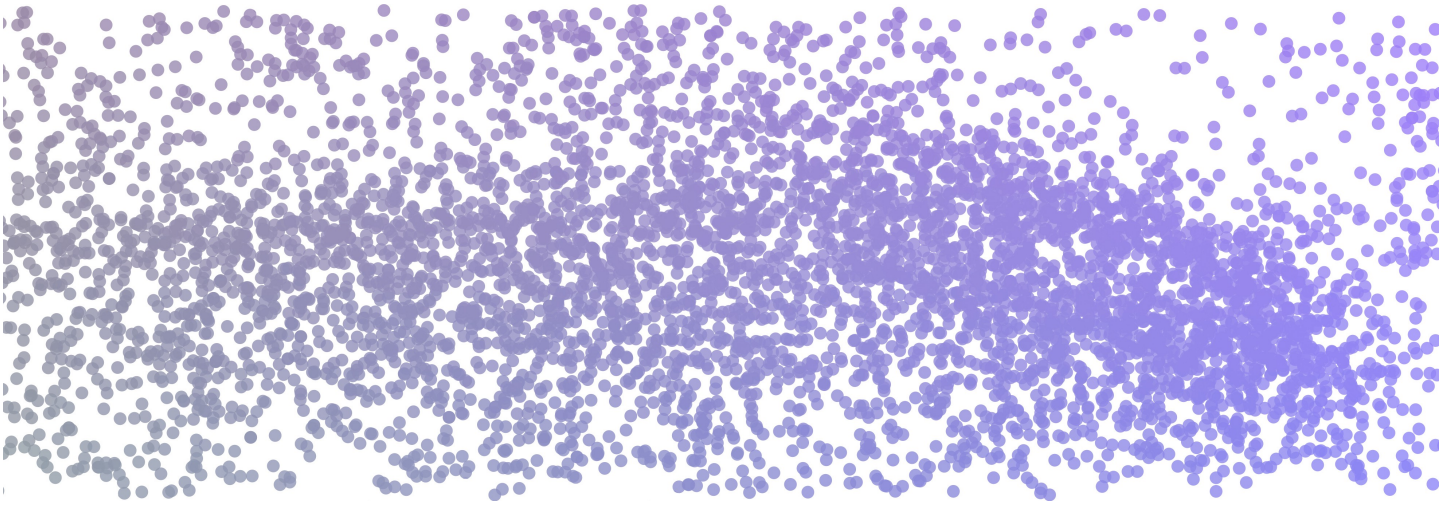
Using Word2vec for Music Recommendations

How we use neural networks to transform billions of streams into better recommendations.

Ramzi Karam
Dec 7, 2017 · 9 min read



Each point represents a song. The closer the points, the more similar the songs are.

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.

With now over 700 million songs streamed every month, **Anghami** is the leading music streaming platform in the MENA region. What this also means, is that the amount of data generated by all those streams 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.

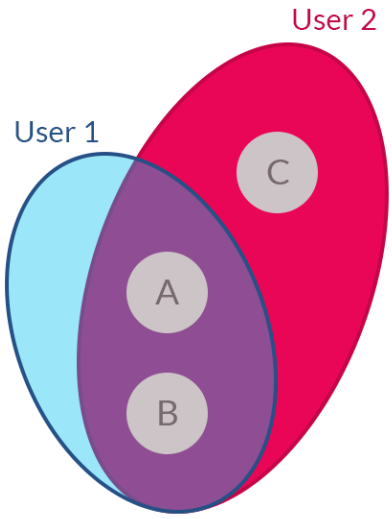
In this article, I will be presenting a neural network approach that we use to extract song embeddings from the wealth of streaming data that we have, and how we use this model to generate relevant recommendations.

Recommendation Systems

Recommendation systems fall under two broad categories:

- **Content-based systems** are recommendation systems that are based on the features of the item we're trying to recommend. When talking about music, this includes for example the genre of the song or how many beats per minute it has.
- **Collaborative Filtering-based systems** are systems that rely on historical usage data to recommend items that other similar users have previously interacted with. These systems are oblivious to the features of the content itself, and base their recommendations on the principle that people who have many songs or artists in common, will generally like the same styles of music.

With enough data, collaborative filtering systems turn out to be effective at recommending relevant items. The basic idea behind collaborative filtering is that if user 1 likes artists A & B, and user 2 likes artists A, B & C, then it is likely that user 1 will also be interested in artist C.



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.

So... What is Word2vec?

Word2vec is a class of neural network models that were initially introduced for learning word embeddings 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. For the rest of this discussion, we will restrict ourselves to the **Skip-gram** model as this is the one that we use.

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:

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:

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 'embeddings' 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. 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:

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. This space is defined by dozens of dimensions, which we cannot easily visualize as humans, but we can use dimensionality reduction techniques such as **t-SNE** to reduce the high dimensional vectors to 2 dimensions, and plot them on a graph:

Each point in the figure above represents a song, and the closer the points are to each other, the more similar the songs are.

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.

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 animation helps visualize the different steps involved in generating those 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.

Conclusion

Word2vec has allowed us to accurately model each song with a vector of coordinates that captures the context in which this song is played in. This allows us to easily identify similar songs, and use vector arithmetic to find a vector that defines each user.

The Word2vec vectors are currently used in production alongside other models for music recommendation features, and mainly music discovery based on a user's listening habits. So next time you discover a good song through recommendations, think about the thousands of people who have played this song before you, and which one of your favorite songs they could have also played next.

• • •

Want to be part of interesting stories like this? We're recruiting!

Machine Learning | Recommender Systems | Data Science | Word2vec | Towards Data Science