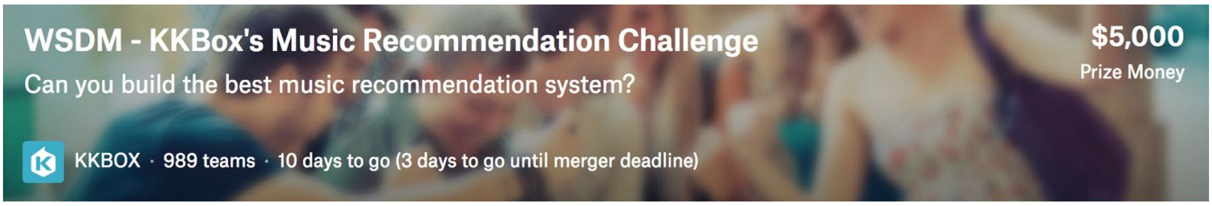
KKBox Repeated Listen Prediction

Cheng Kok Hang (20193070) @samwalker505 Kwok Hin Kwan (20188664) @billykwok
Cheung Tsz Him (20199455) @jamestszhim Shin Wai Ching (20191104) @MarShin

Background

Data analytics and machine learning have been extensively used by software companies to obtain business insights. One important use of it is recommendation system, which allows applications to actively suggest items that users are likely to be interested in. In the space of music streaming services, Spotify is pioneering song-user matching technologies using its hybrid approach to predict users' preference. KKBox, as the major competitor of Spotify in Asia, is trying to catch up and announced a data science challenge in Kaggle.



WSDM - KKBox's Music Recommendation Challenge

Can you build the best music recommendation system?

KKBOX · 989 teams · 10 days to go (3 days to go until merger deadline)

$5,000
Prize Money

Objective

The objective of our project is to take on the challenge - given a song that a KKBox user has listened once, predict whether he/she would listen to the song repeatedly in the coming month. Our predictions will be benchmarked against participants worldwide in Kaggle.

The rest of this report will discuss the dataset, data preprocessing and model training in details, as well as evaluate and discuss the results given by different approaches.

Dataset

The training set contains 7,377,418 examples, associated with 34,379 users and 2,296,833 songs. These three sources of raw data are stored in three separate .csv files.

Training Example
train.csv

- User Id
- Song Id
- Scenario
  - App Tab
  - App Screen Name
  - Detection Type
- Whether listen repeatedly

User Features
users.csv

- User Id
- City
- Age
- Gender
- Registration Date
- Expiration Date

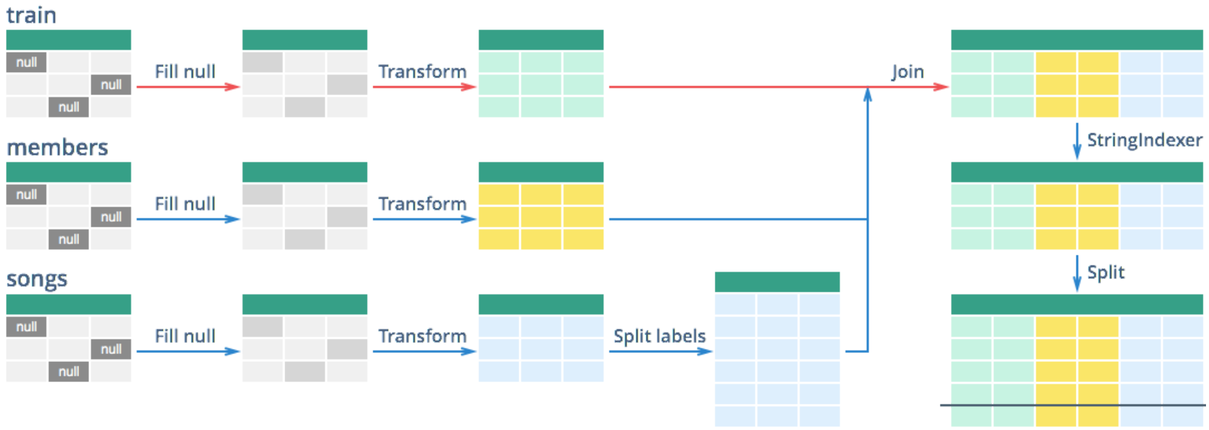
Song Features
songs.csv

- Song Id
- Song Length
- Genres
- Artist
- Composer
- Lyricist
- Language

Since the dataset contains so many records, without cloud computing, it can take days to train the model with such giant dataset. Luckily, Apache Spark and its MLlib library provide a perfect solution to handling massive dataset and demonstrate the power of distributed computing in data analytic tasks.

Data Preprocessing

Real-life datasets typically contain significant noise, corruption and inconsistency. To make unbiased and accurate predictions, we spent a huge amount of effort on data cleansing and transformation to make it follows an organized structure that fits the MLlib requirement and well demonstrate the use of Spark. The overall workflow is shown below.



Throughout the process, we found great opportunities to apply Spark's functionalities and realized how handy Spark is in handling massive data preprocessing problems.

1. Filling null values

Around 30% of the data in KKBox dataset contain null value in at least one field. To handle the problem, we fill the null fields with estimated values obtained by looking up the most similar record to the current one. For example, if the lyricist and composer fields of a song are null, they will be filled by the most popular lyricist and composer of the artist.

Original Records

Song Id	Song Length	Genre	Artist	Composer	Lyricist	Language
1	34189	Folk, Pop	Taylor Swift	Max Martin	Taylor Swift	English
2	41124	Folk	Taylor Swift	null	null	English

Filled Records

Song Id	Song Length	Genre	Artist	Composer	Lyricist	Language
1	34189	Folk, Pop	Taylor Swift	Max Martin	Taylor Swift	English
2	41124	Folk	Taylor Swift	Max Martin	Taylor Swift	English

2. Data transformation

Two types of data transformations have been done.

a) Trivial value replacement

We have discovered unusual values in the dataset, for example, negative "language" label and zero "age". To solve this problem, "0" age entries are replaced by the average age of the entire population. Also, we found that songs with negative language label are all purely instrumental music and should also be interpreted as a type of language. Therefore, we increase the integer label of all songs by two to remove the negative sign.

b) Scale linearization

Some features cannot be directly interpreted by the machine learning model. For example, the registration date of users is a string in “yyyymmdd” format, which cannot be compared among different value. Thus, we map it to the number of days passed from registration to make it comparable and interpretable by the model. For example, if today is 8 Dec, 2017, the string “20171207” will be mapped to 1 and “20161208” will be mapped to 365.

3. Splitting multiple labels

It is quite common that a song is sung by multiple artists, written by multiple composers and lyricized by multiple lyricists. Thus, if we leave it in the dataset, the model will be less robust because a slightly different combination of tags is treated as a completely different label. To tackle this, we split songs with multiple labels into multiple records and feed them separately into the model. The result is the cartesian product of the set of tuples of labels.

Original Record

Song Id	Song Length	Genre	Artist	Composer	...
3	24589	Folk, Pop	Taylor Swift	Max Martin, Taylor Swift	...

Splitted Record

Song Id	Song Length	Genre	Artist	Composer	...
3	24589	Folk	Taylor Swift	Max Martin	...
3	24589	Folk	Taylor Swift	Taylor Swift	...
3	24589	Pop	Taylor Swift	Max Martin	...
3	24589	Pop	Taylor Swift	Taylor Swift	...

4. Joining multiple data sources

After cleaning all three data sources, we perform join operation to merge all three tables. This is can easily done by loading the three tables as `DataFrame` and calling

```
trainDF.join(memberDF, “member_id”, “inner”).join(songDF, “song_id”, “inner”)
```

5. String Indexing

Noticed that almost all of the features are in string format, which cannot be consumed by MLlib. We mapped all string fields into monotonically increasing, non-negative integer indexes. This transformation is done by the `StringIndexer` module and designed to be a `Pipeline` operations applied to both training and test set to make it more scalable.

Original Record

Song Id	Song Length	Genre	Artist	Composer	Lyricist	...
4	24589	Rock	Linkin Park	Kevin Webb	unknown	...

String-indexed Record

Song Id	Song Length	Genre	Artist	Composer	Lyricist	...
4	24589	214	153	652	174	...

Model Training

After preprocessing, we split our training data into 80% training set and 20% evaluation set. Then, we feed our training set to four different models. They are Random Forest, Gradient Boost Tree, Naive Bayesian Classifier and Collaborative Filtering. Features are mapped to a `Vector` class and together with the labels forming the `LabeledPoint` class, required by the data mining models on MLlib.

Platform and Architecture

To support heavy data transformation on large dataset and to allow us to focus more on the logical part of the application, we adopted the Databricks Community Edition, a data analytics platform based on AWS Cloud infrastructure, to serve the Jupyter notebook, which run the python Spark codes.

Data files are uploaded to Databricks File System (DBFS), which is a layer on top of Amazon S3 bucket. This allows persistence data storage even if the cluster is destroyed with the 2 hours limit. Preprocessed intermediate results and trained models are also stored in DBFS for faster computation during run time, so that the redundant pre-processing operations and model training need not be executed every time.

Results

We have established different models for predicting repeated listen for test dataset. All models generate over 50% accuracy which fulfilled our expected delivery. In fact, the accuracy of Random Forest and Gradient Boost Tree can be inferred to pass the benchmark set by Kaggle competition (60%). The detail breakdown is listed as follow.

Metric \ Model	Random Forest	**Gradient Boost Tree**	Naive Bayesian Classifier	Collaborative Filtering*
Accuracy	62.521%	**62.849%**	55.982%	/
Class 0 Precision	49.425%	51.334%	**52.901%**	/
Class 0 Recall	**64.592%**	64.453%	26.774%	/
Class 0 F1 Score	56.000%	**57.150%**	35.555%	/
Class 1 Precision	**74.691%**	73.553%	56.900%	/
Class 1 Recall	61.255%	61.802%	**80.219%**	/
Class 1 F1 Score	**67.309%**	67.167%	66.576%	/
Computation Time	149 s	254 s	**107 s**	> 2 hours

*Collaborative filtering requires the user and song in test set also existed in training set. This involve more complicated split of data. The long training time also limited the testing.

Discussion and Findings

1. Data Source

The performance of our models passes the benchmark. However, we still see much room for improvement.

First, we notice that most features of the data provided by KKBox are categorical values, for example, there are 27,672 artists, 27,310 lyricists, making the distance between data points sparse and may affect the model performance. Metric of comparing distance between artist and lyricists may be developed to mitigate the problem.

Besides, we can expand our feature set with third-party data sources, especially extra information about songs like popularity and lyrics. Also, more robust missing value filling algorithm should be introduced to improve the generalization of the model.

2. Model

From our evaluation, we found that different models have their edges in predicting different labels. It is interesting to investigate further if aggregating the predictions from the three models can produce a better result. We found that running data mining model using MLlib is fast. We expected spark perform significantly faster than native local python program in training the dataset, however, empirical result shows that it is just slightly faster. We believe the data set we are using is still too small for Spark to show its competitive edge.

In addition, we found that the capability of MLlib models is worse than some popular local data mining package like Scikit Learn. In particular, when we test the model using smaller dataset with data imbalance problem, MLlib models did not offer much help.

3. Platform and Architecture

In this project we used Databricks as the cloud computation platform. It requires almost no setup or configuration, however, at the expense of limited flexibility. For example, we cannot run our model more than 2 hours and it is not convenient to share data with each other since the sharing is not available in Community Edition.

Moreover, we cannot control the computing resources of each instance like cluster size, CPU core, etc. Some operations are expected to be run in shorter time in an even more parallel environment. It can be a better idea to launch our customized Spark instance directly on AWS machines for more flexibility.

Data and Source Code Download

The dataset can be obtained from the following webpage.
https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data

Our source code for this project are shown below.

Purpose	Link
Process train	https://goo.gl/rELCqH
Process member	https://goo.gl/Tjxobj
Process song	https://goo.gl/z6YaYS
Process test	https://goo.gl/e3bU1Y

Purpose	Link
Split songs	https://goo.gl/3Lhydj
Aggregate data	https://goo.gl/tL3a8Q
Train and predict	https://goo.gl/Cb9cwb