SMAI PROJECT REPORT

APRIL 29, 2019

AYUSH KUMAR DWIVEDI (2018802002)
AMAN KRISHNA (2018201070)
GYANSHU AZAD SINGH (2018201073)
SHIVANI DIXIT (20172109)

Table of Contents

Table of Figures

Exploiting Music Play Sequence for Music Recommendation

Base Paper: Zhiyong Cheng and Jialie Shen, Lei Zhu, Mohan Kankanhalli, Liqiang Nie, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)

Introduction:

Users leave digital footprints when interacting with various music streaming services. Music play sequence, which contains rich information about personal music preference and song similarity, has been largely ignored in previous music recommender systems. In this paper, author explored the effects of music play sequence (MPS) on developing effective personalized music recommender systems. As one of the most effective recommendation techniques, matrix factorization (MF) aims to learn latent vectors of users and items by decomposing the user-item rating/interaction matrix, which records user-item ratings or interaction times. While MF based approaches have achieved very promising performance, they generally ignore the importance of interaction sequence. In this study, the main objective is to investigate the potential of exploiting MPS in music recommendation. Specifically, paper attempt to leverage the information encoded in MPS into MF methods to improve the recommendation performance.

Motivation:

Music play sequence, which contains rich information about personal music preference and song similarity, has been largely ignored in previous music recommender systems.

Problem Statement:

Incorporate the effect of Music Play Sequence (MPS) along with Matrix Factorization (MF) methods to generate more relevant music recommendations.

Proposed Solution:

Paper proposes to use word embedding techniques in music play sequences to estimate the similarity between songs. The learned similarity is then embedded into matrix factorization to boost the latent feature learning and discovery. Furthermore, the proposed method only considers the k-nearest songs (e.g. K= 5) in the learning process and thus avoids the increase of time complexity.

Overview:

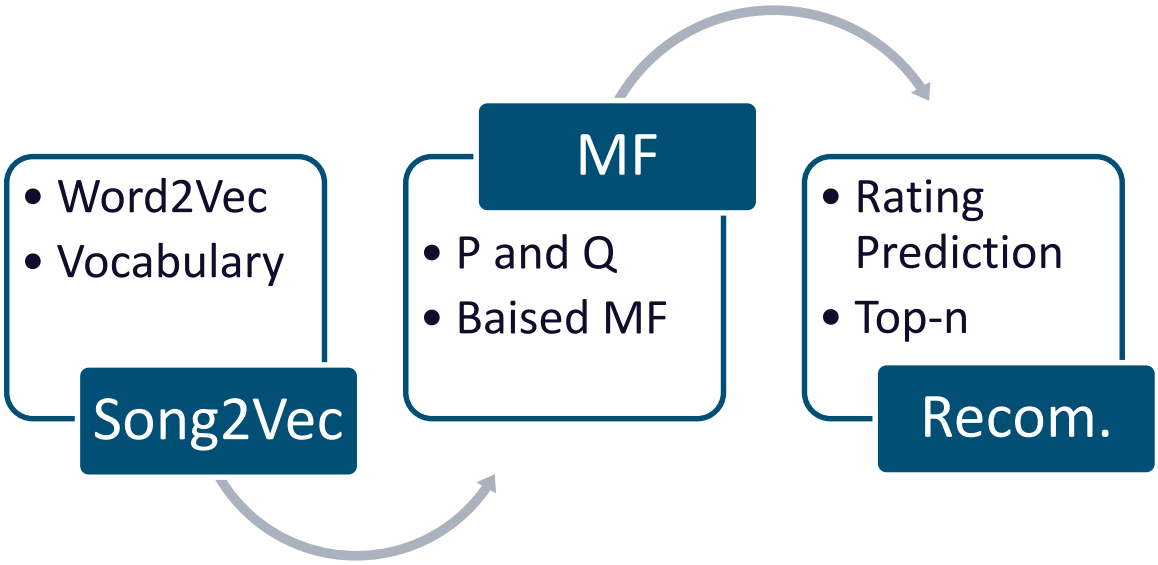

Figure 1: Flow diagram of the complete process

Modules Involved:

The following are the key modules involved in the implementation of the project in chronological order of requirements:

1. Data Processing

Users leave a lot of listening behavior data online when interacting with social music streaming websites. Large scale of play sequences not only record user's music.

We are using Last.fm dataset, which has the following characteristics:

- # of data points : 4752899
- # of unique songs : 72678
- # of unique users : 249

```
user_000001     2009-05-03T15:10:18Z    463a94f1-2713-40b1-9c88-dcc9c0170cae    Minus 8 4e78efc4-e545-47af-9617-05ff816
d86e2   Elysian Fields

user_000001     2009-05-03T15:04:31Z    ad0811ea-e213-451d-b22f-fa1a7f9e0226    Beanfield       fb51d2c4-cc69-4128-92f5
-77ec38d66859   Planetary Deadlock

user_000001     2009-05-03T14:56:25Z    309e2dfc-678e-4d09-a7a4-8eab9525b669    Dj Linus        4277434f-e3c2-41ae-9ce3
-23fd157f9347   Good Morning Love Coffee Is Ready

user_000001     2009-05-03T14:50:51Z    6f3d4a7b-45b2-4c08-9306-8d271e92cb4f    Alif Tree       1151b040-8022-4965-96d2
-8a4605ce456c   Deadly Species

user_000001     2009-05-03T14:46:29Z    463a94f1-2713-40b1-9c88-dcc9c0170cae    Minus 8 f78c95a8-9256-4757-9a9f-213df5c
6854e   Cold Fusion

user_000001     2009-05-03T14:39:20Z    45bdb5be-ec03-484f-b58d-d22afc944b24    Wei-Chi c4fc8802-d186-4c4d-85cd-d5d063b
935c2   Clouds
```

Figure 2: Snapshot of raw data

The following are the preprocessing steps used before using the data to train the model.

1. Exclude the users only listened to less than 10 songs
2. Exclude the songs which have been played by less than 10 users.
3. Exclude the irrelevant data columns i.e. song id, artist id, artist name

We define a play event to be part of a session if it occurs no later than 800 seconds after the previous user play event. We only keep the listening sessions with no less

than 10 songs. Number of sessions 125181 (with each session having 10 or more songs)

Figure 3: Music Play Sequence (in dictionary)

	UserID	TimeStamp	Song
0	1	2009-05-04T23:08:57Z	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007
1	1	2009-05-04T13:54:10Z	Composition 0919 (Live_2009_4_15)
2	1	2009-05-04T13:52:04Z	Mc2 (Live_2009_4_15)
3	1	2009-05-04T13:42:52Z	Hibari (Live_2009_4_15)
4	1	2009-05-04T13:42:11Z	Mc1 (Live_2009_4_15)

Figure 4: Relevant data extracted

2. Matrix Factorization and Biased Matrix Factorization

Matrix factorization techniques have become a dominant methodology within collaborative filtering (analyzes relationships between users and inter-dependencies among products to identify new user-item associations) recommenders. They offer accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels.

Using MF in paper perspective:

When a user gives feedback to a certain song (in our case) they listened (say they can rate from one to five), this collection of feedback can be represented in a form of a matrix. Each row represents users, while each column represents different songs. The matrix will be sparse since not everyone is going to listen/rate every song. One strength of matrix factorization is the fact that it can incorporate implicit feedback, information that is not directly given but can be derived by analyzing user behavior. Using this strength, we can estimate if a user is going to like a song that (he/she) never listened. And if that estimated rating is high, we can recommend that song to the user.

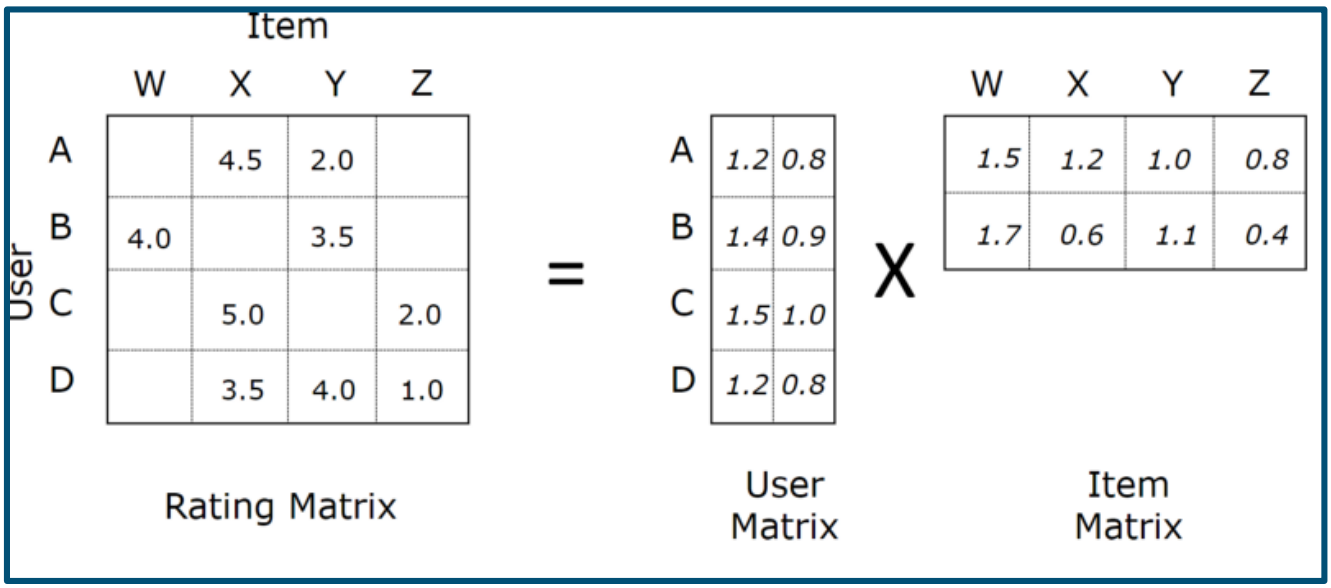

Figure 5: Extracting latent feature matrix

Let there be matrix A with dimensionality of (m,n) this matrix can be viewed as a dot product between two matrix with each matrices having dimensions of (m,k) and (k,n). The concept of matrix factorization can be written mathematically to look something like below.

$$\widehat{r_{ui}} = q_i^T p_u$$

Then we can create an objective function (that we want to minimize) with respect to q and p, which are (m,k) and (k,n) matrices.

$$\min_{q^*,p^*} \sum_{(u,i)\in K} \left(r_{ui} - q_i^T p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

The term on the right is the regularization term, this is added since we do not want our decomposed matrix q and p to over-fit to the original matrix. Since our goal is to generalize the previous ratings in a way that predicts future, unknown ratings, we should not over-fit our model.

Learning Methods:

$$e_{ui} \triangleq r_{ui} - q_i^T p_u$$

One method to find matrix q and p is the gradient descent method. Since we have the loss function defined, take the partial derivative respect to q and p to optimize those values.

$$q_i \leftarrow q_i + \gamma . (e_{ui} . p_u - \lambda . q_i)$$
$$p_u \leftarrow p_u + \gamma . (e_{ui} . q_i - \lambda . p_u)$$

By taking partial derivatives, the update rule would look something like above. But the error surface is not convex, we can also take the alternative approach in which we fix q and p alternatively while optimizing for another.

Biased Matrix Factorization:

Some songs are biased in that it is widely perceived better (or worse) than other songs, and some users may be biased too. These are known as biases or intercepts, and they are independent of any interactions, and it wouldn't be wise to explain these bias terms using our two decomposing matrix q and p. So we include the bias terms into our original equation.

$$\widehat{r_{ui}} = \mu + b_i + b_u + q_i^T p_u$$

And the new objective function would look something like below.

$$\min_{q^*, p^*, b^*} \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda \left(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2 \right)$$

3. Song2Vec

Streaming services have changed the way in which we experience content. While recommendation systems previously focused on presenting you with content you might want to purchase for later consumption, modern streaming platforms have to focus instead on recommending content you can, and will want to, enjoy in the moment. Since any piece of content is immediately accessible, the streaming model enables new methods of discovery in the form of personalized radios or recommendation playlists, in which the focus is now more on generating sequences of similar songs that go well together.

The amount of data generated now a day proves to be an invaluable training set that we can use to teach machine learning models to better understand user tastes, and improve our music recommendations.

Observing global song preferences across all users and applying classic collaborative filtering approaches such as Matrix Factorization on a user-item rating matrix gives us valuable information about how groups of songs could be related. So if a group of users have a large set of common songs they like, we can infer that those are users who have very similar tastes in music, and the songs they listen to are similar to each other.

Those global co-occurrences across multiple users therefore give us valuable information about how songs are related; however, one thing they do not capture is how songs could be locally co-occurring in time. So they might tell us that users who like song A would also probably like song B, but would they have listened to them in the same playlist or radio? We can therefore see the benefit of looking at not just what songs users play across their lifetimes, but at what context they play those songs in. Or in other words, what other songs do they also play before or after them during the same session?

So what we're interested in is a model that can capture not only what songs are similar people generally interested in, but what songs are listened to frequently together in very similar contexts. And this is where Word2vec comes in.

What is Word2vec?

Word2vec is a class of neural network models that were initially introduced for learning word embedding's that are highly useful for Natural Language Processing tasks. In recent years, the technique has also been applied to more general machine learning problems including product recommendations. The neural network takes in a large corpus of text, analyzes it, and for each word in the vocabulary, generates a vector of numbers that represent that word. Those vectors of numbers are what we are after, because as we'll see, they encode important information about the meaning of the word in relation to the context in which it appears.
There are two main models defined: The Continuous Bag-of-Words model and the Skip-gram model. This paper has used the Skip-gram model.

The Word2vec Skip-gram model is a shallow neural network with a single hidden layer that takes in a word as input and tries to predict the context of words around it as output. Let's take the following sentence as an example:

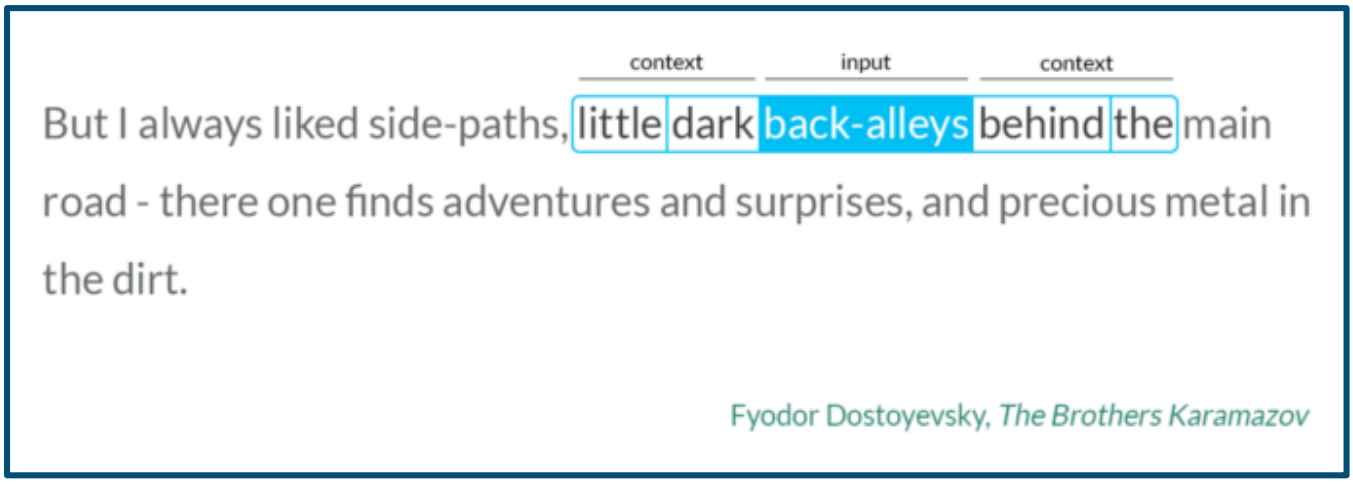

Figure 6: Example sentence for Skip Gram Modelling

In the sentence above, the word 'back-alleys' is our current input word, and the words 'little', 'dark', 'behind' and 'the' are the output words that we would want to predict given that input word. Our neural network looks something like this:

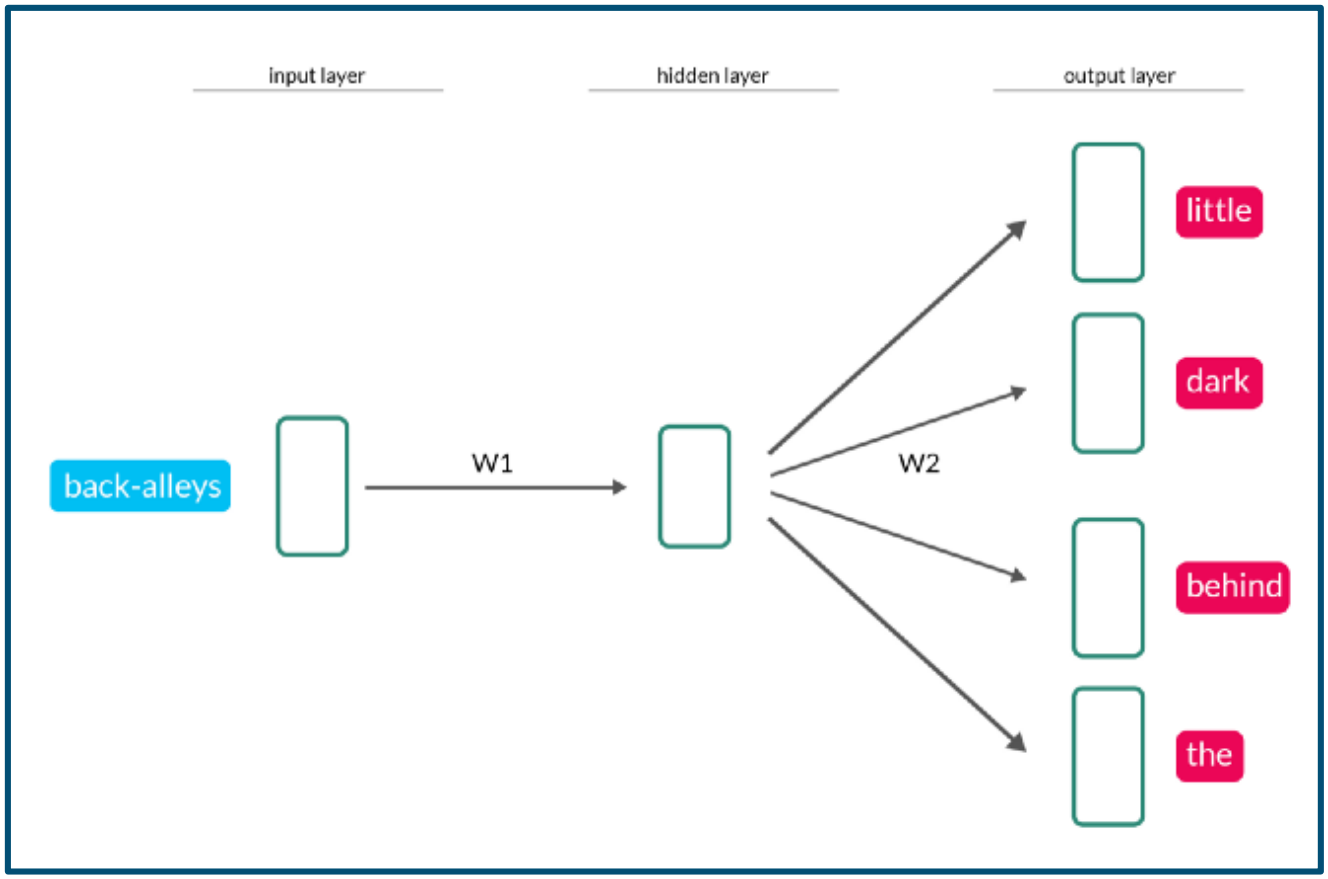

Figure 7: Neural network for Word2Vec

W1 and W2 represent weight matrices that control the weights of the successive transformations we apply on our input to get the output. Training the neural network consists of learning the values of those weight matrices that give us an output that is closest to the training data provided.

Given an input word, we do a first forward propagation pass through the network to get the probability that the output word is the one we expect according to our training data. Since we know with certainty what words we expect at the output, we can measure the error in the prediction, and propagate that error back through the network using backpropagation and adjust the weights through stochastic gradient descent. Through this step, we modify the values of W1 and W2 slightly, so they can more accurately predict the output words we want given that example input word. When we're done with this step, we move our context window to the next word and repeat the above steps again.

We repeat the procedure above for all the sentences in the training set. When we're done, the values of the weight matrices would have converged to the ones that produce the most accurate predictions.

Here's the interesting part: if two different words largely appear in similar contexts, we expect that, given any one of those two words as input, the neural network will output very similar predictions as output. And we have previously mentioned that the values of the weight matrices control the predictions at the output, so if two words appear in similar contexts, we expect the weight matrix values for those two words to be largely similar.

In particular, the weight matrix W1 is a matrix that has as many rows as there are words in our vocabulary, with each row holding the weights associated with a particular word. Therefore, since similar words need to output similar predictions, their rows in the matrix W1 should be similar. The weights associated with each word in this matrix are the 'embedding's' that we are going to use to represent that word.

But how does that relate to music recommendations? Well, we can think of a user's listening queue as a sentence, with each word in that sentence being a song that the user has listened to.

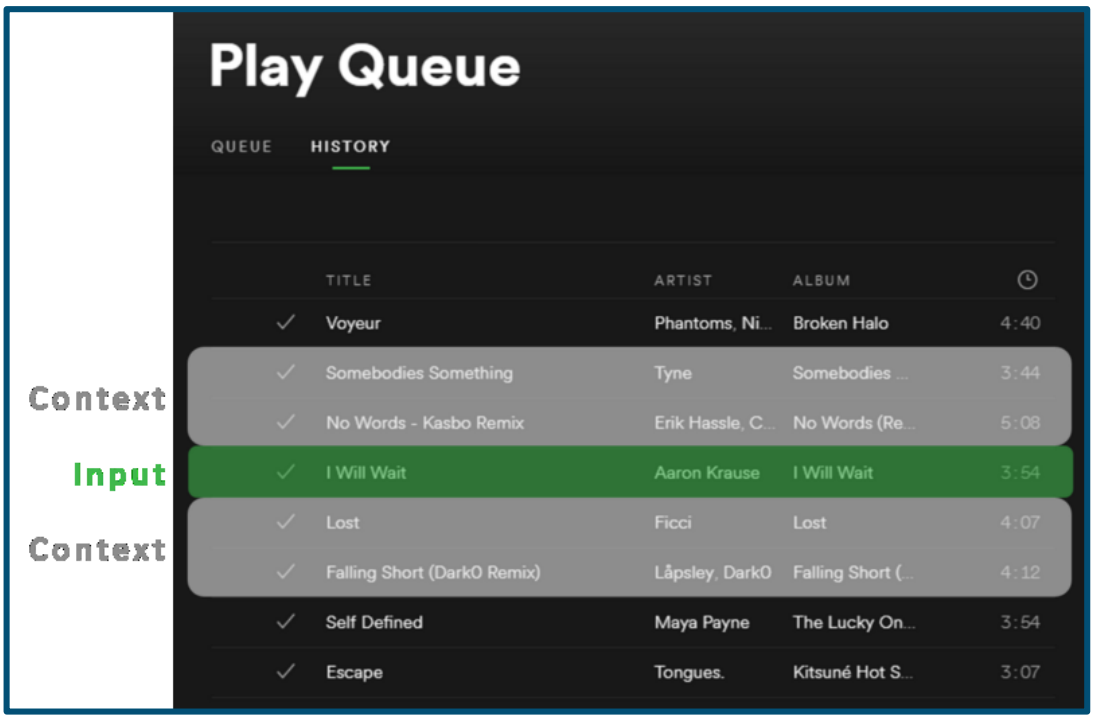

Figure 8: Skip Gram model for Song Sequences

So then, training the Word2vec model on those sentences essentially means that for each song the user has listened to in the past, we're using the songs they have listened to before and after to teach our model that those songs somehow belong to the same context. Here's an idea of what the neural network would look like with songs instead of words:

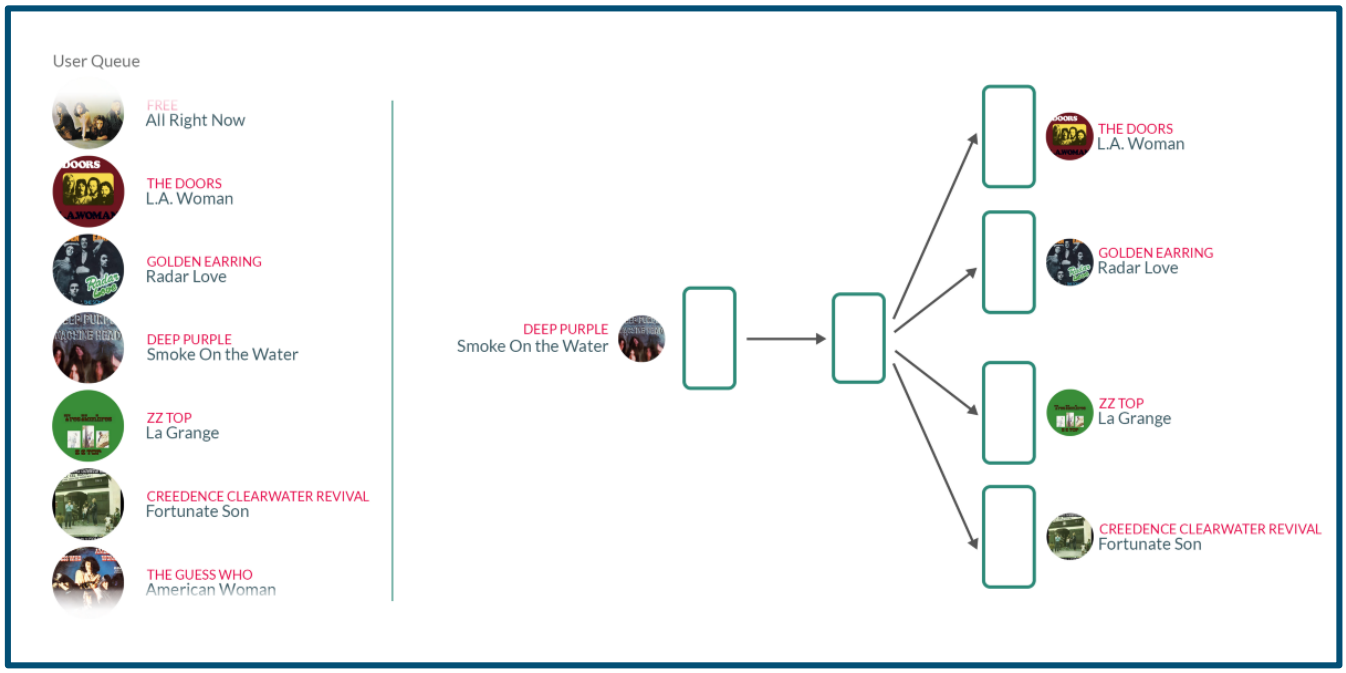

Figure 9: Neural Network for Song2Vec

This is the same approach as the analysis of text discussed above, except instead of textual words we now have a unique identifier for each song. What we get at the end of the training phase is a model where each song is represented by a vector of weights in a high dimensional space. What's interesting about those vectors is that similar songs will have weights that are closer together than songs that are unrelated.

Song Vector Use Cases:

Using Word2vec, we have transformed our problem of finding songs that appear in similar contexts into mathematical numbers that capture those local co-occurrences. We can look at the weights as coordinates in a high dimensional space, with each song being represented by a point in that space. These vectors can be used in a number of

ways, for example, as input features to other machine learning algorithms, but they can also be used on their own to find similar songs.

As we have previously mentioned, the more times two particular songs appear in similar contexts, the closer their coordinates will be. So given a particular seed song, we can find k other similar songs by taking the cosine similarity between the vector of this seed song and all the others while keeping the top k ones that have the highest cosines (which correspond to the lowest angles, and therefore the most similar). For example, the cosine similarity between Lebanese classic singer Fayrouz's Shayef El Baher Shou Kbir and the same singer's Bhebbak Ma Baaref Laych is of 0.98, while the similarity between Shayef El Baher Shou Kbir and Saharna Ya Leil by Elissa — a modern Lebanese pop artist — is only of -0.09. This makes sense, because people who are listening to classics such as Fayrouz's songs are unlikely to alternate them with Elissa's pop music in their queues.

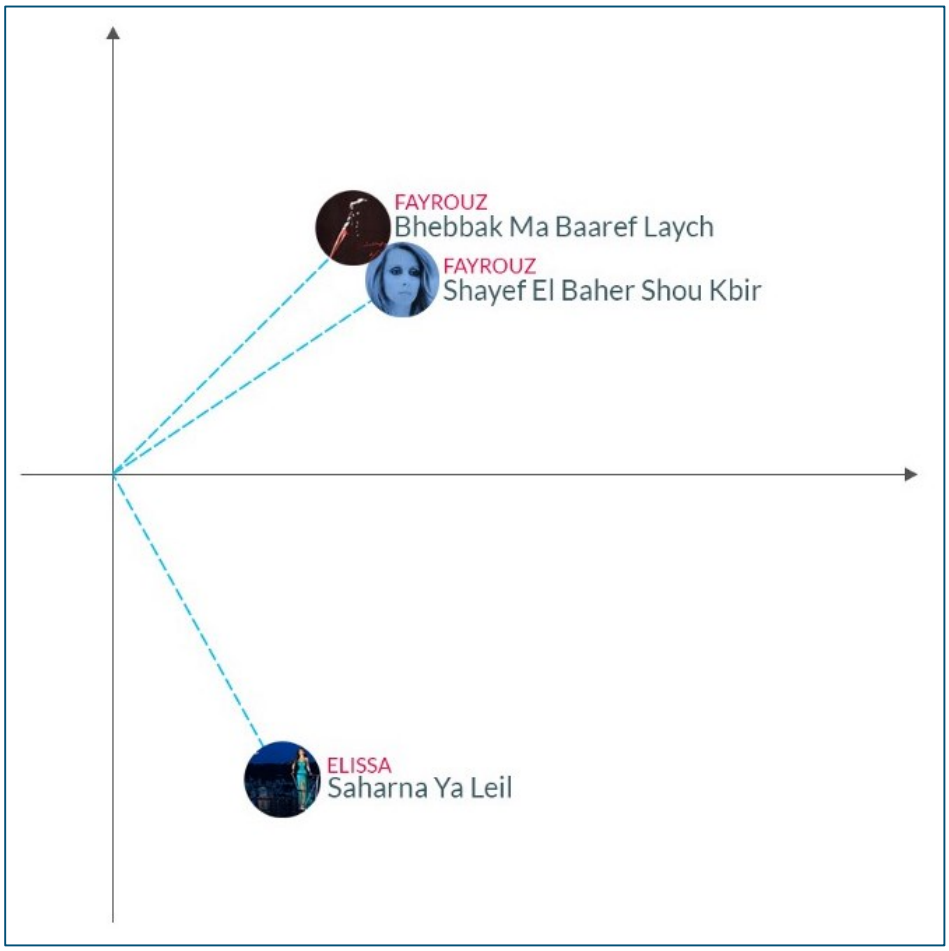

Figure 10: Cosine similarity between Songs

Another interesting way in which we can use the song vectors is to map a user's listening habits onto this space and generate recommendations based on that. Since we're now dealing with vectors, we can use basic arithmetic to add vectors together. Let's take for example a user who has listened to three songs: Keaton Henson's In the Morning, London Grammar's Wasting My Young Years and Daughter's Youth. What we can do is get the vectors corresponding to each of those three songs, and average them out together to find a point that is equidistant from all three songs. What we have essentially done is transform a list of songs that the user has listened to into a vector of coordinates that represents the user in the same vector space in which our songs are located. Now that we have a vector that defines the user, we can use the same technique used previously to find similar songs with points that are close to the user. The following images helps visualize the different steps involved in generating those recommendations:

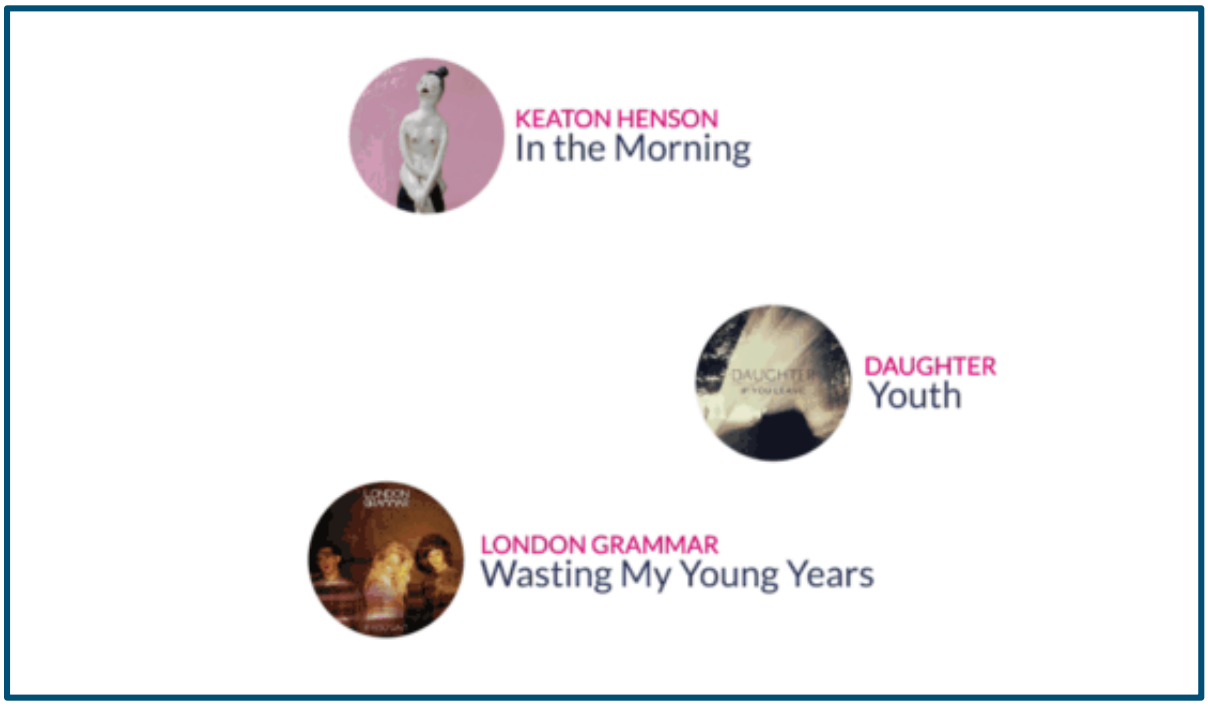

Figure 11: Locate the songs the user has streamed

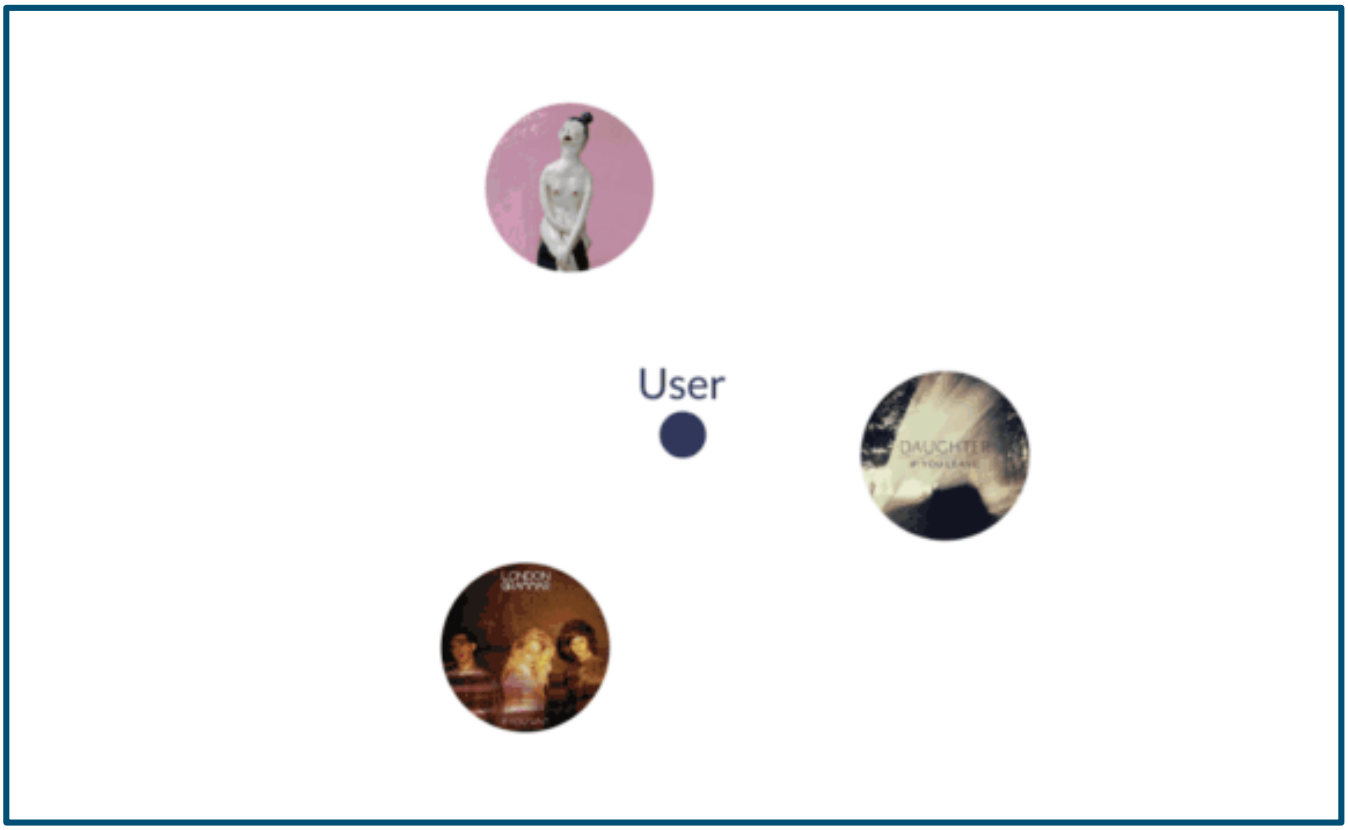

Figure 12: Average their vectors to get a user vector

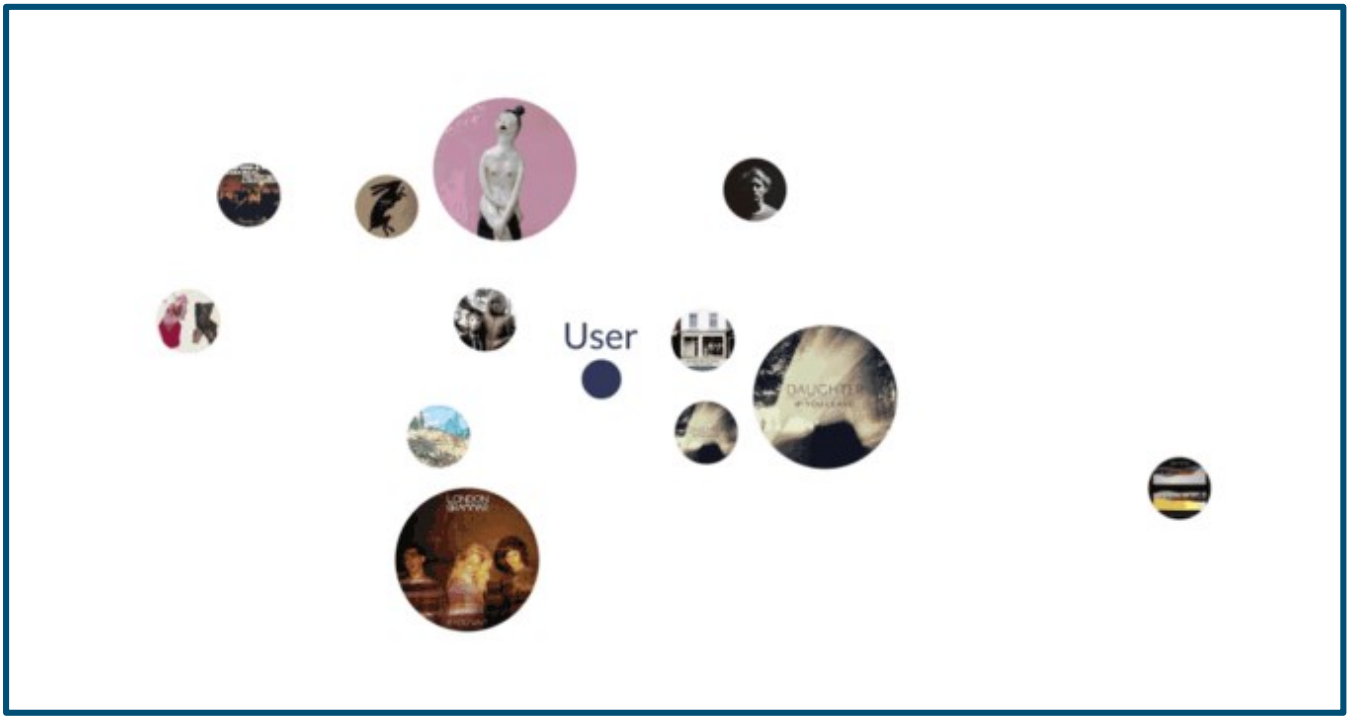

Figure 13: Search for other nearby songs

Figure 14: Keep the songs closest to the user as recommendations

We find songs such as The Head and the Heart's Down in the Valley, Fleet Foxes' Mykonos and Ben Howard's Small Things that are all in the same indie-folk style as our input songs. Remember that we have done all of this not based on a study of the acoustics of the audio, but rather by simply looking at what songs other people have listened to around those particular songs.

4. Experimental Setup

To validate the effects of exploiting music play sequence in recommendation, we evaluate the proposed algorithm over two tasks:

- **Rating prediction** aims to predict the rating $\widehat{r_{ui}}$; of user *u* given to item *i*. It is a standard task to evaluate the performance of recommendation on explicit rating datasets.
- **Top-n recommendation** is to evaluate the accuracy of the top results returned by recommendation methods. Itis often used to evaluate the performance of recommendation methods on implicit datasets

5. Experimental Results

- **Top-n recommendation**
  The following are the experimental results for the top n recommendation based on song2vec module

```
get_song("The Fox")
```

```
Top 6 Songs for: The Fox

Name                          Similarity Score

Start Toge                    0.8167651891708374
Milkshake                     0.7961223125457764
Let'S Call                    0.795258641242981
Leave You                     0.7926040291786194
The End Of                    0.7861700057983398
One Song F                    0.7859086394309998
```

Figure 16: Top 6 recommendation for the song "The Fox"

```
get_song("Bedragaren I Murmansk")
```

```
Top 6 Songs for: Bedragaren I Murmansk

Name                          Similarity Score

Ännu Mera                     0.9780382513999939
Bilder Av                     0.9775434732437134
M Som I                       0.9768289923667908
Sån                 0.9757107496261597
Hur Många                     0.9739065170288086
Trösta Mig                    0.9728430509567261
```

Figure 15: Top 6 recommendation for the song "Bedragaren I Murmansk"

```
get_song("A Little Bit Of Pain")

Top 6 Songs for: A Little Bit Of Pain

Name                                Similarity Score

Make You L                          0.9642292261123657
Ilfracombe                          0.9522489309310913
Autumn'S H                          0.9332128763198853
Silent Tra                          0.9329584836959839
I Would Fo                          0.9248721599578857
Aphrodite                           0.9218252897262573
```

Figure 17: Top 6 recommendation for the "A little Bit Of Pain"

- **Rating prediction**

  The following is the output for RMSE vs d (# of dimensions in latent factor space)

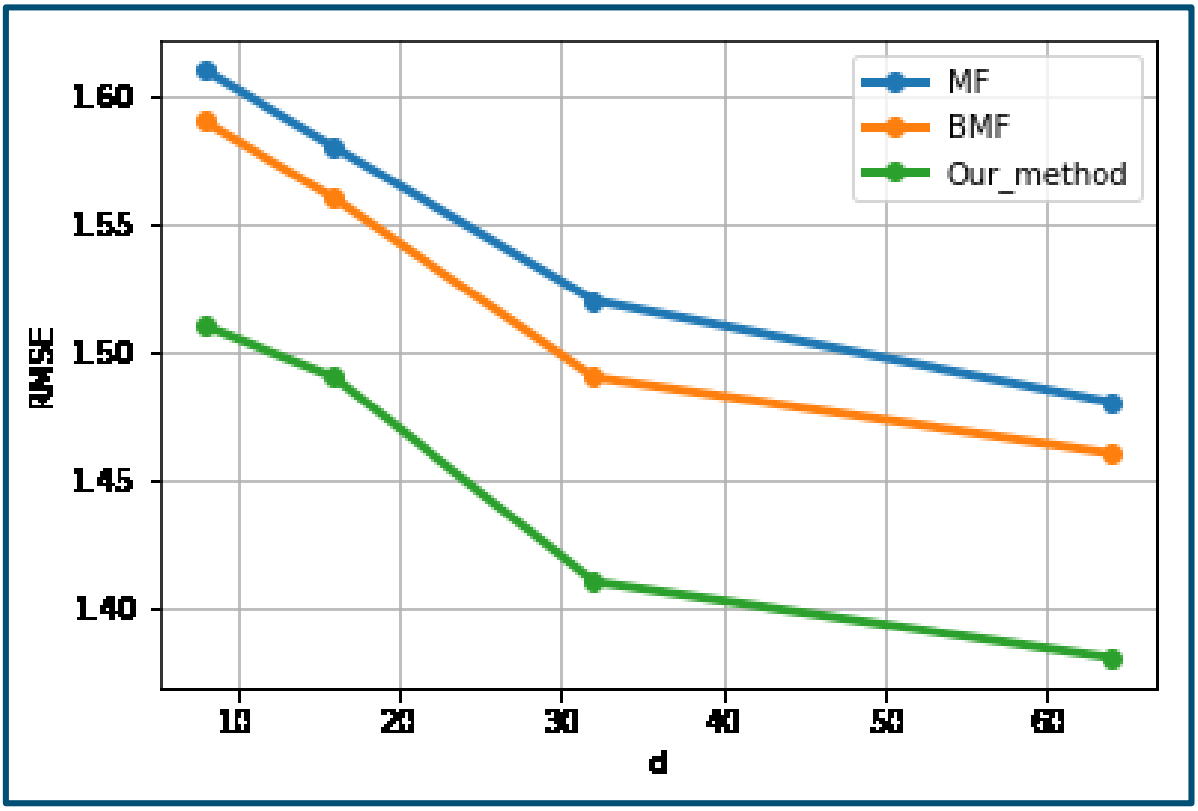

Figure 18: Variation of RMSE vs Dimension of latent feature vector

The following is the output for RMSE vs k (# of nearest songs)

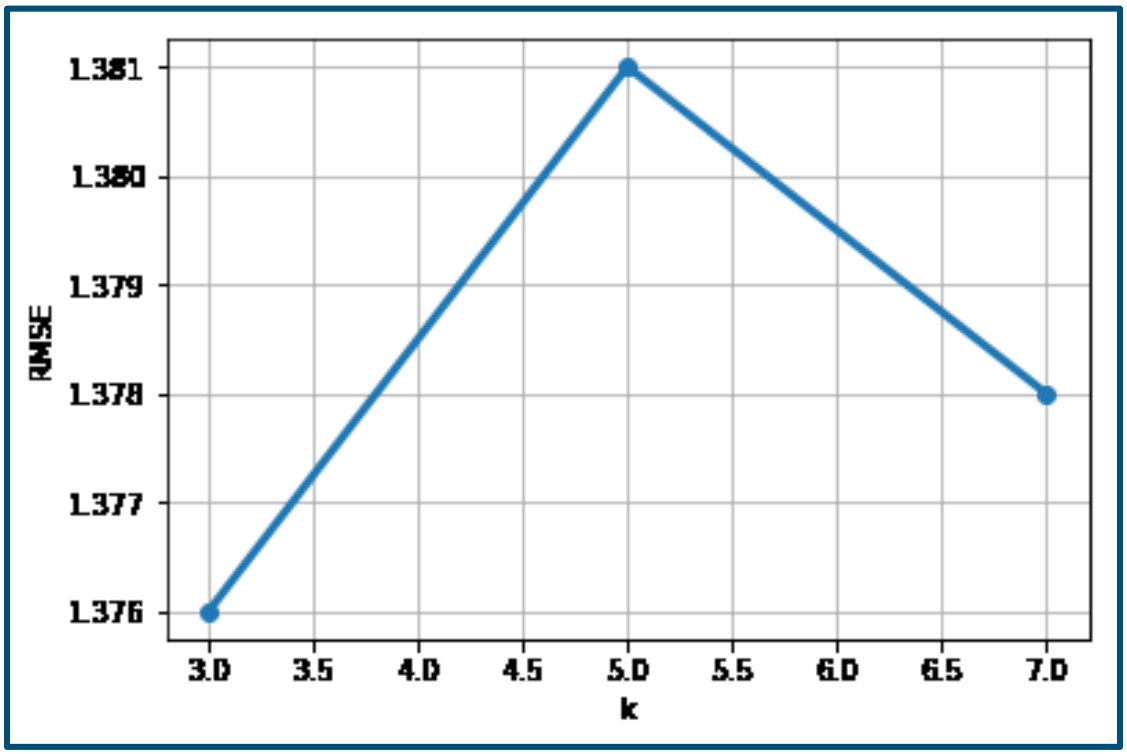

Figure 19: Variation of RMSE vs k (# of nearest neighbors)

6. Future Directions:

- Since the dataset contains artists and artist id's, we can use them to create another method Artist2Vec to generate similarities based on artists.
- We can collect a new dataset which contains more features like demographic information, language and genre which are also relevant. It may also be relevant to collect user specific data, like data of family members, friends etc.

7. References

i. **Song2Vec:**https://towardsdatascience.com/using-word2vec-for-music recommendations-bb9649ac2484

ii. **Matrix factorization:** https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

iii. [KeunhoChoia, DongheeYoob, GunwooKimc, YongmooSuha], *A hybrid online-product recommendation system: Combining implicit rating-based collaborative filtering and sequential pattern analysis*, Electronic Commerce Research and Applications, Volume 11, Issue 4, July–August 2012, Pages 309-317

iv. Song2Vecimplementation: https://github.com/0411tony/Yue/blob/master/recommender/advanced/Song2vec.py

v. Create corpus for Song2Vec: https://github.com/WQtong/MusicRecsys/blob/master/create_song_corpus.ipynb

8. Contributions by the team members

The following are the contributions made by the team members towards successful completion of the project.

- Data Pre-Processing and Cleaning: The initial data preprocessing to reduce the size and to collect the relevant data samples was done by Gyanshu and later on Shivani worked towards extracting relevant features and arranging them into required format. This included, conversion of time stamps to standard format and calculation of time difference and creation of sessions.
- Song2Vec: Ayush primarily worked on complete implementation of Song2Vec. This included creation of a corpus based on music play sequence data and then generating a vocabulary for training a word2vec model customized for songs. Further Ayush and Shivani worked together to obtain top-n recommendation outputs as per the paper. They also created a method which returns the similarity values for k nearest songs to be used in MF cost function.

- Matrix Factorization: Aman primarily worked on implementation of MF and biased MF method which included generation of latent feature vectors learned through gradient dissent and used for generation of rating predictions. Further Gyanshu worked to implement MF including the similarity values from Song2Vec method.

- Documentation was contributed by everyone for the respective part on which the team member worked.