, the accuracy has been improved by more than 5%



```
Accuracy:  0.3619337801340402
```

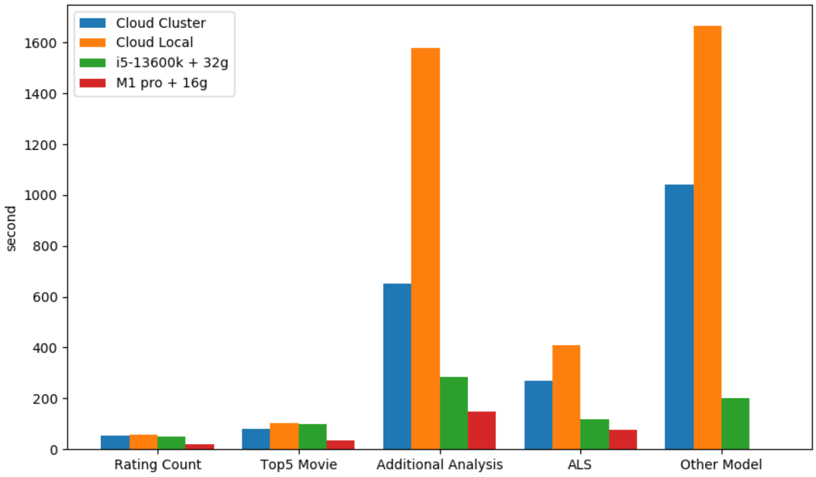
3. Advantages and disadvantages analysis

Alternating Least Squares (ALS):	**Linear regression:**	**Random Forest:**
Pros: ALS is particularly suitable for collaborative filtering scenarios and can effectively handle large sparse datasets. It learns latent feature representations by minimizing the reconstruction error of user and item interactions, which is very effective for capturing user preferences and item characteristics. Cons: ALS is not effective enough for the cold-start problem (recommendation of new users or new items). In addition, ALS models may focus too much on existing interactions and ignore unobserved latent information. Accuracy: ALS tends to perform well on large-scale sparse datasets with high accuracy. However, the performance may be degraded when the data size is small or the data is too dense.	Pros: linear regression models are simple, easy to implement, and computationally efficient. For datasets where the relationship is linearly evident, the model can provide good predictions. Cons: Linear regression assumes a linear relationship between the features and the target variable, which may not hold in many complex real-world data. Also, it is sensitive to outliers. Accuracy: Linear regression can provide high accuracy in scenarios where the features do have a linear relationship with the target variable. However, accuracy may drop in scenarios where the relationship is more complex.	Pros: random forests can automatically handle complex interactions between features and are resistant to outliers and overfitting. Cons: Compared to linear models, random forests are more computationally expensive to train and predict, and the models are less interpretable. Accuracy: Random Forests typically provide high accuracy when working with complex datasets such as the film ratings for this task versus categorical data (data with multiple features or complex relationships between features).

Part IV Performance comparison

1. Table and figure

	Cloud Cluster	Cloud Local	PC	Macbook
Rating Count	53s	56s	48s	17.7s
Top5 Movie	78s	102s	100s	33.9s
Additional Analysis	652s	1579s	283s	148.3s
ALS	270s	408s	117s	77.1s
Other Model	1042s	1666s	202s	NA



2. Analysis

Why servers running slower than personal computers?

Insufficient server configuration: The configuration of the three servers may not be as good as that of personal computers, such as processors, memory, storage, and other aspects, which may have lower performance. Personal computers typically focus more on single machine performance, while servers may prioritize stability and reliability.

Network latency: When using multiple servers, data transmission and communication need to be carried out through the network, while personal computers usually run alone without network latency issues. If the network latency between servers is high, it can lead to an increase in task execution time.

Data distribution and scheduling problems: When using multiple servers, data needs to be transmitted between servers, and tasks need to be scheduled and coordinated. If the data distribution is uneven, resulting in heavy load on certain servers or unreasonable scheduling algorithms, it may lead to a decrease in task execution efficiency.

Insufficient parallelization level: If the task cannot be effectively parallelized, or if the parallelization level is low, even if multiple servers are used, their computing power cannot be fully utilized.

Why cluster may not be much faster than single server?

Small data volume: If the task has a small amount of data and can be easily processed on one server, using multiple servers may not bring significant performance improvements.

Uneven data distribution: If the data distribution is uneven, that is, some data is concentrated on one server while there is less data on other servers, the task may be affected by data skewing, resulting in low utilization of multiple servers.

Network bandwidth limitation: When using multiple servers, data needs to be transmitted between servers. If the network bandwidth is limited, it may lead to bottlenecks in data transmission, making it difficult to fully utilize the computing power of multiple servers.

Tasks cannot be parallelized: Some tasks may not be effectively parallelized, such as dependencies between tasks or certain operations requiring sequential execution. In this case, using multiple servers may not speed up task execution.

Part V Conclusion

In this project, we first learned about RDD and its features and usage. Next, we conducted a deep analysis of the dataset, and then learned and used MLlib and its built-in algorithms. Finally, we compared the performance of Spark on different machines to gain a deeper understanding of its operation.