Real-time music recommendation system based on Spark Streaming

Jiapeng Shi
Master of Data Science and Artificial Intelligence
University of Waterloo
Waterloo, Ontario, Canada
j45shi@uwaterloo.ca

Hanqing Bi
Master of Data Science and Artificial Intelligence
University of Waterloo
Waterloo, Ontario, Canada
h26bi@uwaterloo.ca

Figure 1: Spark Streaming [1].

ABSTRACT

Real-time recommendation systems have become more and more popular recently because they can give personalized recommendations quickly according to the user's current operation, which can improve user stickiness. The real-time recommendation system has two characteristics: it can continuously process the streaming data, and the corresponding recommendation algorithm does not require too much computation. Based on this, this paper builds a music recommendation system based on Apache Kafka and Spark streaming. After simulating the data spread of users' clicks to Apache Kafka, Spark Streaming receives the data in Apache Kafka in real-time and runs the recommendation algorithm to give the latest recommendation song list. To ensure the lightweight of the recommendation algorithm, the ALS algorithm will be used to generate the music similarity matrix offline as the data support of the recommendation algorithm.

CS 651, Fall 2023, University of Waterloo, Ontario, CA

CCS CONCEPTS

• **Information systems** → *Data stream mining*

KEYWORDS

Big Data, Data mining, Real-time recommendation systems, Spark Streaming, Apache Kafka

ACM Reference format:

Jiapeng Shi and Hanqing Bi. 2023. Real-time music recommendation system based on Spark Streaming. In *CS 651 Data-Intensive Distributed Computing. University of Waterloo, Waterloo, ON, CA, 8 pages.*

1 INTRODUCTION

The purpose of the recommendation system is to recommend content that may be of interest to users according to their preferences [2]. In recent years, recommendation systems have rapidly entered people's lives. For example, online shopping platforms will recommend products that users may need based on users' search and purchase records, and short video platforms will recommend the next video that users may be interested in in real time according to users' video playback completion degree and users' likes and favorites. Some music platforms will also provide users with more suitable playlists in real time based on the songs they listen to and the number of times they play them [3]. The main way of recommendation system is to analyze the historical behavior of users and combine the historical behavior of other users for similarity analysis, so as to predict the content that users are more interested in [4].

The recommendation system can be divided into real-time recommendation system and offline recommendation system. In the case of offline recommendation system, it is an analysis of the long-term behavior of users, which requires a long period of historical data, and usually uses more complex algorithms and requires a larger amount of computing resources. The implementation of the recommendation system is different from the offline recommendation system. The algorithm involved in it is relatively efficient and can quickly extract the information of recent user behavior history, quickly make corresponding responses to user behaviors, and make better recommendations according to the current status of users [5].

With the development of the field of machine learning and the continuous improvement of hardware computing power, the content recommended by the recommendation algorithm for users is more and more in line with the taste of users, improving user satisfaction, improving user stickiness for enterprises, and increasing revenue [3]. Therefore, the recommendation system is very important today.

2 LITERATURE SURVEY

2.1 Data Streaming

Data Streaming is a process of continuous transmission from the source of data production to the destination of data, in which data is generated, transmitted, and calculated in a very short time and near real-time speed. It usually refers to a technology that simultaneously processes data in a short time after it is generated. The source of this data production can be a sensor, log, social media, user behavior history [6]. After the data is generated, it is usually received, processed, and distributed by data processing engines, such as Apache Kafka, Apache Storm, Apache Flink, etc. The final result is calculated and produced in a program or framework that processes Streaming data, such as Spark Streaming, Apache Kafka Streams, Apache Flink, Apache Storm, etc. [7]. The final result of the processed and calculated data is stored in the database for application or further processing. The characteristics of Data Streaming are mainly as follows. High throughput and low latency make it possible to quickly generate large amounts of data while being transmitted and processed in a short time [8]. Continuity and unbounded means that the Data Streaming is usually regarded as continuous and infinite. Unlike the data set in batch processing, it can be processed continuously and does not depend on the boundary of the beginning and end of the data set and is highly dependent on time [9]. The system can call resources and compute processing power dynamically according to the demand, which is usually divided into horizontal scaling (adding more processing nodes) and vertical scaling (enhancing the computing power of existing nodes, usually with more fault tolerance and greater flexibility) [10]. In addition, Data Streaming also has high fault tolerance and reliability. By replicating data and backup status in multiple nodes, data transmission and calculation can be restored in a short time when a single node fails, and the output results can still be maintained continuously and completely. In short, Data Streaming is a powerful guarantee and an indispensable tool for making real-time recommendation system [11].

2.2 Spark Streaming

Spark Streaming is an extension module of Apache Spark that was open sourced in 2010 and is considered one of the main platforms for big data processing in the scenario of rapid iterative data processing tasks [12]. Spark Streaming can obtain and process data in real time from Kafka, Flume, Twitter and other channels, while providing a variety of high-level APIs, which can support programming in Scala, Java and Python languages [12]. Same as Spark, Spark Streaming also supports distributed processing, which can handle large-scale data streams and has high dynamic change ability [13]. And support micro batch processing model, stream data can be cut into small data segments and then processed, thus reflecting the advantages of the combination of stream processing and batch processing, for example, it is easy to tradeoff between improved throughput and low latency and find a reasonable balance [14]. Spark Stream is a flexible and efficient framework, with a variety of development environments and support for a variety of components, and is a powerful tool for handling Data Streaming, Data Streaming in dynamic recommendation systems, and final calculations.

2.3 Apache Kafka

Apache Kafka is an open-source distributed stream processing platform based on a publish-subscribe message queue architecture, developed by LinkedIn in 2011 [15]. Producers in Apache Kafka support placing the output to be processed selectively or randomly into the desired partition of the desired topic, and consumers can subscribe to multiple topics at the same time to read data from multiple topics [16]. Apache Kafka can transfer millions of messages per second and has extremely powerful Data Streaming capabilities, which makes it ideal for handling Data Streaming in implementation recommendation systems [17]. At the same time, its distributed characteristics also enhance its horizontal scalability, which can better dynamically allocate resources according to the size of the data stream. Apache

Kafka also has a very powerful ecosystem that works well with big data processing tools such as Apache Spark Streaming, Apache Hadoop, and Apache Storm [18]. Apache Kafka will automatically backup messages between multiple nodes and store all messages on disk, even if an individual node crashes Kafka will run correctly, even if the entire server crashes disk storage features will not cause messages to be lost [19]. In short, Apache Kafka is a widely used and powerful big data processing tool, which undertakes the management and distribution of data flow and processing tasks in the dynamic recommendation system.

3 PROBLEM STATEMENT

The fast response of the recommendation system to user behavior makes it more flexible and sensitive and conforms to the fast pace of the current Internet environment. Music platform companies want to recommend music users may like in real time to improve user viscosity, and many users of music platforms also want to explore more music they are interested in [20]. Therefore, in this article, Spark Steam is used to process real-time data flow, Kafka is used to manage data flow, MongoDB is used to manage data, and machine learning library in Spark is used to analyze long-term historical data to provide data support for real-time recommendation algorithm. In this way, a real-time music recommendation system is implemented to recommend more music that suits the user's taste and style.

4 METHODOLOGY

4.1 Motivation

A machine learning model can be trained to predict what music users will listen to in the future using all of their historical data, but this is the domain of offline recommendation services. However, when it comes to the recommendation system to react quickly to the user's actions, the offline recommendation system has two disadvantages. Firstly, offline recommendation service integrates all historical data of users. When new data is brought by user operation, it takes a long time to retrain the model, which cannot meet the requirements of real-time and rapid response [21]. Secondly, compared with all the historical data of the user, the current operation of the user has little change to the data set, and the offline recommendation system is likely to continue to maintain the original recommendation, rather than quickly respond to the latest operation of the user [21].

Therefore, redesigning a real-time recommendation algorithm to use the offline recommendation results as data support, so as to meet the requirement of fast calculation, and adjust the offline recommendation results according to the recent historical operations of users, so that the real-time recommendation has the ability of dynamic change.

Specifically in this project, calculating the similarity matrix of music through all the historical data of users, that is, each song is more similar to those songs, as the supporting data of real-time recommendation. Then, every time the user takes an action, a real-time recommendation algorithm is run to update the current best song recommendation.

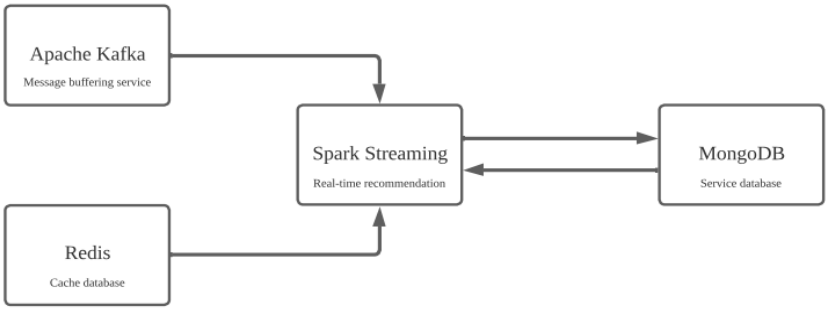

Figure 2: Workflow for Real-time Recommendation System

4.2 Project overview

The user historical behavior data will be loaded into the MongoDB database through the data processing module. Then, the ALS algorithm is used to calculate the feature matrix of the song offline, and then the similarity matrix of the song is obtained by calculating the cosine similarity between any two songs. Finally, the above part is built into a data stream from Kafka to spark streaming, then to real-time recommendation algorithm, and finally to MongoDB, which is the process to carry out real-time recommendation. The process is as follows: Generate a user action stream [UID | MID | SCORE | TIMESTAMP] and send it to a Kafka queue. Spark Streaming will monitor the Kafka queue, obtain the user score data stream filtered out by Kafka in real time, fuse the user's recent score queue data stored in Redis, and submit to the real-time recommendation algorithm; Real-time recommendation algorithm is used to calculate the user's new recommendation results. After the calculation is completed, the new recommendation structure is merged with the recommendation result in the MongoDB database, and the updated recommendation playlist can be generated.

4.3 Data loading

Last.fm is a dataset of music recommendations [22]. For each user in the dataset, user_artists.dat contains a list of their most popular artists and the number of plays, which is the most important things the user behavior data. Artists.dat and tags.dat contain detailed and classified labels for music. In Data loading, the data will be read and loaded into the MongoDB database.

The specific implementation idea is as follows: several sample classes for the original data will be define, then the data from the file will be read through the textFile method of SparkContext, and convert it to DataFrame prepared to be processed, such as Z-score standardization of the number of plays. Compared with the underlying RDD, DataFrame supports SQL-like queries and relational operations, making data processing more intuitive and simpler, and more suitable for processing structured data.

Next, using the casbah library to implement the data path to MongoDB. casbah is a MongoDB driver library for the Scala language that provides an interface to the Scala programming language to access and manipulate MongoDB databases. Finally, the write method provided by Spark SQL is directly used for distributed data insertion.

```
val config = Map(
"mongo.uri" -> "mongodb://localhost:27017/MusicRecommender",
"mongo.db" -> "MusicRecommender"
)

// Define implicit parameters for MongoDB connection configuration
implicit val mongoConfig = MongoConfig(config("mongo.uri"), config("mongo.db"))

// Create a new mongodb connection
val mongoClient = MongoClient(MongoClientURI(mongoConfig.uri))
```

Figure 3: How to Use MongoDB

4.4 Similarity matrix

The feature matrix of songs is calculated by Alternating Least Squares (ALS), and then the similarity matrix of songs is further calculated by this matrix to serve the real-time recommendation system [23].

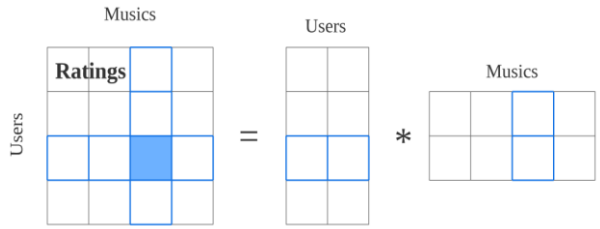

Figure 4: Calculating the Similarity Matix by ALS

When calculating the direct similarity of a song, it is generally necessary to calculate the similarity according to the feature matrix of the song, for example, the song has features such as rhythm, instrument, singer and style, and the similarity can be further calculated through the features. But in the real-time recommendation scenario, only the user's behavior data, that is, the user's operation on different songs. If a song has a series of hidden features, and users like songs with these features because they like these features, then if these hidden features can be extracted from the user's behavior data, the feature matrix of the song can be obtained.

ALS model is a collaborative filtering algorithm. Through matrix decomposition of user behavior matrix R, the user factor vector matrix X and item factor vector matrix Y are obtained, in which the extra dimension k is the hidden feature. The similarity matrix can be obtained by calculating the similarity of each item in Y.

$$R_{m\times n} \approx X_{m\times k} Y^T_{n\times k}$$

The optimization goal of the algorithm is to make the low-rank matrix X and Y as close to R as possible, and the following square error loss function needs to be minimized:

$$\min_{x_*,y_*} \sum_{u,i\ is\ known} (r_{ui} - x_u^T y_i)^2$$

ALS is optimized iteratively in the following way: at each iteration, one of the factor matrices needs to be fixed to update the other matrix factor matrix, and then the updated matrix is fixed, and the other matrix is updated until the model converges.

spark mllib provides the corresponding ALS implementation, which can be called directly: ALS.train(trainData, rank, iterations, lambda) Where trainData is training data, assuming that the number of iterations is 5, rank and lambda are the two parameters to be determined. Where rank is the number of hidden features, i.e., the length of the hidden vector. lambda is a regularization parameter that controls the complexity of the model and prevents overfitting. The best parameters are determined by traversing different combinations of rank, lambda, and calculating the root mean square error (RMSE) (looking at the error between the predicted score and the actual score).

After calculating the characteristic matrix of music, the similarity matrix of music can be obtained by calculating the cosine similarity between each pair of music. It is eventually stored in the MongoDB database as a Music Recs table [24].

$$Sim(p,q) = \frac{\sum_{i=0}^{k}(t_{pi}\times t_{qi})}{\sqrt{\sum_{i=0}^{k} t_{pi}^2}\times\sqrt{\sum_{i=0}^{k} t_{qi}^2}}$$

4.5 Real-time recommendation algorithm

When user clicks on song, an update to user's recommendation results will be triggered. This update is to obtain the first Q similar songs corresponding to song from the song similarity matrix, and then adjust the score in Q according to the collection RK composed of the most recent K songs heard in history (including the latest click on song s). Specifically, Q similar songs are recalculated according to the following method:

$$E_{uq} = \frac{\sum_{r \in RK} sim(q,r)}{sim\_sum}$$

E_{uq} indicates the recommended score of song based on the most recent K operations in the user's history.

$sim(q,r)$ represents the similarity between song q which is clicked by user and song in R. The minimum similarity is set to be 0.5 and when the similarity between song q and song r is below the threshold of 0.5, the two are regarded as irrelevant and ignored.

sim_sum indicates the number of songs whose similarity between q and RK is greater than the minimum threshold.

Firstly, for each song(q) in Q, the similarity between it and K songs that u has heard recently is calculated, and the average similarity is used as the recommended score for song q. If the similarity is high, it means that the song q is in line with the taste of recent u, which can increase the score; On the contrary, it indicates that although q and s are close from the historical data of all users, q does not meet u's recent taste, and the corresponding one is not recommended.

Next, this recommendation set $updatedList$ is merged with the last real-time recommendation result Rec before this real-time recommendation for u. The specific operation is to select K songs with the highest score in the two sets to form a new recommendation set $NewRec$, the formula is as follows:

$$NewRec = topK(i \in Rec \cup updatedList, cmp = E_{ui})$$

In general, the real-time recommendation algorithm flow is basically as follows:

1. User(u) listens to song(s), triggering a calculation of real-time recommendation.
2. Select K songs that are most similar to song s as set Q.
3. Obtain K songs listened by user u in the latest time, including songs of this time, which is set RK.
4. Calculate the average similarity between each song Q in q and each song in RK as the recommendation priority (songs whose similarity is less than 0.5 are ignored), and generate the recommendation set $updatedList$.
5. Merge $updatedList$ with the recommendation result Rec of user(u) last time and take the K values with the highest score (no repetition) to generate a new recommendation result $NewRec$ as the final output.

```
for( candidateMusic <- candidateMusics; userRecentlyRating <- userRecentlyRatings){
    // Get the similarity between the candidate movie and the recently rated movie
    val simScore = getMusicsSimScore( candidateMusic, userRecentlyRating, simMusics )
    // Calculate base recommendation scores for candidate movies
    scores += ( (candidateMusic, simScore) )
}
 scores.groupBy(_._1).map{
    case (mid, scoreList) => (mid, scoreList.map(_._2).sum / scoreList.length)
 }.toArray.sortWith(_._2 > _._2)
```

Figure 5: How to Calculate Scores for Recommendation

4.6 Real-time recommendation system

This step connects the modules mentioned above in series and applies Kafka to manage the data flow to make the entire recommendation system work in tandem. The first is the offline part, which stores the music dataset related code and the music similarity matrix obtained by ALS algorithm into MongoDB. Next, the Redis cluster stores K songs that each user has recently listened to. In particular, start zookeeper and Kafka first, and then create a topic called recommender for the project to read the current data:

```
PS D:\kafka_2.11-2.1.0\kafka_2.11-2.1.0>
.\bin\windows\kafka-server-start.bat ".\config\server.properties"

PS D:\kafka_2.11-2.1.0\kafka_2.11-2.1.0>
.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic recommender
```

Figure 6: Reading Current Data

The data flow of real-time recommendation is as follows: streaming data to Kafka streaming for data cleaning and preprocessing to Spark streaming flow calculation (recommendation algorithm) to Save the recommended song list in MongoDB for front-end call. The real-time recommendation system is written by Spark Streaming, which receives the cached data in Kafka, runs the real-time recommendation algorithm to realize the data processing of real-time recommendation, and merges the structure to the MongoDB database. Spark Streaming applications are able to read data streams directly from Kafka topics using APIs:

```
val kafkaStream = KafkaUtils.createDirectStream[String, String](
ssc,
LocationStrategies.PreferConsistent,
ConsumerStrategies.Subscribe[String, String]( Array(config("kafka.topic")),
kafkaParam))


val ratingStream = kafkaStream.map(
msg => {// Convert the original data UID|MID|TIMESTAMP into a scoring stream})


ratingStream.foreachRDD{rdds =>
rdds.foreach{// Core real-time recommendation algorithm part}
```

Figure 5: Processing Data Stream

History	Style	Click (David Archuleta)	Style	Click (Dokken)	Style	Click (Patrick Wolf)	Style	Click (Scritti Politti)	Style	Click (Robert Palmer)	Style
David Archuleta	Pop/R&B	Dokken	Hard rock	Wynter Gordon	Pop/R&B	Scritti Politti	Post-punk	Robert Palmer	Rock	Ian Van Dahl	Electronic
Panic at the Disco	Pop rock	Michael Jackson	Pop	Michael Jackson	Pop	Whitesnake	Hard rock	Simon Le Bon	Pop	Exposé	Pop
P!nk	Pop rock	Cristina Mel	Gospel	Glenn Tipton	Hard rock	Robert Palmer	Rock	Mike Peters	Rock	Scritti Politti	Post-punk
Alicia Keys	Soul/R&B	Anna Nicole Smith	Pop	Arabesque	Disc/Pop	Dokken	Hard rock	The Alarm	Rock	Heaven 17	Electronic
Katy Perry	Pop rock	Blue Angel	Pop rock	Michael Angelo Batio	Hard rock	Bananarama	Pop	Haircut 100	Pop	The Psychedelic Furs	Post-punk
Jennifer Lopez	Pop/R&B	Mara Maravilha	Pop/Gospel	Patrick Wolf	Electronic	TSA	Hard rock	Bachelor Girl	Pop	The Fixx	Pop rock
Lily Allen	Pop	Cyndi Lauper	Pop	Bonfire	Hard rock	Vixen	Hard rock	Gowan	Rock	Novaspace	Pop

Figure 6: Top 7 Recommendations from Real-time Music Recommendation System

History	Style	Iter 1	Style	Iter 2	Style	Iter 3	Style	Iter 4	Style	Iter 5	Style
David Archuleta	Pop/R&B	Michael Jackson	Pop	Bonfire	Hard rock	Whitesnake	Hard rock	Kirsty MacColl	Pop	Kim Wilde	new wave
Panic at the Disco	Pop rock	Adam Lambert	Pop rock	Live	Rock	Eddie Money	Rock	The Psychedelic Furs	Post-punk	Mike & The Mechanics	Rock
P!nk	Pop rock	Cyndi Lauper	Pop	Nitro	Hard rock	Jamiroquai	Electronic	Kim Wilde	Pop	Scritti Politti	Post-punk
Alicia Keys	Soul/R&B	Joss Stone	Soul/Jazz	The Entire Population Of Hackney	Hard rock	Ruoska	Metal	Heaven 17	Electronic	Heaven 17	Electronic
Katy Perry	Pop rock	Anna Nicole Smith	Pop	Laura Branigan	Pop	Bad Manners	Ska	Madness	Ska	Level 42	new wave
Jennifer Lopez	Pop/R&B	Victoria Silvstedt	Pop	Agonoize	Hard rock	Paul Young	Pop	Thomas Dolby	Electronic	Novaspace	Electronic
Lily Allen	Pop	Cyndi Lauper	Pop	Michael Jackson	Pop	Eddie Money	Rock	Love and Rockets	Post-punk	19	Folk

Figure 7: Top 7 Recommendations from Offline Music Recommendation System

5 EVALUATION

How to average the effect of the music recommendation system is a difficult problem, and in many cases, it is difficult for anyone but the user to give a quantitative description of the quality of the recommendation. The algorithm in this project only uses the user's behavior data. Even if the feature matrix of music is obtained by ALS algorithm, the specific meaning of these hidden features cannot be given. Although there are some indicators such as AUC or MAP that can quantitatively give the quality of evaluation recommendation, AUC is used to evaluate the performance of binary classification model, and it is difficult to split user behavior data into training set, cross-validation set, and test set to calculate

MAP index. Therefore, the evaluation method of examining the recommendation results one by one is adopted here, and the style of music is used as the evaluation standard for whether the music is similar, so as to intuitively evaluate the quality of the recommendation results.

The experiment of this project selects a user who has stored historical behavior data in Redis as the experimental object, simulates the user to play music of different styles of artists for 5 times, and compares the recommended artist list under real-time recommendation with the recommended artist list only using the current click artist as the similar artist. At the beginning, the user's history of playing is mostly pop style and pop rock style, it can be inferred that his current taste is biased towards pop music, pop rock direction. The next four hits were Hard rock, electronic, post-punk, and rock artists. On the whole, offline recommendation has a wide range of style changes per recommended artist, especially after 2, 3, and 4 clicks, and does not do a good job of retaining the user's previous taste toward pop and pop rock. For real-time recommendations, even after 4 clicks, pop and pop rock artists will still be recommended.

There are two drawbacks to offline recommendations that only include similar artists from the most recent click. First, it is easy for users to rotate between artists of one style. For example, artists of pop style are still mostly similar artists of pop style, and they even recommend each other. However, if the historical information contains artists of different styles, the recommended artists will also deviate from different artists. Second, if users click on different artists, the recommendations change too quickly. For example, in the recommendation corresponding to the artist of pop style, there happens to be a similar artist of rock style recommended, then the updated recommendation will belong to rock style, but the similarity of pop style cannot be guaranteed. A user might happen to like an artist who has both rock and pop styles, but if they just recommend rock style, they won't like it very much.

For the real-time recommendation algorithm, it will not only select the artist whose most recent click is similar to the artist, but also consider how similar it is to the historical artist. Therefore, although the style of artists will deviate from Hard rock, electronic, post-punk, rock and other styles, they will choose the style artists who are closest to pop and pop rock among the styles of Hard rock, electronic, post-punk, and rock. Seeing from the table above, even by the 4th click, the recommendations are still dominated by artists in the pop and rock genres. The benefit of such a recommendation effect is that the recommended content will be updated in real time and will not deviate too far from the user's recent taste, which is a gradual change process, which is easier for users to find their real interest points or find songs that can both make themselves refreshing but not too weird.

6 CONCLUSION AND FUTURE SCOPE

In this paper, a real-time music recommendation system is implemented. Algorithmically, it has both an artist similarity calculated based on all historical data of all users, and a real-time recommendation algorithm based on that data. In order to make the system run smoothly, using spark to process offline data, build a streaming data processing pipeline based on spark streaming and Kafka, and use MongoDB and Redis NoSQL databases to store recommended data and cache user history behavior respectively. After the data pipeline is built, the four clicks of the user are simulated, the recommendation effect is evaluated by checking one by one, and the advantages of the real-time recommendation system are analyzed.

Of course, there is a lot of room for improvement in this project. The real-time recommendation algorithm can be more detailed, and it is rough to measure only the average similarity of historical data. It can increase the penalty amount or add different weights according to the distance of time. If there is not only a music similarity matrix derived from user behavior data, but also data from other channels to give music similarity evaluation, more data support can be provided. Second, the evaluation method of the experiment is also rough. Artists' styles are very diverse, and their similarity is difficult to evaluate with a single style label. If a test set can be established to evaluate the quality of the recommendation system quantitatively, a better algorithm can be trained.

REFERENCES

[1] David Virgil Naranjo. Spark Structured Streaming output mode. In *Big Data and Scala Blog.* Accessed on November 5, 2021. Available at: https://dvirgiln.github.io/spark-structured-streaming-output-modes/.

[2] Deepjyoti Roy and Mala Dutta. 2022. A systematic review and research perspective on recommender systems. In *Journal of Big Data*, 9, 59. DOI: https://doi.org/10.1186/s40537-022-00592-5.

[3] Arpita Chaudhuri, Debasis Samanta and Monalisa Sarma. 2021. Modeling User Behaviour in Research Paper Recommendation System. Accessed on November 7. Available at: https://ar5iv.labs.arxiv.org/html/2107.07831.

[4] Pablo Castells. 2023. Recommender Systems: A Primer. In *"Advanced Topics for Information Retrieval" by R*. Baeza-Yates and O. Alonso. Accessed on November 7. Available at: https://arxiv.org/abs/2302.02579.

[5] Deepjyoti Roy and Mala Dutta. 2022. A systematic review and research perspective on recommender systems. In *Journal of Big Data*, 9, 59. DOI: https://doi.org/10.1186/s40537-022-00592-5.

[6] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. *SIGMOD* Rec. 34, 2 (Jun, 2005), 18–26. DOI: https://doi.org/10.1145/1083784.1083789.

[7] May Thet Tun, Dim En Nyaung, and Myat Pwint Phyu. 2019. Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming. In *Proceedings of the 2019 International Conference on Advanced Information Technologies (ICAIT 2019)*, 25-30. DOI: https://doi.org/10.1109/AITC.2019.8920960.

[8] Thilina Buddhika, Ryan Stern, Kira Lindburg, Kathleen Ericson, and Shrideep Pallickara. 2017. Online Scheduling and Interference Alleviation for Low-Latency, High-Throughput Processing of Data Streams. In *IEEE Transactions on Parallel and Distributed Systems*, 28, 12, 3553-3569. DOI: https://doi.org/10.1109/TPDS.2017.2723403.

[9] Arvind Arasu, Brian Babcock, Shivnath Babu, et al. 2016. STREAM: The Stanford Data Stream Management System. In *Data Stream Management: Processing High-Speed Data Streams*, 317-336. DOI: https://doi.org/10.1007/978-3-540-28608- 0_16.

[10] Daniel Warneke and Odej Kao. 2011. Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. In *IEEE Transactions on Parallel and Distributed Systems*, 22, 6, 985-997. DOI: https://doi.org/10.1109/TPDS.2011.65.

[11] Matei Zaharia, Tathagata Das, Haoyuan Li, et al. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*, ACM Press, New York, NY, 423–438. DOI: https://doi.org/10.1145/2517349.2522737.

[12] Michael Armbrust, Doug Bateman, Reynold Xin, et al. 2016. Introduction to Spark 2.0 for Database Researchers. In *Proceedings of the 2016 International*

Conference on Management of Data (SIGMOD '16), ACM Press, San Francisco, CA, 2193–2194. DOI: https://doi.org/10.1145/2882903.2912565.

[13] Dazhao Cheng, Xiaobo Zhou, Yu Wang, et al. 2018. Adaptive Scheduling Parallel Jobs with Dynamic Batching in Spark Streaming. In *IEEE Transactions on Parallel and Distributed Systems*, 29, 12, 2672-2685. DOI: https://doi.org/10.1109/TPDS.2018.2846234.

[14] Ahmed S. Abdelhamid, Ahmed R. Mahmood, Anas Daghistani, et al. 2020. Prompt: Dynamic Data-Partitioning for Distributed Micro-Batch Stream Processing Systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20),* ACM Press, New York, NY, 2455–2469. DOI: https://doi.org/10.1145/3318464.3389713.

[15] Rishika Shree, Tanupriya Choudhury, Subhash Chand Gupta, et al. 2017. KAFKA: The modern platform for data management and analysis in the big data domain. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET),* 1-5. DOI: https://doi.org/10.1109/TEL-NET.2017.8343593.

[16] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, et al. 2019. A Survey of Distributed Data Stream Processing Frameworks. In *IEEE Access*, 7, 154300-154316. DOI: https://doi.org/10.1109/ACCESS.2019.2946884.

[17] Han Wu, Zhihao Shang, and Katinka Wolter. 2020. Learning to Reliably Deliver Streaming Data with Apache Kafka. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN),* 564-571. DOI: https://doi.org/10.1109/DSN48063.2020.00068.

[18] Shubham Vyas, Rajesh Kumar Tyagi, Charu Jain, et al. 2021. Literature Review: A Comparative Study of Real-Time Streaming Technologies and Apache Kafka. In *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT),* 146-153. DOI: https://doi.org/10.1109/CCICT53244.2021.00038.

[19] Nishant Garg. 2013. *Apache Kafka*. Packt, Birmingham, UK.

[20] Yasi Huang, Yazhao Huang, and Pengcheng Li. 2021. Research on Emotional Marketing and Copywriting of NetEase Cloud Music. In *Proceedings of the 2021 International Conference on Public Relations and Social Sciences (ICPRSS 2021),* Atlantis Press, 911-915. DOI: https://doi.org/10.2991/assehr.k.211020.280.

[21] Rocío Cañamares, Pablo Castells, and Alistair Moffat. 2020. Offline evaluation options for recommender systems. In *Information Retrieval Journal*, 23, 4, 387-410. DOI: https://doi.org/10.1007/s10791-020-09371-3.

[22] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, et al. 2011. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*. Available at: http://millionsongdataset.com/.

[23] Gábor Takács and Domonkos Tikk. 2012. Alternating Least Squares for Personalized Ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys '12)*, ACM Press, New York, NY, 83–90. DOI: https://doi.org/10.1145/2365952.2365972.

[24] Faisal Rahutomo, Teruaki Kitasuka and Masayoshi Aritsugi. 2012. Semantic Cosine Similarity. In *ResearchGate*. Available at: https://www.researchgate.net/publication/262525676.