Playlist Generation using Emotion Recognition and Semantic Textual Similarity

Daniel King **Thomas Hayter**

1 Introduction

Similarly to the proposal, try to convince your reader that your project is interesting. Introduce and motivate your problem. Discuss the formulation of your task.

2 Related Work

Short reports do not necessarily need a dedicated related work section; if you are running out of space, you can at least cover relevant work briefly, for example when substantiating your motivations or conclusions.

3 Methodology

The methodology of the project can be split up into different steps, which can be seen in the flowchart below. Each of the different steps will be explored in detail.

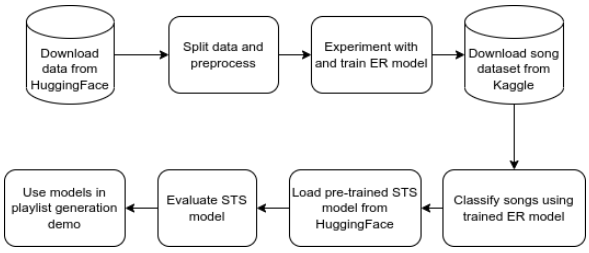

Figure 1: Flow chart showing the steps of the project

3.1 Emotion Recognition Model

The first task to be performed was to train ane motion recognition model to classify a given input of text into an emotion class. The dataset used was the dair-ai/emotion dataset (DAIR.AI, 2023) from HuggingFace. The dataset consists sentences that have been classified into one of six separate emotion classes - sadness, joy, love, anger, fear and surprise. The dataset contains two separate splits; one with 20,000 samples split into a training, testing and validation set, and another with 417,000 samples in one set. For this project, the larger split with 417,000 samples was chosen to train the model.

After being read in, the dataset was checked for imbalances between the labels in the dataset, which was immediately obviously present. The sadness and joy classes had 121,187 and 141,067 samples respectively, whils the surprise class only have 14,972 samples. To combat this imbalance, downsampling was performed to randomly select 5,000 samples from each of the classes to form a new dataset of 30,000 samples. Due to a lack of dedicated hardware to quickly train models, a smaller dataset of only 2,000 samples was created using the same downsampling method - to be used for experimenting with hyperparameters before training the final model. The larger dataset was split into a training and testing set, with a 0.85:0.15 train:test split; resulting in 25,500 samples being used for training and 4,500 for testing. The smaller dataset was also split.

The approach decided upon to create the model was to fine-tune a pre-trained BERT (Devlin et al., 2019) model, by minimising cross entropy loss on the predictions of the classes. This was done using PyTorch and transformers. To preprocess the data, all of the data was encoded using the BertTokeniser, which tokenised the data, then returned both the input ids of all of the tokens, as well as the attention mask for the given input, so that padded values could be ignored (padding was used to ensure all inputs were of the same length). Before tuning the final model, an experiment was run with different model parameters to see which gave the best performance, and this experimentation will be discussed in more detail in the evaluation section. Using the best found parameters, the model was then fine-tuned on the larger training set by minimising cross entropy loss. To speed up the training process, Adam optimiser was used when training. The model returns logits, so an argmax function was used to get the predicted class from the output

of the model. Metrics such as the precision, recall and f1 score across the different classes were calculated. The final trained model was then saved using PyTorch to be loaded in later when making predictions.

3.1.1 Generating Song-Emotion Dataset

To have songs to generate playlists from, the Spotify Million song dataset (Mahajan, 2022) from kaggle was used. The dataset consists of songs with the artist, song title, a link to the song and song lyrics. To create the emotion-song dataset, the lyrics of the songs are first cleaned to remove newline characters, then passed into the trained emotion recognition model. The logits from the output of the model were taken, and any class with positive logits returned was said to be an emotion 'present' in the current song lyrics. Using this output of the emotion classes for each song, a dictionary of emotions mapping to a list of indices of the songs containing the emotions was constructed to be used alongside the song dataset in the final demo.

3.2 STS Model

To calculate the semantic textual similarity between two songs, a MiniLM (Wang et al.) model, which is a distilled version of a large transformer model scuh as BERT (Devlin et al., 2019) was used. The particular model in question had 6 layers, was finetuned on a dataset of 1B sentence pairs and was loaded from HuggingFace (Sentence-Transformers, 2022) using sentence-transformers. To compare the similarity between embeddings generated between by this model, a function to compute cosine similarity was written and used. All pre-processing/encoding needed was performed by the model itself, so no such code was necessary when using this model. An experiment to evaluate the performance of this model was written and run, but shall be discussed in the evaluation section of this report.

3.3 Demo

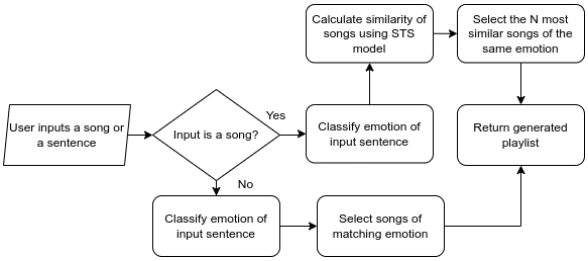

Figure 2: Flow chart showing the program flow of the demo

The flow chart in Figure 2 depicts the code flow of the final demo notebook. Both the ER model and the STS model are loaded into the notebook, as well as the song dataset and the song-emotion dictionary. The user is allowed to generate playlists in two different ways. First, they can enter an input sentence, which will have its emotion detected by the ER model, and then a playlist, of length defined by the user, with songs of the same emotion as the input sentence will be generated. The second option is to input song lyrics. In this option, the emotion of the song is detected and songs from the same emotion are selected as candidates for the playlist. Then, the STS model calculates the similarity between the input lyrics and the songs in the database, and the top N most similar songs are selected to be put into the playlist - where N is the length of the playlist defined by the user.

4 Evaluation

Similarly, describe your experimental setup and what you are trying to demonstrate with the experiment, and your expectations. Try to formulate a hypothesis such as "By doing X, we expect Y to happen, because . . . . If Y does not happen, then it means Z, because...". Present the results of your experiments and show how they support or contradict your hypothesis in a coherent manner, making use of graphs and tables as necessary.

4.1 ER Model

When experimenting with the performance of the ER model, the factors to consider were the learning rate of the model, and how many epochs of training to do. To find the optimal values of these parameters, a grid search was performed using 3 learning rate values of 1e-3, 1e-4, and 1e-5 and 3 epoch numbers of 10, 20, and 30. The data was trained on

the training set and the loss on the testing set measured at each of the epoch values. By increasing the number of epochs, and using a low learning rate, we expect the model to perform better as it would allow for the best fit onto the training data. However, if this doesn't result in the best performance, it is most likely due to the fact that the model has overfit on the training data and so performs worse on the training data. The results of the experiment can be seen in Table 1.

Table 1: A table showing the Cross Entropy loss of the model with different hyperparameter values

Learning rate 1e-3		
10 epochs	20 epochs	30 epochs
1.91	1.84	1.79
Learning rate 1e-4		
10 epochs	20 epochs	30 epochs
0.83	0.94	1.04
Learning rate 1e-5		
10 epochs	20 epochs	30 epochs
0.41	0.48	0.48

From this table, we can see that reducing the learning rate did greatly increase performance. With a high learning rate of 1e-3, the weight updates are too large and the model fails to converge onto a well performing weight settings. In contrast, with a much lower learning rate of 1e-5, the model quickly converges on a good performing set of weights. Unlike what I expected, the model actually performs better with a lower number of epochs, with 10 epochs being the best performing settings. I believe this is because at the larger number of epochs, the model overfit on the training data and thus didn't perform as well when shown unseen data.

4.2 STS Model

To evaluate the performance of the STS model on song data, a custom testing metric was written. Taking a random song from a given emotion class, two other songs were selected - one from the same emotion class, and another from a random differing emotion class. A prediction is considered 'correct' if the model marks the song from the same emotion class as more similar than the song from the different emotion class. We expect the number of correctly classified songs to be at least 60%, as it shows that the model is definitely classifying songs of the same emotions as more similar. However, if this doesn't come to pass, it is hard to say whether it is due to the performance of the model, or the fact that two songs from the same emotion class aren't necessarily more textually similar.

Table 2: A table showing the % accuracy of the STS model across 3 runs of the experiment

Accuracy		
Repeat 1	Repeat 2	Repeat 3
0.57	0.78	0.6

Table 2 shows the results of the experiment across 3 iterations. The average accuracy across the 3 iterations is 65%, which is promising accuracy. However, due to the large standard deviation of 9.3%, and a range of 21% across the 3 iterations, it is fair to say that this is quite an unreliable metric of accuracy for the model. This unreliability is most likely due to the fact that songs from different emotion classes can still be textually similar, which greatly skews the performance of the metric.

4.3 Playlist Generation

There is no mathematical way to evaluate the performance of the final playlist generation, as the performance is subjective. However, in future, it would be possible to run some form of user testing, where users marked songs selected in the generated playlist as appropriate or inappropriate, and some form of performance metric could be calculated based on the percentage of appropriate songs that the system selects.

5 Discussion

The results you obtained should be analysed, interpreted and discussed. Do they correspond to what you have expected? Are they surprising or unexpected? Try to find an explanation for what you have observed, but be careful about making statements that are too strong or too general. Are your explanations substantiated by your experimental evidence? How do your results relate to what others have done? This is another point in the report where you could link back to existing literature, especially when comparing your results with those obtained in previous work. Depending on the depth of the discussion, this can be either its own section, or can be merged with the evaluation section.

Maybe show the confusion matrix and the recall scores here? I put pictures in the images folder but

I don't really get this vs evaluation part. Might be easier to do them both together.

6 Conclusion

Summarise what you have done to address the problem, and what it potentially means. What could be some reasonable follow-on work based on what you have done?

References

DAIR.AI. 2023. dair-ai/emotion dataset.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Shrirang Mahajan. 2022. Spotify million song dataset.

Sentence-Transformers. 2022. all-minilm-l6-v2.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. *MINILM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*.