EMERGIFY

Spotify recommender system using pyspark and Kafka streaming

INFORMATION SYSTEM FOR BIG DATA

23 novembre 2021

DOMAIN OF THE PROBLEM

At the time of liquid music and the development of digital markets, a user's listening experience has changed considerably thanks to the spread of new tools and streaming platforms that have changed their most consolidated habits. With the application, in the music sector, of concepts typical of Artificial Intelligence and recommendation techniques, new trends have emerged that have redefined its contours promoting in the listener the birth of new needs. With the rapid development of web, mobile apps and IoT devices, a huge volume of data are created every day. In fact, these data are not only large, but also generated quickly and with a variety of different formats. Many companies are forced to face the problem of processing these large data in near real-time.

In this context, Streaming music services, such as Spotify, have worked to not only offer a wide catalogue of content, but a real musical experience aimed at increasing the satisfaction of its users through the help of computer algorithms. The proposals, following the implementation of these automatic systems, become increasingly diversified, personalized and tailored to the tastes of the profiles created. By the way, playlists acquire prominence which, with their targeted advice, aim to offer the right song at the right time. For this project are going to use a content-based recommendation system and new technologies studied during the Information Systems for big data course.

The software, takes songs from different playlists and recommends similar songs from a streaming data source. To perform this process we will use Kafka for data streaming, pyspark dataframe and Spark SQL to perform spark and streamlit operations to view it all. We will also use MLlib of pysparl for KMeans and PCA analysis.

Spotify's personalized playlists

In recent years the well-known on-demand music streaming service, made in Sweden, Spotify, has given great importance to the creation of playlists based on machine learning, expanding this section available to the user by offering new songs that will be based on different criteria, such as favourite artists, decades and genres.

The playlists names will be clear and will help the user to accurately understand what to expect. For example, in recent days, Spotify has presented 2 new playlists “On Repeat" and "Repeat Rewind". Specifically, On Repeat includes a list of some most listened tracks in the last 30 days, substantially grouping the current preferences of its users. While, Repeat Rewind is composed of the songs that the user in question has listened to the most in recent months. These playlists are both composed of 30 songs and will be automatically updated every 5 days as the user uses the music service. We can then remember some of the most popular and famous playlists such as New Music Friday, Release Radar and Discover Weekly.

A NEW PLAYLIST

1. EMERGING ARTISTS

The system will create a playlist of emerging songs that, due to features (instrumentalness, liveness, mode, moudness, speechiness, etc.), are similar to our favourite songs. The playlist will then be shown in the users' home page.

The principles behind this idea are based on the TikTok model: one of the reasons for the success of this platform is that the published content is not only shown to a small circle of friends (see Instagram or Facebook) but can be shown to any user who is using the application anywhere in the world. The "opportunity" to go viral leads more and more users to publish more and more content by making TikTok's fortunes.

On the same concept we think that, if this same opportunity is given to emerging artists, the number of content published on Spotify will increase and listen to it. In fact, in addition to the artist, it will also benefit the user experience who will have new music to listen to and "discover"..

ARCHITECTURE

In this project, we will follow the Lambda Architecture to implement a a recommender system by using several open-source software. This is a consolidated architectural pattern in the Big Data field described by Nathan Marz, known for being one of the main contributors to the Apache Storm open source project. This pattern implements a generic methodology capable of providing a modular, fault tolerant and real-time organization of its Big Data processing system.
This architecture provides a general approach for applying a function, then a processing, on an arbitrary dataset obtaining the result with a low latency. No single tool provides the ability to dynamically process arbitrary data streams in real time. For this it is necessary to use a group of technologies divided into separate modules in which each module deals with a specific role.
At the speed layers, we use Apache Kafka and Apache Spark Streaming for real-time data processing. At the batch layer, we use Apache Spark and several Apache Spark library, spark SQL lets us query structured data inside Spark programs, using either SQL or a familiar DataFrame API, while we use MLlib for KMeans clustering. We'll send our final data frame to the recommender system, which using a content-based filter will create our

playlist. Everything will be displayed on streamlit and the playlist created will be shown in the user's spotify account.

DATA SOURCES

For this project we used 2 types of different data sources:

- every_fresh_find.csv : a public playlist of Spotify, which contains 4471 songs of emerging artist from all around the world. Each song has an ID, name, artist and features associated with the song, like: Instrumentals, Liveness, Acoustics, Valence, Danceability, Energy.

- A random song retrieved from our Spotify liked song playlist, with the same features of the songs in every_fresh_find.csv, in order to make the filter content based.

STREAMING DATA USING KAFKA PRODUCER

Kafka is a distributed streaming platform and a publish-subscribe messaging system. It is a platform used for collecting and delivering large volumes of data, the actual time it is captured. Kafka is scalable, durable, reliable and fast. It is used to send or ingest data over a cluster. Once the data is captured from the sensors, Kafka transmits the data to the system and also sends the data over the cluster for processing. Apache Kafka is a publish-subscribe messaging platform implemented as a distributed commit log, suitable for both offline and online message streaming. Kafka is a solution to the real-time problems of any software solution to deal with real-time data and route it to multiple consumers quickly.

ZOOKEEPER

Zookeeper is a top-level software developed by Apache that acts as a centralized service and is used to maintain naming and configuration data and to provide flexible and robust synchronization within distributed systems. Zookeeper keeps track of status of the Kafka cluster nodes and it also keeps track of Kafka topics, partitions etc.

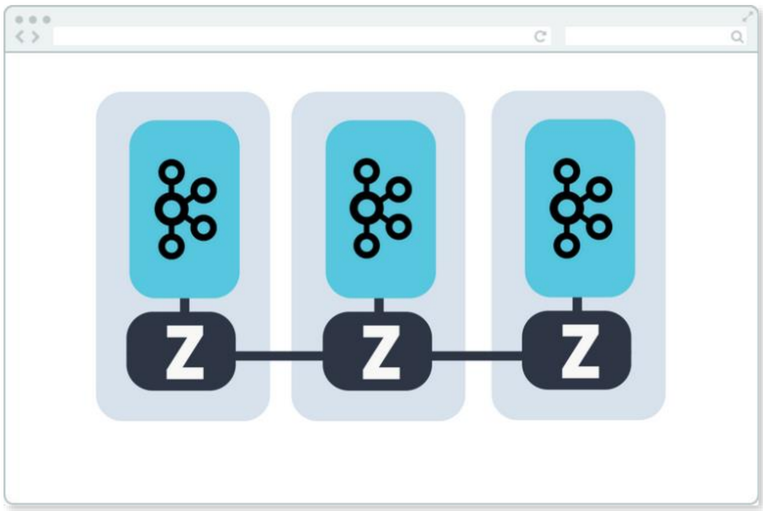


Image describes one Kafka cluster and one Zookeeper in three different servers, and shows how the Zookeeper's collaborate with each other.

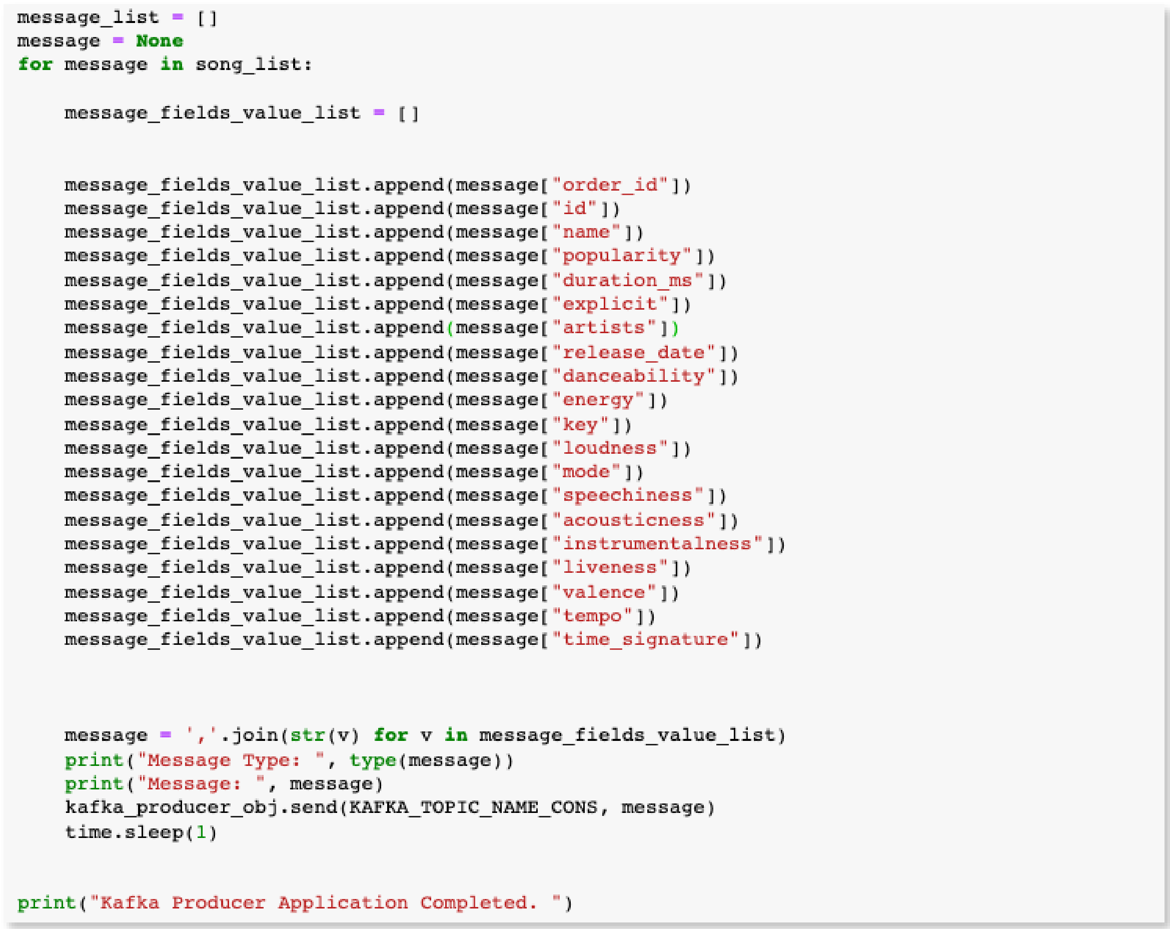
As said before, all Kafka records are organized into topics. Producer applications write data to topics and consumer applications read from topics. The first step, after starting zookeeper, is to create the topic for our application.

```
$ kafka-topics --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic test
```

Initially, we have a CSV file which contains all the data of our song. We use this CSV and put it in the same directory where the Kafka producer code will run.Now let’s create a kafkaProducer object.

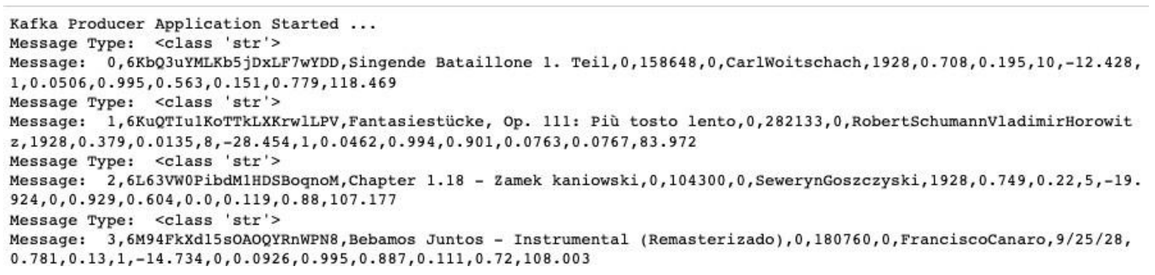
```
KAFKA_TOPIC_NAME_CONS = "test"
KAFKA_BOOTSTRAP_SERVERS_CONS = 'localhost:9092'

if __name__ == "__main__":
    print("Kafka Producer Application Started ... ")

    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       api_version=(0,11,5),
                                       value_serializer=lambda x: x.encode('utf-8'))
```

Then with the following code we get the CSV data into the stream. This is Kafka's producer code where data can be streamed as it is sent to a topic we specify. In the consumer code, we can retrieve this data with the argument.



```
message_list = []
message = None
for message in song_list:

    message_fields_value_list = []


    message_fields_value_list.append(message["order_id"])
    message_fields_value_list.append(message["id"])
    message_fields_value_list.append(message["name"])
    message_fields_value_list.append(message["popularity"])
    message_fields_value_list.append(message["duration_ms"])
    message_fields_value_list.append(message["explicit"])
    message_fields_value_list.append(message["artists"])
    message_fields_value_list.append(message["release_date"])
    message_fields_value_list.append(message["danceability"])
    message_fields_value_list.append(message["energy"])
    message_fields_value_list.append(message["key"])
    message_fields_value_list.append(message["loudness"])
    message_fields_value_list.append(message["mode"])
    message_fields_value_list.append(message["speechiness"])
    message_fields_value_list.append(message["acousticness"])
    message_fields_value_list.append(message["instrumentalness"])
    message_fields_value_list.append(message["liveness"])
    message_fields_value_list.append(message["valence"])
    message_fields_value_list.append(message["tempo"])
    message_fields_value_list.append(message["time_signature"])



    message = ','.join(str(v) for v in message_fields_value_list)
    print("Message Type: ", type(message))
    print("Message: ", message)
    kafka_producer_obj.send(KAFKA_TOPIC_NAME_CONS, message)
    time.sleep(1)


print("Kafka Producer Application Completed. ")
```

We will see an output like this if kafka starts transmitting data

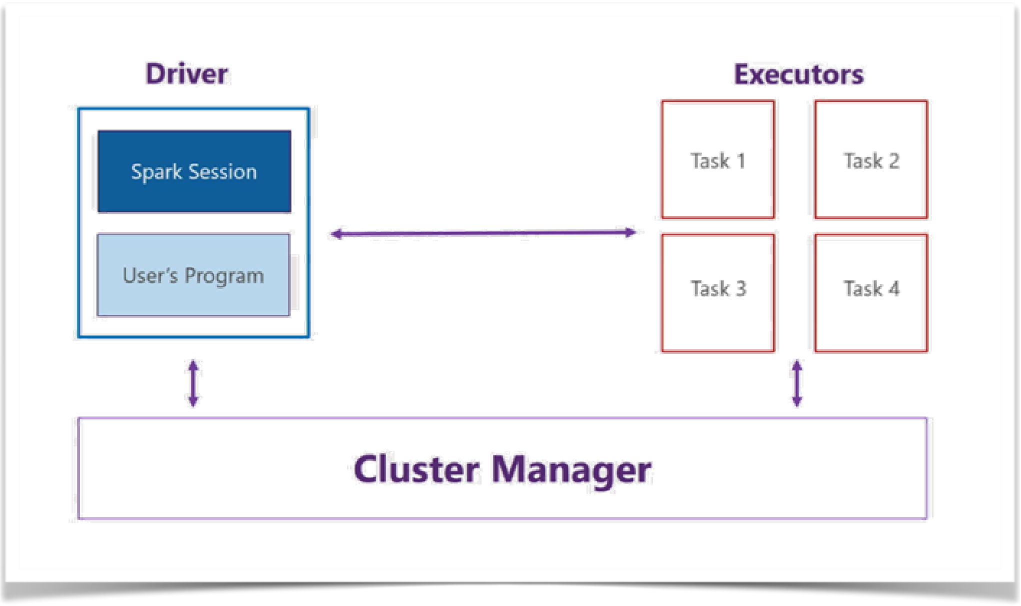


```
Kafka Producer Application Started ...
Message Type:  <class 'str'>
Message:  0,6KbQ3uYMLKb5jDxLF7wYDD,Singende Bataillone 1. Teil,0,158648,0,CarlWoitschach,1928,0.708,0.195,10,-12.428,
1,0.0506,0.995,0.563,0.151,0.779,118.469
Message Type:  <class 'str'>
Message:  1,6KuQTIu1KoTTkLXKrwlLPV,Fantasiestücke, Op. 111: Più tosto lento,0,282133,0,RobertSchumannVladimirHorowit
z,1928,0.379,0.0135,8,-28.454,1,0.0462,0.994,0.901,0.0763,0.0767,83.972
Message Type:  <class 'str'>
Message:  2,6L63VW0PibdM1HDSBoqnoM,Chapter 1.18 - Zamek kaniowski,0,104300,0,SewerynGoszczyski,1928,0.749,0.22,5,-19.
924,0,0.929,0.604,0.0,0.119,0.88,107.177
Message Type:  <class 'str'>
Message:  3,6M94FkXd15sOAOQYRnWPN8,Bebamos Juntos - Instrumental (Remasterizado),0,180760,0,FranciscoCanaro,9/25/28,
0.781,0.13,1,-14.734,0,0.0926,0.995,0.887,0.111,0.72,108.003
```

On the customer side, we used Spark technology to retrieve the streaming data from the producer and process it in parallel.

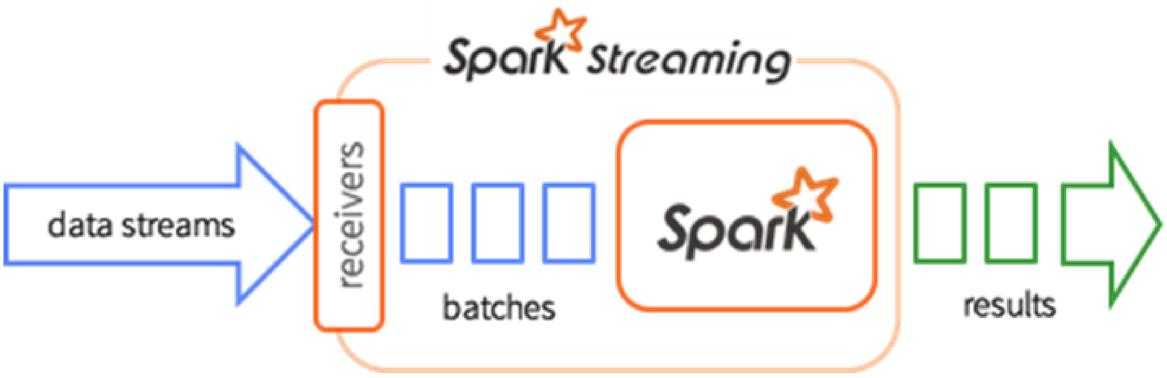
SPARK TECHNOLOGY

Apache Spark is an open source project for large scale distributed computations. Spark has several engines. Spark SQL for doing batch processing, Spark Streaming for real-time processing, Mllib which contains several machine learning algorithms and utilities for advanced analytical problems, and GraphX for graph computations.
In order to process the streaming data, we initially built a Spark Session. The Spark Session is the entry point for Spark applications. As we can see in the figure, a Spark Architecture consists of a central coordinator called driver, many distributed workers called executors and a Cluster Manager. The driver comprises the Spark session which accepts the program and divides it into smaller tasks managed by the executors. Each executor, or work node, receives a task from the driver and performs that task. The executors are located in an entity known as a cluster. Then the Cluster Manager communicates with both the driver and executors to: Manage resource allocation, Manage program division, Manage program execution.



SPARK STREAMING

After initializing Spark Session, we used the Spark Streaming engine to retrieve the data from the producer. Spark Streaming allows streams through a fast, scalable, fault-tolerant process. Internally, it works as follows. Spark Streaming receives live input data streams (also called input DStream) and divides the data into micro-batches, which are then processed by the Spark engine as a series of small-batch jobs with low latency, to generate final results in batches. In particular, when data arrives, the receivers divide it into partitions of RDDs (Resilient Distributed Dataset) which is Spark's abstraction of an immutable, distributed dataset that contains data from a certain time interval.



Specifically, we used the function *spark.readStream* that reads the live data streams from the topic that we created in Kafka and put it in a value column of a streaming DataFrame. Values are binary format, so they are converted into String and we also added a timestamp to each row as data arrives.

SPARK SQL

Next, we created the schema for the streaming data. We created it in such a way that it matches the data coming from the producer. The schema is created with Spark SQL.

Spark SQL is Apache Spark's module for working with structured data. Spark SQL allows to query structured data inside Spark programs, using either SQL or DataFrame API.
We first used the function *createOrReplaceTempView* in order to create an SQL View so that the streaming data can be put into a temporary View and then we applied *spark.sql* command to run a SQL query on the table and to register it on a dataframe. We also used the *writeStream* function to write the dataframe to memory. Specifically, we set a processing time of 5 seconds in append mode to get all the data incoming from the producer.

Spotify API

Next, we used Spotify API to retrieve a random song from our Spotify favorite songs. The Spotify Web API allows to retrieve JSON metadata about music artists, albums, and tracks, directly from the Spotify Data Catalogue. Moreover, it provides access to user related data, like playlists and music that the user saves in the *Your Music* library. Such access is enabled through selective authorization, by the user.
To implement that, we used *Spotipy* python library and we set the credentials necessary to get access to all music data provided by the Spotify platform (a Client ID and Client Secret). In order to access our personal data, we also set a redirect URI from which retrieving an access token for the user authorization. We retrieved data about id, name, artists, duration_ms, popularity and integrated them with audio features of acousticness, speechiness, key, liveness, instrumentalness, energy, tempo, loudness, danceability, and valence. We saved the collected data into a pandas dataframe. From the data collected by Spotify API, we selected a random song which we converted into spark Dataframe in order to combine it with the songs previously collected in Spark. We also selected the features

extracted from spark data collection and those coming from the Spotify API in such a way that they matched appropriately.

SPARK’S MACHINE LEARNING LIBRARY (MLlib)

Built on top of Spark, MLlib is a scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.
Spark MLLib integrates with other Spark components such as Spark SQL, Spark Streaming, and DataFrames, it allows for preprocessing, munging, training of models, and making predictions at scale on data.
It is also possible to use models trained in MLlib to make predictions in Structured Streaming.

FEATURE ENGINEERING

After retrieving our favourite song from Spotify API, our next step was to delete all undesired columns from our Spark dataframe, such as “order ID” or “artist ID”. Then, we appended our favourite song data to the dataframe using Sparks Union operation.

At this point, it was important to represent all the sound features (danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness , valence, tempo) in a single column, and for that purpose we utilized VectorAssembler function from pyspark.ml.feature library. After normalizing the feature vector through StandardScaler, we attached it with the other columns of the dataframe.

K-MEANS CLUSTERING

In consideration of the fact that a possible scenario for our system would be to have very large data incoming, we have decided to perform K-Means Clustering to our data, so that the recommendation is performed on a clustered dataset which is created in correspondence of the cluster where our favourite song falls in.
To do that, we again used MLLib and specifically pyspark.ml.clustering library for the K-means and pyspark.ml.evaluation for the Clustering Evaluator.

RECOMMENDATION SYSTEM

For what concerns the recommendation system, we defined a function that first gets details of our favourite song and then drops the data of that song in order to avoid any effect those information could have on the recommendation.
Next, we considered all the numerical columns of our dataset and calculated Manhattan distances between the favourite song and the streaming data for each numerical feature.
We appended and added those values to a new column called "distances" in our dataset and then we sorted our data to be ascending by the distance feature.
The result of this function is a dataset that contains songs similar to our favourite song's numerical values and thus will be recommended.
Before calling the function that we have just illustrated, we have converted the data frame into a Pandas type for ease of operation.
In addition, we can specify the number of songs we want to obtain as a recommendation by passing that number as a parameter.
Finally, we saved the recommended songs dataset into a .csv file, which will be accessed for visualization in the Streamlit application.

VISUALIZATION

Streamlit is an open-source app framework that makes it possible to easily turn data scripts into shareable dashboards and web apps.
We have decided to first visualize relevant information about the logged user Spotify account, such as “Latest added Song” and “Top 10 Artists”, to better understand personal tastes of the user.
To obtain those information we used Spotipy library and accessed the Spotify API data.
Next, we displayed a list of the recommended songs and plotted a graph of the sound features of those songs along with our favourite song characteristics. Through this graph it is possible to observe all possible similarity in the features.

SPOTIFY APP

Finally, to achieve a sense of completeness in our project work, we have decided to directly upload the newly created list songs into a playlist on our personal Spotify account.
To do that, we again used Spotify API functionalities to access our account, create a playlist, named as we desire, and add the recommended songs by extracting the song ids from the .csv dataset that we previously saved.
In that way, it is possible to verify the accuracy of our model in the most comfortable way: just by opening Spotify app and listening to the suggested songs!