Implementing the Prod2vec Algorithm as a Music Recommender in Python

Sunday 24th March 2019

Patrick Knott

Supervisor:

Assoc. Prof. Yue Xu

1. Executive Summary

This project implemented and evaluated the word2vec algorithm as a recommender for online music streaming services. Music streaming services offer both a free service for users, as well as an enhanced service for a subscription fee. Higher quality song recommendations increase user willingness to pay for the service. This project found that the Continuous-Bag-of-Words (CBOW) word2vec model gave the best suggestions.

This project assessed four variants of the algorithm. The primary results are that SkipGram and CBOW gave accurate recommendations, with CBOW model's recommendations being more unexpected than those of SkipGram. User2vec seemed to work but requires real world testing to quantify. The project also found that bagged-prod2vec worked, but not as well as the standard data input method (i.e. grouped by user rather than listening session). A variant on a collaborative filtering matrix was implemented and assessed, finding that it did not work.

A small dataset was used for this analysis so the latter two word2vec variants and the modified collaborative filtering method may be more effective in a larger scale implementation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Recommending has become a vital tool for ecommerce (Zhang, 2017). However, music streaming services suffer from a problem with the quality of their recommendations. This project will investigate the efficacy of a relatively new recommendation method, word2vec embeddings. One streaming service, Anghami, uses a variation of word2vec as part of its recommendations, but does not release any information about its details or effectiveness (Karam, 2017).

The internet age has brought with it a problem of consumer over-choice (Arditi, 2018). The abundance of options leave consumers overwhelmed. Pitney Bowes (2018) found that 61% of online shoppers are disappointed with the experience. In many different online markets, recommender systems have been shown to improve the user experience and to increase sales (Zhang, 2017).

Music distribution is shifting online, initially with music sales, and more recently with streaming services. Last year, a quarter of profits for music content owners came from streaming subscriptions alone (Arditi, 2018). Hand-in-hand with this shift to ecommerce, the problems stemming from a tidal wave of offerings have begun to face music consumers. Spotify have 30 million different songs, far beyond the abilities of a person to negotiate (Hall, 2019). It has been found that the average music consumer triples their annual spend if they swap from purchasing music, to streaming music (Liang, 2015). Most streaming services offer a "freemium" model, where there are advertisement funded options for users, as well as enhanced memberships available for a subscription fee.

Accurate and unexpected recommendations for listeners will increase their likelihood of paying for a streaming service subscription (Zalmanson, 2016), greatly increasing profits for the music industry.

ENVIRONMENTAL SCAN

Before 2013, online recommendations were typically performed by collaborative-filtering, where behaviours of users are compared, or content-based filtering where characteristics of the item are extracted by domain specialists or by mathematical tools. However, these techniques tend not to work well with music for a variety of reasons, such as the strong social and geographic impacts on listener tastes (Van den Oord, 2013). Most collaborative-filtering techniques require explicit user feedback (Mcfee, 2012) (e.g. a rating out of five stars) which are rarely available with songs. Another problem specific to music is that content-based methods lead to obvious, and therefore unhelpful recommendations (Van den Oord, 2013). It is also much more computationally efficient to find relationships between songs than between users for music streaming sites (Barkan, 2016) as they tend to have many more customers than songs available (Hall, 2019). So, before 2013 music recommendation algorithms were greedy. They were built to either recommend the most popular songs, recommend other songs of the same genre, or recommend other songs by an artist that a user already listened to (Wang, 2014).

In 2013 (Mikolov) a pivotal paper in recommender scholarship was published. A natural language processing technique called word2vec was announced. It is an algorithm which develops an understanding of words by revealing their relationships to other words, derived solely from their proximity to each other within a text. This method seems to work because it functions similar to our intuition and reveals pertinent underlying relationships (Meyer, 2016). The

process has been altered, with great success, for a variety of other tasks such as recruitment consulting (Elsafty, 2018), image recognition, graph traversal, ad placement, and the relationships between goods and services (Grbovic, 2016). This latter application is how word2vec has become an especially powerful technique for recommender systems (Liang, 2015). By finding the relationships between products themselves, accurate recommendations can be made based on as little as one previous purchase by the consumer.

In 2015 (Grbovic), word2vec was used to recommend product advertisements to Yahoo email users by analysing their online shopping receipts. The paper announced a new technique the authors named bagged-prod2vec. The idea was to allocate stronger connections to items within a single receipt, rather than treat all items in all the customer's receipts as single sequence. After A-B testing on real users, they found that bagged-prod2vec resulted in a 9% higher click-through and conversion rate (Grbovic, 2015). This paper will adapt the bagged-prod2vec technique by partitioning a particular users music streaming history into chronologically distinct sequences.

Caselle-Dupre (2018) found that the following hyperparameters were most likely to be impactful on prediction accuracy of word2vec recommenders:

- Window size (how many items either side of the target item should be included as model input)
- Negative-sampling basket size (when calculating the cost function for each training input, how many non-pairings should be contrasted with the actual item pairing)
- Number of epochs (how many times to loop through the full training data set when building the model)
- Batch-size (how many song pairs to include when performing one step of the training)

In 1989 Cybenko's Universal Approximation Theorem showed that it is the number of hidden neurons not the number of hidden layers that most impacts accuracy of neural nets. So, the number of neurons will be included as a hyperparameter to test.

Glorot (2010) found that weight initialization methods can have a large impact on neural net training times. However, there are many hyperparameters more likely to have unique behaviours for a music model so different initialization techniques were not evaluated. Finally, the functions for the training cost, optimization and activations were investigated.

During the literature review, it was found that one of the problems with using the collaborative filtering technique for music streaming recommendations was that users don't give ratings to songs (as opposed to, for example, movies). The researcher speculated that the number of times a user listens to a particular song could be used as a proxy for a rating. So, as well as the word2vec models, a pseudo-rating variant of a collaborative filtering recommender was built and assessed.

THE WORD2VEC ALGORITHM

By finding relationships between a sequence of words in a text, the words can be placed into a vector space, where their positions with respect to one another reveal semantic and syntactic meanings of the word. Figure 1 shows a vector space developed from a set of texts. In the bottom left of Figure 1 the capital city names and the corresponding country names have the same difference between them. Or on the right-hand side, the adjectives slow, fast and long have the same difference between their different variations. With enough data, any

ordered sequence can be placed into an embedding which contains information about the relationships between the elements.

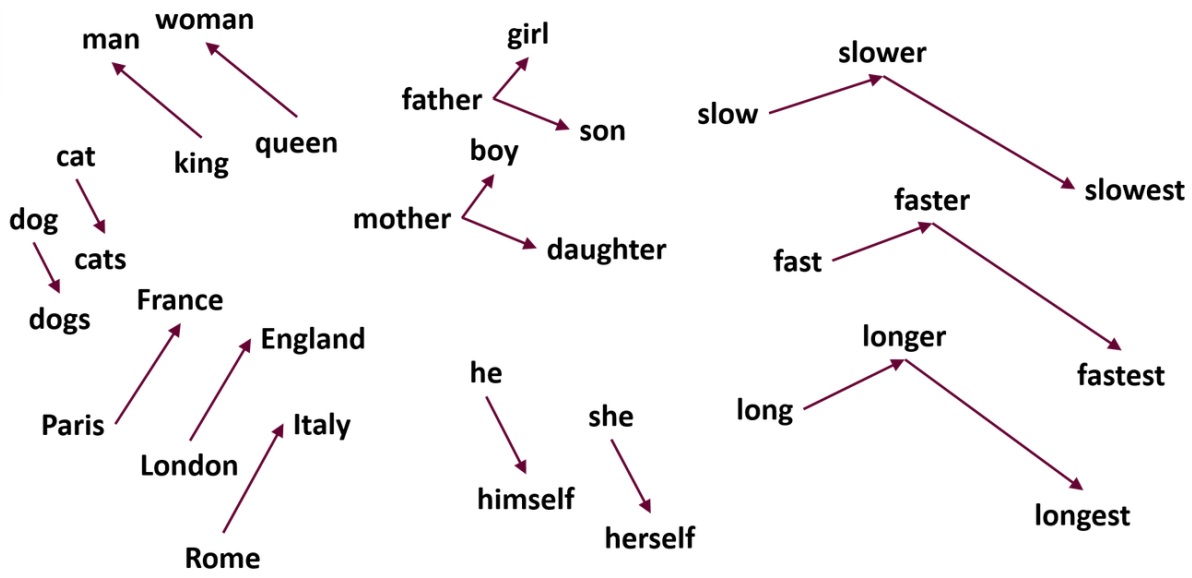


Figure 1. Example of word2vec embedding for words

Starting with a sequence of songs B-C-D (or a sequence of words, YouTube videos, Amazon purchases etc.), word2vec breaks them into inputs and predictions. The two training variants tested are shown in Figure 1. In SkipGram, one song is used to predict multiple songs around it, whereas Continuous-Bag-of-Words uses multiple songs to predict one song in the middle. In practice, the predictions are actually broken into one-to-one input-predictions pairs. For example, C-to-B and C-to-D for SkipGrams, or B-to-C and D-to-C for CBOW (Figure 2).

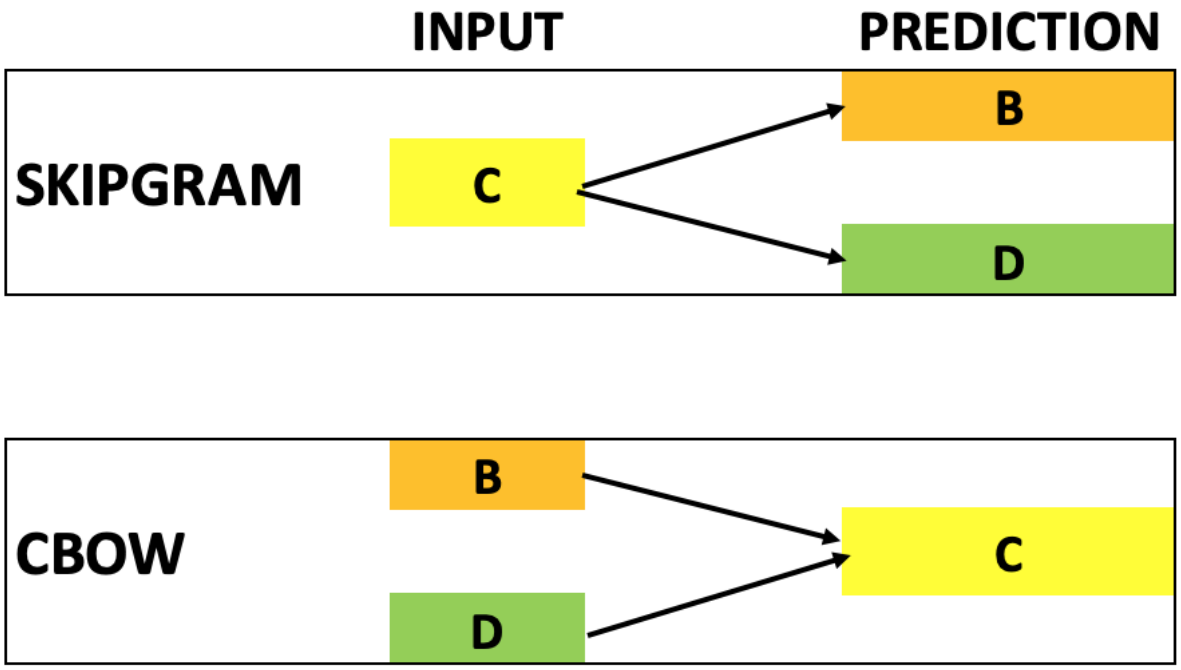


Figure 2. The SkipGram and CBOW model input concepts

Figure 2 shows a set of input-prediction pairs. Notice on the right that both correct and incorrect pairs are generated. From this sequence of songs, B-C-D, a correct prediction is calculated, B-to-C, and it is then labelled as correct. Then an incorrect prediction is created with the input song B and a randomly selected non-output song, in this case F. This process is repeated for every song in the user's listening history. The ratio of correct to incorrect predictions, and the size of the window of correct predictions were two of the most important hyperparameters tested.

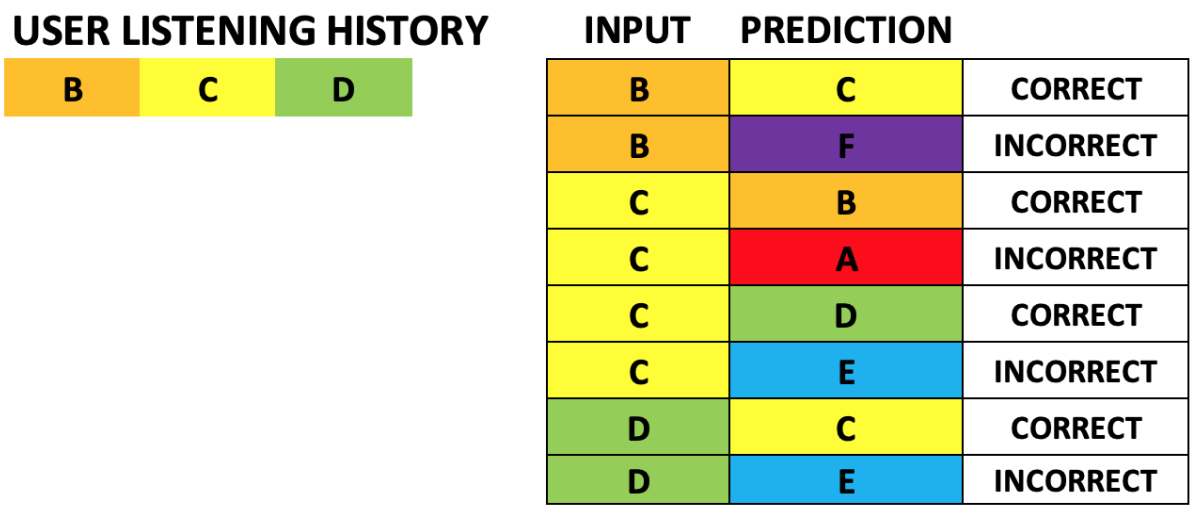


Figure 3. Example of SkipGram model input tuples

Having generated the correct and incorrect input-prediction pairs, the neural net is built (Figure 3). The letters A through F are the full set of unique songs. The model input layer and output layer are each a vector, with an element for each unique song. Each element in the input vector on the left, connects to each hidden layer element, each of which connects to each output layer element. Each of these connections has a weight, just a scalar multiplier. One input-prediction pair is selected. The input element corresponding to the input song is filled with a value of one, the other inputs are all set to zero. Then, these values are multiplied by each of the weights and summed to calculate the values in the hidden layer, which in turn are multiplied by each of the weights connecting to the output layer. Then, this output layer is compared to what the output layer should be. The prediction song from the pair should be a one for a correct prediction, or a zero for an incorrect prediction, with all the other outputs zero. For each element in the output vector, the differences between the actual output

and what the output should be are calculated. Then, back propagation is used to make a small adjustment to each weight in both set of connections to slightly improve the prediction. After this training step, randomly select another input-prediction pair, (maybe a correct one, maybe an incorrect one) and repeat for every song pair generated in the SkipGram or CBOW pre-processing stage.

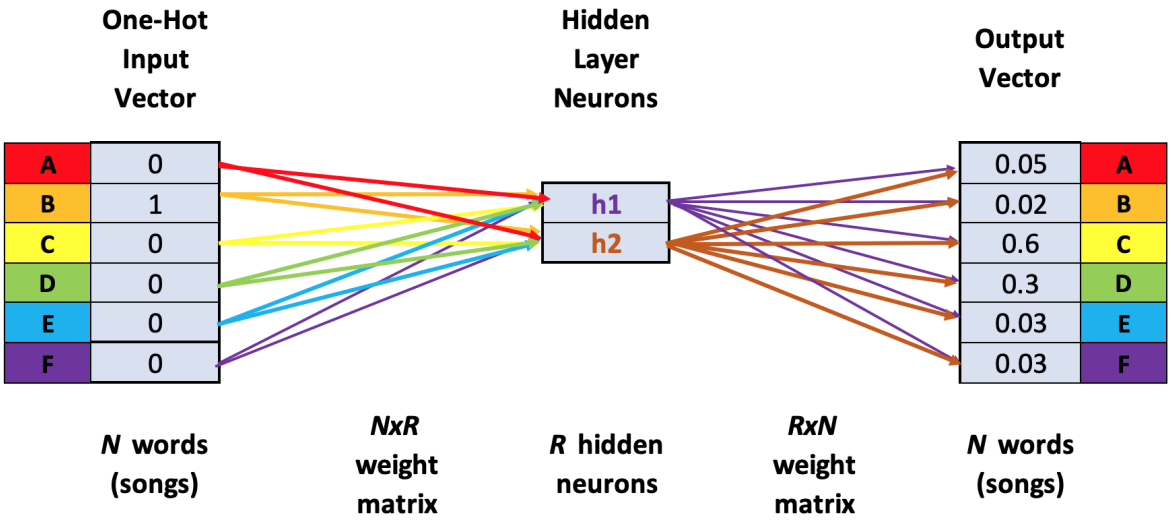


Figure 4. Example of word2vec neural net for songs

After this process ends, discard the output layer, the set of connections from the hidden layer to the output, and the hidden layer. The set of connections from the inputs to the hidden layer are the set of song embedding vectors. For example, in Figure 4 above the neural net contains two hidden layer elements so the word2vec vector embeddings for song 'A' are a two-element vector [1.3, -0.2]

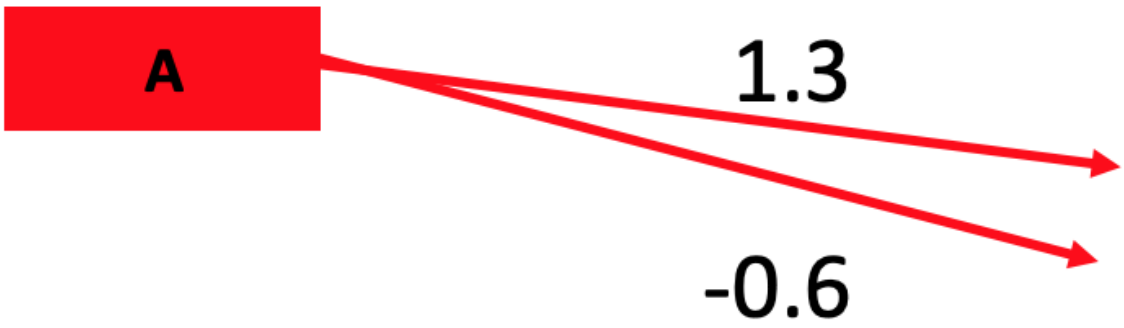


Figure 5. Example of connection layer weightings for a song

One-to-one song recommendations can be made by finding the closest other songs to a particular song in the vector space.

Some recommenders use an alteration called user2vec which embeds users into the same space as the items, then uses simple distance measures to make recommendations (Karam, 2017).

A different method of generating the input-prediction pairs, called bagged-prod2vec was evaluated too. It takes place before the SkipGram or CBOW stage. Originally used for placing ads beside Yahoo email apps, bagged-prod2vec would parse online shopping receipt emails to generate a sequence of purchases. However, it would treat the products on one receipt as a sub-sequence, giving stronger associations within a recept than between different receipts. In this project it was adapted to create sequences of songs which are listened to in one sitting, this was calculated as those songs starting less than fifteen minutes from each other.

OBJECTIVE

The quality of recommender systems for music have lagged behind other markets. This project sought to implement and assess a word2vec algorithm as

a recommender for music streaming customers. More accurate recommendations for music streamers will encourage users to commit to a particular streaming service, which in turn will increase a user's willingness to pay for its services (Zalmanson, 2016).

KEY DELIVERABLES

1. Create music specific data preparation code
2. Implement word2vec library for music recommendation
3. Original code for CBOW input preparation, bagged-prod2vec pre-processing, user2vec creation and evaluation
4. Compare CBOW and SkipGram input methods
5. Assess bagged-prod2vec's efficacy on music
6. Derive original code for pseudo-rating collaborative filtering

LIMITATIONS

1. the dataset used may have distorted the evaluations of the different recommender models
2. quantitatively evaluating the user2vec method requires real world testing.

Neural networks are computationally expensive. So, a relatively small dataset was used for the project. The full LastFM dataset contains 20 million play events and one thousand users. But the project was run on a personal laptop computer. Building models with the full dataset would overload the RAM and the program would shut down. A subset containing 8 users, 515 thousand play-events and 76 thousand unique songs was used. For some models (batch-size 1) this still took eleven hours to run a single epoch of training.

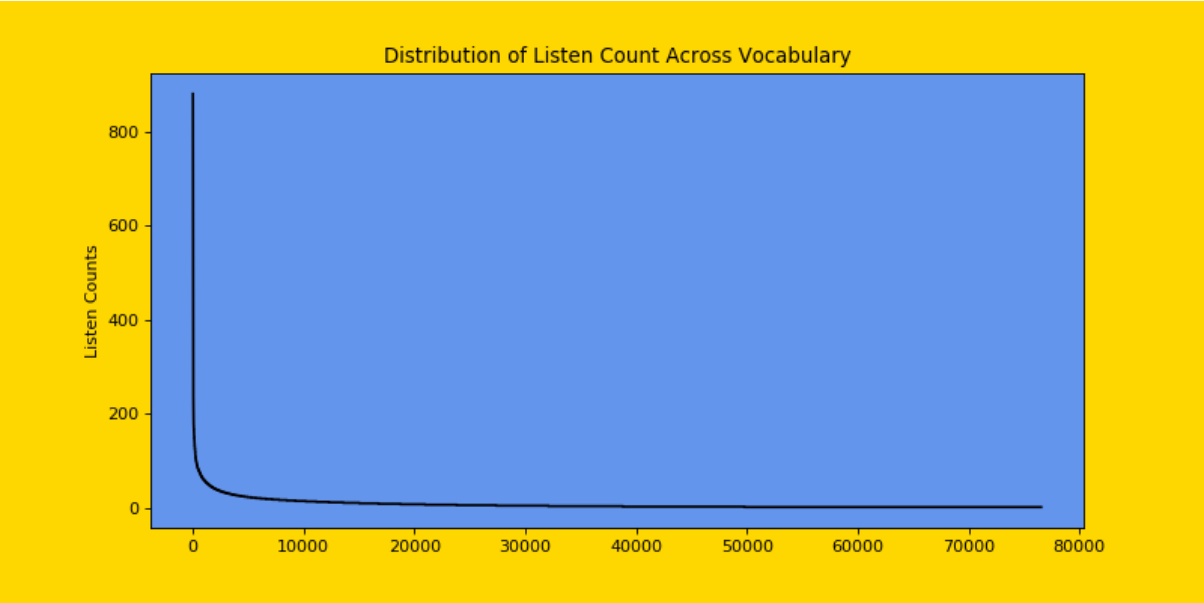


Figure 6. Plot of song listen-count frequency distribution

Though the songs are played an average of six times each, the median play-count is three. Figure 6 shows the listen-count of each of the seventy-six thousand unique songs, with the maximum count of nearly nine-hundred, but quickly tailing off.

There was very little listening history overlap between the users. Only nine-hundred and nine songs, 1.2% of the total, were listened to by more than one person.

It is possible that bagged-prod2vec, user2vec and the pseudo-rating collaborative filtering models would be more effective with a larger dataset, a dataset with more overlap between user listening histories, or even a different source of data. This project should not be interpreted as proof that these methods are inferior or ineffective.

The other major limitation of this study is that without the ability to test on real users, the user2vec model could not be evaluated with precision. The other

models had labelled data for calculating the accuracy of suggestions. However, implicit in the user2vec variant is that some of the songs closest to a user in the vector space will be songs the user has not heard before. The only way to assess the user2vec was to count how many of the closest songs to a user were in the user's listening history. But the input data for the project took into account the sequence of songs listened to. So, to add chronology to the user2vec suggestions would require A-B testing.

PROJECT MANAGEMENT

As is common for software development, this project was managed with an agile methodology. There were many unknowns about development timeframes, computational timeframes, and the complexity of the code. So, a provisional scope was defined within a MoSCoW prioritization. Weekly progress meetings with the supervisor sometimes involved a renegotiation of the deliverables. This was in response to the timings, and in response to some of the results of intermediate stages. At the supervisor's suggestion, a publicly available dataset of LastFM streaming service user history was used.

Table 1. MoSCoW prioritization

PRIORITY LEVEL	TASK
Must (60%)	• Weekly meetings with supervisor • Contingency time (every sprint) • Learn Python (most sprints) • Perform literature review (over several sprints) • Write Project Plan Pitch • Write Project Plan Report • Clean data • Learn about Neural Networks (NN) • Learn about Skip-Gram (SG), Continuous-Bag-of-Words (CBOW), bagged-prod2vec (B2V) • Learn about collaborative-filtering (CF) • Write SG, CBOW, B2V equivalent: playlist2vec code • Research and select existing libraries for NN • Combine existing NN library with my SG, CBOW, B2V code • Research and select existing libraries for CF • Transform both datasets into collaborative-filtering matrix • Write and execute algorithm test & debugging code • Code, run training and hyperparameter grid-search experiments • Code and run prediction evaluation methods on NN • Code and run prediction evaluation methods on CF model • Compare with collaborative-filtering model • Write presentation • Write report
Should (20%)	• Assess user2vec (U2V) • Assess transfer-learning between models for different datasets • Assess sub-sampling (SS)
Could (20%)	• Include metadata into P2V vectors (Artist, time of day, day of week, sequence) • Compare closest song-vector method to partitioning into clusters and then choose song-vector from closest cluster (as per Yahoo email ads) • Repeat modelling with no negative-sampling • Write code for NN instead of using a library

PRIORITY LEVEL	TASK
Wont (0%)	• Investigate social-engagement techniques as an alternative method to encourage user subscriptions • Respond to cold start and warm start • Assess song addition methods (i.e. after a model is trained) • Assess different cost functions • Assess different non-linear output activation functions • Assess multiple hidden layers in NN • Optimize code efficiency from a hardware perspective • Hybridize with content-based methods

Table 2. Deliverables

MUST DELIVERABLES	SHOULD DELIVERABLES	COULD DELIVERABLES
Project plan, presentation, report	U2V code, experiments, results, and analysis	Metadata-inclusive code, experiments, results, and analysis
SG, CBOW, B2V code	Transfer-learning code, experiments, results, and analysis	Clustering code, experiments, results, and analysis
Code linking different libraries	SS code, experiments, results, and analysis	No negative-sampling code, experiments, results, and analysis
Algorithm testing code and results		Compare with other model types
NN experimental code and results for both datasets		Original NN code
NN prediction analysis code, results and evaluation		
CF experimental code and results for both datasets		
CF prediction analysis code, results and evaluation		
Prediction performance comparison between and CF		
Results and recommendations for music recommender hyperparameters		
Final talk and report		

SUBSTANTIAL CHANGES FROM INITIAL SCOPE:

1. only used one dataset
2. user2vec was performed
3. hyperparameter recommendations were not made
4. one-to-one song recommendations were assessed
5. weight initialization methods were not assessed
6. the collaborative filtering method was original code, not an existing library

PROJECT METHODOLOGY

To assess possible solutions to the problem of inaccurate music recommendations, this project trained and tested different models for a sample of music streaming listener history. Using historical data to develop and evaluate machine learning models is a common data science practice.

Music website LastFM release an anonymised data set of user-listening history. The full text file contains twenty million play-events for ninety-two users. Each row of the data represents a single play-event. It includes a user identifier, the time and date of the play-event, a hexadecimal identifier for the song, as well as the names of the artist and the song.

The first step of the pre-processing was to remove the hexadecimal feature from the data. The artist and song name strings were then concatenated to give a unique identifier for each song. The songs for each user are in reverse chronological order, so these were reversed again to give the correct sequence of songs for each user. The full listener history was too large for the hardware available, so a subset needed to be selected. The number of play-events and average play count for each song were calculated for every user in the data (see

Figure 7). Eight users with similar play-event counts, middle range average play counts, and a combined 515000 play-events were selected.

```
>>> # calculate actual playevent-count
... for i in [1, 11, 20, 21, 32, 66, 67, 73]:
...     totalSongs = len(UserSonglist.iloc[i, :].dropna())
...     uniqueSongs = UserSonglist.iloc[i, :].dropna().nunique()
...     print("User", i+1,
...           "#play-events", totalSongs,
...           "average", round(totalSongs/uniqueSongs, 1))
...
User 2 #play-events 57438 average 6.7
User 12 #play-events 75876 average 5.8
User 21 #play-events 70446 average 7.0
User 22 #play-events 51474 average 8.8
User 33 #play-events 96436 average 13.3
User 67 #play-events 51367 average 7.1
User 68 #play-events 74364 average 4.6
User 74 #play-events 73724 average 4.6
```

Figure 7. Total and average play-events per user

After removing the rest of the users, the data was converted into the formats required for the standard word2vec models, the bagged-prod2vec model, and the pseudo-rating collaborative filtering model. The word2vec model's input was a list of lists, where each outer list element represented a user, and the inner list was the sequence of songs they listened to. The bagged-prod2vec was similar, except the outer list was not just a particular user but a particular user's particular listening session. These were calculated as any sequence of songs where each adjacent pair of songs began within fifteen minutes of each other. The pseudo-ratings were stored in a matrix. A row for each user, and a column for each unique song in the dataset. The elements are a listen count for the user-song pairing.

For the word2vec models, the next step was to create a pair of Python dictionaries, one to convert each unique song into a numeric identifier, and the

other to do the reverse. The sequences of numeric encoded songs were then input to either the SkipGram or CBOW functions to create the input-prediction pairs, and a corresponding Boolean to indicate whether it is a correct or incorrect prediction. Hyperparameters grid searches were performed to find the optimum combinations required to give the best models, measured via the prediction accuracy of the separate testing partition.

Using these hyperparameter sets, models were built and the song embedding vectors were saved for both the SkipGram and CBOW variants. The testing accuracies generated during and at the end of the model training were used as the assessment for one-to-one predictions.

The listening history for each user was used to generate their user vector. The average of the weighted total of their song history was calculated with the song embedding vectors. For each user, the cosine proximity for each song was calculated and saved.

To qualitatively assess the user2vec variant, the following tests were performed for each of the users. The twenty closest songs were output and then compared to the user's history to see if the songs had been listened to before. Then, the twenty most listened to songs for the user had their cosine similarity displayed and averaged. Next, the twenty most similar songs in the CBOW vector space had their similarity looked up in the SkipGram vector space, and vice versa.

The final user2vec analysis was to investigate the genres of songs with similar vectors. The most listened to song from the whole data set had its ten closest songs in each vector space calculated, and separately, the same was done for a mid-level popularity song. Then, all the song's genres were collated from a

music database. The genres for the recommended songs were compared to the genres of the original song for each vector space.

The bagged-prod2vec variants were only investigated during the training of the models. Their inferiority was substantial at this stage, so no further analysis was required.

The pseudo-rating collaborative filtering matrix was then evaluated. The lack of overlap between users caused a major problem with this model type. Typically, for collaborative filtering, the similarity between users is calculated as the similarity between their listening histories (the respective rows of the matrix). But, with so few songs shared between users, there were too few common songs to calculate user similarities. At the supervisor's suggestion, the user2vec similarities were used instead. All songs (i.e. columns) with fewer than four listeners were dropped from the matrix, leaving eleven unique songs. Each user's listen counts for each song were mean-centred to give the pseudo-ratings from each user a similar scale. Then, for each song in each user's row, a rating prediction was calculated with the other user's pseudo-ratings, weighted by the similarity of the pairwise user vectors. These predictions were plotted with the actual pseudo-ratings to compare predictions to reality.

SYSTEM DEVELOPMENT

The implementation environment, language, packages, code, methods, parameter settings and user manual will be included with the separate technical submission, as discussed with the Supervisor.

KEY RESULTS

1. Word2vec gives accurate music recommendations
2. CBOW give more unexpected suggestions than SkipGram

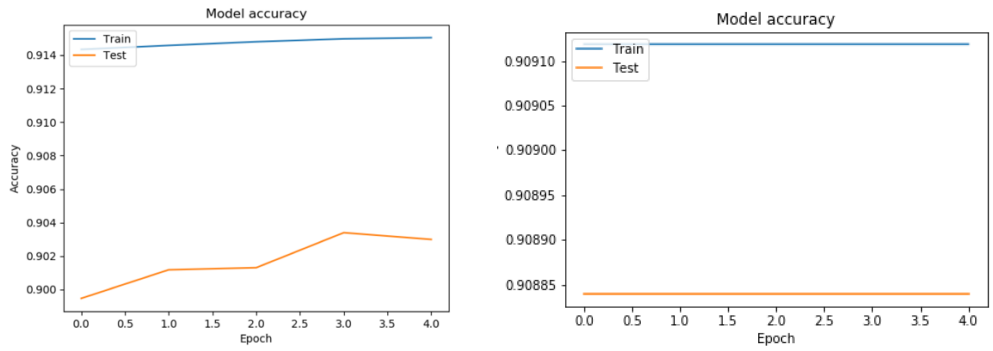
3. Bagged-prod2vec is effective but much less accurately
4. User2vec is functional but not quantifiable without experimental testing
5. Pseudo-rating collaborative filtering does not work

NB These results are specific to the dataset used for this project, 2.5% of the LastFM user listening history.

EVALUATION & DISCUSSION

Word2vec works well as a music recommender. Skipgram and CBOW have very different predictions, but similar accuracy of about 90% for one-to-one song predictions. CBOW predictions were less obvious than those of SkipGram. Bagged-prod2vec worked, but its one-to-one accuracy was lower at 76%. The user2vec models had very different accuracy for the different users. Collaborative filtering did not work.

Figures 8 and 9 show the test set accuracy during training of a SkipGram and CBOW model. Despite the top-heavy data, 90% of the time, both the SkipGram and CBOW models could accurately predict if a random pair were a correct or incorrect pairing.



Figures 8 and 9. SkipGram and CBOW training accuracies

Some models, particularly with CBOW, seemed to barely improve during training. In this model, improvement was only noticeable from the "model loss",

a measure of the difference between the actual and ideal output vector (Figure 10). Both models worked best with a window size of three, and with ten incorrect samples for each correct one. So, each of the five hundred thousand play-events had sixty training pairs, and on average, each unique song had a little more than four-hundred training pairs. This could be the reason why the accuracy could become so high so early in the first epoch of CBOW model training.

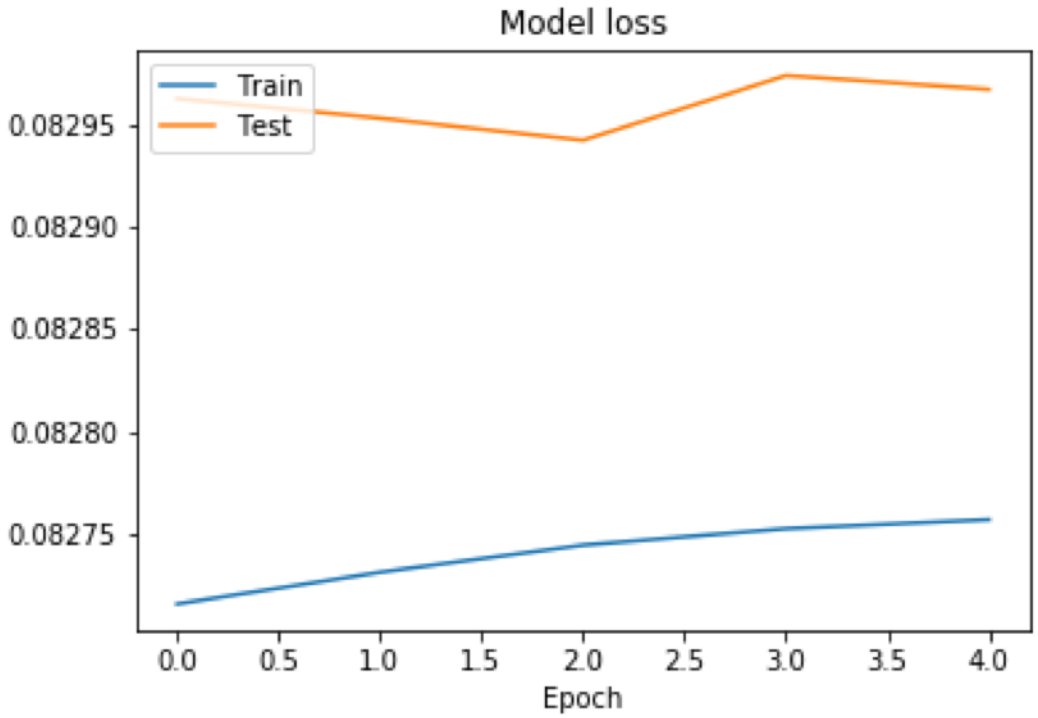


Figure 10. CBOW training model loss

Figure 11 is an example of how significant the window size and ratio of incorrect samples could be. This model used a window size of one and ratio of one incorrect pair for each correct pair. The accuracy of this model fluctuated between slightly below and slightly above 50%. Keeping all other hyperparameters the same, just changing the window and negative sampling rate would reduce the model's predictions from 90% to 50%.

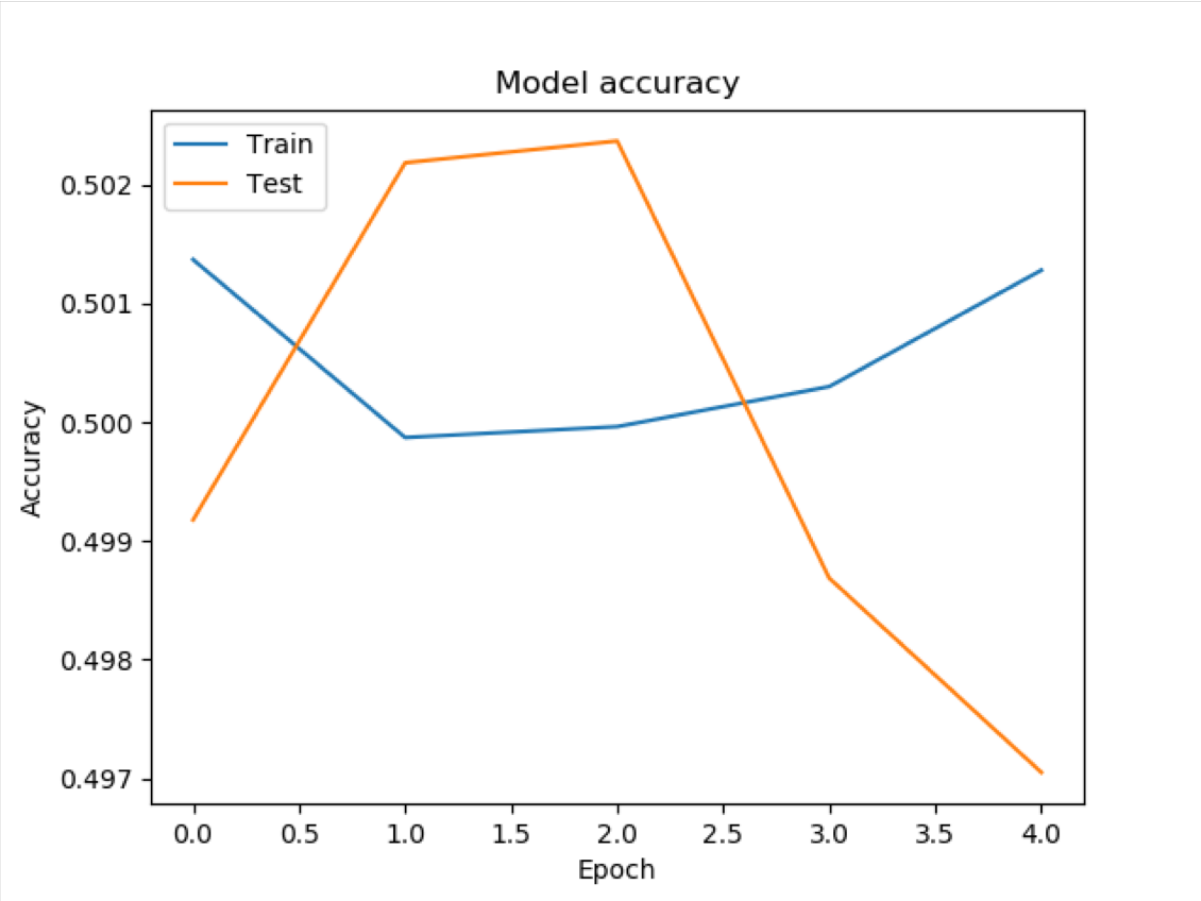


Figure 11. Model training accuracy of window-size 1 with batch-size of 1

There were also hyperparameters to tune for the model training process, both the architecture of the neural net and some within iteration parameters. The most significant were the number of neurons (i.e. hidden layer elements), batch-size (it was more accurate to train with multiple pairs at once), and epochs (the number of times to cycle through the full set of input-prediction pairs).

```
# Hyperparameters~~~~~~~~~~~~~
optimizer = ['Adam', 'SGD', 'RMSprop', 'Adagrad',
             'Adadelta', 'Adamax', 'Nadam']
activation = ['relu', 'softmax', 'relu', 'sigmoid',
              'hard_sigmoid', 'linear', 'softplus',
              'softsign', 'tanh']
neurons = [10, 40, 100]
loss = ['mean_squared_error', 'mean_absolute_error']
batch_size = [1, 1000, 5000, 50000]
epochs = [2, 6, 10]
validation_split = [0.1, 0.3]
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 12. Hyperparameter sets for early models

The functions for the training cost, optimization and activation were found to have little effect on training time or accuracy. Both models worked best with forty neurons, but CBOW needed smaller batches and fewer epochs than SkipGram. This paper had sought out to make hyperparameter suggestions for future word2vec music recommenders. However, it was discovered that the optimal hyperparameters were highly dependent on the dataset used. Early on during the project, while trying to determine the largest dataset which could run on the hardware available it became apparent that even small changes to the data would drastically affect the hyperparameters. So, this was discussed with the Supervisor and removed from the project scope

```
# SkipGram Hyperparameters~~~~
neurons = 40
epochs = 5
loss = 'mean_squared_error'
optimizer = 'Adam'
batchSize = 1000
validationSplit = 0.1
activation='relu'
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
# CBOW Hyperparameters~~~~~~~~
neurons = 40
epochs = 2
loss = 'mean_squared_error'
optimizer = 'Adam'
batchSize = 500
validationSplit = 0.1
activation = 'relu'
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figures 13 and 14. Optimum hyperparameters for SkipGram and CBOW models

Next were the user2vec models. Figures 15 and 16 shows the three songs closest to user 6's user vector in the two vector spaces. User 6 has listened to the first

two song recommendations but not the third from the SkipGram vector space in the left-hand Figure, and listened to the second song but not the other two from the CBOW recommendation vector space in the right-Figure. On average, 70% SkipGram user2vec predictions and 90% of CBOW user2vec predictions were not in the user's listening history. Of note is that even though both models had a similar one-to-one song prediction accuracy during the training of the models, the similarities between songs and user's in the vector spaces are very different. The SkipGram model found the most similar songs to users had a cosine similarity of 96%, but the CBOW model were in the low 60%. This difference is not indicative of better matches in the SkipGram model, but solely a result of the vector space architecture. The differences in similarity between objects in a vector space are only comparable within the vector space, not between models.

```
Songs closest to user 6 :

 0.9641          Cellard'Or: Scheinwelt
                         In user's history

 0.9626          Bonobo: Nothing Owed
                         In user's history

 0.9619          Stephen Sondheim: Johanna
NOT              in user's history
```

```
User 6  user2vec consists of  16225 songs.

Closest song runtime: 261.30839014053345
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Songs closest to user 6 :

 0.6095          The Expos: Before Breakfast
NOT              in user's history

 0.6016          Bums: Elefanten Bums
                         In user's history

 0.6011          The Adored: We Don'T Want You Around
NOT              in user's history
```

Figure 15 and 16. User 6's closest songs to their user2vec embedding in the SkipGram and CBOW vector spaces

Figures 17 and 18 show the similarities for user fives most listened to songs with their user vectors, in the SkipGram and CBOW vector spaces respectively. A significant difference between the two is that SkipGram songs are closer to the maximum similarity for the user (high 80% from a maximum of mid 90%) than are the CBOW songs (at most 20% from a maximum of low 60%). This aligns with a recurring aspect of word2vec models which is that CBOW gives less obvious recommendations than do SkipGram models.

```
Similarities of user 5's 20 most frequently listened to songs:
#0: 0.89916 Sheryl Crow: Real Gone
#1: 0.87302 Jupiter One: Mystery Man
#2: 0.84302 Coldplay: Viva La Vida
#3: 0.88664 Red Hot Chili Peppers: Road Trippin'
#4: 0.87629 Barenaked Ladies: If I Had $1000000
```

Figure 17. SkipGram similarities for user 5's user vector to their 5 most listened to songs

```
Similarities of user 5's 20 most frequently listened to songs:
#0: 0.20497 Sheryl Crow: Real Gone
#1: 0.10441 Jupiter One: Mystery Man
#2: 0.12693 Coldplay: Viva La Vida
#3: 0.05156 Red Hot Chili Peppers: Road Trippin'
#4: 0.08445 Barenaked Ladies: If I Had $1000000
```

Figure 18. CBOW similarities for user 5's user vector to their 5 most listened to songs

Figure 19 shows the next user2vec investigation technique. In the CBOW vector space, the twenty songs closest to the user 0's user vector were found. Then, the similarities between user 0 and these songs in the SkipGram vector space were calculated. The reverse process with the closest SkipGram songs evaluated in the CBOW vector space was performed too. The results for this are not what was expected. The literature review and the two previous user2vec analyses suggested that the CBOW method would give more unexpected recommendations. However, the CBOW closest songs each have a very high similarity in the SkipGram vector space, in this example an average of 89% out of a maximum similarity of 96%. This suggests that the CBOW model recommendations are the more obvious suggestions, as they are so similar in the SkipGram space. Whereas the SkipGram closest songs, when measured in the CBOW vector space have much lower similarities, 10% out of a maximum of 60%. This would suggest that the SkipGram song recommendations are more unexpected. No reason for this discrepancy was created or found.

```
                                    ########################
                                           User 0
                                    ########################

CBOW model similarity of top 20 SG songs:          SG model similarity of top 20 CBOW songs:
0.0655                                              0.8798
0.0781                                              0.9533
0.0931                                              0.9544
0.1031                                              0.8203
0.243                                               0.9101
0.1446                                              0.8314
0.0414                                              0.8581
0.095                                               0.9548
0.0005                                              0.9085
0.0966                                              0.8741
0.1938                                              0.8531
0.0516                                              0.9546
0.1065                                              0.8825
0.0146                                              0.8958
0.0138                                              0.9154
0.151                                               0.9841
0.0435                                              0.8208
0.1436                                              0.8361
0.1507                                              0.9863
0.1951                                              0.8932
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CBOW Average relevance: 0.1013                      SG Average relevance: 0.8983
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 19. Similarities of closest songs in one vector space to a user, as measured in the other vector space

Figure 20 is an example of the expected difference between the SkipGram and CBOW models. The red, blue, and yellow highlights are for the genres of different songs. The most listened to song in the data set was by "Ryoujoku No Ame" by Japanese heavy metal band Dir En Grey. So, the closest songs to it in both the SkipGram and CBOW vector spaces were calculated. Then online music database Discogs was consulted to find the genre of each of the closest songs. In yellow are the skipgram songs. The original song was heavy metal, the skipgram recommendations are mostly the same or similar genre as the original song. Whereas the CBOW recommendations, in blue, are very different genres to heavy metal. This fits with the thrust of the research from the literature review that CBOW gives more unexpected recommendations than SkipGram but does not fit with the inferences from the previous analysis of user2vec above.

Closest songs most listened-to song: Dir En Grey: Ryoujoku No Ame (Heavy Metal)
(listen count: 880)
~~~~~~

SG: 雅-Miyavi-: Kekkonshiki No Uta ~Kisetsu Hazure No Wedding March~ (Heavy Metal)
CBOW: The Beatles: Got To Get You Into My Life (Pop Rock)

SG: Efdemin: Le Ratafia (Techno)
CBOW: Michiru Oshima,Moscow International Symphony Orchestra: Sad Resolution (Soundtrack)

SG: Jesu: Blind And Faithless (Rock)
CBOW: Jimmy Edgar: Sheer, Make, Serve (Techno)

SG: Mucc: Saishuu Ressha 70'S Ver. (Rock)
CBOW: 高木正勝: Private Drawing (Soundtrack)

SG: Dir En Grey: Ain'T Afraid To Die (Heavy Metal)
CBOW: Wax Poetic: Dreamin' (Electronic)

SG: Zbigniew Preisner: Piotr (Soundtrack)
CBOW: Peaches: Hit It Hard (Electro)

SG: Mucc: Kasa Ga Nai (Rock)
CBOW: Hauschka: Alma (Classical)

SG: Placebo: Where Is My Mind (Rock)
CBOW: Sum 41: Dave'S Possessed Hair / It'S What We'Re All About (Pop Punk)

SG: Sads: Boukyaku No Sora (Rock)
CBOW: Nat King Cole: Smile (Jazz)

Figure 20. Closest songs and their genres for the nine closest songs in each vector space to the most listened to songs
~~~~~~

Also note, that only one of the recommended songs is the same band as the original song, Dir En Grey. This is surprising because Dir En Grey make up five of the nine most listened to songs in the whole dataset (Figure 21), but the models didn't just recommend them, a marked improvement compared to a greedy algorithm.

```
('Dir En Grey: Ryoujoku No Ame', 880),
("Dir En Grey: Ain'T Afraid To Die", 855),
('雅-Miyavi-: Itoshii Hito', 753),
('植松伸夫: Somnus Memoria', 730),
('雅-Miyavi-: Kekkonshiki No Uta ~Kisetsu Hazure No Wedding March~', 630),
('Dir En Grey: Conceived Sorrow', 613),
("Mucc: Saishuu Ressha 70'S Ver.", 560),
('Dir En Grey: Namamekashiki Ansoku, Tamerai Ni Hohoemi', 505),
('Dir En Grey: The Pledge', 467),
```

Figure 21. The most listened to songs in the dataset and their listen counts

Figure 22 shows the genres of the nine closest songs to a less popular song, "The Arms of Orion by Prince". This time, the genres of the CBOW songs are more similar to the original. Prince's song is a Pop Ballad, and all the CBOW songs are Rock, Indie Rock or Pop Ballad. The only surprising suggestion is the Techno song recommended by the SkipGram model, the sixth yellow highlight.

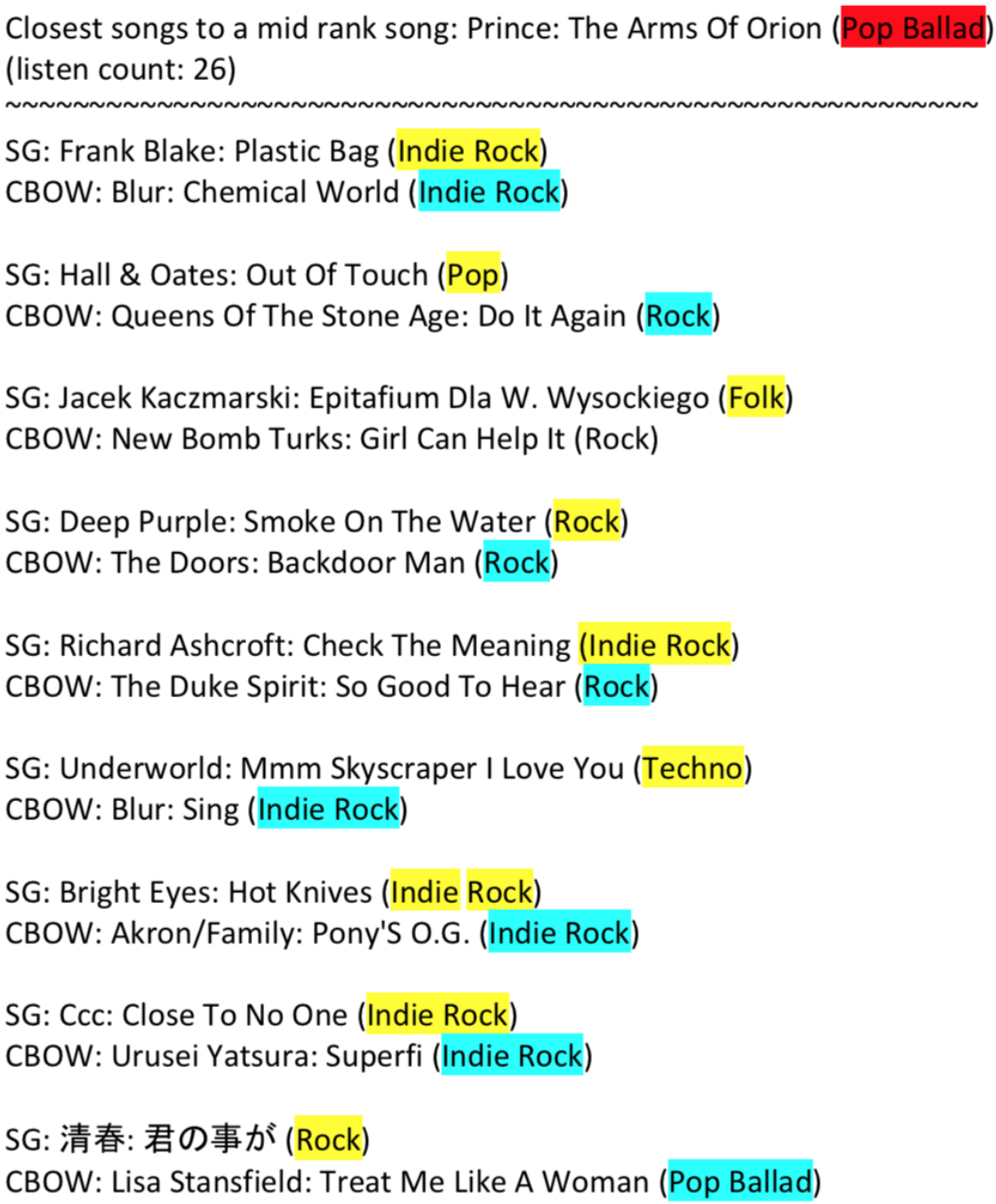
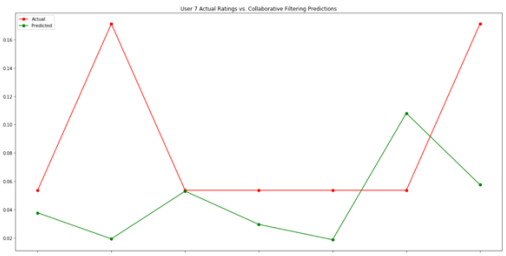
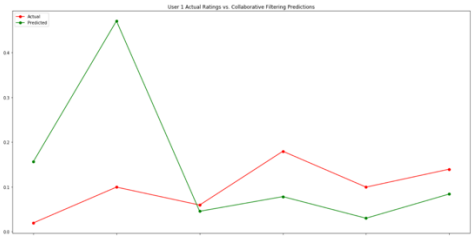


Figure 22. Genres of the closest songs to Prince's "Arms of Orion"

Collaborative filtering was not functional with this data. Figures 23 and 24 show the difference between the actual pseudo-rating and the predicted pseudo-rating for one user each. The red line is the user's actual normalised listen count, the green the predicted. The plot on the right show some similarity in shape but

the magnitudes are wrong. This may be a function of the small dataset or may be an indication that the method doesn't work





Figures 23 and 24. Actual and predicted pseudo-ratings for users 7 and 1

CONCLUSION

This paper set out to investigate the efficacy of the word2vec algorithm as a recommender for online music streaming services. It tested four variants of the algorithm. It found that SkipGram and Continuous-Bag-of-Words gave similar accuracies for one-to-one song predictions, with most of the evidence suggesting that the CBOW recommendations are less obvious. It then found that the bagged-prod2vec model gave one-to-one predictions at a lower accuracy. The user2vec method was investigated, but without any definitive testing method. From qualitative analysis it appears to work, but it would require A-B testing on real users to quantify. The pseudo-rating collaborative filtering matrix did not work, but there is some suggestion in the results that it may with more data.

RECOMMENDATIONS

This paper has shown that word2vec models can be applied to music recommenders for online streaming services. Despite some counter evidence, it suggests that CBOW models will give less obvious suggestions. It has been shown in other word2vec applications that more surprising recommendations are more likely to encourage users to spend more money (Zalmanson, 2016), so this project would recommend one-to-one predictions based on a CBOW model.

This study also creates several questions for future research in this area. The relatively small dataset used may have led to results which do not hold when the dataset scales up. It is possible that bagged-prod2vec predictions will improve relative to other word2vec with more data, or that SkipGram and CBOW will diverge. User2vec shows some promise but needs controlled experiments, rather than data analysis to accurately assess. The pseudo-rating collaborative filtering may also work with more data.

The major question posed by this project is about the difference in suggestions between the SkipGram and CBOW models. Most of the analysis fit with the literature review that CBOW is more unexpected in its recommendations, but for each of the eight users in this model, comparing the user2vec suggestions in the other vector space gave counter intuitive results.

REFERENCES

Arditi, D. (2018). Digital subscriptions: The unending consumption of music in the digital e ra. *Popular Music and Society*, *41*(3), 302-318.

Barkan, O., & Koenigstein, N. (2016, September). Item2vec: neural item embedding for c ollaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)* (pp. 1-6). IEEE.

Caselles-Dupré, H., Lesaint, F., & Royo-Letelier, J. (2018, September). Word2vec applied to recommendation: Hyperparameters matter. In *Proceedings of the 12th ACM Conference on Recommender Systems* (pp. 352-356). ACM.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303-314.

Elsafty, A., Riedl, M., & Biemann, C. (2018). Document-based Recommender System for Job Postings using Dense Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)* (Vol. 3, pp. 216-224).

Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., & Sharp, D. (2015, August). E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1809-1818). ACM.

Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., & Sharp, D. (2015, August). E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1809-1818). ACM.

Hall, P. (2019, March 12). *Apple Music vs. Spotify*. Retrieved from https://www.digitaltrends.com/music/apple-music-vs-spotify/

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Karam, R. (2017, December 7). *Using Word2vec for Music Recommendations.* Retrieved from https://towardsdatascience.com/using-word2vec-for-music-recommendations-bb9649ac2484

Liang, D., Zhan, M., & Ellis, D. P. (2015, October). Content-Aware Collaborative Music Recommendation Using Pre-trained Neural Networks. In *ISMIR* (pp. 295-301).

McFee, B., Barrington, L., & Lanckriet, G. (2012). Learning content similarity for music recommendation. *IEEE transactions on audio, speech, and language processing*, *20*(8), 2207-2218

Meyer, D. (2016). *How exactly does word2vec work?*. Retrieved from https://pdfs.semanticscholar.org/49ed/be35390224dc0c19aefe4eb28312e70b7e79.pdf

Mikolov et al. (2013) *US Patent No. 9,037,464 B1*. Retrieved from https://patents.google.com/patent/US9037464B1/en

Orendorff, A. (2019, February 14). *Global Ecommerce Statistics and Trends to Launch Your Business Beyond Borders*. Retrieved from https://www.shopify.com/enterprise/global-ecommerce-statistics#1

Pitney Bowes Global Ecommerce Study Finds We Are Shopping Online More Frequently and Frustrated More Often. (2018, October 24). Retrieved from https://www.pitneybowes.com/us/ecommerce-study/press-release.html

Rendle, S. (2010, December). Factorization machines. In *2010 IEEE International Conference on Data Mining* (pp. 995-1000). IEEE.

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Van den Oord, A., Dieleman, S., & Schrauwen, B. (2013). Deep content-based music recommendation. In *Advances in neural information processing systems* (pp. 2643-2651).

Vasile, F., Smirnova, E., & Conneau, A. (2016, September). Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems* (pp. 225-232). ACM.

Wang, X., & Wang, Y. (2013, November). Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 627-636). ACM.

Wang, X., Wang, Y., Hsu, D., & Wang, Y. (2014). Exploration in interactive personalized music recommendation: a reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *11*(1), 7.

Zhang, S., Yao, L., & Sun, A. (2017). Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*.