CS464 Machine Learning
Final Report

“Song Lyrics Decade and Popularity Classification”

Group Members
Elif Kurtay 21803373
Cansu Moran 21803665
Atakan Dönmez 21803481
Öykü Irmak Hatipoğlu 21802791
Elif Gamze Güliter 21802870

1. Introduction

In this project, we performed two different classification tasks on song lyrics. First, we classified song lyrics into decades to observe what kind of distinction the song lyrics have in different decades, for example 70s, 80s or 90s. The second classification task we performed was whether that song is popular in a particular decade or not. We used different text vectorization techniques with two models to perform these classification tasks: Recurrent Neural Network(RNN) and Multinomial Naive Bayes model.

RNN is a class of artificial neural networks that is popular in text classification tasks. It processes sequence input by iterating through the elements and passes the outputs from one timestep to their input on the next timestep. In our project we used RNN with Long Short-Term Memory (LSTM). Naive Bayes is a probabilistic classifier based on applying Bayes’ Theorem with independent assumptions between the features. In our project, we used a Multinomial Naive Bayes classifier.

RNN with the LSTM model achieved 33% accuracy on period classification and 73% accuracy on popularity classification(70s popularity classification) on our test set with the GloVe word vectorization technique whereas it achieved 32% accuracy on period classification and 69% accuracy on popularity classification(90s popularity classification) on our test set with the Word2Vec word vectorization technique. The Multinomial Naive Bayes classifier, on the other hand, achieved approximately 38% accuracy on period classification and 71% on popularity prediction (70s popularity classification).

The information about the data, methods used, models and the results are discussed in the following sections.

2. Problem Description

The problem we chose is to predict the period (decade) of a given song based on its lyrics and to predict its popularity (whether it is popular or not) in any given period. In order to achieve that, we decided to use text classification techniques (Naive Bayes and Recurrent Neural Networks) in Machine Learning by forming our own dataset with songs from Spotify Data 1921-2020 dataset on Kaggle[1] and lyrics of these songs retrieved with Musixmatch API [2].

Whether a song will be more or less popular in a different decade was an interesting question for us. However, when we get a prediction from our models of a song's popularity in a different period, we do not have ground truth values for that result and thus cannot provide a metric of accuracy or strength of the guess. Therefore, we decided to classify the popularity of every song in their own periods. This way, we are providing results and scores of our models. However, in order to satisfy our curiosity, we have used our pre-trained models to see the results of popularity classification of songs from other periods which can be performed with the models provided in the scripts. Those results are not included in this report.

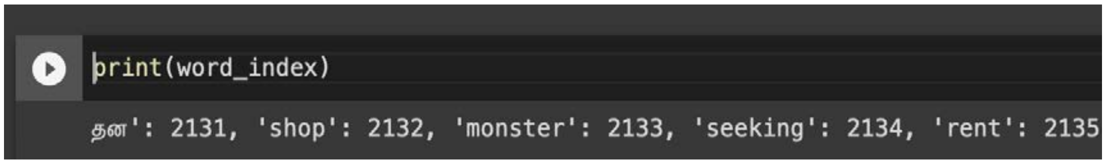
3. Methods

3.1. About the Data

We used the Spotify Dataset 1921-2020 on Kaggle[3]. The dataset contains over 169.000 songs from the year 1921 to the year 2020, each song entry includes information on the song such as its unique Spotify id, names of the artists, name of the song, tempo, loudness, etc. However, it does not include the lyrics of the songs nor the language of songs. Hence, songs from the dataset were needed to be selected on the grounds of availability of lyrics and being in the English language before the preprocessing.

[1] E. Negi, "Spotify-data 1921-2020," *Kaggle*, 29-Aug-2020. [Online]. Available: https://www.kaggle.com/ektanegi/spotifydata-19212020. [Accessed: 30-Oct-2021]
[2] "Build with lyrics," *Musixmatch Developer*. [Online]. Available: https://developer.musixmatch.com/. [Accessed: 26-Nov-2021].
[3] E. Negi, "Spotify-data 1921-2020," *Kaggle*, 29-Aug-2020. [Online]. Available: https://www.kaggle.com/ektanegi/spotifydata-19212020. [Accessed: 30-Oct-2021].

Since the classification is performed on English lyrics, any song in a language other than English is removed from the dataset by the use of a language detection model in Python's "langdetect" package [4]. This library is a machine learning classification library. Therefore, it is observed that results on the same data can sometimes alternate. However, since we could not modify the algorithm or have the time for a manual control, we accepted possible wrong labeling as noise in the data. The reason we did not first retrieve the lyrics was due to time constraints because the lyric extraction API was taking very long. After some calculation, we realized that retrieving all the lyrics in the database would have taken approximately 12 days. Therefore, we wanted to reduce the data as much as possible before extracting the lyrics. Hence, the language detection model is performed on the name of the song and in case of no result (could be due to no letters in the song name), on the artists' names. This technique eliminated errors in labeling. However, since songs with English names do not only contain English words, the language accuracy is not perfect as seen from the resulting data. In Figure 1, it can be noticed that the first word is not English. We tried to detect the languages once again after lyrical data was retrieved but this time full lyric predictions labeled everything in English and word by word detection deleted a lot of proper English words. Due to the unstable nature of the language detection model on single words or too long sequences, we decided to keep the lyrics with the noise.



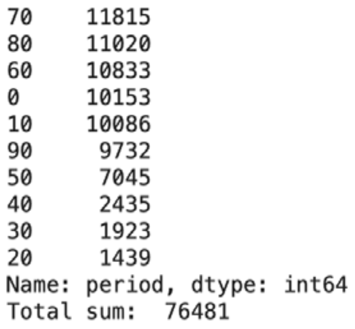
```
print(word_index)
தன': 2131, 'shop': 2132, 'monster': 2133, 'seeking': 2134, 'rent': 2135
```

Figure 1: An example of words found in the final dataset used for training the models

Before starting to retrieve lyrics on the songs, it is noticed that some tracks are completely instrumental and thus do not have any lyrics. However, two available features in the dataset were Spotify's audio features: speechiness and instrumentalness. According to Spotify's Web API Documentation, speechiness values under 0.33 are most likely to represent non-speech-like tracks and instrumentalness values above 0.5 are intended to represent instrumental tracks without vocal sounds, but confidence is higher as the value approaches 1.0 [5]. After trials and adjustments done on the dataset based on these values, the optimal parameter is found to be selecting the songs with instrumentalness value greater than 0.5 while removing others from the dataset. The number of songs in the selected method can be seen in Figure 2. The reason for this selection comes from the fact that trying to perfect all lyrical output caused us to lose all our data and the remaining numbers can be observed in Figure 3. This little amount of data would not be enough to train a RNN model. Even Naive Bayes could not do any selection better than random

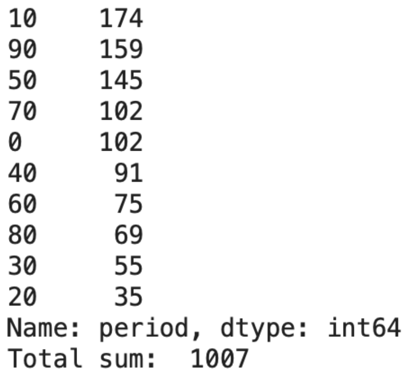
[4] "Langdetect." *PyPI*, [Online]. Available: https://pypi.org/project/langdetect/. [Accessed: 30-Oct-2021].
[5] "Web API reference: Spotify for developers," *Home*. [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features. [Accessed: 23-Dec-2021].

for this amount of data. However, the selected criteria minimized noisy data while keeping the numbers appropriate for Machine Learning models.



```
70    11815
80    11020
60    10833
0     10153
10    10086
90     9732
50     7045
40     2435
30     1923
20     1439
Name: period, dtype: int64
Total sum:  76481
```

Figure 2: Number of English songs with instrumentalness value less than 0.5 per period.

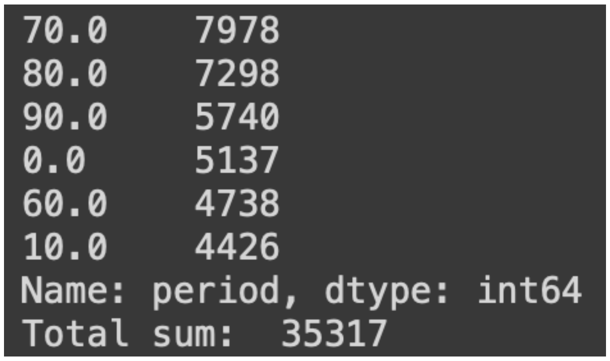


```
10    174
90    159
50    145
70    102
0     102
40     91
60     75
80     69
30     55
20     35
Name: period, dtype: int64
Total sum:  1007
```

Figure 3: Number of songs in each period with instrumentalness value lower than 0.5 and speechiness value larger than 0.33.

However, after filtering the data based on language, and instrumentalness, the number of songs reduced significantly, especially for earlier periods and we decided to perform 6 label classification per decade starting from 1960's to 2010's for the period classification task. In order to retrieve the lyrics in the filtered dataset, we used MusixMatch API[6]. Figure 4 shows the number of lyrics we were able to retrieve using the API.

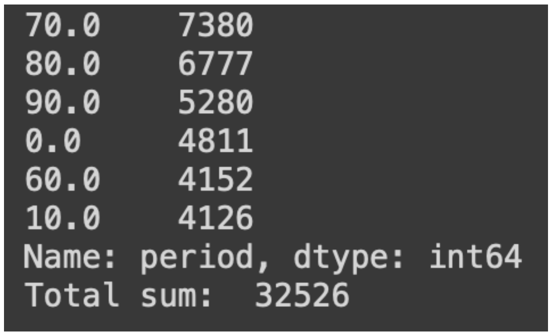
[6] "Build with lyrics," *Musixmatch Developer*. [Online]. Available: https://developer.musixmatch.com/. [Accessed: 26-Dec-2021].



```
70.0    7978
80.0    7298
90.0    5740
0.0     5137
60.0    4738
10.0    4426
Name: period, dtype: int64
Total sum:  35317
```

Figure 4: Number of lyrics retrieved per period

After the completion of the dataset with lyrics, data preprocessing is done to prepare the data for text vectorization techniques to be used for their related Machine Learning models. At the first stage, the songs in the dataset where the lyrics appeared as if they were retrieved successfully but were actually unsuccessful with a float value or a "nan" value as lyrics were cleaned. Figure 5 shows the number of remaining lyrics after cleaning the songs. While we were able to clean such unsuccessful results with nan value as lyrics from the dataset, we still had noise in the data that we weren't able to clean. In some cases, the API retrieved data that was a string (hence we couldn't filter it in the same way we filtered the nan values) but not lyrics of that particular song and had random information in it. Since detecting such noise was not possible, these noises were left in the data. After, the lyrics are cleaned by the removal of punctuations, the removal of stopwords (with an addition of new lyrical stop words such as "verse" and "chorus"), and lemmatization of the words.



```
70.0    7380
80.0    6777
90.0    5280
0.0     4811
60.0    4152
10.0    4126
Name: period, dtype: int64
Total sum:  32526
```

Figure 5: Number of songs remaining in the cleaned dataset

Labels have also gone through preprocessing, since there was neither a period feature nor a binary popularity label. Period information is extracted from the release year, if not available the release date of the tracks. The popularity score (0-100) available in the dataset is transformed into a binary popularity label according to the average popularity score of each period.

3.2. Text Vectorization and Models

In order to predict the period and popularity of a song, we selected two text classification models: Multinomial Naive Bayes and RNN with LSTM. We wanted to observe the difference neural networks can create on textual classification. For RNN, we used three different word vectorization techniques: Word-to-Vector (Word2Vec), Global Vector (GloVe) and Bag-of-Words (BoW) with Term Frequency-Inverse Document Frequency (TF-IDF). We will use BoW with TF-IDF also for the Multinomial Naive Bayes model.

3.3. Libraries and Tools Used

- The dataset is Spotify Data 1921-2020 dataset on Kaggle [7].
- We used Musixmatch API [8] to extract the lyrics of each song in the Kaggle dataset.
- We used the "langdetect" package found in Python 3.7 in order to determine the language of a song.
- We used the MultinomialNB classifier that sklearn provides as our NB classifier. Furthermore, we used the TfidfVectorizer package that sklearn provides in order to provide a tfidf matrix to the model.
- We used the NLTK library to remove stopwords from our data and lemmatization. The RegEx library named "re" from Python to remove non-alphanumeric characters in the lyrics[9].
- We used sklearn's metric package for scoring the predictions of the Naive Bayes model.
- We used PyTorch to implement the RNN with LSTM.

3.4. Models

3.4.1. Multinomial Naive Bayes Model with TF-IDF

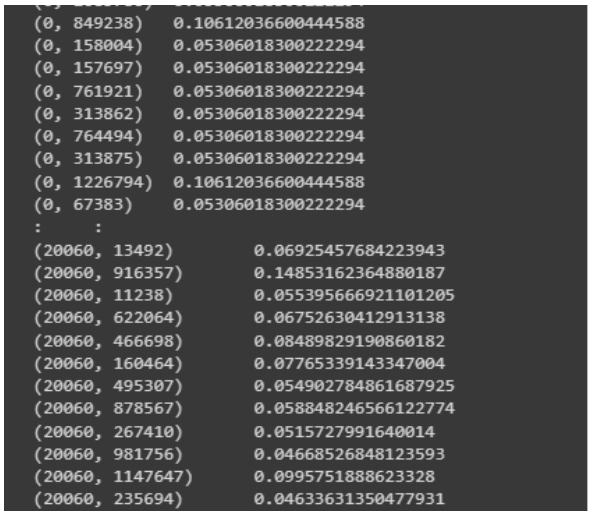
Naive Bayes is one of the best known models for text classification tasks. In our model we tried Bag Of Words with the TF-IDF text vectorization. Term Frequency - Inverse Document Frequency (TF-IDF) is a statistical measurement that reflects how important a word is for a document by looking at occurences of words in a document and among other documents. TF is the measurement of how frequent that term is in a document, IDF is the measurement of how much information that word provides by looking at its occurrences among other documents.

[7] E. Negi, "Spotify-data 1921-2020," *Kaggle*, 29-Aug-2020. [Online]. Available: https://www.kaggle.com/ektanegi/spotifydata-19212020. [Accessed: 30-Oct-2021].

[8] "Build with lyrics," *Musixmatch Developer*. [Online]. Available: https://developer.musixmatch.com/. [Accessed: 26-Nov-2021].

[9] "Re - Regular Expression Operations." Re - Regular Expression Operations - Python 3.10.0 Documentation, [Online]. Available: https://docs.python.org/3/library/re.html. [Accessed: 28-Nov-2021].

Sklearn offers a TF-IDF vectorizer called TfidfVectorizer. There are different attributes of this vectorizer. We experimented on the range of n-grams. We observed that trigram, which is the highest range available, yielded the best results and continued with this range. Using this vectorizer, we produced the TF-IDF matrix for our model. Figure 6 is a snapshot of a TF-IDF matrix of our data. The numbers in the matrix indicate how important a word is for a document. This matrix has a shape of (20061, 1387372) which means that there are 20061 songs where each song has 1387372 features. There are a total of 38615 different words in all the data. If we have selected to perform a unigram selection, the matrix would have the shape (20061, 38615).



```
(0, 849238)   0.10612036600444588
(0, 158004)   0.05306018300222294
(0, 157697)   0.05306018300222294
(0, 761921)   0.05306018300222294
(0, 313862)   0.05306018300222294
(0, 764494)   0.05306018300222294
(0, 313875)   0.05306018300222294
(0, 1226794)  0.10612036600444588
(0, 67383)    0.05306018300222294
:     :
(20060, 13492)        0.06925457684223943
(20060, 916357)       0.14853162364880187
(20060, 11238)        0.055395666921101205
(20060, 622064)       0.06752630412913138
(20060, 466698)       0.08489829190860182
(20060, 160464)       0.07765339143347004
(20060, 495307)       0.054902784861687925
(20060, 878567)       0.058848246566122774
(20060, 267410)       0.0515727991640014
(20060, 981756)       0.04668526848123593
(20060, 1147647)      0.0995751888623328
(20060, 235694)       0.04633631350477931
```

Figure 6: TF-IDF matrix used in the model

We used %80 of the TF-IDF matrix to train our model and the rest for testing. The period classification task has six labels: 60’s, 70’s, 80’s, 90’s,00’s and 10’s. The popularity classification is a binary classification and indicates whether a song is popular in its own period or not.

3.4.2. RNN with LSTM

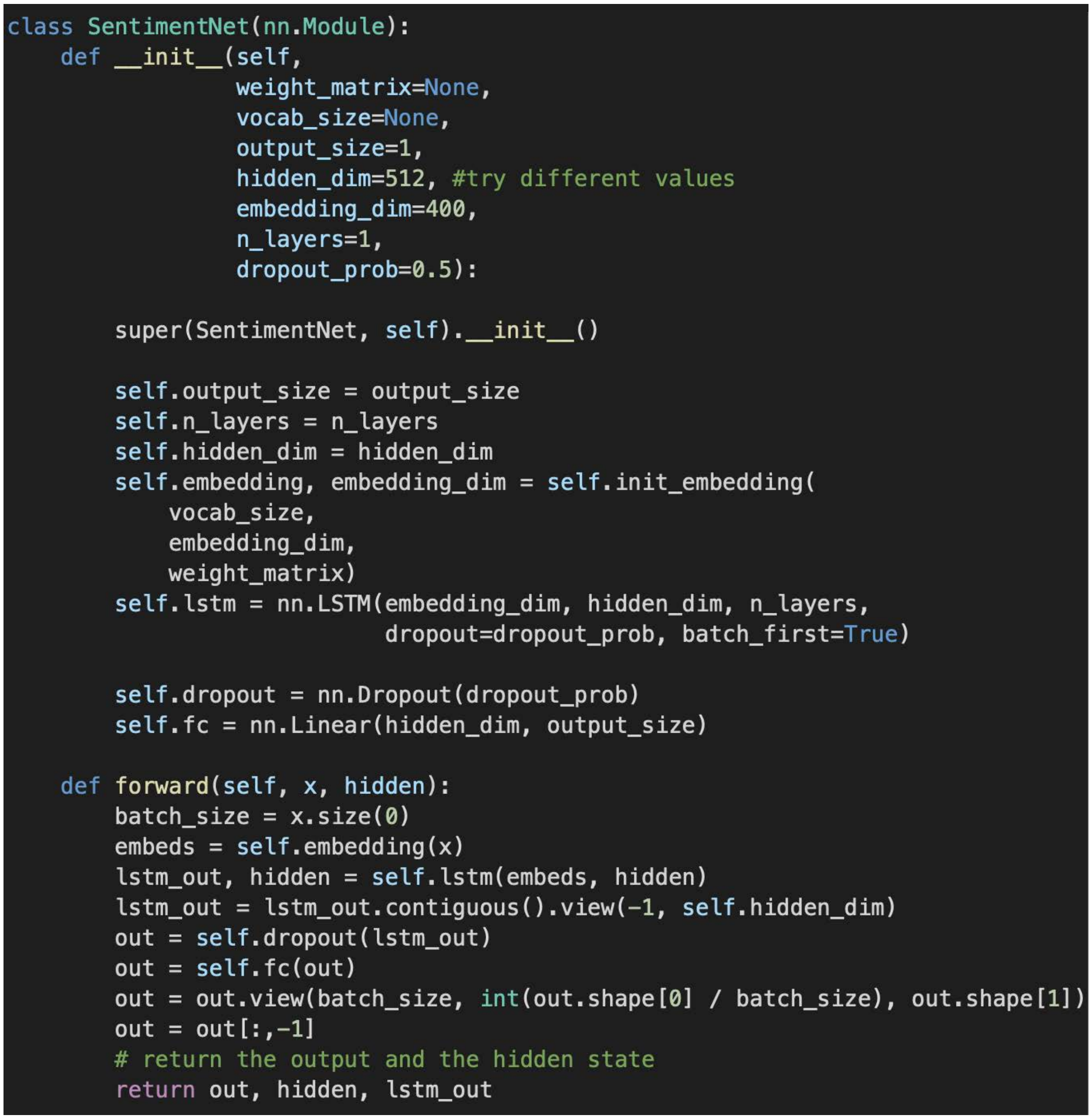
We used RNN with LSTM as our second model for period and popularity classification of lyrics. We chose RNN in addition to the Multinomial Naive Bayes model to observe how neural networks perform compared to traditional machine learning models in a text classification problem. While choosing our RNN model, we took into consideration previous work done on similar lyrics classification problems in addition to pros and cons of different RNN architectures. We observed that RNN with LSTM was a commonly used approach for lyrics classification problems. While our problem has not been implemented before, we were able to find lyrics based genre classification models which we took as a starting point. After observing the abundance of RNN with LSTM models on lyrics based classification problems, we did our research on standard RNNs versus gated RNN architectures in addition to LSTM vs GRU models. Due to the vanishing gradient problem of RNNs with long word sequences[10], we decided to implement a gated RNN which has the property of maintaining information (i.e words in the beginning of the lyrics) for a longer period of time over a standard RNN. When deciding between GRU and LSTM models, we decided to use LSTMs mainly because GRU is a newer model with less resources. In addition, LSTMs are argued to have the capability of maintaining information for longer compared to GRUs, which was a determining factor in our choice. However, since LSTMs are more complex, they are more prone to overfitting, which we have experienced in our experiments (see section Results), which was a limitation of our model.

We used a Neural Network called SentimentNet as our base RNN architecture[11]. This network was originally used for Sentiment Analysis in customer reviews. The network was aimed at the binary classification of positive and negative sentiments and the model was trained using 1 million reviews. This model was given tokenized word sequences as an input without any word embeddings such as Word2Vec or GloVe. It was able to get 93.8% accuracy on its task. For our case, we modified the architecture to include Word2Vec and GloVe embeddings as input weight matrices. We took a lyrics genre classification repository on GitHub that was based on SentimentNet as a reference as their problem was similar to ours[12]. However, when we read the code, we found out that the model was trained to predict if a song's lyrics belong to the Rock genre or not, thus the model was for binary classification. We turned the model into a multiclass classification model by replacing PyTorch's BinaryCrossEntropyLoss with the CrossEntropyLoss (which is NLLLoss and a softmax last layer activation function) and we erased the sigmoid last layer activation function as we already have a softmax function. The Github repository presented that they have achieved at most 92.4% accuracy for 15 epoch and 190k samples on this model using GloVe embeddings.

[10] M. Phi, "Illustrated guide to LSTM's and GRU's: A step by step explanation," *Medium*, 28-Jun-2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed: 24-Dec-2021].

[11] G. Loye, "Long short-term memory: From zero to hero with pytorch," *FloydHub Blog*, 16-Aug-2019. [Online]. Available: https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch. [Accessed: 25-Dec-2021].

[12] Nbandhi, "AI-ml/LyricsAnalysisGenrePredictionLSTM.ipynb at main · NBANDHI/AI-ML," GitHub. [Online]. Available: https://github.com/Nbandhi/AI-ML/blob/main/LyricAnalysis/LyricsAnalysisGenrePredictionLSTM.ipynb. [Accessed: 25-Dec-2021].



```
class SentimentNet(nn.Module):
    def __init__(self,
                 weight_matrix=None,
                 vocab_size=None,
                 output_size=1,
                 hidden_dim=512, #try different values
                 embedding_dim=400,
                 n_layers=1,
                 dropout_prob=0.5):

        super(SentimentNet, self).__init__()

        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim
        self.embedding, embedding_dim = self.init_embedding(
            vocab_size,
            embedding_dim,
            weight_matrix)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,
                            dropout=dropout_prob, batch_first=True)

        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(hidden_dim, output_size)

    def forward(self, x, hidden):
        batch_size = x.size(0)
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)
        out = self.dropout(lstm_out)
        out = self.fc(out)
        out = out.view(batch_size, int(out.shape[0] / batch_size), out.shape[1])
        out = out[:,-1]
        # return the output and the hidden state
        return out, hidden, lstm_out
```

Figure 7: Our modified architecture of the SentimentNet model.

The model starts with an embedding layer which applies the pretrained GloVe and word2vec weights to the tokenized input lyrics sequences. Then, the output of the Embedding Layer is passed to the LSTM followed by a Dropout layer. After that, the nodes are passed to a fully connected layer which outputs 6 nodes for each decade class. The results are passed through a softmax layer when the loss is being calculated. The given model with output size 6 is used for period classification. For popularity classification on the other hand, the same model was used with output size 2.

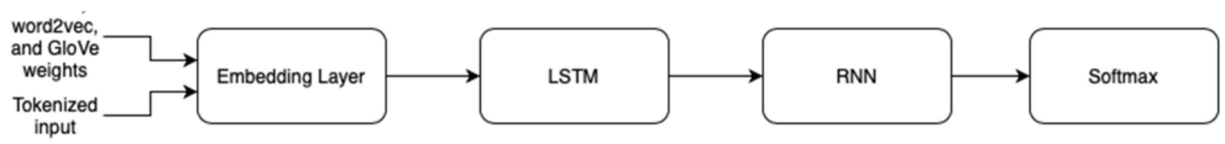


Figure 8: High-level representation of the modified SentimentNet model.

In the training, we used the Adam optimizer and CrossEntropyLoss as a loss function as it is used for multiclass classification in PyTorch and by default it also passes the inputs through a softmax layer to normalize the probabilities of each class (their sum of probabilities will be one) and find the class with the highest probability which is the predicted class.

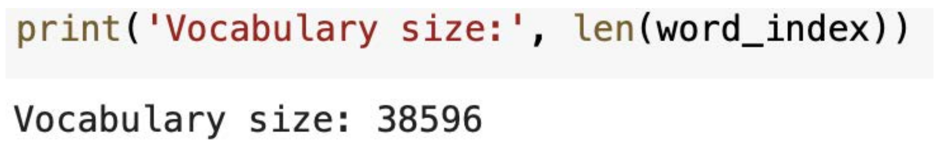
The hyperparameters to be tuned in the period prediction model were the learning rate (0.001, 0.0001, 0.00001), number of epochs (10, 30, 50), hidden dimensions size (64, 128, 512), batch size (8, 16, 32, 64, 128, 256), GloVe embeddings dimension (50, 100, 200, 300), and Dropout probability (0.2, 0.5). Similar hyperparameter values were also tested for popularity prediction. Since there were less number of training samples in popularity prediction(the dataset was divided into 6 decades - i.e each decade was trained separately) and since popularity classification is a simpler task with only two classes, the hyperparameters values used for tuning the model were chosen to reflect the simplicity of the problem. Hence the values used for popularity prediction model were the learning rate (0.001, 0.0001), number of epochs (10, 15), hidden dimensions size (4, 8, 16, 32), batch size (4, 8, 16), GloVe embeddings dimension (300), and Dropout probability (0.5).

During model training, the best model is saved if the training loss has decreased. We did not use Early Stopping to stop training as we wanted to keep the number of epochs constant and we wanted the graphs to clearly show overfitting in some cases. Since the best model is only saved if the validation loss decreases, the overfitted model was not chosen for the test evaluation.

For the GloVe method[13], we used pre-trained word embeddings as a weight matrix input for our model. The pre-trained GloVe embeddings come in 4 different dimensions which are 50, 100, 200, and 300. The increase in these embedding dimensions means the ability of the vector to represent each word also increases while also increasing the computation complexity as matrix gets larger. We needed a fixed sequence length for each lyric so we arbitrarily chose 200. However, when we did our first training, the results were not great thus we looked at the min, max and mean length of the lyrics in our data. The mean length of the lyrics were around 70, so we rounded it to 100 and defined the maximum sequence length as 100. For lyrics shorter than 100 words, we added padding by placing 0's before the tokenized lyrics to increase the sequence's size to 100 while adding words (0) that have no weight on training.

[13] J. Pennington, *Glove: Global vectors for word representation*. [Online]. Available: https://nlp.stanford.edu/projects/glove. [Accessed: 25-Dec-2021].

Both Word2Vec[14] and GloVe were used as the input weight matrices for our models to be embedded with the tokenized input lyrics. The Word2Vec method produces weights according to the local context of the words in each document, that is, the information only about adjacent words. On the other hand, the GloVe method also considers the global information of the words through all documents alongside the local information within the documents. While we could have constructed our own weights using these methods, this would be a costly process as we need to compute counts of thousands of words in thousands of documents, so we decided to use pre trained weights for each word embedding method.



```
print('Vocabulary size:', len(word_index))

Vocabulary size: 38596
```

Figure 9: Vocabulary size of the dataset used.

The original data was in period order (labels were ordered sequentially as 60, 70, 80, 90, 00, 10). It was shuffled so that each of the training, validation, and test sets get similar proportions of each decade. We splitted the data as 70% for training and 30% for testing and then we also splitted the 30% testing data into 50% validation and 50% test data. The model used training data to update the hidden layer weights and the validation data was used to display the performance of the model. Finally, the best model is evaluated using the test set and this is how our test loss and accuracy results are produced.

In the data we used to train our period prediction models, the decades were given as numbers such as 60, 80, 10. In order to feed these labels into our RNN model and in order for the loss function to calculate the loss, we needed to convert these labels into one hot encoding. Otherwise, the correlation between class labels 80 and 90 will be different than the correlation between 00 and 90. We also modified our model to produce 6 output nodes which represent each class. The loss function is calculated between the one hot encoded ground truth label and the model output. The similar approach was used for the popularity prediction model. Instead of making a binary classification with one bit output, we used 2 class classification with one hot encoding, to be able to utilize the same code we used for the period classification model.

It should be noted that during popularity prediction, each decade was trained within itself. In other words, 6 separate popularity prediction models were trained. This was done to eliminate any confusion in the model. For example, if the popularity determining factors in song lyrics

[14] "Gensim: Topic modelling for humans," *models.word2vec – Word2vec embeddings - gensim*, 22-Dec-2021. [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html. [Accessed: 26-Dec-2021].

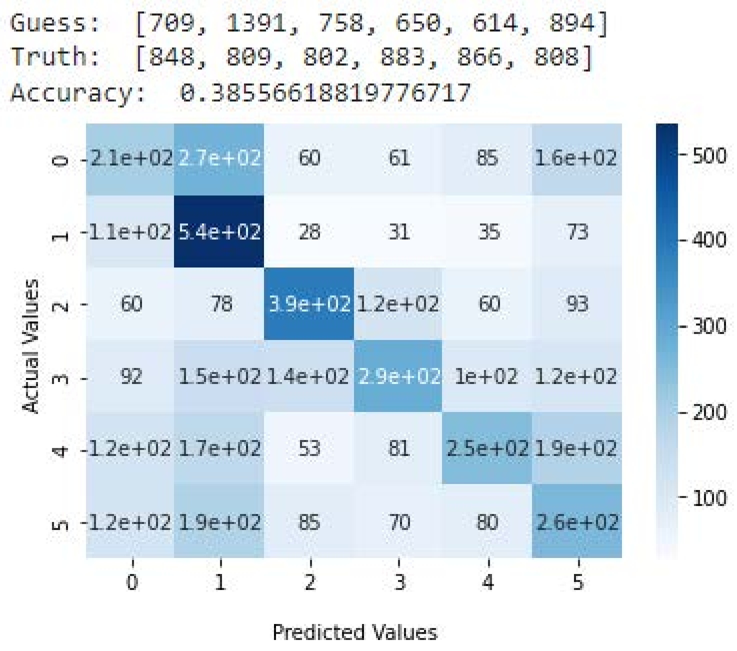
have changed over the decades, it would not make sense to train the popularity classification independent of the decade the song was released in. M song that was popular in the 70s might not be as popular if it was released in the 60s, which was essentially one of the questions we were trying to answer with our project. Therefore, for each decade, a popularity classifier that only takes the songs released in that decade for training and validation was trained. Hence, we can test whether a song that was maybe released in the 70s would be popular in any other decade in addition to its own decade using the saved trained models for each decade.

4. Results

4.1. Results of Period (Decade) Classification

4.1.1 Multinomial Naive Bayes

After creating the TF-IDF matrix and splitting it into test and train, we acquired ~38% accuracy for a six-labeled classification task. The accuracy score oscillates between 35-42% due to the difference in splitting the complete dataset since a shuffle function is used to make the splitting random. Figure 10 has information on the results of the classification predictions of the model. The Guess and Truth arrays show the number of songs per period in each category. The period labels [0. 10. 60. 70. 80. 90.] are encoded to [0 1 2 3 4 5] respectively.



```
Guess:  [709, 1391, 758, 650, 614, 894]
Truth:  [848, 809, 802, 883, 866, 808]
Accuracy:  0.38556618819776717
```

Figure 10: Guess (prediction) and truth values, accuracy score and the confusion matrix of period classification with Multinomial Naive Bayes

4.1.2 Recurrent Neural Networks with LSTM

The RNN with LSTM model was first trained using the GloVe word vectorization technique. During this process, hyperparameter tuning was performed on the model to achieve the best results. After determining the hyperparameter values, other word vectorization techniques were applied to the same model with the same values to observe the performance difference of using different word vectorization techniques. We tried a smaller subset of hyperparameter values with Word2Vec to confirm that using the hyperparameter values obtained from the best performing RNN model with GloVe is the same as the best performing RNN model with Word2Vec. Hence, while the section with results of RNN using GloVe includes different steps taken to optimize the model, section RNN with Word2Vec includes the experimentations done to observe whether there is another set of parameters for RNN with Word2Vec that performs better than the obtained best model parameters of RNN with GloVe.

4.1.2.1 RNN using Glove

The model was first trained with the initial hyperparameters listed in Table 1, some of which were taken from the values used in the initial model[15]. We based our model on the dataset we have prepared.

Hyperparameter	Value
Hidden Layer Dimension	512
Epochs	10
Batch Size	512
Learning Rate	1e-04
Dropout Probability	0.5
GloVe Embedding Dimension	50

Table 1: Hyperparameter values used in the first experimentation with unbalanced dataset

After the training, we achieved the results shown in Figure 11 and Figure 12. When we looked at the predictions of the model on the test set, we realized that it was only predicting the period of the samples as either 70s or 80s, which we traced back to the class imbalance present in our data.

[15] Nbandhi, "AI-ml/LyricsAnalysisGenrePredictionLSTM.ipynb at main · NBANDHI/AI-ML," *GitHub*. [Online]. Available: https://github.com/Nbandhi/AI-ML/blob/main/LyricAnalysis/LyricsAnalysisGenrePredictionLSTM.ipynb. [Accessed: 24-Dec-2021].

Therefore, we balanced our dataset via undersampling. Figure 13 shows the number of samples per period in the balanced and unbalanced dataset.

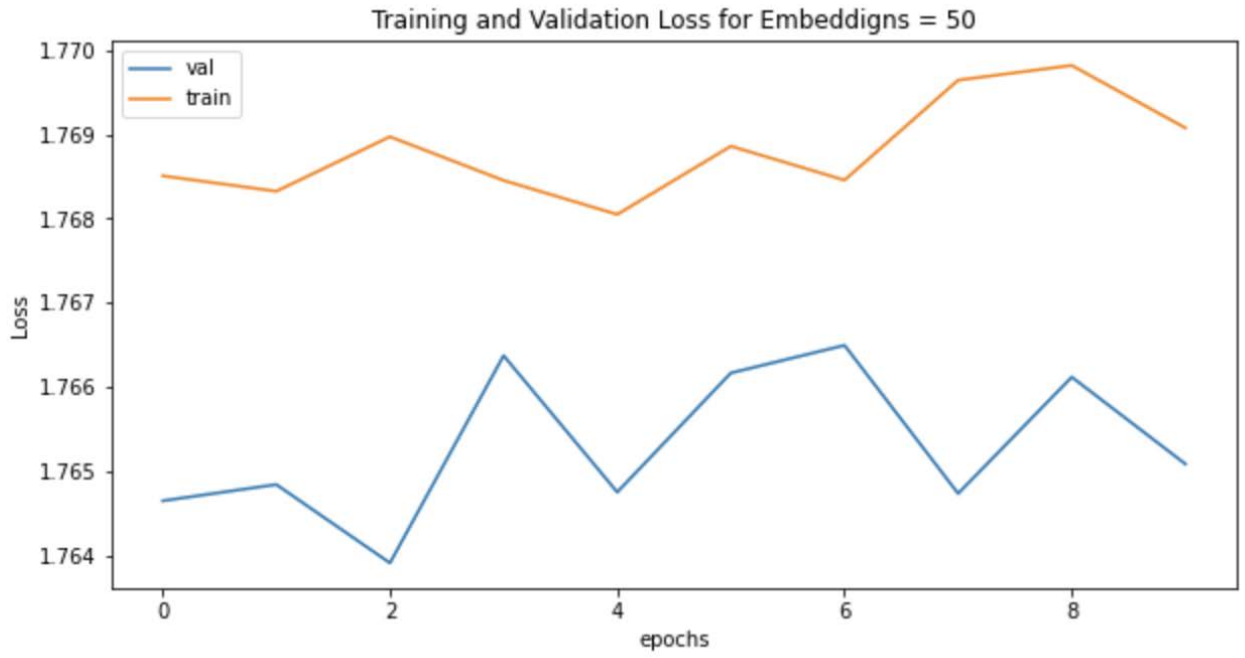


Figure 11: Loss graph of the model on the imbalanced dataset

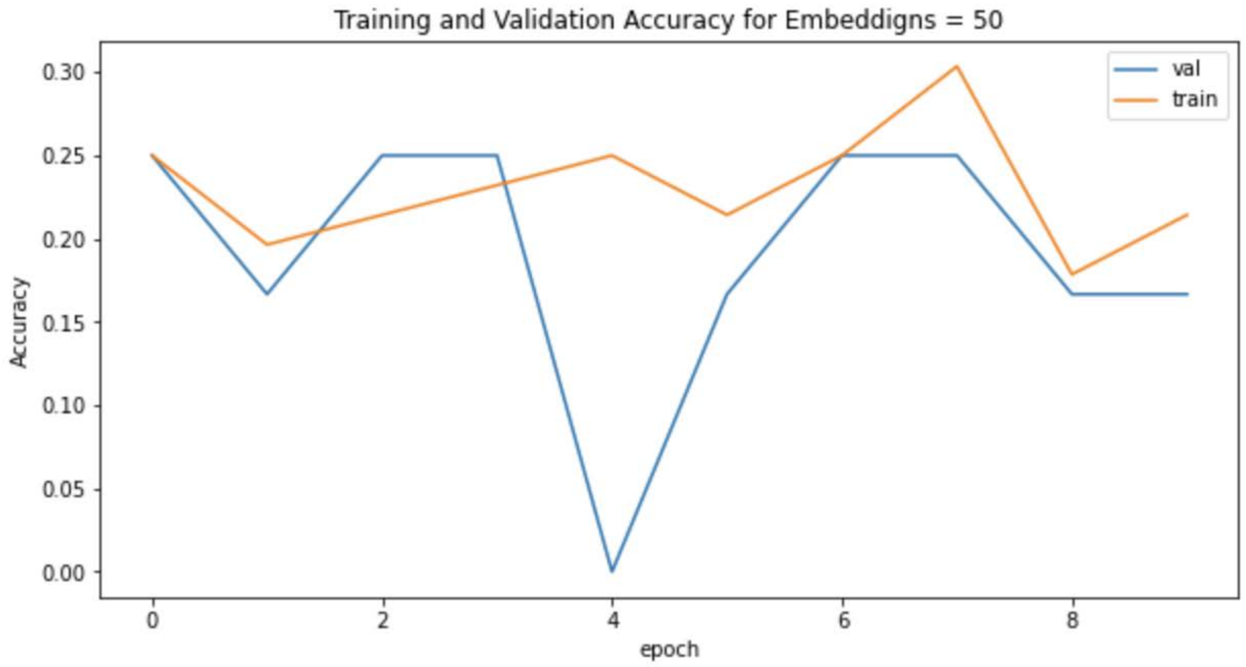


Figure 12: Accuracy graph of the model on the imbalanced dataset

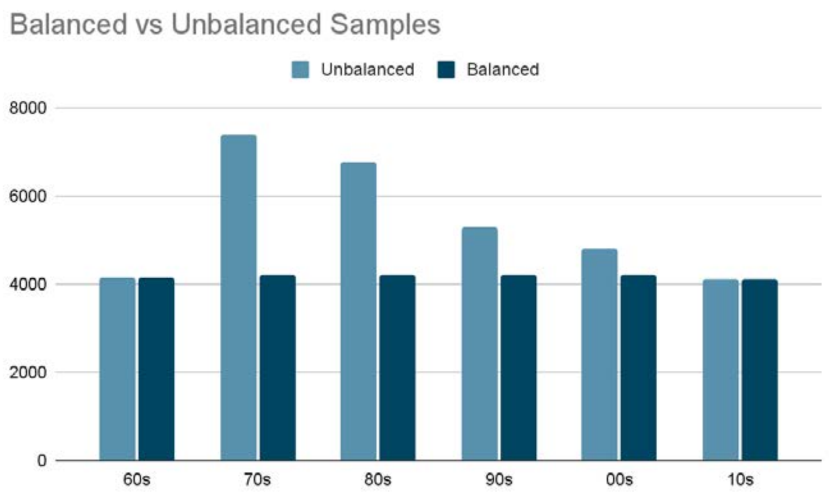


Figure 13: Number of samples per period in the balanced and unbalanced dataset

After balancing the dataset, we also changed certain hyperparameter values to values in Table 2 to make the model fit better to our dataset.

Hyperparameter	Value
Hidden Layer Dimension	512
Epochs	50
Batch Size	256
Learning Rate	1e-03
Dropout Probability	0.2
GloVe Embedding Dimension	50

Table 2: Hyperparameter values of the first experimentation with the balanced dataset

It should be noted that we didn't have an early stopping mechanism in our model. However, at each epoch, we saved the model if it was the best performing model achieved so far. In other words, if the validation loss has decreased at an epoch compared to the validation loss of the last best performing model, we save the new model with lower validation loss as the best model. Hence, if the model overfits, the overfitted model is not used during testing. Instead, the model that was encountered at an earlier epoch with best validation loss is used. With that in consideration, we eliminated the effect of the number of epochs the model has been trained on from the risk of overfitting. We also lowered batch size, since we had substantially less samples than the dataset used in the initial model we based our model on. The results of the model with the hyperparameter values are shown in Figure 14 and 15.

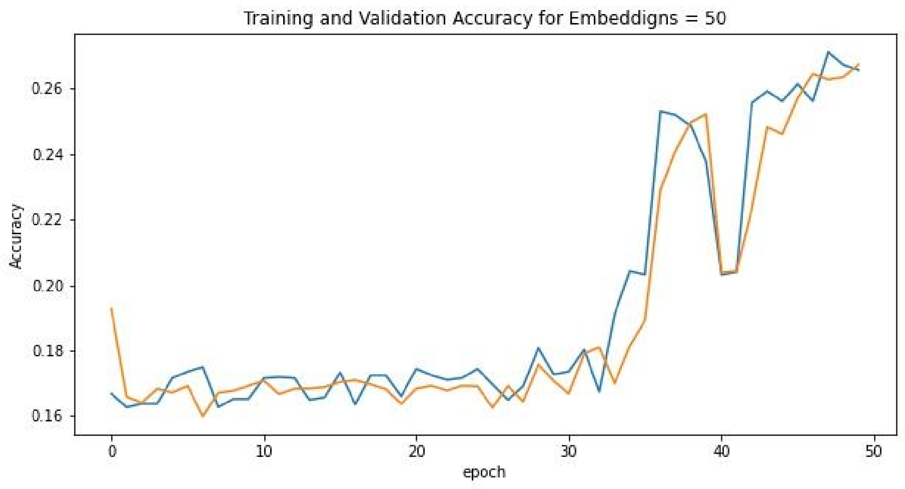


Figure 14: Accuracy graph of the model on the balanced dataset

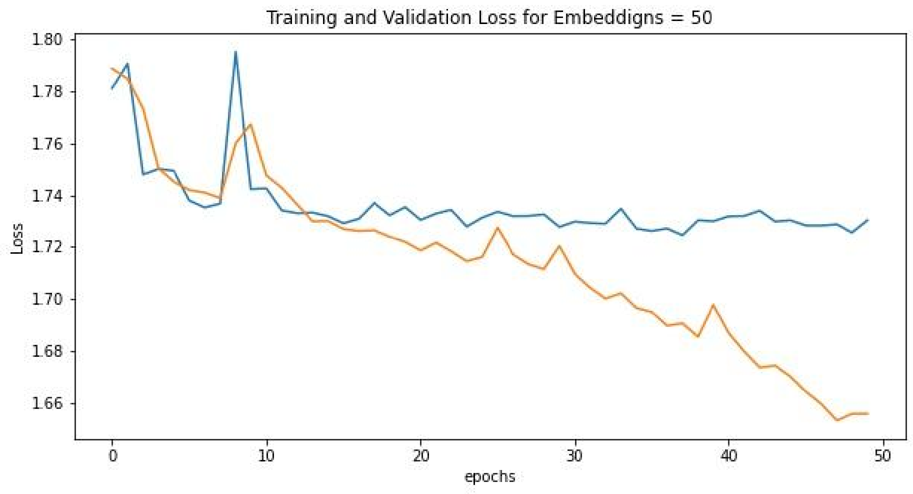


Figure 15: Loss graph of the model on the balanced dataset

As seen on Figure 15, the model starts overfitting around the 20th epoch. In addition, the sudden peaks in our loss graph might be caused by the high learning rate we have at the beginning or just some unlucky batches that maximized the loss around the 10th epoch. With this model, we were able to reach 25.9% test accuracy, which is better than guessing a random period for each song(⅙ probability of choosing the right period). After observing the overfitting problem of the model, we decided to reduce the complexity of the model, to make the model continue to learn for a longer period of time(i.e reduce overfitting at earlier epochs). Some steps we took to reduce the overfitting problem were: reducing the number of hidden layer dimensions, reducing the batch size and increasing the dropout probability. In addition we experimented with different learning rates to find the best performing model. All the different hyperparameter values that we experimented with in addition to their test results can be found in Appendix B. We have observed that these steps taken to reduce the overfitting problem were partially effective, i.e they slightly improved the model and the test results. Another important parameter we changed was the GloVe embedding dimension. Since higher dimension GloVe embeddings increased the preciseness of the weights given to each word, we used the highest available GloVe embedding dimension in our best performing model. Figure 16 and 17 shows the accuracy and loss graphs of the best performing model. The model was able to reach 33.2% accuracy on the test set. The hyperparameter values of the model are given in Table 3. As seen from the graphs, the overfitting issue still continues with the model. Further improvements that can be done on the model are given in detail in the Discussion Section. Further experiments with the hyperparameters can be found in Appendix B for decade classification.

Hyperparameter	Value
Hidden Layer Dimension	64
Epochs	30
Batch Size	8
Learning Rate	1e-04
Dropout Probability	0.5
GloVe Embedding Dimension	300

Table 3: Hyperparameter values of the best performing model

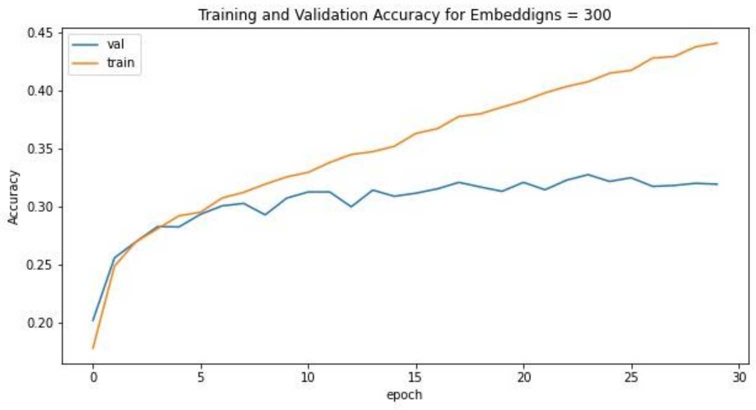


Figure 16: Accuracy graph of the best performing model

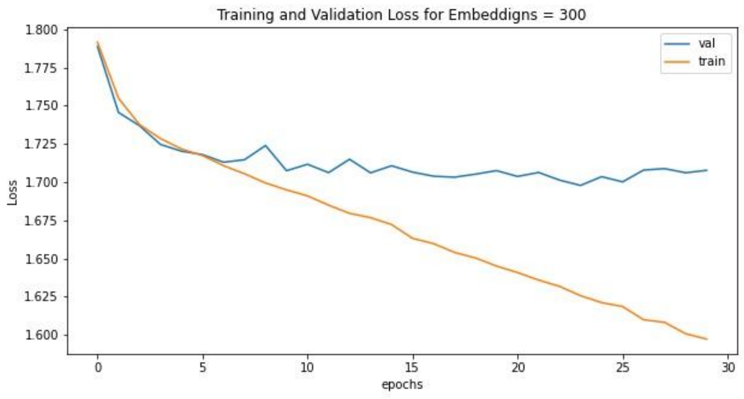


Figure 17: Loss graph of the best performing model

4.1.2.2 RNN using Word2Vec

The parameters for RNN using Word2Vec were selected as the hyperparameters of the tests done with GloVe. This allows us to eliminate the differences between models as much as we can when comparing results.

The parameters from GloVe are also tested to be fit for the Word2Vec model (see Appendix D) and they are as follows:

Hyperparameter	Value
Hidden Layer Dimension	64
Epochs	30
Batch Size	8
Learning Rate	1e-04
Dropout Probability	0.5

Table 4: Hyperparameter values to be used with Word2Vec

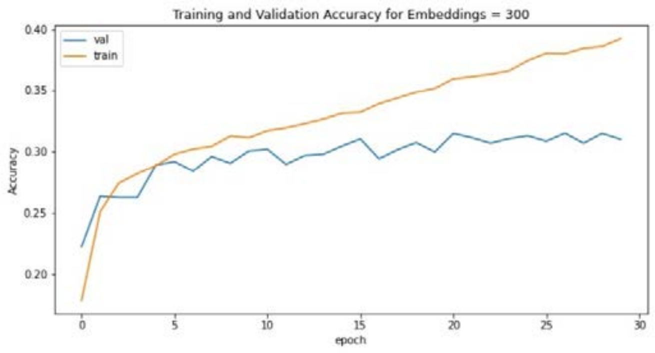


Figure 18: Accuracy graph of the model with Word2Vec

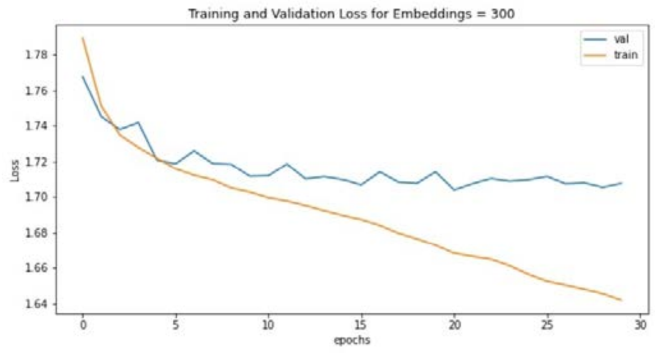
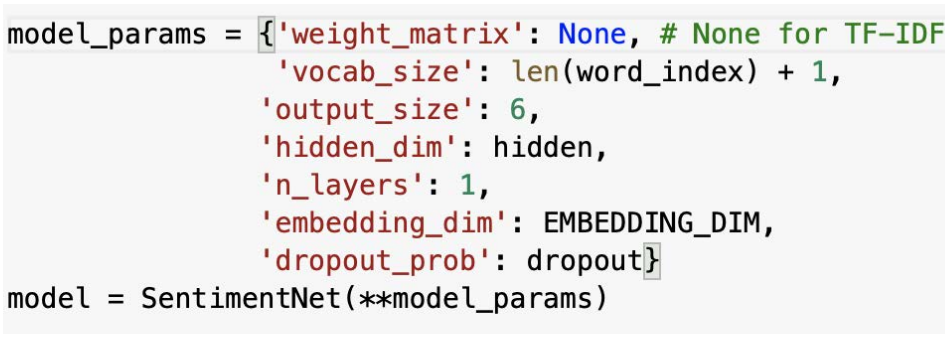


Figure 19: Loss graph of the model with Word2Vec

An accuracy of 31.8% and loss of 1.707 is acquired from the model.

4.1.2.3 RNN using TF-IDF

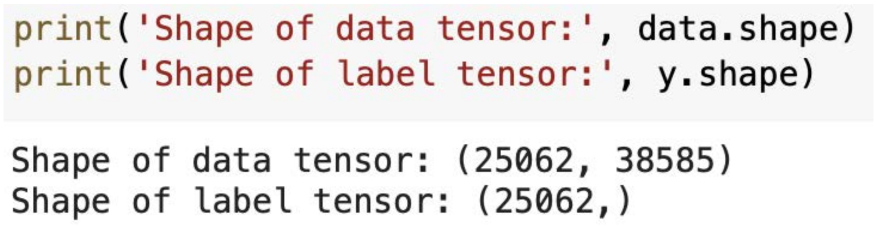
A model using the scoring system of TF-IDF for RNN was aimed to be trained. This model would provide a direct difference between a neural network and a probabilistic ML classifier. The TF-IDF matrix produced included document-specific grades for each word in a specific document. When a different document did not include a word, it did not have a score for that word. This system of vectorizing is different from those of GloVe and Word2Vec since it does not produce a vocabulary with weights but rather it produces document specific information. Therefore, unlike GloVe and Word2Vec, the training of TF-IDF vectors does not need a weight matrix so the weight matrices oın the Embeddings layer of our network is set to None which can be seen from Figure 20. Then, instead of using tokenized lyric sequences, the TF-IDF vectors are given to the model as input.



```
model_params = {'weight_matrix': None, # None for TF-IDF
                 'vocab_size': len(word_index) + 1,
                'output_size': 6,
                'hidden_dim': hidden,
                'n_layers': 1,
                'embedding_dim': EMBEDDING_DIM,
                'dropout_prob': dropout}
model = SentimentNet(**model_params)
```

Figure 20: Model parameters to train the TF-IDF vectors.

Unfortunately, the matrix became too large in size for Google Colab to create.

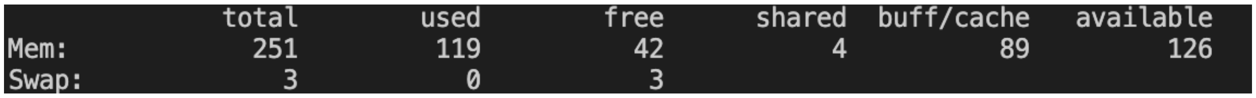


```
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', y.shape)

Shape of data tensor: (25062, 38585)
Shape of label tensor: (25062,)
```

Figure 21: TF-IDF matrix size.

We decided to modify the lyrical data by only getting unique words in each song. This lowered the memory required for the matrix. However, it was still too big for Google Colab to train the model with that data. Hence, we used a server with 256 GB RAM (Google Colab has only 12 GB RAM available for free use) to create the matrix and train the model. While we were able to create the matrix in the server, the execution time had certain issues and it was too large for RNN models, therefore we weren't able get any results as running only 1 epoch took 3 days and we aimed for 50 epochs. We have observed that our model uses 120 GB of RAM which can be seen from Figure 22.



```
              total        used        free      shared  buff/cache   available
Mem:            251         119          42           4          89         126
Swap:             3           0           3
```

Figure 22: Memory usage of our RNN with TF-IDF

4.2. Results of Popularity Classification

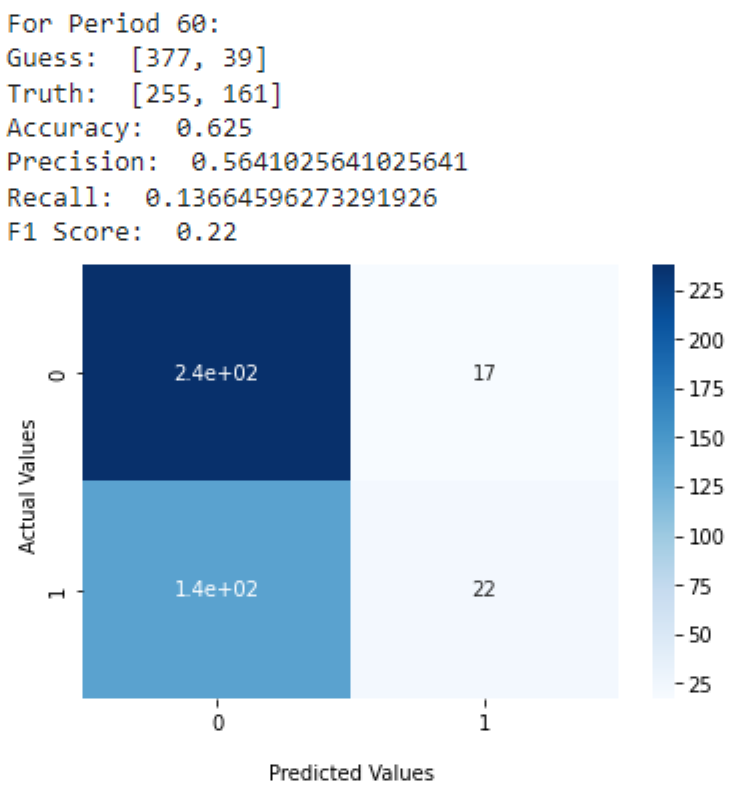
4.2.1 Multinomial Naive Bayes

After performing a period classification task, we added a second classification task which also uses the TF-IDF matrix created in the same way as period classification. This classification predicts whether that song was popular in its decade or not. It is a binary classification task and uses "pop_class" labels in the train data, which is 0 if the song's popularity score was lower than the mean score in its own period, and otherwise 1. For this classification, the data is divided into smaller dataset per period. Since the size gets a lot smaller, 90% of the data is used for training, unlike the 80% in period classification.

Table 5 has accuracy scores per period. For example, for a song produced in the 1960's, the accuracy of its popularity prediction is ~62%. The average accuracy score on popularity classification is 60%. Even though an accuracy above 60% seems better than a random guess for binary classification and higher result than the RNN model, we understand that this is not the case for the Naive Bayes classifier when we look at the precision, recall and F1 scores of the predictions. The average F1 score is 0.16 and there is only one period going over 0.25. This score shows us that the model predicts one label much more than the other one. Hence, it fails to build an accurate probabilistic relation between words in the lyrics and the popularity of those songs. More detailed score information per period can be found in the following figures.

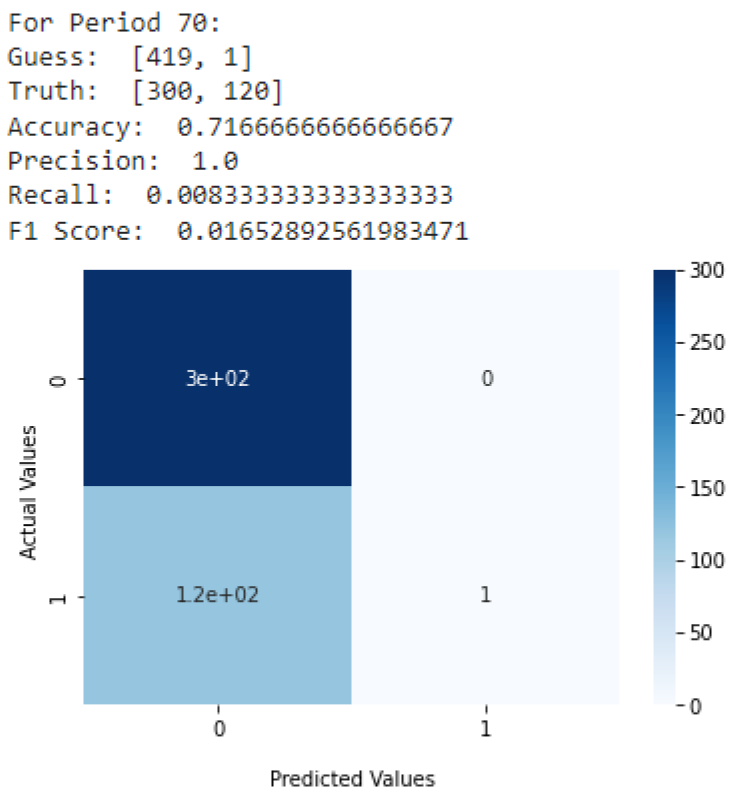
Period	Accuracy	F1 Scores
60s	0.625	0.22
70s	0.716	0.02
80s	0.638	0.05
90s	0.523	0.14
00s	0.593	0.17
10s	0.549	0.40

Table 5: Accuracies of popularity classification of songs per period



```
For Period 60:
Guess:  [377, 39]
Truth:  [255, 161]
Accuracy:  0.625
Precision:  0.5641025641025641
Recall:  0.13664596273291926
F1 Score:  0.22
```

Figure 23: Metric scores and confusion matrix for the period 60s in popularity classification.



```
For Period 70:
Guess:  [419, 1]
Truth:  [300, 120]
Accuracy:  0.7166666666666667
Precision:  1.0
Recall:  0.008333333333333333
F1 Score:  0.01652892561983471
```

Figure 24: Metric scores and confusion matrix for the period 70s in popularity classification.

```
For Period 80:
Guess:  [414, 6]
Truth:  [266, 154]
Accuracy:  0.638095238095238
Precision:  0.6666666666666666
Recall:  0.025974025974025976
F1 Score:  0.05
```

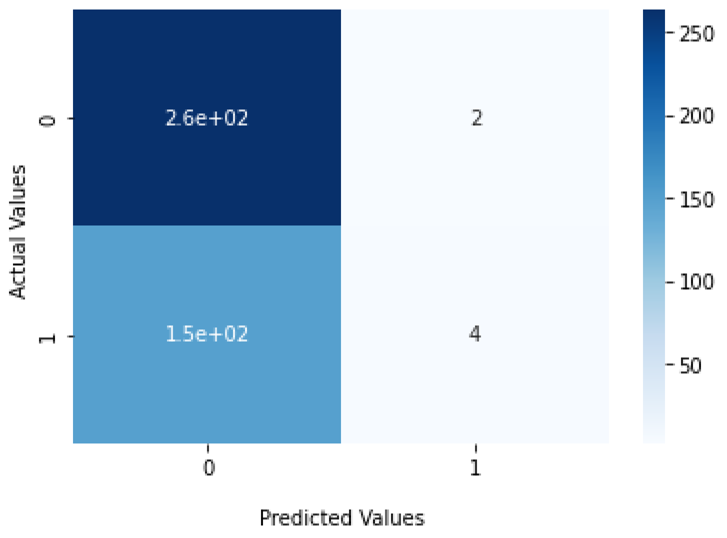


Figure 25: Metric scores and confusion matrix for the period 80s in popularity classification.

```
For Period 90:
Guess:  [390, 30]
Truth:  [216, 204]
Accuracy:  0.5238095238095238
Precision:  0.5666666666666667
Recall:  0.08333333333333333
F1 Score:  0.14529914529914528
```

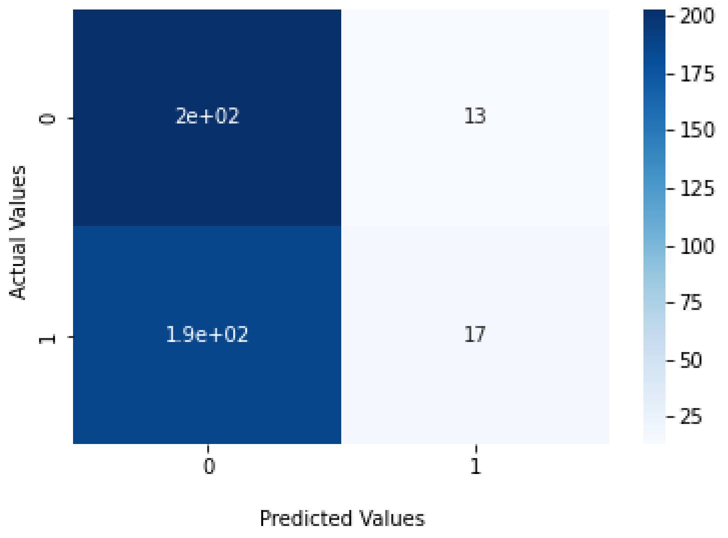


Figure 26: Metric scores and confusion matrix for the period 90s in popularity classification.

```
For Period 0:
Guess:  [392, 28]
Truth:  [241, 179]
Accuracy:  0.5928571428571429
Precision:  0.6428571428571429
Recall:  0.1005586592178771
F1 Score:  0.17391304347826086
```

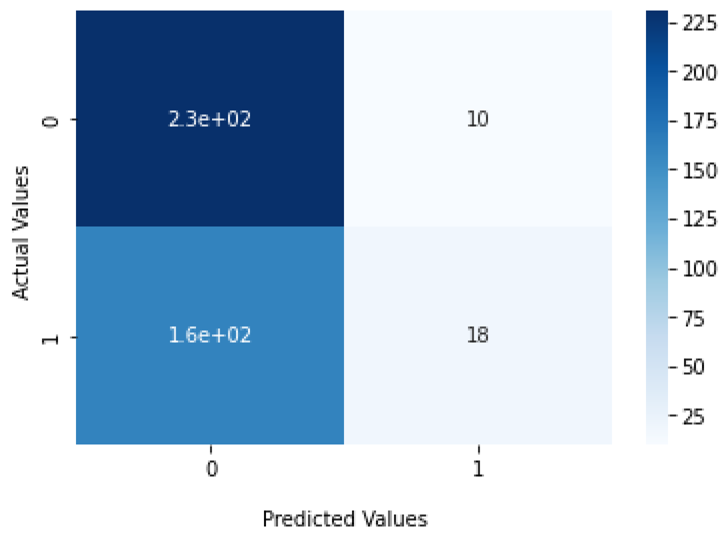


Figure 27: Metric scores and confusion matrix for the period 00s in popularity classification.

```
For Period 10:
Guess:  [303, 110]
Truth:  [211, 202]
Accuracy:  0.549636803874092
Precision:  0.5727272727272728
Recall:  0.3118811881188119
F1 Score:  0.4038461538461538
```

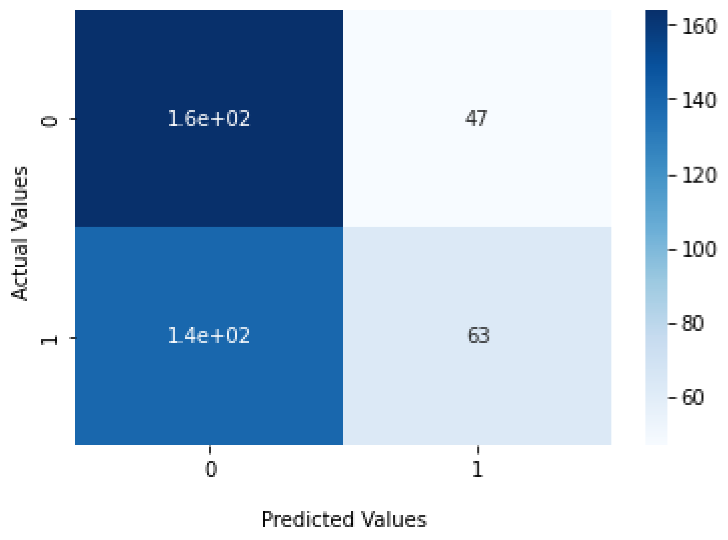


Figure 28: Metric scores and confusion matrix for the period 10s in popularity classification.

4.2.2. Recurrent Neural Networks with LSTM

For popularity prediction, the model used for period prediction was used with small modifications. These modifications were necessary since popularity classification was a two class classification task. Similar to period classification, the popularity classification model was first trained and optimized on the RNN with GloVe embedding and then trained using the Word2Vec embedding with the same hyperparameter values of the best performing model.

4.2.2.1 RNN using GloVe

Since popularity classification tasks were done on each period separately, the number of data per period decreased substantially. Hence, we needed to tune the model to reflect the change in the number of training samples and the classification task. The model was simplified by decreasing the number of dimensions in the hidden layer, and decreasing the batch size. After experimenting with the hyperparameter values, the best performing model was obtained using the hyperparameter values listed in Table 6. We made sure that within each decade, there was no class imbalance, i.e the number of songs that were popular (1) were similar to the number of songs that were not popular (0). This was ensured during the labeling of the songs. The mean popularity score was found for each decade and songs with higher than mean popularity scores were labeled as 1 and others were labeled as 0. Although we tried optimizing the model, we still had an overfitting issue, which will be further discussed in the Discussion section. The test accuracy of the popularity classifier of each decade is given in Table 7 and accuracy and loss graphs of each classifier are displayed below. Further experiments with the hyperparameters can be found in Appendix C for popularity classification.

Hyperparameter	Value
Hidden Layer Dimension	8
Epochs	15
Batch Size	8
Learning Rate	1e-03
Dropout Probability	0.5
GloVe Embedding Dimension	300

Table 6: Hyperparameter values of the best performing popularity classification model

Period	Accuracy
60s	0.583
70s	0.735
80s	0.643
90s	0.551
00s	0.590
10s	0.553

Table 7: Performance of the model in popularity classification task, given per period

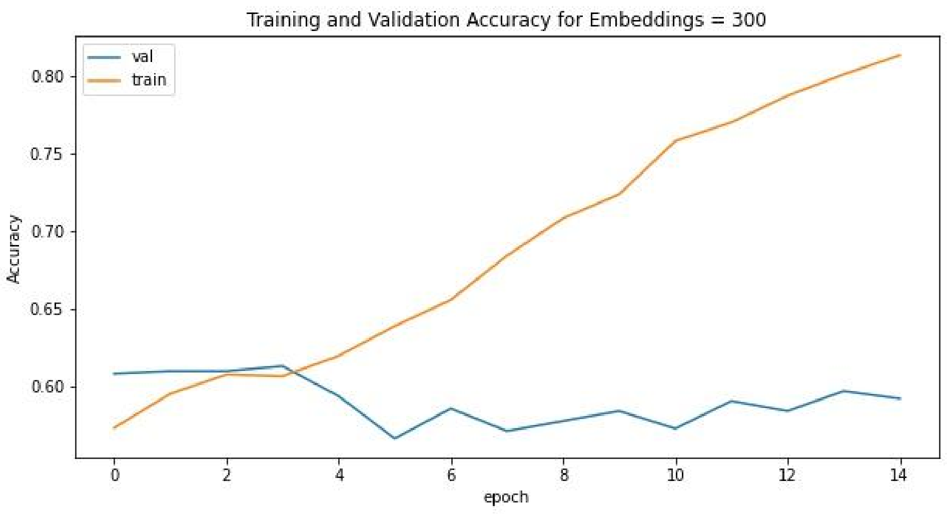


Figure 29: Accuracy graph of 60s popularity classification

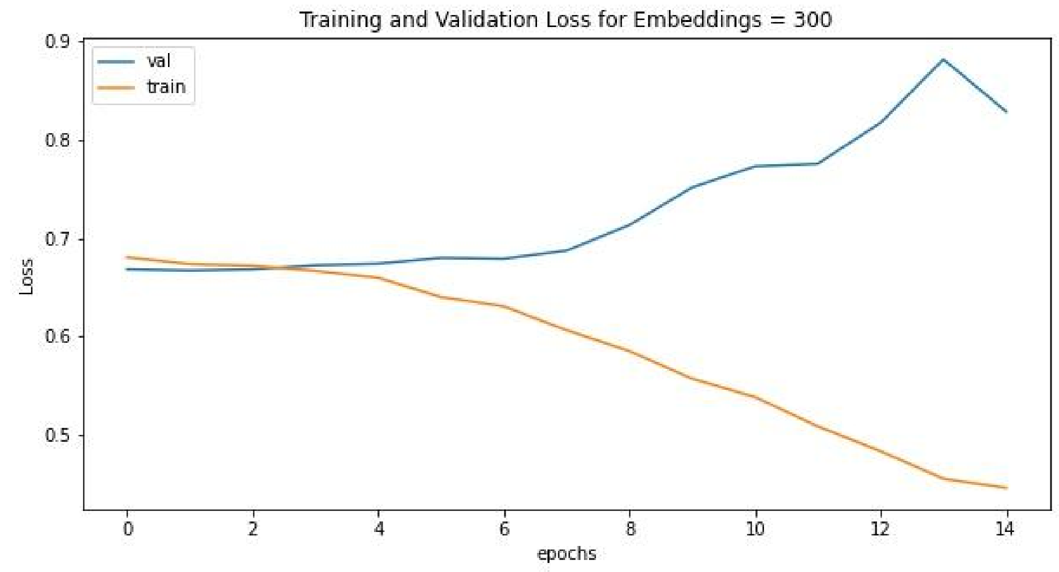


Figure 30: Loss graph of 60s popularity classification

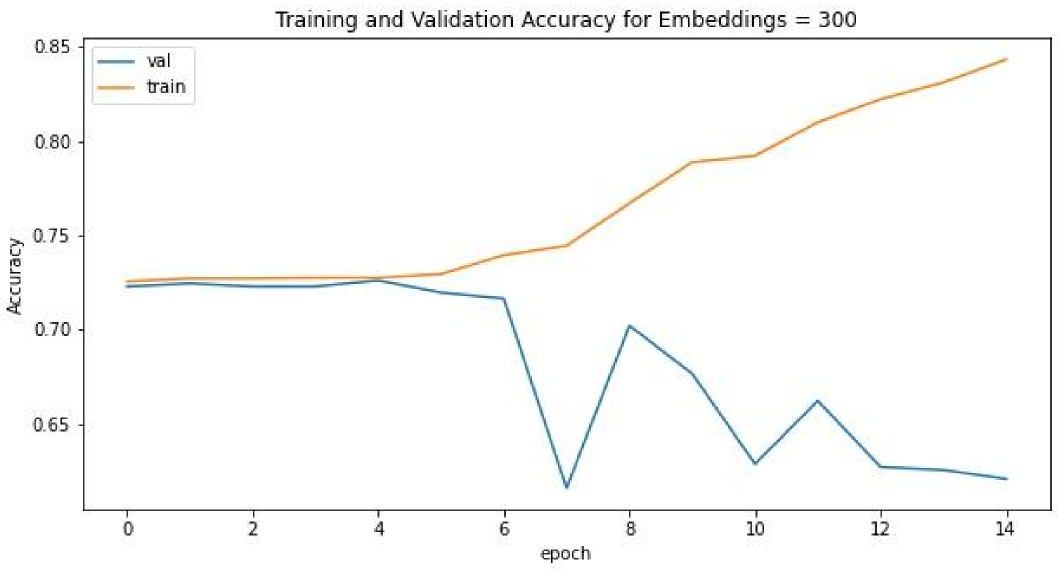


Figure 31: Accuracy graph of 70s popularity classification

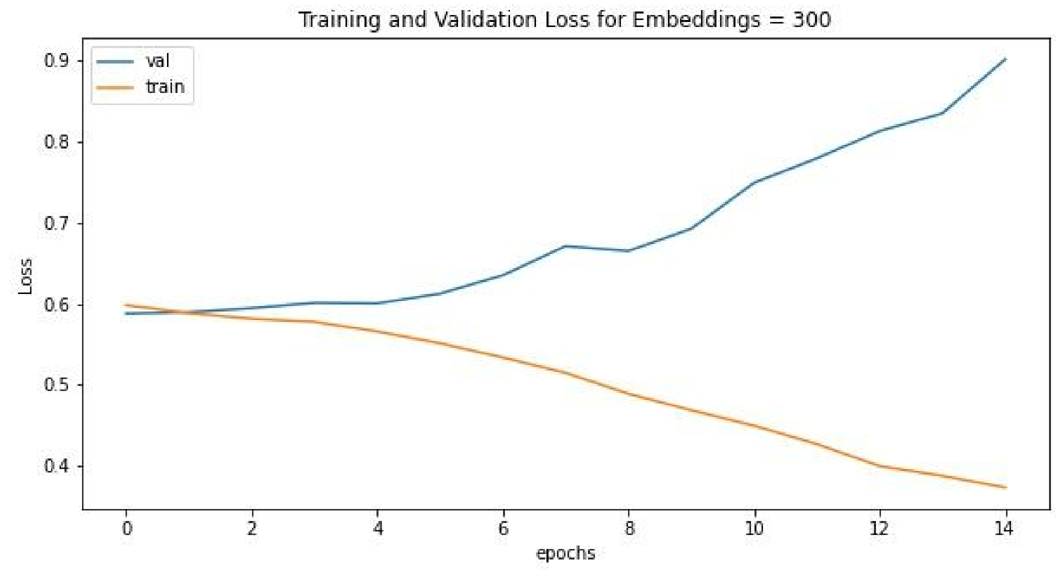


Figure 32: Loss graph of 70s popularity classification

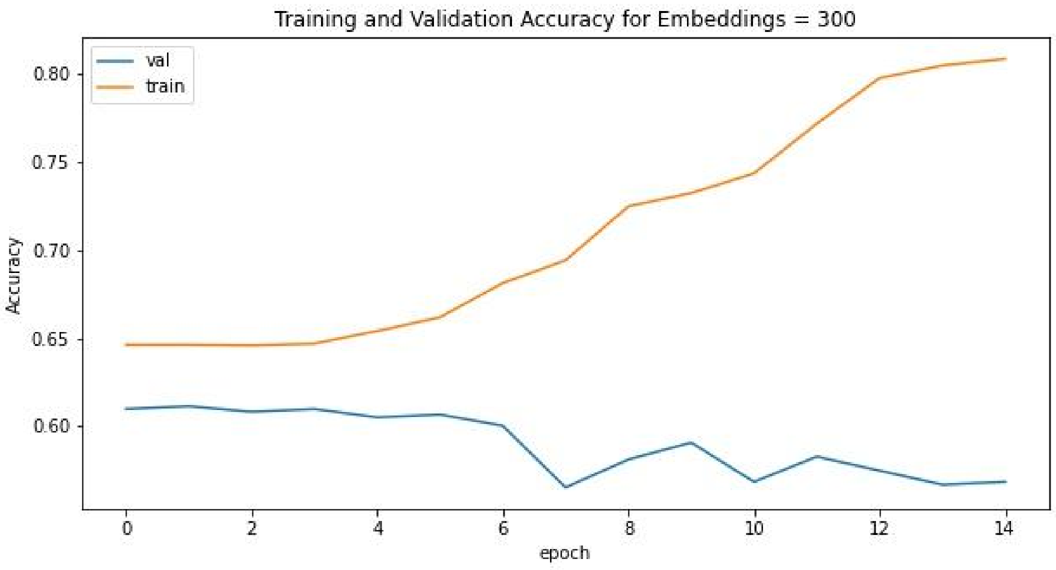


Figure 33: Accuracy graph of 80s popularity classification

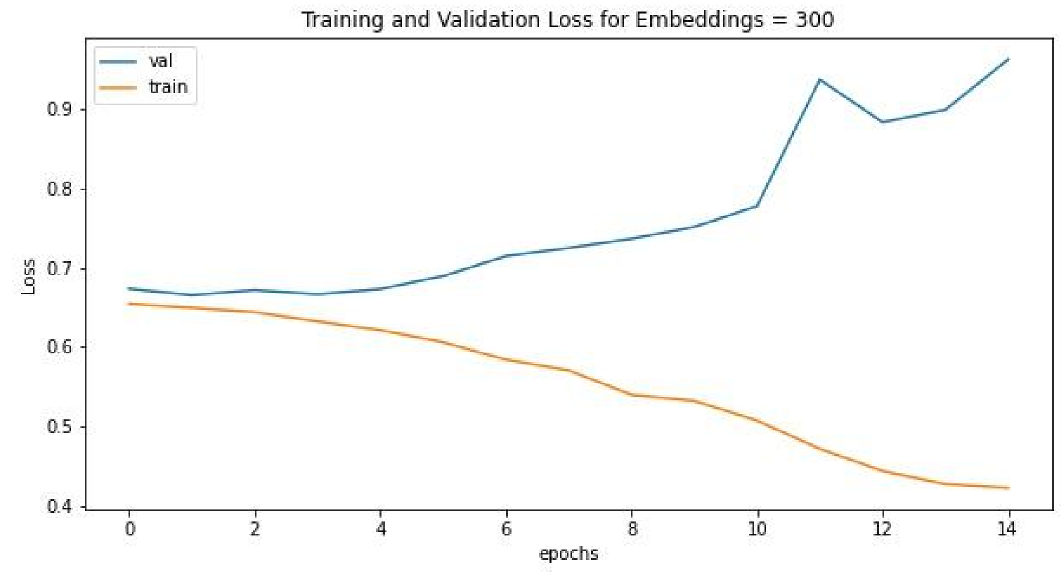


Figure 34: Loss graph of 80s popularity classification

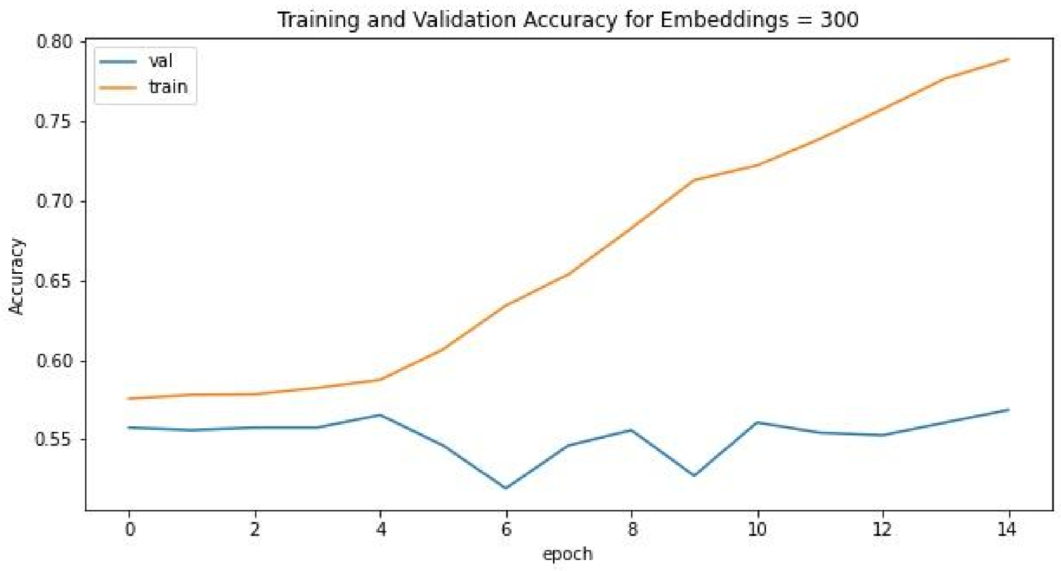


Figure 35: Accuracy graph of 90s popularity classification

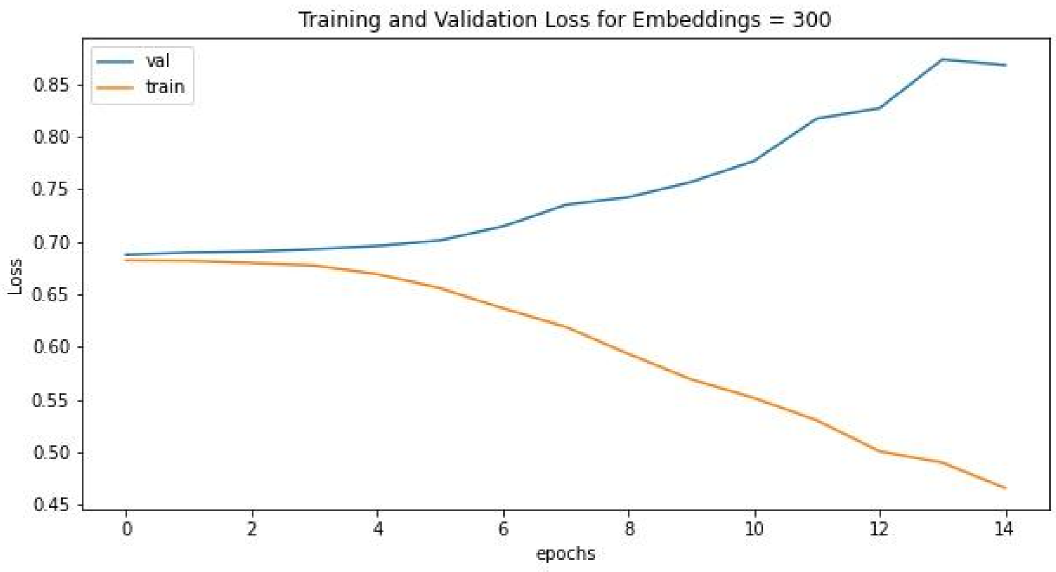


Figure 36: Loss graph of 90s popularity classification

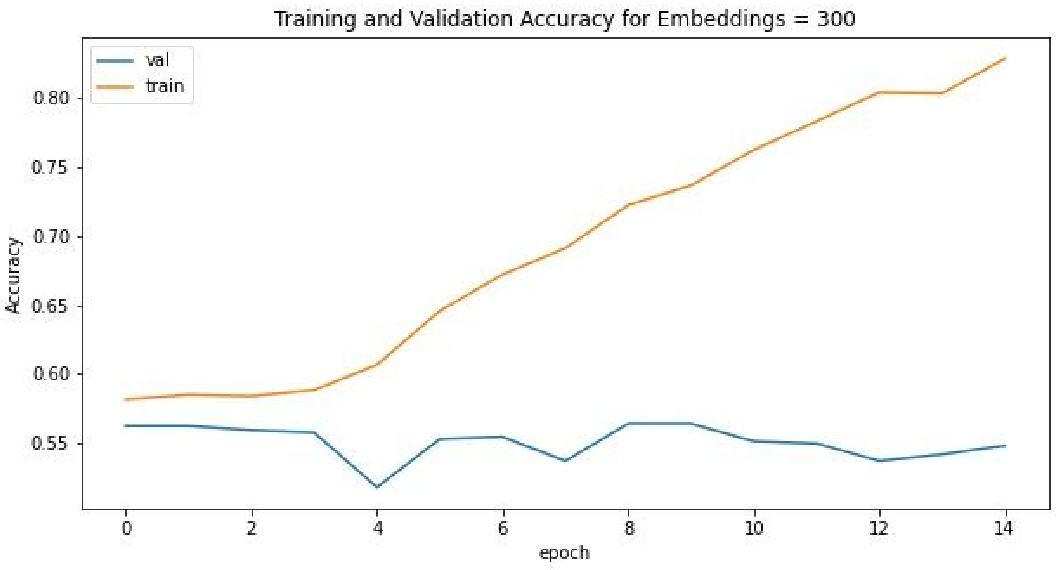


Figure 37: Accuracy graph of 00s popularity classification

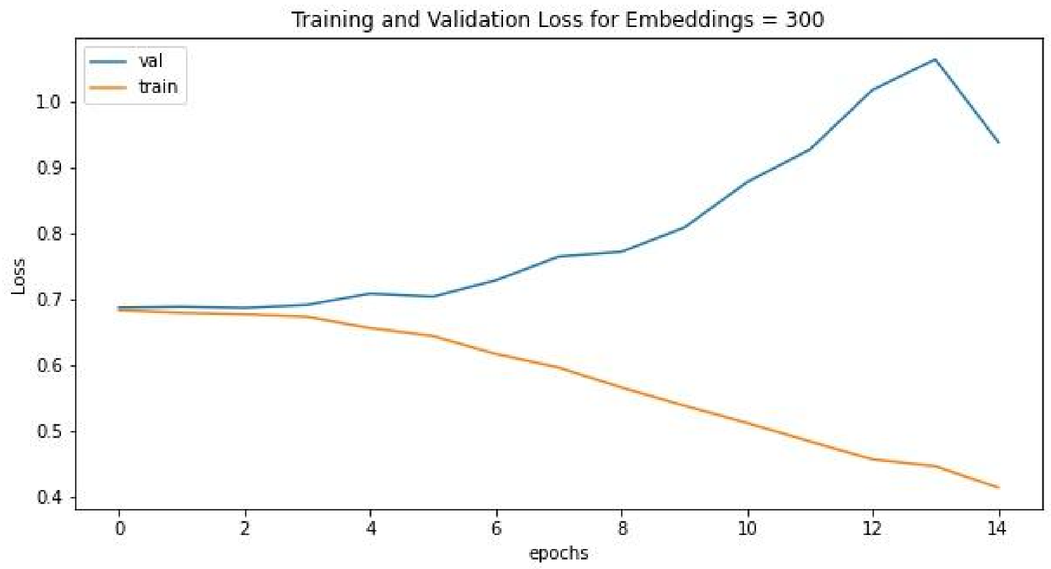


Figure 38: Loss graph of 00s popularity classification

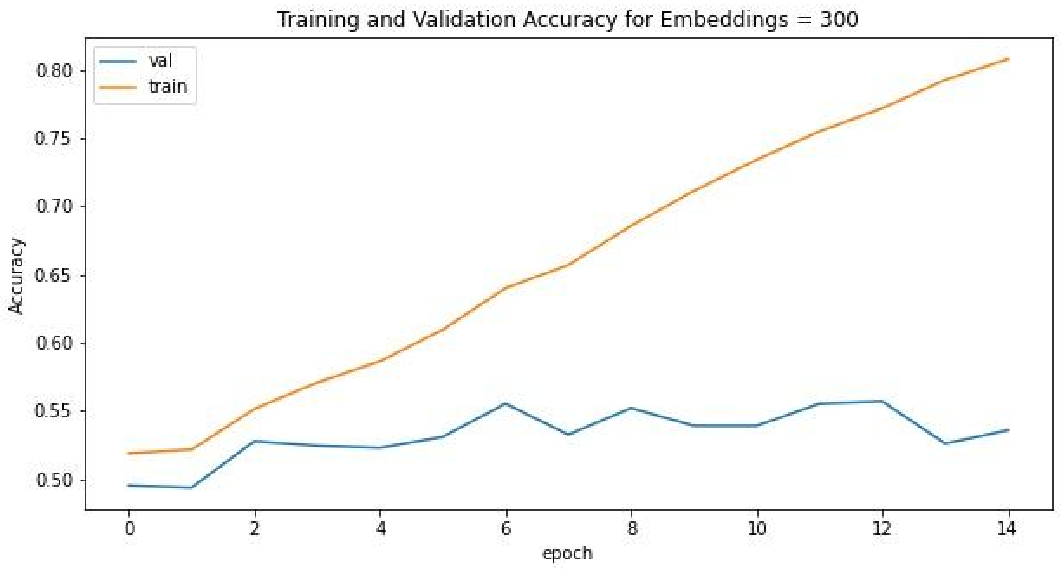


Figure 39: Accuracy graph of 10s popularity classification

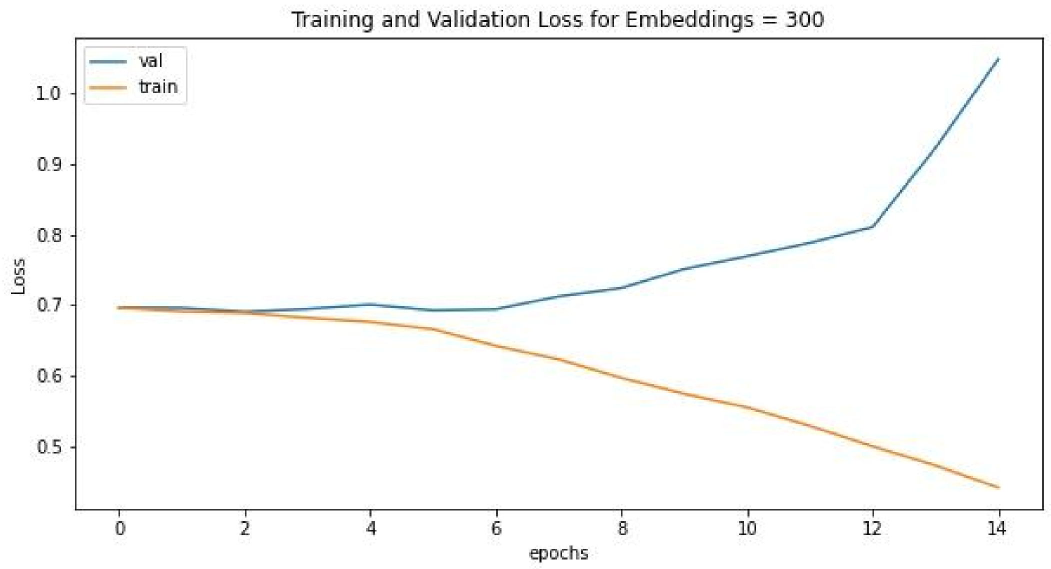


Figure 40: Loss graph of 10s popularity classification

4.2.2.2 RNN using Word2Vec

From our trials of period classification we know that Word2Vec and GloVe operate on similar parameters optimally therefore the hyperparameters of GloVe will be used again to eliminate model-related differences in the results.

Period	Accuracy	Loss
60s	0.565	0.692
70s	0.687	0.548
80s	0.561	0.679
90s	0.692	0.525
00s	0.684	0.586
10s	0.679	0.588

Table 8: Test accuracy and loss of popularity classification of songs per period

The accuracy and loss graphs of popularity classification of each period are displayed below.

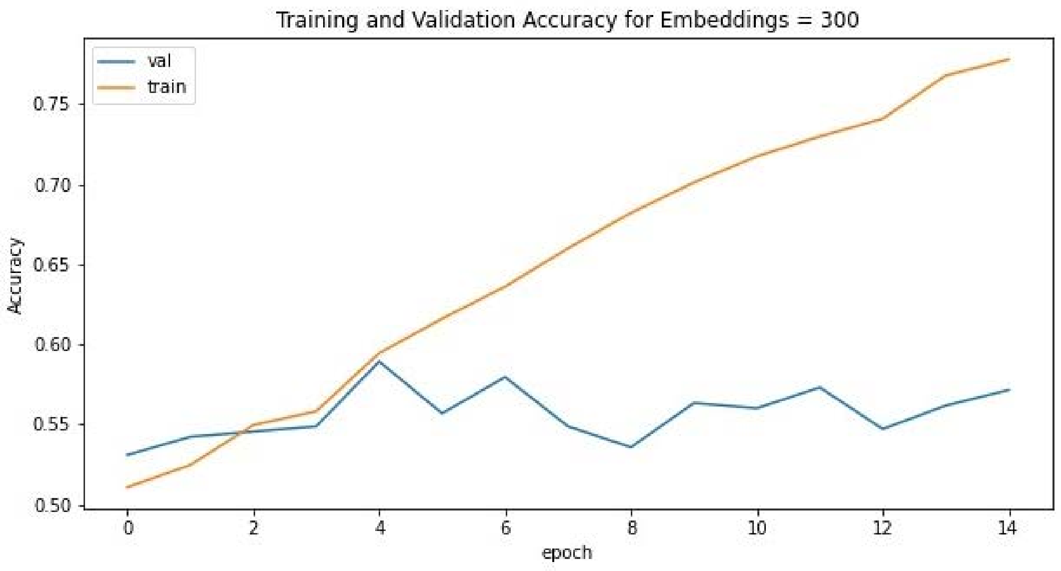


Figure 41: Accuracy graph of 60s popularity classification

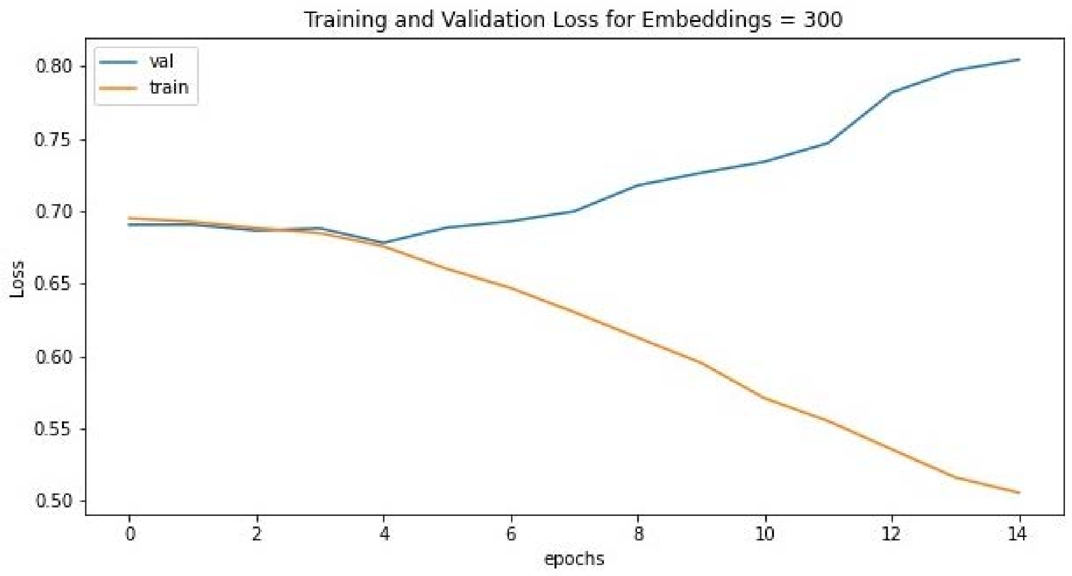


Figure 42: Loss graph of 60s popularity classification

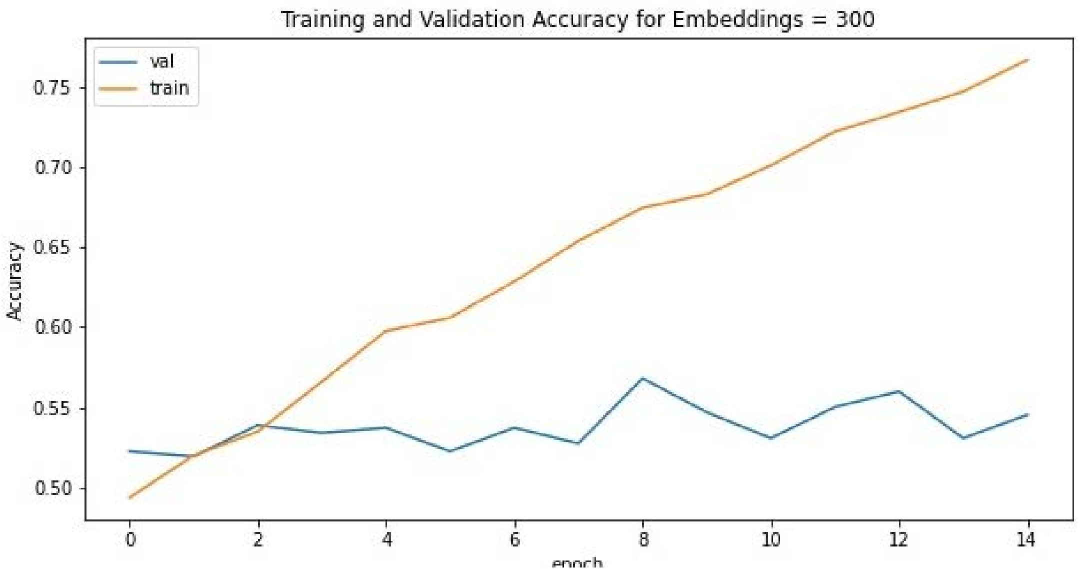


Figure 43: Accuracy graph of 70s popularity classification

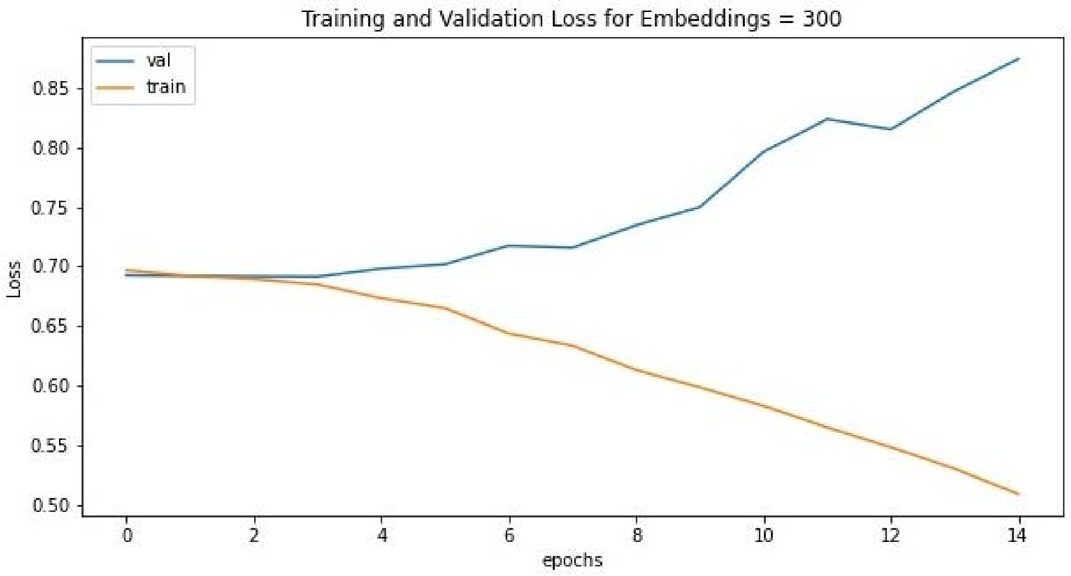


Figure 44: Loss graph of 70s popularity classification

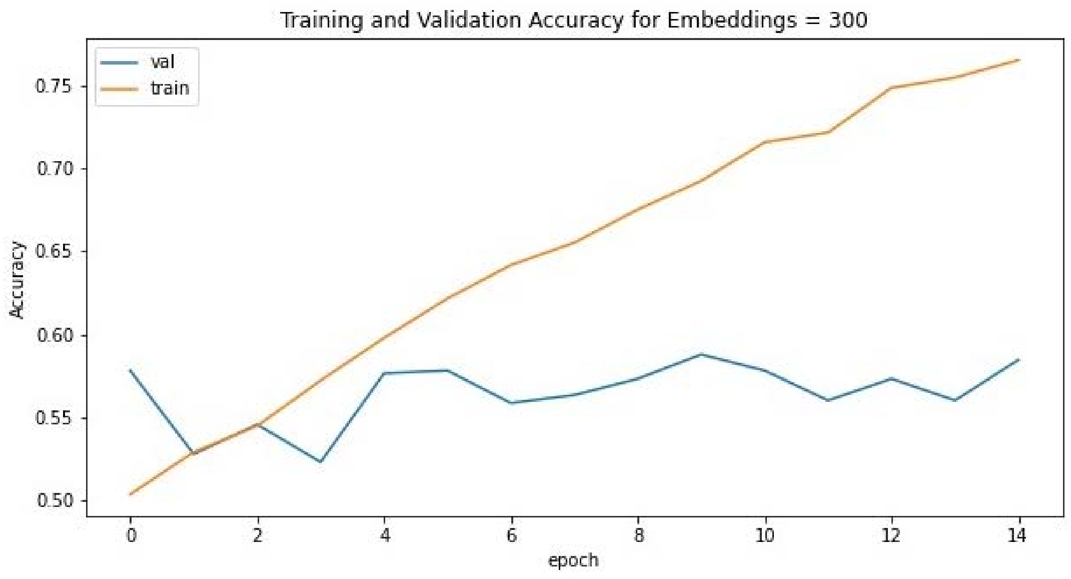


Figure 45: Accuracy graph of 80s popularity classification

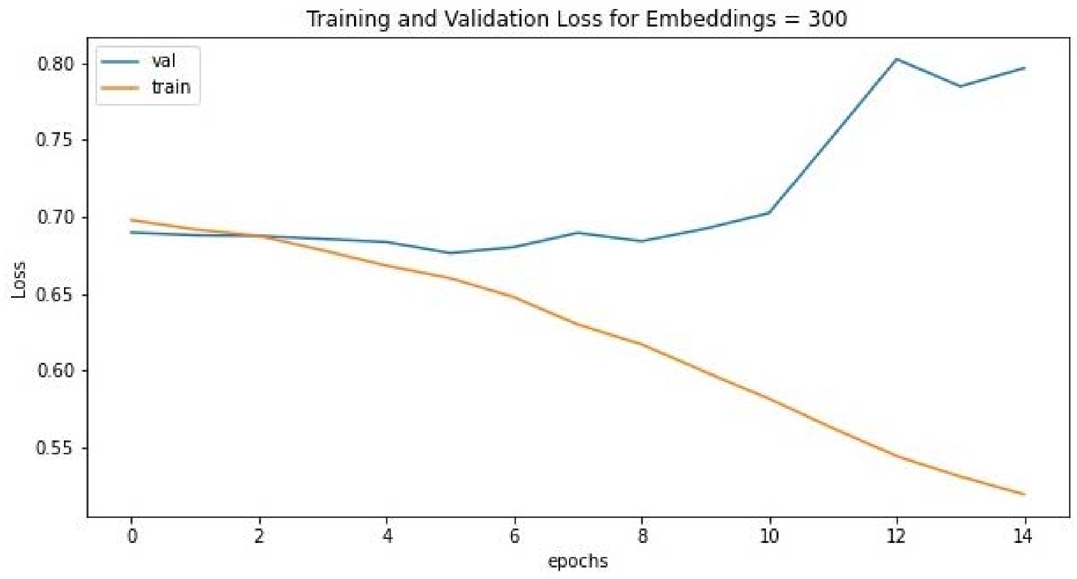


Figure 46: Loss graph of 80s popularity classification

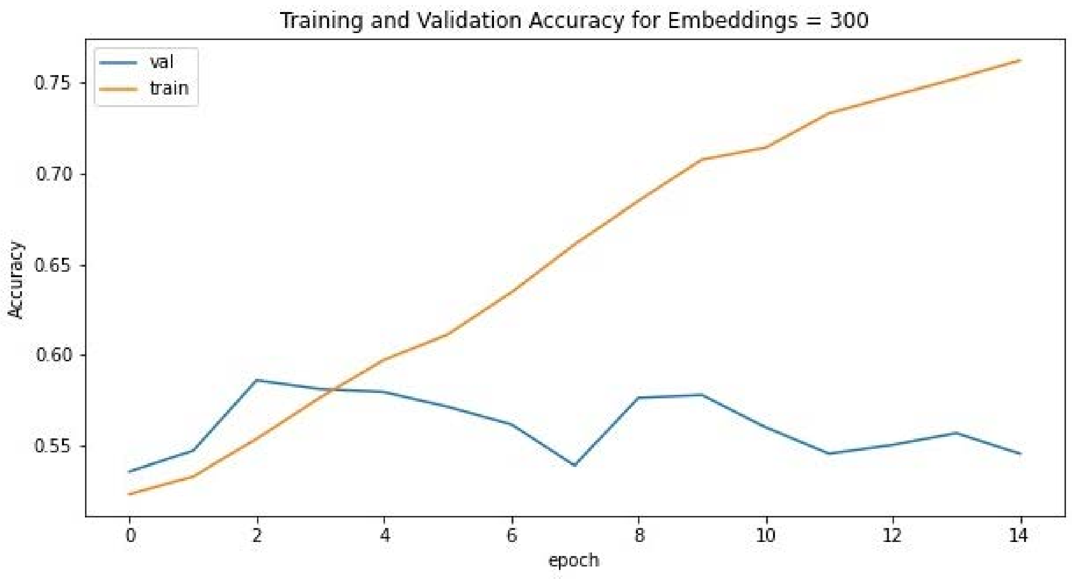


Figure 47: Accuracy graph of 90s popularity classification

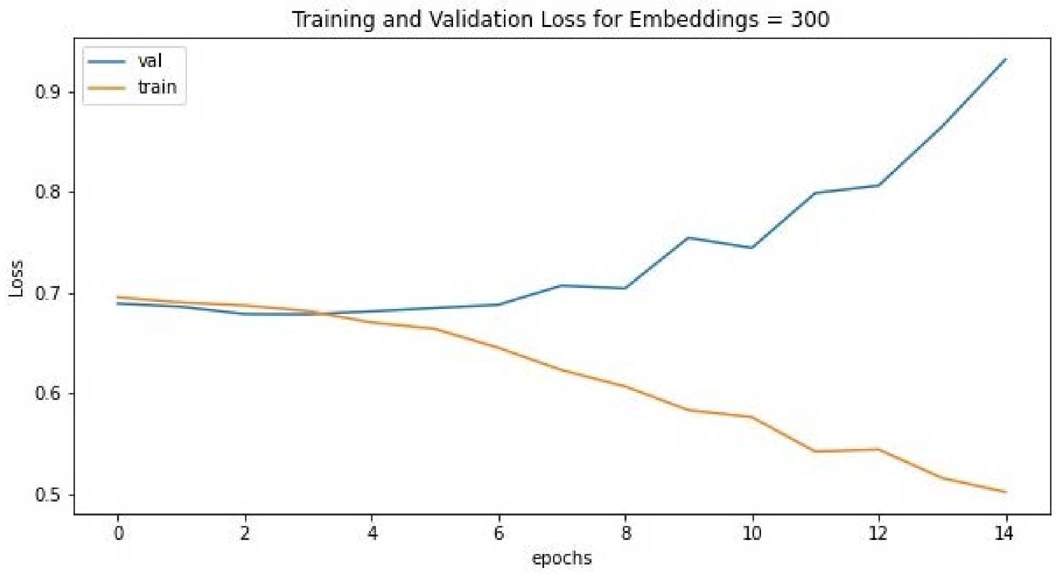


Figure 48: Loss graph of 90s popularity classification

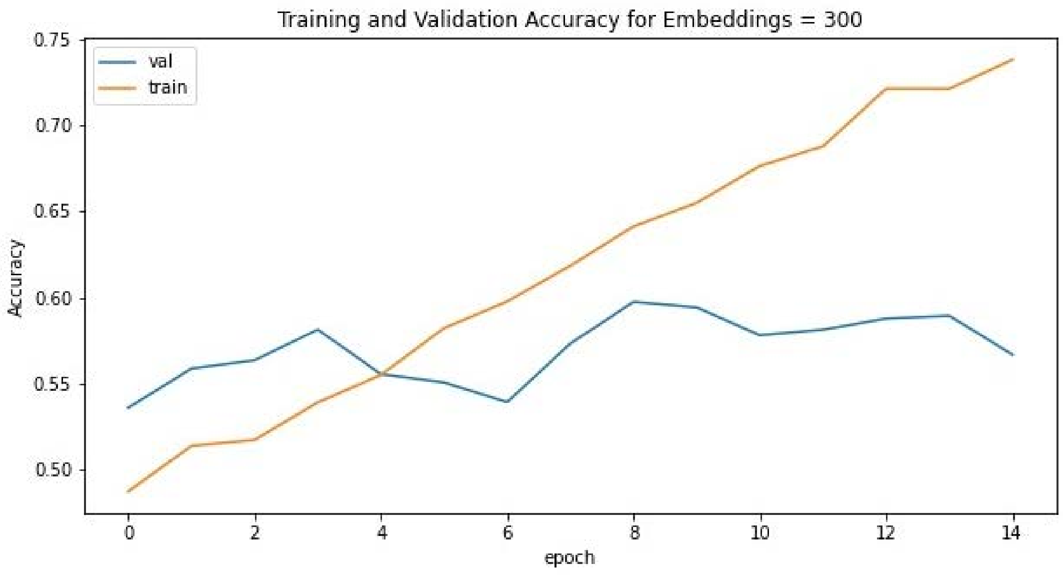


Figure 49: Accuracy graph of 00s popularity classification

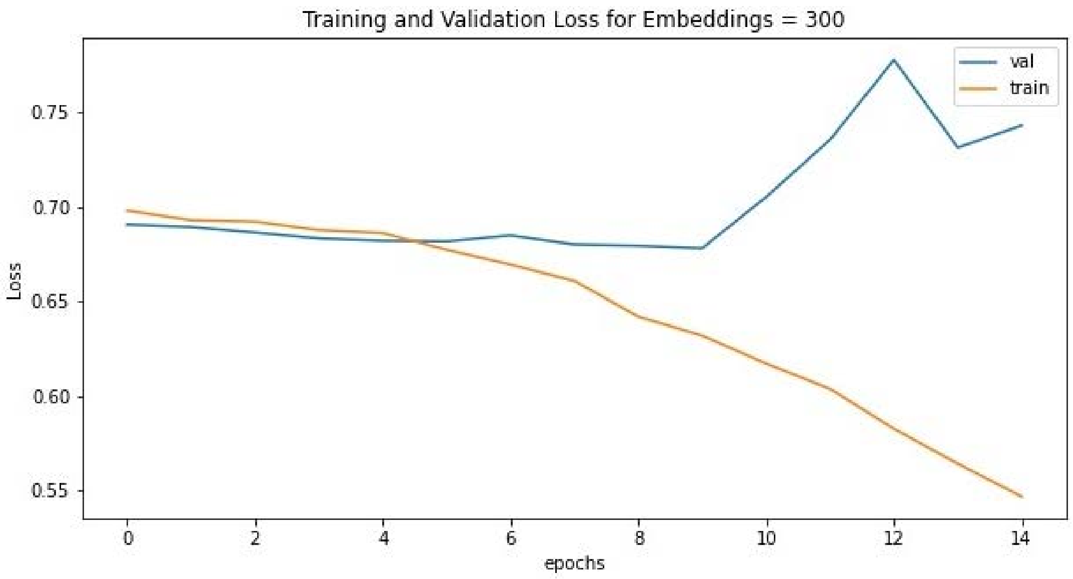


Figure 50: Loss graph of 00s popularity classification

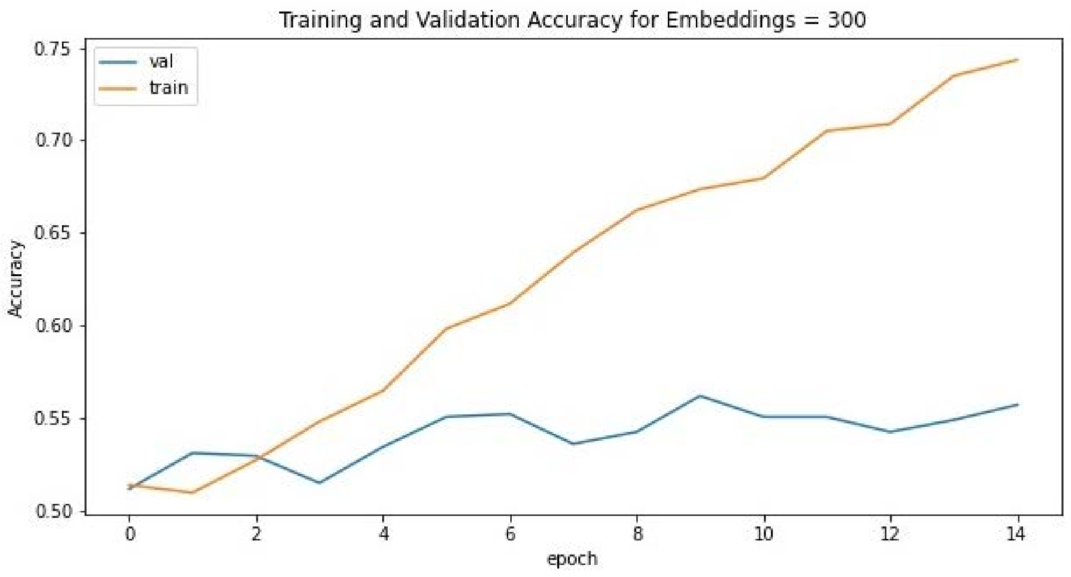


Figure 51: Accuracy graph of 10s popularity classification

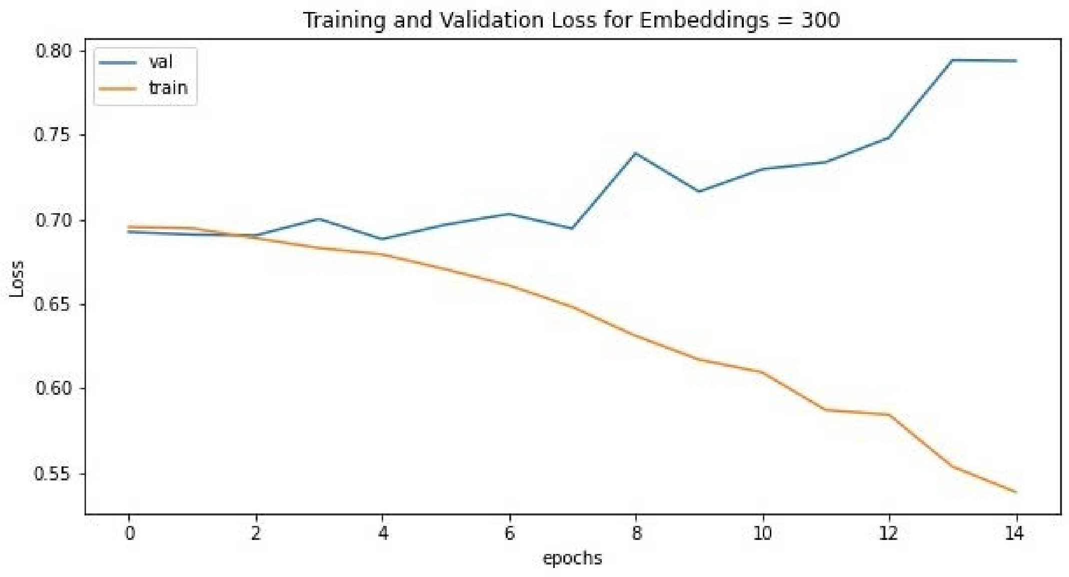


Figure 52: Loss graph of 10s popularity classification

5. Discussion

Data Preprocessing

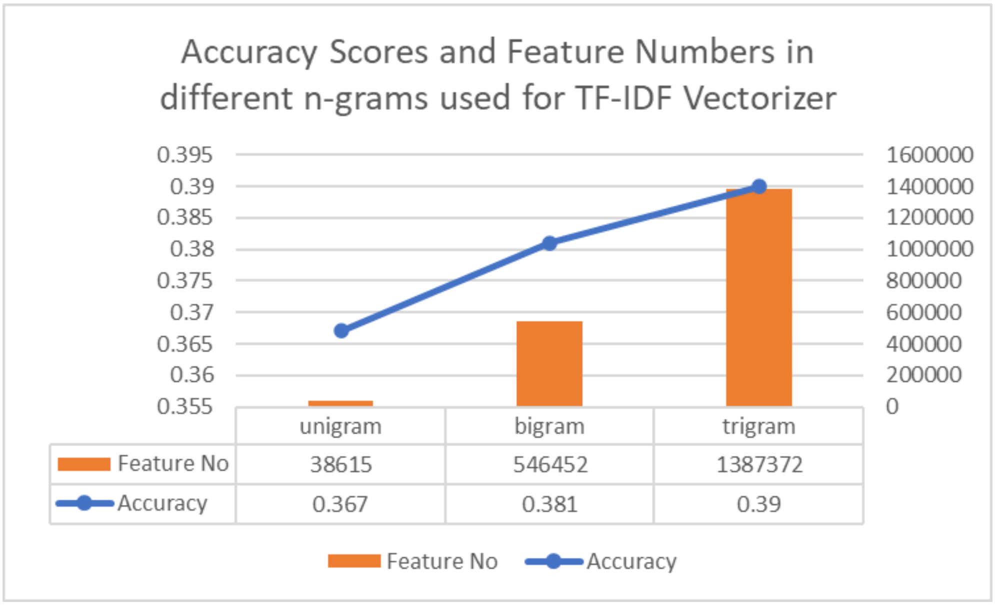
Preprocessing the data in such a way that eliminates the noise nearly completely was unsustainable as when we took the Spotify Web API Documentation and our findings to get rid of as much noise as possible we were left with an extremely small pool of data. Even performing a model like Naive Bayes becomes unreasonable with such a small dataset with a very large amount of features. Therefore, compromises had to be made as noise, sample size, and feature size all affected the model's accuracy.

Furthermore, making sure the lyrics were in English also became a tough obstacle. Songs can have English title and non-English lyrics or vice versa or even have both English and non-English lyrics. Methods of word vectorization function within the constraints of a language, therefore non-English lyrics or parts of lyrics do not work with the models. Categorizing the songs by their languages so that different models work with different languages is also not an option because our goal is steered specifically towards songs in English and as was mentioned previously such a categorization even with the inclusion of both lyrics and titles labeled everything as English.

Moreover, many songs contain non-lyrical content which creates noise even after the removal of stop-words as some came in the form of a string that provided random information that was not part of the lyrics. When such strings that neither show up as "NaN" or are not part of the stop-words they easily pass through the filters of preprocessing, it is not possible to remove them without doing it manually. Since that would require an extreme amount of time due to the size of the dataset and since such problems are faults of the API or the dataset, it leaves us no option. Having more data which was at the same time not as cluttered as it is with noise would have been ideal as it would have helped our models come with more accurate results.

Multinomial Naive Bayes

On the TfidfVectorizer, as the n-gram number increased, the accuracy scores also increased as can be observed in Figure 53. This shows that the sequential information is valuable and important in lyrical text data. Even though the size of the matrix also grew with n, since the classification task is done with a simple probabilistic classifier, the time and computation required for results did not have a significant difference. Hence, we selected trigram for better results. Trigram vectorization is also used in popularity classification.



	unigram	bigram	trigram
Feature No	38615	546452	1387372
Accuracy	0.367	0.381	0.39

Figure 53: The graph and data table of accuracy results and the feature numbers in the classification of periods with TF-IDF Vectorizer.

There are many reasons for the predictions of the Multinomial Naive Bayes model resulting in lower accuracies: the noise in the data, not having enough data, and not valuing sequential information of the words due to the Bag of Words approach. Another very important reason could be due to the amount of features. Having 1387372 number features for a dataset of 20061 can be expected to be too much for the classifier to find out the important features. A strict and detailed feature selection could have helped with the performance of the classifier. However, we wanted to keep the dataset the same for all classifiers for direct comparison. Therefore, we did not change the features. In addition, there are limitations of the TF-IDF vectorization. For example, it only checks for the occurrence of a word in a document, It does not check the place of a word in a document and its relevance. Therefore, the unsatisfying results were expected[16].

Furthermore, the unsuccessful scores can also lead to the suggestion that there are no significant lyrical differences between periods. It is seen that non-neural lyrical classifiers, such as SVMs, k-NN, and Naive Bayes, are found to have struggled to achieve a classification accuracy higher than 50% in previous experiments[17] as well.

[16] P. Ahuja, “How good (or bad) is traditional TF-IDF text mining technique?,” *Medium*, 31-Jan-2020. [Online]. Available: https://medium.com/analytics-vidhya/how-good-or-bad-is-traditional-tf-idf-text-mining-technique-304aec920009. [Accessed: 26-Dec-2021].

[17] Tsaptsinos, Alexandros. “Music Genre Classification by Lyrics using a Hierarchical Attention Network” ICME, Stanford University. [Accessed in: 26-Nov-2021].

Beside the results of the models, when we take a closer look into the confusion matrix to find a relation between the predictions and periods, we can observe that songs from 2010s have one of the biggest accuracies, or F1 score. This suggests a bigger difference in the lyrical data between the 2010s and the rest.

Recurrent Neural Networks with LSTM

Early stopping was not utilized, however it did not negatively affect the results of our models. This is due to the fact that the best model saves the last epoch only if it provides a decrease in loss. This also allowed us to view overfitting very clearly in our graphs. In cases where overfitting was obviously apparent the epoch number has been cut down for the sole purpose of reducing training time.

After the data was processed the dataset was reduced very heavily. Neural networks tend to yield much better results with more data thus, we could have achieved higher accuracy results had we had more data. Also, less noise in the data could have helped us reach a better accuracy as well. We prefered to use LSTMs as it gives better control and results over the data. LSTM is usually setback by the time and memory costs, however according to the dataset we have we decided that trade-off was worth it. Especially due to the fact that the noise in the data and the size of the data made it clear that accuracy was a bigger concern compared to how much data has to be processed or stored. Another problem that LSTM faces is that it is easy to overfit. We tried to overcome that shortcoming by doing a wide range of tests to see when the model starts to overfit and adjusting the parameters according to it. Another option we had was to use GRUs which need less data to generalize. It would have helped with our problem of overfitting but we prefered LSTM for its better end results. Another way to combat overfitting is through data. Perhaps a different dataset with less noise or more samples would yield less overfitting and higher accuracy. It is also possible to use Cross Validation in order to simulate a bigger sample size. Other regularization techniques such as batch normalization can also be applied to prevent overfitting.

For popularity prediction, we achieved different accuracies for different periods ranging from 0.565 to 0.692 using the Word2Vec vectorization method and from 0.551 to 0.735 using the GloVe vectorization method. This difference between accuracies may be caused by the data quality for each period indicating that periods with high accuracies such as 70s have less noise in the data. The hit rate by the API massively contributes to the data quality, therefore a higher hit rate could result in better accuracy for each period. This might also show a correlation between the importance of lyrics in a song's popularity during a certain period.

6. Conclusions

The question we wanted to answer was whether there was a relation between song lyrics and periods and whether song lyrics have a relationship with popularity. Assuming we used optimal models, with accuracy scores keeping under 40% for period classification and under 70% for binary population, we cannot suggest a strong correlation between lyrics and decades/popularity of songs. Therefore, the answer to our problem would be that there is no direct relation between lyrics and periods. However, it is important to consider that in practice we could not obtain an optimal dataset. In addition, we did not perform feature selection to have the models work on the same and full dataset. Skipping feature selection and accepting very noisy data could have led us to miss the relation that might exist in reality.

We were predicting better results from the RNN model due to the importance of sequences in a lyrical data. A text vectorization model valuing the position of the words and sequences should have better understanding of the lyrical data. However, against our expectations, an ML model performed better than a neural network model. This unexpected result is caused by the trouble we had with our RNN model. It was too complex, hence it overfit to the data and the small number of training samples we had for RNN actually made it even worse because neural networks need large amounts of data - while this need might be less significant with ML models. In the case of having better data -more in amount, clean and less noisy- and we didn't have an overfitting problem we might have achieved better results with RNN than ML. Nonetheless, we discovered different ways to represent string data in our exploration with text vectorization and had a clearer comprehension of the difference and working principles of a neural network versus a probabilistic ML model.

For future directions, we mentioned some improvement suggestions in the discussion section of this report. These can be listed as: better data quality (less noise), more data samples, for RNN more regularization techniques such as batch normalization can be applied and GRU can be used instead of LSTM as LSTM is more prone to overfitting.

7. Appendix

Appendix A: Work Allocation

WP	Work Package	Members Involved
1	Background Research	All Members
2	Reports	All Members
3	Retrieving lyrics and dataset formation	Cansu Moran
4	Data Selection and Preprocessing	Elif Kurtay
5	Implementation of Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM)	Öykü Hatipoğlu
6	GloVe integration to RNN model	Öykü Hatipoğlu
7	TF-IDF integration to RNN model	Öykü Hatipoğlu, Elif Kurtay
8	Word2Vec integration to RNN model	Atakan Dönmez
9	Multinomial Naive Bayes with TF-IDF	Elif Kurtay, Gamze Güliter
10	Hyperparameter Optimization for RNN	Cansu Moran, Öykü Hatipoğlu
11	Result Analysis	All Members

Table 9: Work packages and members involved in each package

Table 8 displays our division of work into work packages and the responsible members for each package. **WP1** includes research on similar previous work and learning tools that will be used for the project. **WP2** includes writing the project reports. **WP3** includes the formation of the dataset by retrieving lyrics for the selected dataset. **WP4** consists of a pre-selection of the dataset for lyric retrieval and preprocessing the dataset into a clean format required for text vectorization. Some of these activities in **WP3** and **WP4** are the retrieval of song lyrics from MusixMatch API, deleting non English or non verbal songs, and determining the period of the songs. **WP5** has the implementation of an RNN model suitable for our dataset. The neural network model will utilize Word2Vec, Global Vector(GloVe) and Bag-of-Words(BoW) with TF-IDF word vectorization techniques while Multinomial Naive Bayes will only utilize Bag-of-Words with TF-IDF. **WP6, WP7,** and **WP8** belong to the text vectorization process where each word in string lyrical data is transformed into numerical representations in different ways. These work packages include vectorizing the dataset and using them with RNN. **WP9** includes the implementation of the Multinomial Naive Bayes model with TF-IDF text vectorization. In **WP10**, hyperparameter optimization is performed on RNN. The **WP11** package includes a detailed analysis of the results obtained from different models using different word vectorization techniques and the comparison of these results with other models.

Appendix B

GloVe Hyperparameter Tests

For the purposes of finding the optimal parameters specifically using GloVe the model was tested on the balanced dataset with epochs of 30 and 50; batch sizes of 8, 16, 32, 64, 128, 256; learning rates of 10^-3, 10^-4, 10^-5; hidden layers of 64, 128, 512; dropout probability of 0.2 and 0.5; and GloVe embedding dimensions of 50, 100, 200, and 300 and the following results were collected.

Epochs	Batch Size	Learning Rate	Embedding Dimension	Hidden Layer	Dropout Probability	Test Accuracy	Test Loss
30	8	1,00E-04	300	64	0.5	**1.693**	**0.332**
30	16	1,00E-04	50	64	0.2	**1.725**	**0.288**
30	16	1,00E-04	300	64	0.5	**1.708**	**0.316**
30	32	1,00E-05	50	64	0.5	**1.761**	**0.245**
30	32	1,00E-04	50	64	0.2	**1.725**	**0.291**
30	32	1,00E-04	300	64	0.2	**1.723**	**0.295**
30	32	1,00E-04	300	64	0.5	**1.711**	**0.303**
30	32	1,00E-04	200	64	0.2	**1.729**	**0.274**

30	32	1,00E-04	100	64	0.2	**1.717**	**0.296**
30	32	1,00E-04	200	64	0.5	**1.726**	**0.284**
50	8	1,00E-03	50	128	0.5	**1.716**	**0.312**
50	16	1,00E-03	50	128	0.5	**1.724**	**0.278**
50	16	1,00E-04	50	64	0.5	**1.722**	**0.294**
50	16	1,00E-03	50	64	0.5	**1.722**	**0.294**
50	32	1,00E-03	50	128	0.5	**1.729**	**0.293**
50	32	1,00E-04	50	128	0.5	**1.723**	**0.289**
50	32	1,00E-04	50	64	0.5	**1.729**	**0.278**
50	32	1,00E-03	50	64	0.5	**1.734**	**0.276**
50	32	1,00E-05	50	64	0.5	**1.745**	**0.259**
50	64	1,00E-04	50	128	0.5	**1.721**	**0.288**
50	128	1,00E-03	50	128	0.5	**1.727**	**0.281**
50	256	1,00E-03	50	512	0.5	**1.731**	**0.259**

Table 10: Tests for different parameters with GloVe for decade classification.

Appendix C

Epochs	Batch Size	Learning Rate	Embedding Dimension	Hidden Layer	Dropout Probability	Test Accuracy	Test Loss

10	4	1,00E-03	300	16	0.5	0.685	0.544
15	4	1,00E-03	300	16	0.5	0.691	0.535
15	4	1,00E-03	300	8	0.5	0.679	0.583
15	4	1,00E-03	300	4	0.5	0.688	0.549
15	4	1,00E-03	300	8	0.5	0.576	0.735
15	4	1,00E-03	300	8	0.5	0.648	0.643
15	4	1,00E-03	300	8	0.5	0.687	0.551
15	4	1,00E-03	300	8	0.5	0.678	0.590
10	4	1,00E-03	300	16	0.5	0.685	0.544

Table 11: Tests for different parameters with GloVe for popularity classification.

Appendix D

Word2Vec Hyperparameter Tests

For the purposes of finding the optimal parameters specifically using Word2Vec the model was tested with epochs of 50; batch sizes of 8, 16, 32; learning rates of 10^-3, 10^-4, 10^-5; hidden layers of 64, 128 and the following results were collected. Similar parameters as the parameters of GloVe hyperparameters were chosen to see if some adjustments could yield better results.

Epochs	Hidden Layer	Learning Rate	Batch Size	Test Accuracy	Test Loss
50	64	10^-3	8	**0.303**	**1.708**
50	64	10^-3	16	**0.298**	**1.707**

50	64	10^-3	32	**0.294**	**1.719**
50	64	10^-4	8	**0.319**	**1.711**
50	64	10^-4	16	**0.306**	**1.710**
50	64	10^-4	32	**0.313**	**1.712**
50	64	10^-5	8	**0.283**	**1.724**
50	64	10^-5	16	**0.279**	**1.73**
50	64	10^-5	32	**0.277**	**1.728**
50	128	10^-3	8	**0.302**	**1.709**
50	128	10^-3	16	**0.299**	**1.709**
50	128	10^-3	32	**0.292**	**1.721**
50	128	10^-4	8	**0.315**	**1.714**
50	128	10^-4	16	**0.314**	**1.715**
50	128	10^-4	32	**0.308**	**1.715**
50	128	10^-5	8	**0.284**	**1.727**
50	128	10^-5	16	**0.285**	**1.729**
50	128	10^-5	32	**0.274**	**1.733**

Table 12: Tests for different parameters with Word2Vec for decade classification.