Karma Police - Radiohead + Metallica = Where Is My Mind?

Beatriz Borges*, Manon Michel*, Julien Vignoud*
GROUP 2, TRACK 3 - MUSIC
*All authors had equal contribution.
Department of Computer Science, EPFL, Switzerland

PROJECT INFORMATION
Project done in the Spring 2021 semester class of Machine Learning for Behavioral Data (CS-421), at EPFL.

***Abstract*—Motivated by the increasing importance of user recommendation in music streaming platforms, we explore a technique tailored to capture the information encoded in a song listening context, based on existing natural language processing architectures. Such context is defined by the neighboring songs in the listening order and may be used to improve music recommendation and shape user profiles. We find that the context is tightly coupled with the song's artist and determine that popular songs don't necessarily share similar context.**

I. INTRODUCTION

Music displayed a sudden burst of popularity with the ascension of music streaming platforms, such as Spotify [1], Apple Music [2] or Last.fm [3]. Along with the increased number of listens, collecting user data became more accessible due to the nature of the platforms. This new information creates new opportunities to improve user listening experience. Among others, music recommendation or finding user with similar music tastes are examples of how data can be leveraged. In fact, Last.fm provides both through their so-called *Audioscrobbler* system, which also offers a way for the users to see their music listening habits.
A particular aspect of listening habits is the order in which a user listens to music. For instance, one can listen to music according to artist, genre, or without any obvious pattern. Being able to identify this particular listening pattern for each user would yield considerable insights for music recommendation or to establish a user profile and groups of user with similar listening patterns.
In order to capture this ordering, we create song representations as vectors derived from a song's context, i.e., from the preceding and following songs in the chronological order of the listening history. To this mean, we used Word2Vec [4] architectures, the Skip-gram and Continuous Bag-of-Words models. The goal of our research is to assess what information we can extract from a song listening context, such as the artist, the genre, or if the embeddings define a measure of similarity.

II. DATASET

A. Data Description

The dataset [5] represents the full listening history, from 2005 till May 5th 2009, for nearly 1,000 users. This dataset is composed of data collected from Last.fm API [3], using the `user.getRecentTracks()` method. It contains two data tables. The first, data on the user listening history, is comprised of the following tuples : `<user-id, timestamp, artist-mbid, artist-name, song-mbid, song-title>` tuples. The second, data on the users for which we have a listening history. The user data is based on tuples of the form `<userid, gender, age (int|empty), country, signup_date>`.

B. Exploratory Analysis

The dataset contains around 20 millions records of user listens, composed of one million different songs from 200 thousands different artists. Figure 1 shows the distribution of song's occurrences, i.e., the distribution of the number of times songs have been listened to. An enormous majority of songs have very few listens while a few of them account for almost all the listens. This is a power-law distribution, similar to Zipf's law for word occurrences, and as a consequence it is necessary to use robust estimators.

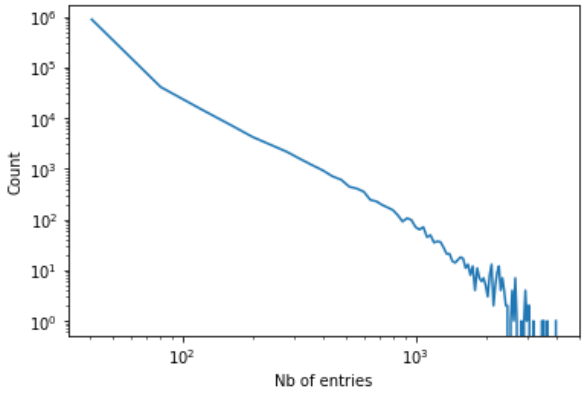


Figure 1: Distribution of the occurrence per songs on log-log scale

In theory, data collection spanned from 2005 until 2009. Yet, records in the dataset are found up until 2014. As shown

on Figure 2, the number of entries drops abruptly mid-2009 but stays positive until 2014.

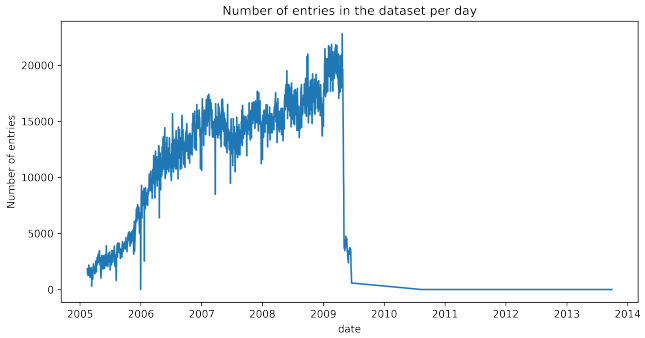


Figure 2: Number of entries with respect to time

Data exploration revealed that a important amount of ID's are missing. About 11% of track ID's and 3% of artist ID's are missing. Moreover, track and artist names are not consistent through the records. Even with the same ID, some songs or artists have different names. For instance *Yael Naim* and *Yael Naïm* or *[Disney]* and *Disney*. Yet different names may sometimes be meaningful, for example one can argue that *My Way (Duet With Luciano Pavarotti)* and *My way* are different songs despite the same ID, thus it is the ID's that should instead be different. Having missing ID's and unreliable names is an issue that is necessary to address and we describe in Section III how we designed our solution.

Figure 3 shows the most frequent artists: *Radiohead*, *The Beatles*, *Nine Inch Nails*, etc. Most artists are rock bands, presenting a discrepancy with the global charts of 2009 according to the Billboard magazine, containing mainly pop songs. This difference illustrates that the relatively small number of users is indeed insufficient to have a representative sample of the global population.

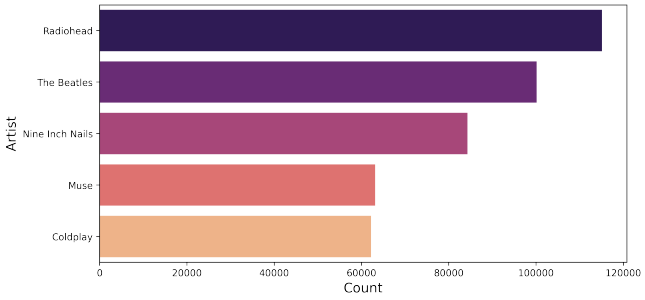


Figure 3: Most frequent artists in the dataset

An analysis of the user demographics showed that data was unreliable. An example is illustrated in Figure 4, where the user age distribution is drawn, containing 3-year-old to 100-year-old users. Moreover, a significant amount of demographics data is missing, as it is probably not required to fill at registration. Along with the bias induced by the insufficient number of users, this unreliability proved that the dataset cannot be used to draw demographic conclusions on general music tastes or global listening patterns.

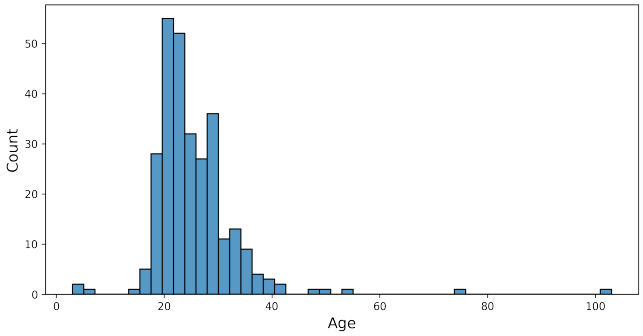


Figure 4: User age distribution

III. The Proposed Approach

A. Tags

An important information on a song is its genre. Indeed, in order to evaluate the information encoded in the vector representations, genre is a meaningful element. However, song genres are not already included in the dataset. Therefore, we used the Last.fm API to query the song tags, which are not song genres but keywords chosen by users for the songs. Tags are a superset of genres and each comes with a count of how many users applied this tag. The count is upper-bounded by 100 and only users that used Last.fm for a certain amount of time can apply them. Some examples are given in Table I. Since tags are applied by users, using them required some pre-processing, for instance to filter out tags composed of the name of the artist. The most frequent tags after data cleaning are *indie*, *alternative*, *electronic*, *indie rock* and *pop*. On the technical aspect, the tags are available at the `getTopTags` Last.fm endpoint, and due to the considerable amount of songs in the dataset, querying all the tags took 2 days while using multi-threading to simultaneously send multiple requests (within the limit allowed by Last.fm).

Table I: Examples of Last.fm tags for two songs

Paranoid Android - Radiohead		**My Way - Frank Sinatra**	
Tag	**Count**	**Tag**	**Count**
alternative	100	jazz	100
alternative rock	87	oldies	81
rock	69	frank sinatra	81
radiohead	56	swing	60
indie	48	easy listening	45

B. Data Cleaning

To improve the data quality and solve the previously identified issues of missing ID's and non-cohesive artist and track names, we employed the MusicBrainz Database [6]. This decision was backed by the fact the original dataset

relied on this dataset in order to assign each artist and track with an ID, whenever it existed. To tackle this problem, besides the original dataset, three new versions of it are generated, with different levels of data unification. The unmodified dataset will henceforth be referred to as the **Original Dataset**.

Firstly, we unify the artists' information by filling in missing ID's. To achieve this, we start by extracting the Original Dataset's artist's name and applying a simple transformation to it, in order to remove entries where one artist features another, and trying to fill in the missing ID by matching only the main artist with MusicBrainz's Dataset name and sort name (a variant of the artist's official name, often used when sorting artists by name) fields. For entries still not identified, we match the lower-cased versions of the Original Dataset's artist name with both its lower-cased name and sort name counterparts in the MusicBrainz Database. Finally, to address the variation of artist names, we now merge the Original Dataset with the MusicBrainz Database one, but instead of the names, we merge on the artist ID's, and, on match, replace the artist's name with its official version as defined in MusicBrainz. This resulted in the **Artist-Merged Dataset**.

A very similar process was done to clean the tracks' information, filling in missing ID's by an analogous process. Instead of removing featured artist, track names featuring "live" (followed usually by a specific date) were modified to contain only the track name, and a no-parenthesis variant was also created. Merges were then performed based on these fields and their MusicBrainz Database track name counterpart. For tracks still unidentified, the lower-cased versions of these fields were used for the matching. This resulted in the **Doubly-Merged Dataset**.

Finally, to unify track ID's, as with artist ID's, a merge was performed based on matching track ID's between the Doubly-Merged Dataset and the MusicBrainz Database, and, on match, the track names were replaced by their official names. This resulted in the fourth and final dataset, **Renamed Doubly-Merged Dataset**.

Though this process did not result in a perfectly clean final dataset, it significantly improved the cohesiveness of the data. However, in rare cases, matches were approximate, as there existed, for example, several artists with exactly the same name in the MusicBrainz Database, but with different ID's and maybe even different creation years, but with so many missing fields that discriminating between them was impossible even for a human annotator. Due to this, instead of just using the Renamed Doubly-Merged Dataset from this point onward, all four datasets were kept and treated as a hyper-parameter.

C. Song Embeddings

In order to study listening contexts, or even song similarities, we first need to create an appropriate representation for each song. To achieve this, we're applying the ideas from Word2Vec to our dataset. To transform our data into an equivalent format as the one used to produce those embeddings, each unique song is considered as an unique word, and each user's listening history is treated as a sentence-equivalent. This transforms our dataset into a document with approximately 1000 sentences, of varying length. To identify each unique song, each artist and track names are assigned a code. If two entries have the same artist name, they will also have the same artist code, and if they had different values, their codes will be different. The artist and track codes are then combined to form a single identifier that encodes both values, and with which only songs which had the exact same artist and track name will have the same identifier.

Having transformed our data into a document with several sentences made up of words, we now need to apply the Word2Vec models to it. The original paper proposes two different models: the Continuous Bag of Words (CBOW) model, which is trained on a word's surrounding context and aims at predicting the center missing word; and the Skip-Gram model which does the opposite, receiving a single word as input and trying to predict its surrounding context. The model choice was considered another hyper-parameter, as both have their trade-offs. According to the authors of the Word2Vec paper, Skip-Gram can better represent words that occur infrequently, and works well with small datasets, while CBOW is faster to train (according to their values, about 3 times faster to train), and can better represent frequent words.

The underlying dataset and the model architecture are, however, not the only hyper-parameter choice we're facing. Furthermore, are these models are being built outside of a literary context, the propose default values are often non-optimal and can visibly affect performance [7]. In light of this, informed by music-datasets appropriate range of values for several other hyper-parameters, we employed grid search and three classification tasks to fine-tune the models and find their optimal hyper-parameters.

D. Fine-tuning

Selecting the best model for our data was not a trivial process. First, due to computational and temporal restriction, only a subset of the hyper-parameter search space was considered. Secondly, we needed to develop concrete metrics with which to assess each model's performance.

To evaluate each model, the following three classification tasks were defined:

1) Predicting whether any two given songs come from the same context
2) Predicting whether any two given songs are from the same artist
3) Predicting whether any two given songs share the same top tag

For all three tasks, randomly generated inputs were created by concatenating two songs embeddings into a single vector, and using that as an input, a random forest classifier was trained and evaluated on the correct classification of the entries. All three are binary classification tasks. The first is labeled `1` when two songs were consecutively played at least once by at least one user, the second when two songs share the same artist, and finally, the third when they have the same top tag.

With regards to the search space, a grid search was used over the following hyper-parameters and values:

- **The underlying dataset** - between four possible choices: Original Dataset, Artist-Merged Dataset, Doubly-Merged Dataset and Renamed Doubly-Merged Dataset.
- **The model architecture** - between CBOW and Skip-Gram architectures.
- **Embedding dimension -** between vectors of size 20, 60 and 100.
- **Context window size -** between sizes of 3, 5 and 10.
- **Minimum occurrences -** between a minimum count of 1 and 5.
- **Number of negative sampling words to be drawn -** between 5, 10, and 20 words.
- **Negative sampling distribution's exponent -** between an exponent value of -0.8, -0.5 and 0.

These values already present a more limited search space than we would have liked, and we would have liked to explore at least a few more embedding dimensions and negative sampling factor values. Despite this, running a search over all these would be computationally prohibitive for us, given our equipment and available time-frame. As such, the fine-tuning was done in two steps:

1) First, to reduce the number of models that needed to be created and trained by four-fold, the underlying dataset was fixed to the Original Dataset. In total, 324 models were trained at this stage, 162 of them CBOW models and the other 162 Skip-Gram ones. Their performance was evaluated on the first and second classification tasks. Originally, we had planned to create models for all the four datasets, but just training (not evaluating) the 162 Skip-Gram models took more than 18 hours on our fastest computer, and the evaluation of each group of 162 took, per each task, more than 10 hours on the same setup.
2) After obtaining the performance results on the first two tasks, the best 15 configurations, per model architecture, were recorded, and used to train models with these configurations but now across the four different datasets. As such, a total of 120 models were created and trained – 30 per each dataset. These models were then evaluated on both the third task, which could be used a type of validation set, as it had not been used in the model selection itself, and over the three tasks, in order to see which models would perform well across a myriad of tasks (where the third task served less as a pseudo-validation set and more as an assurance the model had not over-fitted to those tasks and was able to generalize and perform at least reasonably well on new, previously unseen, tasks).

IV. Experimental Evaluation

A. Fine-tuning results

After training the initial 324 models, they were evaluated on the first two classification tasks, using their F1 score, accuracy and Area Under the Curve (AUC) values. Giving equal weight to the three metrics and to the two tasks, the top 15 best models per Word2Vec architecture were selected. Figure 5 displays the distribution of the results of each type.

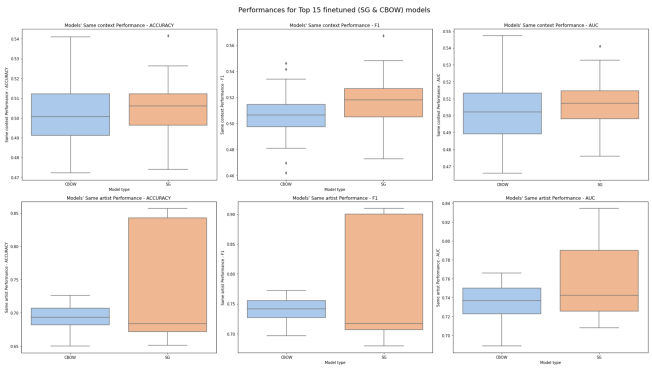


Figure 5: Overall performance results for the first two tasks, same context and same artist classification, separated by models' architecture type.

These total 30 model configurations were then used to train the models for the other three datasets. Figure 6 displays the distribution of the 120 models' performance on the third task.

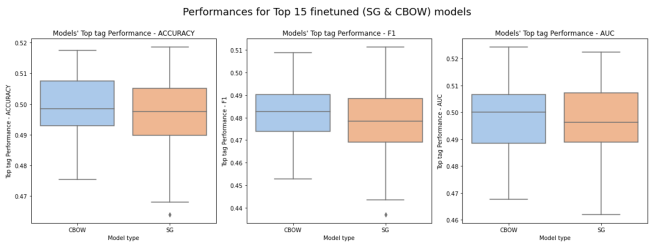


Figure 6: Overall performance results for the third task, same top tag classification over the four different datasets, separated by models' architecture type.

To select the best performing model, all three metrics and tasks were given the same weight, and the overall best scoring model selected. Table II, below, provides a summary

of the results. For better readability, instead of presenting all three performance metrics, only the F1 scores are displayed.

Context	Artist	Top Tag	Architecture	Config.	Dataset
0.531792	**0.906468**	0.492147	Skip-Gram	108	Artist-Merged
0.527273	0.903707	**0.497399**	Skip-Gram	19	Doubly-Merged
0.524815	0.905106	0.482940	Skip-Gram	108	Doubly-Merged

Table II: F1 scores of the top 3 overall best models on the three classification tasks as well as their corresponding architecture, configuration and dataset.

It is possible to observe that see configuration 108 appears to perform well regardless of the specific dataset it was trained on. The overall best performing model's hyper-paramenter values are the following:

- **underlying dataset:** Artist-Merged Dataset;
- **model architecture:** Skip-Gram;
- **embeddings dimension:** 100;
- **context window size:** 3;
- **minimum occurrences:** 1;
- **negative sampling words to be drawn:** 5;
- **negative sampling distribution's exponent:** -0.8.

It's easy to infer why this configuration might be leading to success on the classification tasks: much consideration is being given to less frequent songs. The Skip-Gram architecture itself performs better on infrequent words than its CBOW counterpart, since it is trained on single word inputs (and thus less prone to over-fitting on common words). The hyper-parameter values of a minimum count of one occurrence as well as such a negative sampling exponent also translate the extra effort model is giving to these rarely occurring titles.

B. Baseline Comparison

In order to quantitatively assess if the song representations hold some value and encoded some information we evaluated them using different classification tasks. We compared the performances of the song embeddings on 4 different classification tasks:

1) Predict if two songs were sampled from the same context (in fact successive songs)
2) Predict if two songs are from the same artist
3) Predict the artist of a song (among the top 10 artists)
4) Predict the first tag of a song (among the top 10 tags)

For tasks 1) and 2), identical in objective to the first two classification tasks used for the models' fine-tuned, two songs are used as input, and their embeddings concatenated into a single vector. For task 4), the tag ordering is defined according to the user count of each tag (as a measure of user agreement) and then by alphabetical order for determinism.

To perform the classifications using the embeddings we used a random forest classifier, for its great representative power and the possibility to perform hyper-parameter fine-tuning, while it is often too time consuming for neural networks. For the evaluation metric we chose the F1 score as the classification tasks sometimes have a class imbalance and we don't give more importance in predicting correctly any of the class. Tasks 3) and 4) are multi-class predictions and we chose to use weighted F1 score, again due to the class imbalance.

The performances of the song representations, which we will call *Song2Vec*, are compared to 3 different models: a random predictor, a majority predictor and a random forest with input embeddings created using TF-IDF. To create those embeddings using TF-IDF, users acted as documents and songs were considered for terms, resulting in vector representations of dimension equal to the number of users. For each task, data was split into a training set and a test set and both random forest models were fine-tuned using 5-fold cross-validation grid search on the training set.

The results are given in Table III.

Classification tasks	Song2Vec	TF-IDF	Random	Majority
Same context	0.474	0.495	**0.507**	0.0
Same artist	**0.903**	0.495	0.608	0.846
Top artists	0.169	**0.7**	0.034	0.012
Top tags	0.226	**0.725**	0.109	0.074

Table III: F1 score of the different models on the classification tasks

On the same context classification task, all models performed near random, except the majority, which predicted the negative class. An explanation is that the task is intrinsically too hard and songs from the same context among a 20-million-song dataset would require more data on which to train the models.

However, for the same artist classification we can see that our embeddings performed significantly better than all the other baselines. We might conclude from this relatively large difference that songs from the same artist share similar listening contexts, for instance that users listen to songs from the same artist successively.

On the two last tasks, predicting the artist and tag of a given song from a pool of the respective ten most popular ones, our model performs better than the random and majority predictors yet TF-IDF significantly outperforms all models. A conclusion may be that songs from the top artists or top tags are very different in terms of context. For instance, it is possible that songs from *Radiohead* or from *The Beatles* are listened in different context, especially if it holds that users listen to music by artist. The great performances of TF-IDF may also be explained by the choice of using popular artist and songs. We are measuring popularity using the frequency and TF-IDF is defined according to the term and document frequencies, therefore it is probable that TF-IDF values are highly correlated to popularity.

C. Qualitative Interpretation

In order to gain insight on our song embeddings we perform a dimensionality reduction on the embeddings coloured by the top ten artists and top ten tags as shown in Figures 7 and 8. From this we can identify a clear pattern between song artists and song tags through our song embeddings. Notably, in Figure 7 one can identify a pink cluster corresponding to the band *Nine Inch Nails*, similarly for Figure 8 the brown cluster representing the *rock* tag is located on the same tangent. This is coherent with respect to the fact that Nine Inch Nails is a rock band.

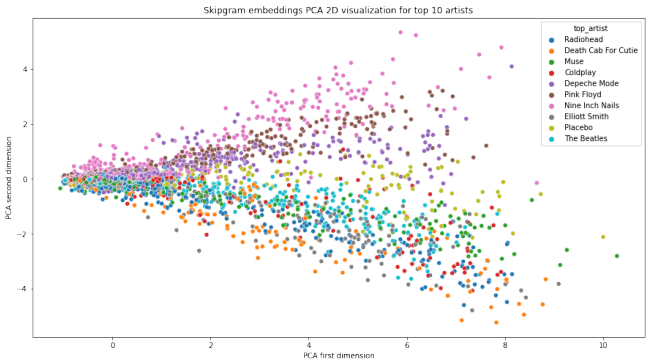


Figure 7: 2D PCA coloured by Top 10 Artists

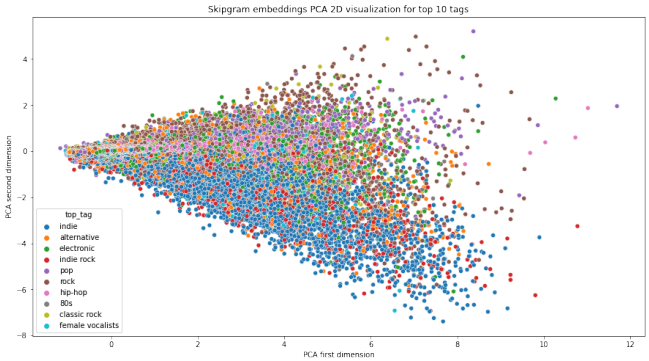


Figure 8: 2D PCA coloured by Top 10 Tags

In order to gain further insight on the Principal Component Analysis (PCA), we used plotly [8]. We added several attributes to the hover data to analyse the points corresponding to each song, namely:

- `PC1` : The value of the first principal component,
- `PC2` : The value of the second principal component,
- `artist_name` : The artist name for that song,
- `track_name` : The song name,
- `top_tag` : The top tag for that son,
- `occur` : The number of times that particular song appeared in the samples,
- `usr_count` : The number of unique users who listened to that song,

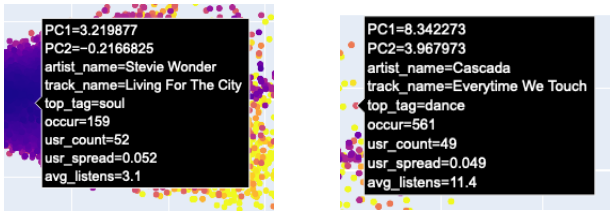


Figure 9: Hover Data Examples

- `usr_spread` : The ratio of unique listeners to total users in our sample,
- `avg_listens` : The average number of listens for that song per user.

A couple of examples can be seen on Figure 9.

By plotting the dimensionality reduction coloured by `occur` (Figure 10) and `usr_count` (Figure 11, we can gain insight on our principal components. Namely, we can see that song occurrence seems to increase along the x-axis, which indicates that the first principal component encodes notions of song popularity in terms of total listens. On the other hand, from Figure 11 we can observe a dark line along the center of the y-axis which seems to lighten out symmetrically along the latter. This could indicate that the second principal component encodes the song popularity in terms of number of unique users who listen to that song.

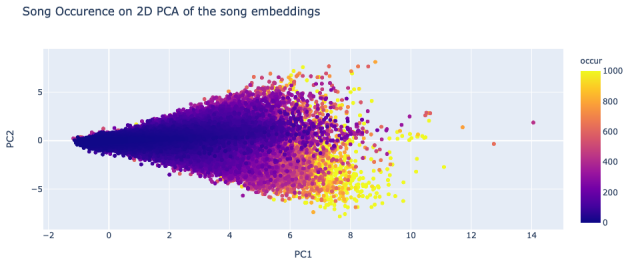


Figure 10: 2D PCA coloured by Song Occurence

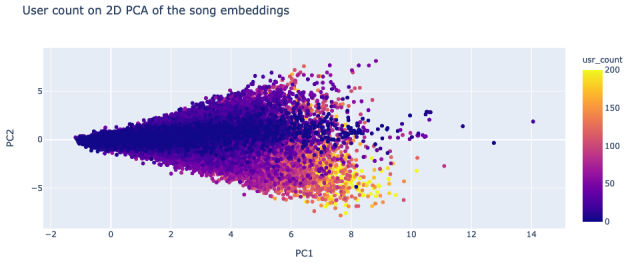


Figure 11: 2D PCA coloured by Song Listeners

We leveraged the K-means clustering algorithm [9] in order to extract further information from our song embeddings. We can see from Figure 13 that the clusters

follow a pattern similar to the PCA. We used the Bayesian Information Criterion (BIC) and distortion measures to choose the number of clusters, illustrated in Figure 12. We didn't use the Silhouette score due to computational constraints. For example with three clusters we see that they are isolated by song occurrence. Likewise for five clusters, with an additional split along the y-axis which represents the popularity in terms of user listens. The split along the y-axis also encodes tag information as illustrated by Figure 8.

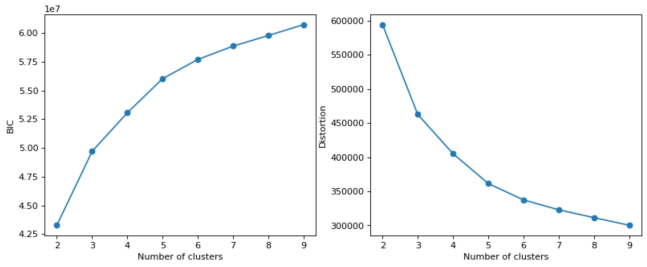
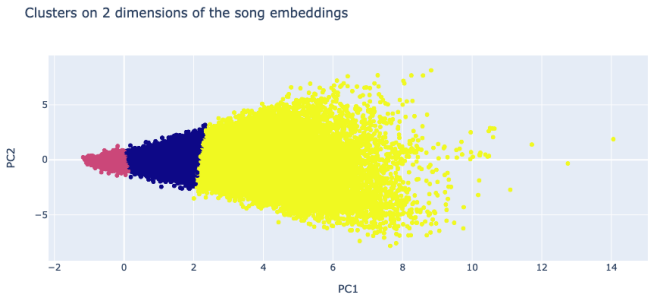
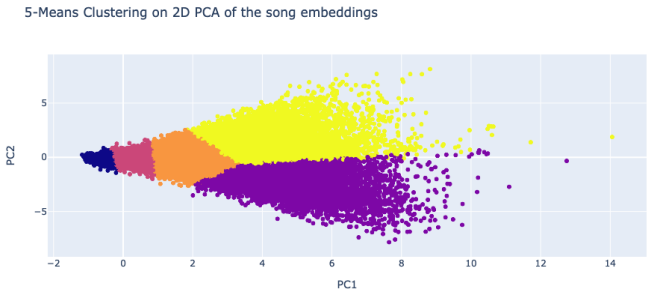


Figure 12: BIC (left) and distortion (right) with respect to the number of clusters



(a) 3-means



(b) 5-means

Figure 13: K-means clustering

V. Discussion and Implications

We centered our research on uncovering what information can be extracted from a song's listening context. Our hypothesis was that the embeddings would encode information about genre, context-based similarity, and possibly even artist. Section IV-C highlights our findings. Notably, as anticipated, the song's listening context indeed encoded information about genre (through tags) and artists. In addition to this, information regarding popularity, both in terms of listens and users, is encoded in the embeddings.

In terms of performance, TF-IDF outdid our expectations. We expected our song embeddings to perform better as they were specifically designed for this music context, as opposed to TF-IDF which is more generic. It therefore shows that TF-IDF adapts well to more specific contexts.

It is also possible that more training data, or more powerful computers that would allow us to train bigger embeddings could lead to a better performance. Indeed, it is important to note that the TF-IDF vectors have a dimensionality of almost 1000, as opposed to our best model with a dimensionality of only 100, so it seems plausible that TF-IDF may be encoding more information.

One final conclusion that can also be drawn is that performances are highly dependent on the task and the fine-tuning should ideally be performed for each task rather than as an overall process.

VI. Conclusions and Future Work

Music recommendations are not an easy problem to tackle. We created fully reproducible song embeddings tailored specifically to the Last.fm data, in order to assess how well they managed to capture users' listening behaviors, in terms of genre, context and artist information. From the comparison between our model and the three baselines, detailed in Section IV-B, we can conclude that our embeddings manage to encode useful information and, subsequently, do indeed possess some value.

One possible next step would be comparing an optimized version of our model against fine-tuned baselines, and see whether our embeddings remain competitive. As mentioned in Section V, though we were limited by the dataset size and computational power, for future work, it would be beneficial to train on a bigger dataset and it could also be interesting to train bigger embeddings so they'd be, dimensionality-wise, on equal grounds with TF-IDF.

Our song embeddings allow us to extract various insights regarding the song's listening contexts. It would be interesting to go further still, and investigate what other aspects are encoded in the embeddings, from the temporality of user listening patters. Intuitively, it seems likely that we'd be able to extract different clusters for songs (mainly) listened to in different seasons.

In addition to this, for our research, we limited ourselves to the top tag per song. For a final future work suggestion, more tags could be used per song, as well as integrating the `tag_count` in our analysis, although the latter is capped at 100 which limits the its insightfulness.

REFERENCES

[1] D. Ek, *Spotify*, 2006, https://www.spotify.com/.

[2] A. Inc, *Apple Music*, 2015, https://music.apple.com/.

[3] F. Miller, *Last.fm*, 2002, https://www.last.fm/.

[4] T. M. et al., *Efficient Estimation of Word Representations in Vector Space*, Google Inc., Sep. 2013.

[5] Òscar Celma, *Music Recommendation Datasets for Research*, 2010, http://ocelma.net/MusicRecommendationDataset/.

[6] M. Foundation, *MusicBrainz*, 2004, https://musicbrainz.org/.

[7] H. Caselles-Dupré, F. Lesaint, and J. Royo-Letelier, "Word2vec applied to recommendation: Hyperparameters matter," *CoRR*, vol. abs/1804.04212, 2018. [Online]. Available: http://arxiv.org/abs/1804.04212

[8] P. T. Inc. (2015) Collaborative data science. Montreal, QC. [Online]. Available: https://plot.ly

[9] X. Jin and J. Han, *K-Means Clustering*. Boston, MA: Springer US, 2010, pp. 563–564. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_425