

Big Data Report

Task 1 – PySpark Implementation of Recommendation System

1

Firstly, the FilmTrust dataset was loaded using Pyspark, and a series of data-cleaning steps were applied to ensure its suitability for building a recommendation system. The cleaning process focused on improving the clarity and reliability of the dataset while maintaining its integrity for downstream tasks such as plotting, graphing, or model development.

Secondly, to enhance the dataset's readability, the raw data was split into three columns: 'user_id,' 'item_id,' and 'rating.' This made it easier to interpret and for anyone reviewing the code to understand the data whilst reading through. Outliers or invalid data were filtered out, such as the ratings, which were limited to between 0 and 5, to avoid any erroneous or skewed results once plotted.

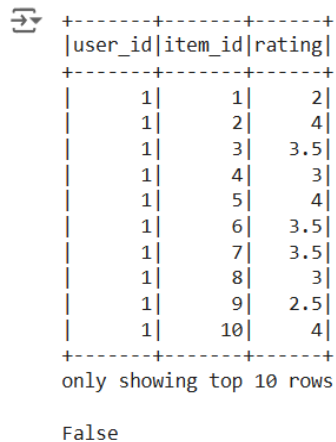
Furthermore, null values were removed as they can cause inaccuracy in the analysis when recommending movies and during model training. Given that the data provided is entirely numerical, not even column names are included; these steps were the standard best practices suitable for cleaning and preparing the data. By removing invalid, null, and duplicate values, the dataset is less likely to cause any inaccuracies and, therefore, more suitable for creating an accurate, unbiased recommendation system.

Finally, the cleaned dataset should now represent the FilmTrust dataset accurately and without redundancy. This makes it suitable for building the recommendation system because it relies on the user's rating to infer the public's preferences and what is most popular with the general public. By addressing these issues during cleaning, the cleaned dataset can give reliable insights and better performance during modeling.

You will also find everything mentioned above within my code, with comments to explain and briefly label which part is which.

Big Data Report

Here is a screenshot of my output:



```
+-----+-----+-----+
|user_id|item_id|rating|
+-----+-----+-----+
|      1|      1|      2|
|      1|      2|      4|
|      1|      3|     3.5|
|      1|      4|      3|
|      1|      5|      4|
|      1|      6|     3.5|
|      1|      7|     3.5|
|      1|      8|      3|
|      1|      9|     2.5|
|      1|     10|      4|
+-----+-----+-----+
only showing top 10 rows
False
```

2

An exploratory analysis was performed to gain insight into the dataset by starting with a statistical summary table of the ratings; this included the mean, minimum, and maximum values. To complement this, the rating distribution was plotted using a histogram, which provided a clear visual representation of the spread of ratings across the dataset of movies. This step helped to identify trends or anomalies in the user ratings.

To make the statistical summary accessible to stakeholders, the columns were given names to enhance readability for non-technical audiences. Next, an analysis of the top ten and lowest ten rated movie tables was created by calculating the total number of ratings for each movie to show which movies were the most popular and which were the least. This provided valuable insights for the creation of the recommendation system.

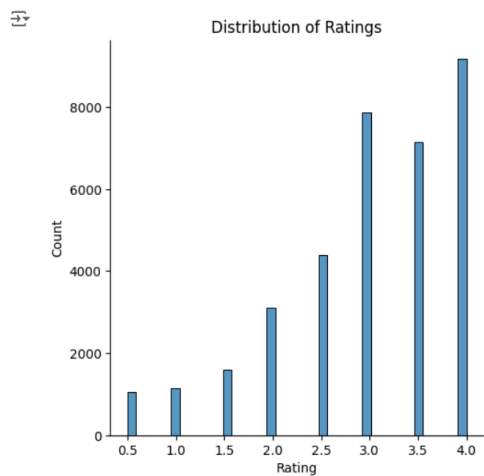
Additionally, a list of the top ten reviewers who provided the most reviews in the dataset was created, as these users' contributions could be particularly valuable for validating the consistency and credibility of the movie ratings. Finally, I calculated and displayed the number of users to movie ratio to understand how the scale of user participation impacted the movie ratings. Finally, the ratio of users to movies was calculated to know how the scale of user participation impacted a movie's ratings.

This analysis laid a strong foundation for understanding and preparing the dataset for further tasks, such as developing a recommendation system.

Big Data Report

Output:

Graph



Statistics, Top and Lowest rated movies, etc:

Statistics For Movie Ratings

Statistics	Movie Ratings
count	35497
mean	3.0028030537791928
stddev	0.9186923817926762
min	0.5
max	4.0

Top Rated Movies

Movie ID	Movie Rating Total
7	1044
11	931
2	915
207	883
1	866
17	815
13	807
215	761
12	756
10	750

Big Data Report

Lowest Rated Movies

Movie ID	Movie Rating Total
1361	1
1957	1
1746	1
1512	1
1808	1
853	1
1870	1
1903	1
1897	1
2034	1

Top Ten Users Who Rate The Most Movies

user_id	Total Movie Ratings
272	244
1187	232
161	197
3	193
969	174
79	172
199	155
702	147
670	137
1039	129

Number of unique users: 1508

Number of unique movies: 2071

3

The recommendation system used the ASL model; it was deemed the most suitable model for scalability due to the large datasets provided and the potentiality of new data being added to them in the future. Using a memory-based model would have proven quicker to adapt to new data since it accesses the data directly; however, memory-based models have to update more periodically, which can prove to be resource-intensive, especially with larger datasets. Using a model-based recommendation system instead would cut down the processing time, as a model-based system has a significantly shorter computation time (can be up to 10 times shorter) and proves to be more efficient at handling the data. It can be argued that a memory-based recommendation system is more straightforward to implement initially and doesn't require too much setup; however, this is primarily overshadowed by its drawbacks, as while it is easier to use, this also means it can require more complex algorithms implemented and more resources may be necessary to develop the model.

Big Data Report

In contrast, the model-based system is more efficient and accurate because it doesn't require one to go through the entire dataset each time it creates a recommendation. The dataset used in this instance is large and only has data relevant to the movie ratings; as previously mentioned, there is also a high likelihood that more information will be added to the data as more users join or start rating new movies, which is why for ease of scalability and the sake of accuracy using a model-based was the most optimal choice in this occasion even though memory based has its positives for use for this specific dataset it wasn't as valuable.

MAE (mean area error) was used to evaluate the model because it is easy to interpret; it is calculated by taking the absolute value of each error, adding them together, and dividing them by the number of errors present. It directly represents the average error without it being by the variability of error magnitudes, unlike RMSE, which involves squaring each of the errors, doing similar, but then taking the square root rather than simply dividing them; this process is more likely to give weight to more significant error which can distort the average error. The results of MAE were 0.43, as shown in the screenshots below. This model's limitation is that it may struggle with making recommendations for new users with no historical data because it relies on collaborative filtering. Also, if this data were made more sparse, where users have rated only a few movies, the ALS wouldn't capture user preferences as effectively, meaning it would perform subliminally. Cross-validation could be included to improve the model further for optimising the parameters and increasing the predictive accuracy.

0.43027785365383775

User ID	Movie Recommendation ID	Predicted Rating
1	1809	3.866673
3	1336	3.9864798
5	1259	3.8470514
6	2030	3.881222
9	1336	4.0961885
12	1336	4.7148447
13	1021	5.2231984
15	1021	4.4833612
16	1416	4.10922
17	1973	4.619214

only showing top 10 rows

Big Data Report

Task 2 – MapReduce for Margie Travel

1

The flight database is loaded into Pyspark and converted into an RDD to transform and aggregate tasks in MapReduce. The departure airport column is extracted and connected with the airport they are flying to; extracting the columns made it easier to complete tasks by knowing where the flights were leaving to collect a count of how many flights had happened per airport.

The reduceByKey was used to count the number of flights precisely and efficiently as the operation aggregates the total number of flights for each used airport. It also ensures the data is distributed and scalable in case it ever needs to handle larger datasets. To identify which airports weren't used, all airports were extracted from the departure and arrival columns using a flat map operation and a distinct operation that stopped the information for each flight being printed repeatedly in the output.

The total number of flights is displayed in the output inside of a table in descending order for better readability. Separating the task into smaller sections with comments makes the code easy to understand and debug. Sorting the airport output in descending order ensures that it is user-friendly. They can understand the context while efficiently looking it over and identifying the most and least used airports.

```
Airports not used as departure points:  
SFO  
SIN  
HKG  
LAX  
FRA  
DXB
```

```
Flight count from each airport:  
Airport DEN has 46 flights.  
Airport CAN has 37 flights.  
Airport IAH has 37 flights.  
Airport ATL has 36 flights.  
Airport ORD has 33 flights.  
Airport KUL has 33 flights.  
Airport CGK has 27 flights.  
Airport JFK has 25 flights.  
Airport LHR has 25 flights.  
Airport CDG has 21 flights.  
Airport CLT has 21 flights.  
Airport PVG has 20 flights.  
Airport LAS has 17 flights.  
Airport BKK has 17 flights.  
Airport AMS has 15 flights.  
Airport FCO has 15 flights.  
Airport MUC has 14 flights.  
Airport MAD has 13 flights.  
Airport PEK has 13 flights.  
Airport HND has 13 flights.  
Airport DFW has 11 flights.  
Airport MIA has 11 flights.
```

Big Data Report

2

Similar to the previous task, the dataset was changed into an RDD to use MapReduce to complete the task. The flight ID and passenger ID were extracted as pairs to find how many passengers were on each flight by using the reduceByKey function to calculate the total number of unique passengers per flight.

The distinct function was also implemented to prevent duplicate entries, which would cause errors. All the relevant flight details were extracted and organised into a structured format so that each flight ID output would include all relevant information required for the task and the specific flight (such as the number of passengers). The join function adds the passenger amount count to the relevant flight ID.

The Unix timestamps are converted to HH:MM format as required by the task, improving the departure times' readability. The final RDD is mapped to make sure no duplicates are in the output and is displayed in a list, which is then formatted into a table with appropriate headers so that the output is more legible for company employees to read if they must have access to this dataset's output.

Flight ID	Departure Airport	Arrival Airport	Dept Time	Total Passengers
SQU6245R	DEN	FRA	17:14	17
MBA8071P	KUL	PEK	17:04	13
MOO1786A	MAD	FRA	16:56	11
GMO5938W	LHR	PEK	17:11	17
DAU2617A	CGK	SFO	17:23	12
ATT7791R	AMS	DEN	17:13	12
VYU9214I	ORD	DXB	17:18	12
VYW5940P	LAS	SIN	17:26	11
WSK1289Z	CLT	DEN	16:59	18
FYL5866L	ATL	HKG	17:25	14
SOH3431A	ORD	MIA	17:00	15
WPW9201U	DFW	PEK	17:21	9
ULZ8130D	CAN	DFW	17:23	18
BER7172M	KUL	LAS	17:26	12
VDC9164W	FCO	LAS	17:18	11
YZO4444S	BKK	MIA	17:28	16
XOY7948U	ATL	LHR	17:07	15
XXQ4064B	JFK	FRA	17:05	19
DKZ3042O	MIA	SFO	17:05	10
EWB6301Y	CAN	DFW	17:22	7
KJR6646J	IAH	BKK	17:26	15
RPG3351U	HND	CAN	16:59	11
PME8178S	DEN	PEK	17:13	15
HUR0974O	DEN	PVG	17:15	6
RUM0422W	MUC	MAD	16:58	10
QHU1140O	CDG	LAS	17:14	14
HZT2506M	IAH	AMS	17:12	10
TMV7633W	CGK	DXB	17:05	11
JVY9791G	PVG	FCO	17:16	17
XIL3623J	PEK	LAX	17:13	11

Big Data Report

3

In this task, the goal is to calculate the line-of-sight (nautical) miles for each flight and then find out who has the highest air miles saved up in total. The solution is implemented using MapReduce principles to satisfy the task requirement; it is also optimal for usage on large datasets such as this one as it is a highly efficient distributed model well suited for big data processing.

The data is loaded in an RDD format and then parsed to extract the relevant columns for each flight; this allows us to clean the data and transform it to focus on the specific fields needed for the following calculations. To calculate the line-of-sight distance, the airport data must also be parsed to extract each airport's codes, latitude, and longitude. The geographical coordinates of each airport are necessary for finding the line-of-sight distance by converting the date into a dictionary, and this information can be accessed easily for further use.

The line-of-sight was calculated using the Haversine formula, commonly used for calculating the distance between two points using latitude and longitude. Using MapReduce to complete this task is optimal as it can calculate the distance between the departure and arrival points for each flight using the map function, making it more efficient. Flight_distance_rdd is transformed into a new RDD called passenger_miles_rdd, which uses the MapReduce principles of mapping each flight to a key-value pair (passenger_ID, Distance) and then uses the reduceByKey function to aggregate the total miles per unique passenger.

The takeOrdered function retrieves the top passenger with the most air miles accumulated, as it is the most efficient way to do so from an RDD. Finally, PrettyTable displayed the output in a neat table format.

Passenger with the Highest Air Miles:

+-----+-----+	
Passenger ID	Total Air Miles
+-----+-----+	
UES9151GS5	131879.99042795028
+-----+-----+	

Big Data Report

Task 3 – Big Data Tools and Technology Appraisal

In task 1, a movie recommendation system was developed using Pyspark and an ALS model. This was optimal for handling the scalability of large datasets by using ALS, a model-based rather than a memory-based because the memory-based needs to go through the entire database directly every time it creates a recommendation, which becomes resource expensive overtime, whilst model-based methods create models from the data which is then used to generate recommendations quickly. ALS models are one of the most commonly used models therefore, it was efficient and compliant with completing the objectives of task 1 however, with more research, there could be a model method that fits more closely to this dataset. More validation checks could have been implemented, as whilst there is a check for duplicates within the dataset, an efficient cleaning code has not been implemented for removing all duplicate data from the dataset.

In task 2, the requirement was to use MapReduce to complete all three tasks. MapReduce was the ideal method to use in this scenario as you are required to filter through lots of data for the three tasks given. MapReduce can cut this data into smaller manageable chunks used for scalability, especially in the last task where you have to load and join together two different datasets to find the distance and fill in the data for the full passenger flight information.

However, MapReduce also has its own limitations, such as potentially having a high latency since it needs to go through multiple stages (mapping, shuffling, and reducing) this makes it less ideal than using a model-based method if the company needs real-time processing. For example, using MapReduce for Task 1 wouldn't be ideal. Most movie rating companies would need real-time updates as consumers worldwide are constantly logging in and looking for new recommendations. MapReduce shows its uses in scalability compared to PySpark but falls short in updating information in real time. Therefore, whilst PySpark is definitely helpful for movie recommendations, if it had been used for task 2, it would have taken longer to process the data than MapReduce, so it was the most optimal choice.

If MapReduce fails a task, this can slow down the entire job significantly as it tries to recover, which isn't a prominent disadvantage in task 2, but in task 1, this would prove a significant issue.

In conclusion, the ALS model and MapReduce used in their respective tasks are the best choices given for their specific tasks as they have larger limitations than advantages should the task they are used for be reversed.

Big Data Report

References:

1. Aditya, P., Budi, I. and Munajat, Q. (2016). *A Comparative Analysis of Memory-based and Model-based Collaborative Filtering on the Implementation of Recommender System for E-commerce in Indonesia : A Case Study PT X*. [online] Available at: <https://qoribmunajat.github.io/files/comparative-analysis-memory-based-model-based-recommendation-systems.pdf> [Accessed 5 Jan. 2025].
2. Willmott, C. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, [online] 30(1), pp.79–82. doi:<https://doi.org/10.3354/cr030079>. [Accessed 5 Jan. 2025].
3. Wikipedia Contributors (2019). *Haversine formula*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Haversine_formula. [Accessed 5 Jan. 2025].
4. Guo, G., Zhang, J. and Yorke-Smith, N. (2016). A Novel Evidence-Based Bayesian Similarity Measure for Recommender Systems. *ACM Transactions on The Web*, 10(2), pp.1–30. doi:<https://doi.org/10.1145/2856037>. [Accessed 5 Jan. 2025].