EECS E6893
Big Data Analytics
Group 18

Real-time Music Recommendation System

Presenter:
Zhejian Jin, Yixing Wang, Zhiqing Wang

Project phase:
Final Report, 12/22/2021

Contents

- Data
- Two-stage recommendation system
- Solution architecture
- System evaluation

Data Overview

- Offline music pool:

500 random songs from Spotify

name	artist	dancibility	energy	key	loudness	speechiness	instrumentalness	liveness	valence
Stuck with U (wit	Ariana Grande	0.597	0.45	8	-6.658	0.0418	0	0.382	
Can I Call You To	Dayglow	0.641	0.842	9	-7.27	0.0292	0.91	0.419	
Feel Good Inc.	Gorillaz	0.818	0.705	6	-6.679	0.177	0.00233	0.613	
Way 2 Sexy (with	Drake	0.803	0.597	11	-6.035	0.141	4.50E-06	0.323	
Have Mercy	Chlöe	0.903	0.535	9	-6.434	0.0742	0	0.345	
Congratulations	Post Malone	0.63	0.804	6	-4.183	0.0363	0	0.253	
MONTERO (Call	Lil Nas X	0.593	0.503	8	-6.725	0.22	0	0.405	
Christmas Tree F	Taylor Swift	0.599	0.681	7	-4.5	0.0331	0	0.324	
I Saw Mommy Ki	The Jackson 5	0.583	0.712	3	-8.222	0.0538	1.02E-06	0.817	
Jealous	Eyedress	0.474	0.91	9	-10.431	0.0462	0.859	0.476	
MONTERO (Call	Lil Nas X	0.593	0.503	8	-6.725	0.22	0	0.405	
Happy Xmas (Wa	John Lennon	0.321	0.64	2	-10.023	0.0324	0	0.718	
Way 2 Sexy (with	Drake	0.803	0.597	11	-6.035	0.141	4.50E-06	0.323	
Break from Toron	PARTYNEXTDO	0.596	0.678	9	-5.18	0.0335	0.004	0.418	
Don't Stop Believ	Journey	0.5	0.748	4	-9.072	0.0363	0	0.447	
abcdefu	GAYLE	0.695	0.54	4	-5.692	0.0493	0	0.367	

- Current users behavior:

Liked tracks with music features

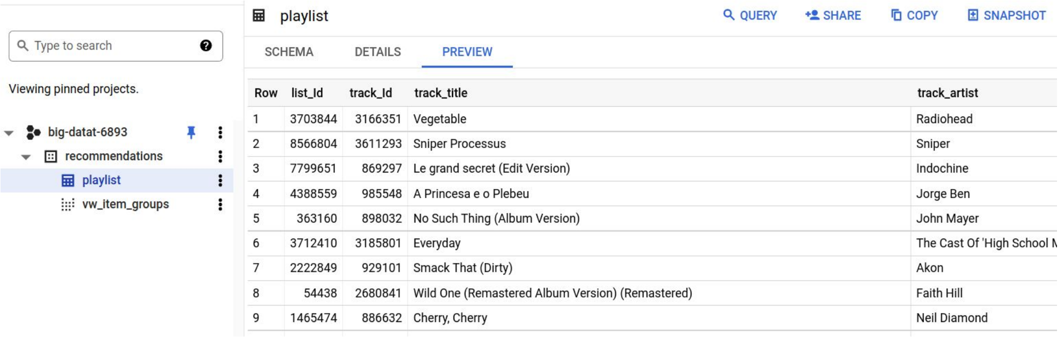
Followed artists with music genre

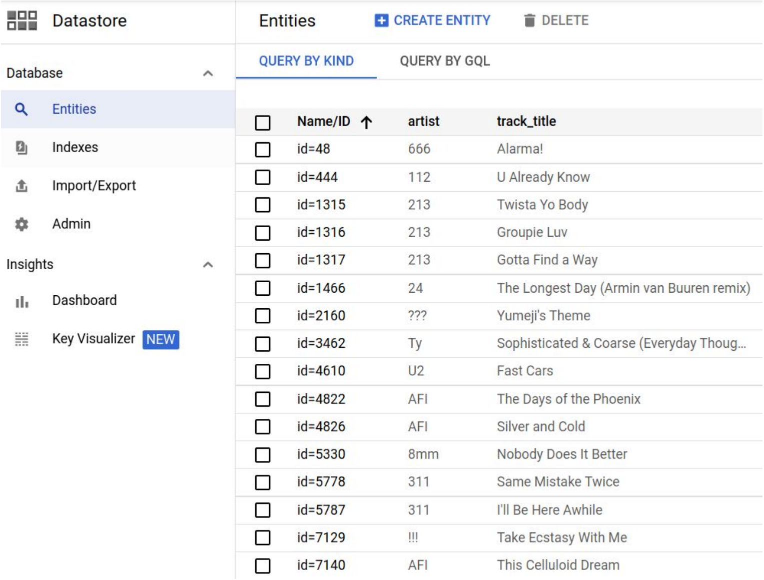
Listening history with music features

id	name	artists	duration_s	popularity	added_at	acousticness
3USxtqRwSYz57	Heat Waves	Glass Animals	238.805	88	2021-11-15T16:4	0.44
5HCyWlXZPP0y	STAY (with Justi	The Kid LAROI	141.805	98	2021-11-15T16:4	0.0383
00Blm7zeNqgYL	One Right Now (	Post Malone	193.506	90	2021-11-15T16:4	0.0361
0gplL1WMoJ6iYa	Easy On Me	Adele	224.694	100	2021-11-15T16:4	0.578
6zSpb8dQRaw0	Cold Heart - PNA	Elton John	202.735	95	2021-11-15T16:4	0.034
6bQfNiqyCX7Ua	Shivers	Ed Sheeran	207.853	64	2021-11-15T16:4	0.281
2dLLR6qlu5UJ5	Royals	Lorde	190.185	76	2021-09-11T03:1	0.121
3tCwjWLicbjsMC	Rude	MAGIC!	224.773	67	2021-09-10T04:2	0.0435
6OtClsQZ64Vs1	Good Life	OneRepublic	253.306	72	2021-09-10T03:5	0.0771
7soJgKhQTO8hl	One Call Away	Charlie Puth	194.453	66	2021-09-10T03:2	0.403
4iJyoBOLtHqaG	Peaches (feat. D	Justin Bieber	198.081	91	2021-09-02T23:3	0.321

Real-time item matching

1. Import the playlist table to BigQuery.
2. Create the vw_item_groups view that contains the item data used to compute item co-occurrence.
3. Export song title and artist information to Datastore to make it available for lookup when making similar song recommendations.

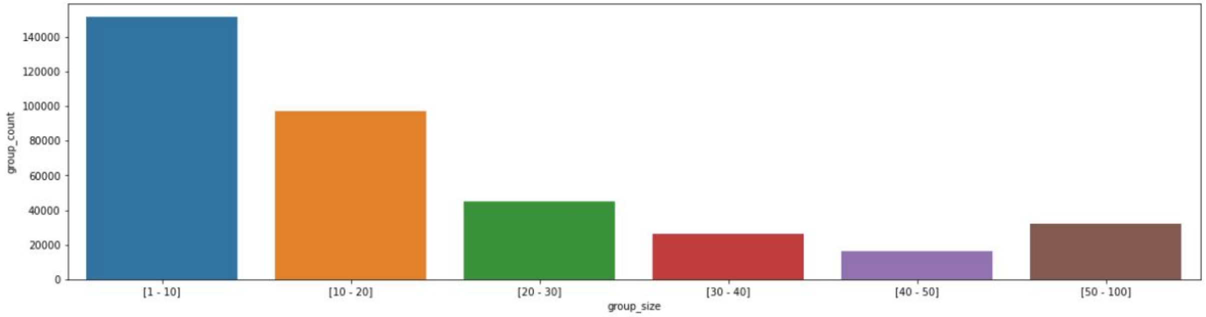




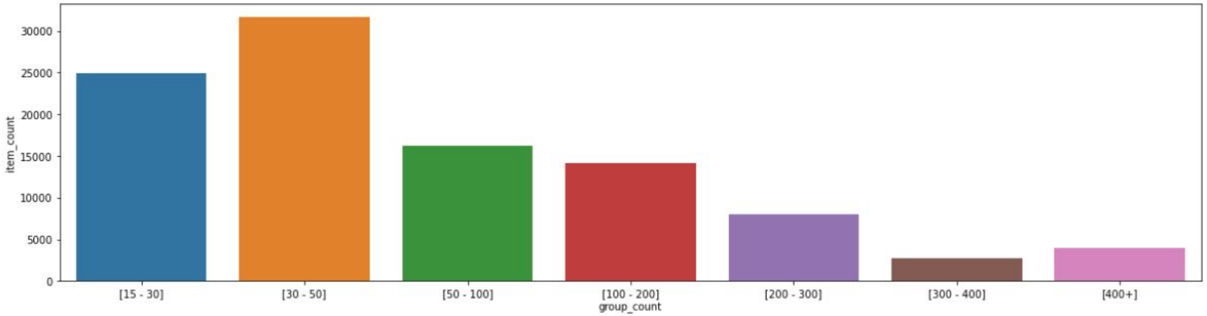
DataSet Exploration

dataset: `bigquery-samples.playlists`

playlist size distribution:



song occurrence distribution:



Methods overview

- Clustering the music pool
    a. Different users prefer one or several types of songs, we need to classify them and make predictions
    b. Finding the best K is important
- Turning implicit features(liked songs, feature of songs) into explicit ratings
    a. Use K-means to get the types, and get rating by considering the relative ratio of a type of songs to all songs
    b. If type A songs makes up a much more proportion of A’s liked songs than the music pool, we can conclude that the user may have a higher rating for it
- Using ALS to get the recall results:
    a. ALS is a kind of matrix factorization method which can recommend both for the item and user

Clustering: recommend for different types

raw data

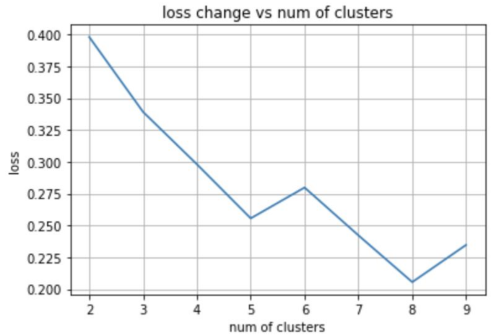
numeric values

energy	key	loudness	speechiness	instrumentalness	liveness	valence	tempo
0.783	6	-4.023	0.062	7.96E-6	0.119	0.775	172.041
0.431	10	-8.81	0.0578	1.42E-5	0.106	0.137	143.875
0.671	0	-5.077	0.0337	2.54E-6	0.0921	0.699	110.054
0.612	2	-7.222	0.112	5.7E-6	0.37	0.178	180.917
0.424	5	-9.579	0.324	0.0	0.0834	0.153	145.887
0.521	10	-9.461	0.0329	0.149	0.123	0.337	85.012
0.501	1	-6.339	0.0592	0.0	0.114	0.119	83.5
0.806	11	-8.262	0.0886	0.0542	0.0703	0.637	124.927
0.597	11	-6.035	0.141	4.5E-6	0.323	0.331	136.008
0.662	1	-6.903	0.292	0.0	0.534	0.389	112.511

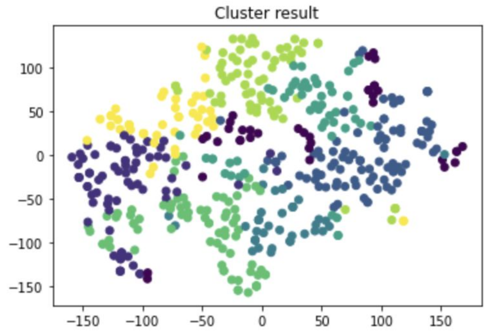
normalize combine

features
[0.82905982905982...
[0.44823109380071...
[0.70788704965920...
[0.64405496051065...
[0.44065779508817...
[0.54560207724764...
[0.52396408092610...
[0.85394352482960...
[0.62782646326950...
[0.69814995131450...

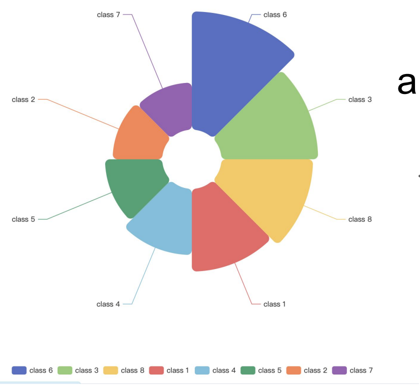
K means: find best K



clustering result

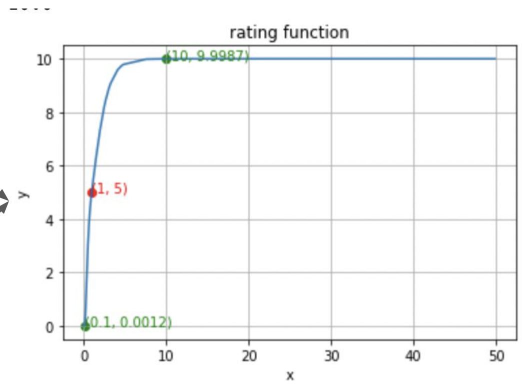


Rating: get rating according to the relative ratio

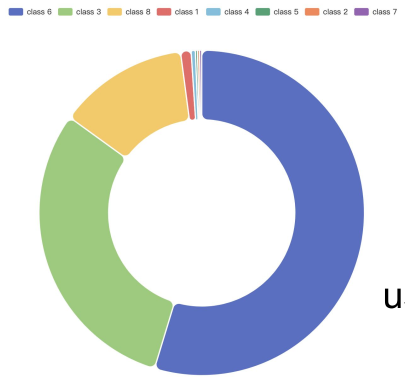


all data distribution

give rating considering the relative ratio



prediction	rating	userid	target
Row(prediction=0)	5.021052507	0	1
Row(prediction=2)	9.979441337	0	1
Row(prediction=2)	1.128520011	0	1
Row(prediction=4)	1.128520011	0	1
Row(prediction=2)	3.622152942	0	1
Row(prediction=2)	9.979441337	0	1
Row(prediction=7)	5.021052507	0	1
Row(prediction=6)	3.817198528	0	1
Row(prediction=2)	8.899590206	0	1
Row(prediction=0)	9.979441337	0	1



rating function maps the ratio into [0,10] and also maintains: $x + 1/x = 10$

user data distribution

Recall: get the recommendation

only needs user, item and rating

popularity	id_num	name_num
75	5.0	5.0
70	2.0	1.0
69	4.0	0.0
77	0.0	2.0
75	1.0	4.0
79	6.0	3.0
72	3.0	6.0

ALS model

implicitPrefs = True

evaluator = ‘rmse’

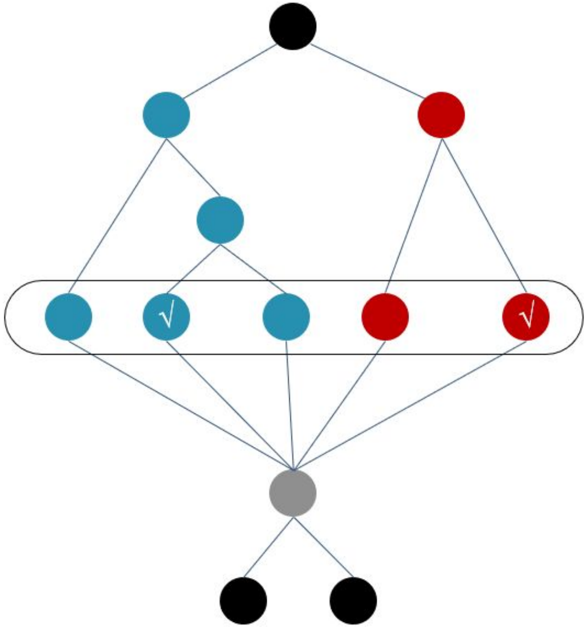
```python
def alsRecall(df):
    (training, test) = df.randomSplit([0.8, 0.2])
    # rating is inferred from other signals, set implicitPrefs to True to get better results
    als = ALS(maxIter=15, regParam=0.01,implicitPrefs=True,userCol="userid", itemCol="songid", ratingCol="rating",
            coldStartStrategy="drop")
    model = als.fit(training)
    predictions = model.transform(test)
    evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                    predictionCol="prediction")
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))

    # Generate top 10 song recommendations for each user
    userRecs = model.recommendForAllUsers(10)
    # Generate top 10 user recommendations for each song
    songRecs = model.recommendForAllItems(3)

    return userRecs, songRecs
```

recall result for user and item

songid	recommendations
0	[{4, 1.0620573}, ...
1	[{5, 0.99533343},...
2	[{2, 1.0019244}, ...
3	[{8, 1.0007138}, ...
4	[{2, 1.0523099}, ...
5	[{5, 1.0485523}, ...
6	[{9, 1.0863023}, ...
7	[{7, 1.0497539}, ...
8	[{9, 0.9993777}, ...
9	[{2, 0.99854606},...
10	[{6, 0.9884141}, ...

userid	recommendations
0	[{13, 1.0887454},...
1	[{8, 1.1075128}, ...
2	[{4, 1.0469949}, ...
3	[{17, 1.0808561},...
4	[{0, 1.0625092}, ...
5	[{20, 1.0492959},...
6	[{1, 1.0471842}, ...
7	[{91, 1.0537599},...
8	[{24, 1.0051345},...
9	[{6, 1.0894992}, ...

Sorting: get a more accurate result



GBDT Classifier

0 1 0 0 1

Logistic Regression

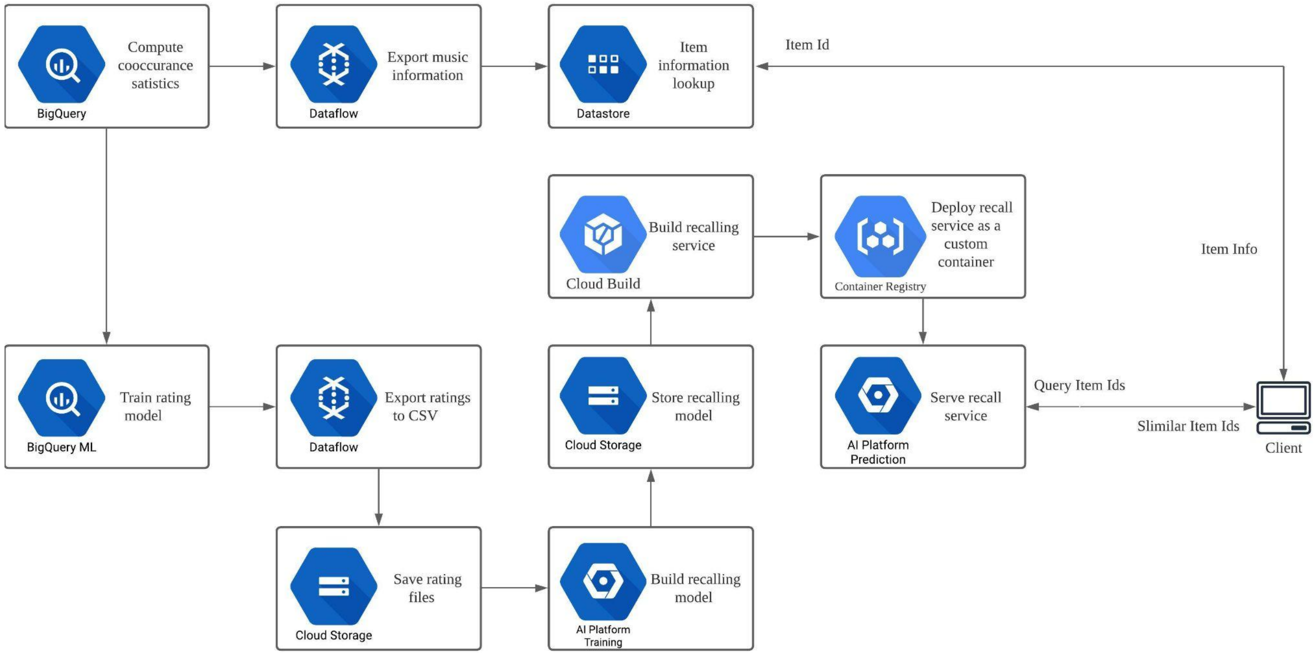
```
[91]    training's binary_logloss: 0.257858
[92]    training's binary_logloss: 0.257542
[93]    training's binary_logloss: 0.257119
[94]    training's binary_logloss: 0.256816
[95]    training's binary_logloss: 0.256407
[96]    training's binary_logloss: 0.25589
[97]    training's binary_logloss: 0.255327
[98]    training's binary_logloss: 0.254804
[99]    training's binary_logloss: 0.254198
[100]   training's binary_logloss: 0.253666
Save model...
Start predicting...
(400, 100)
[[ 1  1  1  1  1  3 11 10  1 12  1  0  2  2  2  1  1  1  1  9  1  1 11 10
   9 10  1  7  6  1  1 11  1 11 11 11  0 11 11 11  1  6  3  0  3  1  1  1
   1  1  1  0  0  1  0  1  2  1  1  1  3  8  6  3  7  6  2  3  3  6  2  1
   2  1  1 11 11 10 11 11 11  2  2  2  0  8  8  8  1  8  6  6  1  6  1  2
   0  0  2  0]
 [ 8  2  4  8  4  8 10  0 10 10 11  4  3  3 11  9  9  9  6  2  3  3  9  4
   6  0  0  8  0  0  5  0  5  0  0  6 13  6  6  5  2  1  2  8  2  5  5  5
   5  6  5 10 11  5  3  3  6  3  3  3  7  0  6  7  0  6 10  6  6  4  6  9
   6  9  9 10 10  9  9  9  8  4  4  4  9  8 11  8  3 11  1  1  2  1  2  6
   3  5  7 10]
```

Use GBDT to find the best combination of features

```
[0.98589794 0.01410206]
[0.98051233 0.01948767]
[0.98314042 0.01685958]
[0.981483   0.018517  ]
[0.97089102 0.02910898]
[0.77317924 0.22682076]
[0.61707597 0.38292403]
```

The best accuracy is 38.2%, which is much higher than the percentage of positive samples in training set

Solution Architecture



Performance Evaluation

Primary consideration for real-time music recommendations

- Latency & Throughput
  - Container Registry: providing scalability that meets throughput requirements
  - Datastore: a quick and highly scalable NoSQL

Load test (Machine type : n1-standard-4 (4 vCPUs, 15 GB RAM))

- Under a load test with 20 concurrent users, this architecture provides recommendations at a median latency of about 40 ms and performs at a throughput of 16 requests per second.

Future improvements:

Use in-memory service such as Redis or Memcached to replace Datastore for faster latency

Reference

1. https://www.analyticsvidhya.com/blog/2021/06/spotify-recommendation-system-using-pyspark-and-kafka-streaming/
2. https://docs.confluent.io/5.5.1/streams/kafka-streams-examples/docs/index.html#prerequisites
3. https://github.com/HanXiaoyang/pyspark-recommendation-demo
4. https://github.com/learn-co-curriculum/als-recommender-system-pyspark-lab

Q & A