Hit Song Classification with Audio Descriptors and Lyrics

- Ishank Sharma
- Dissertation Director: Dr. Maria Striki

Can you predict the NEXT hit song to enter Billboard Top-100 charts ?

The Billboard Hot 100 is the music industry standard record chart in the United States for songs, published weekly by Billboard magazine. For our task we use the Billboard Top-100 annual hit list.

MOTIVATION

- Hit Song Science (HSS) aims to infer whether a song's commercial success full scale release of the song.
- It makes the assumption that audio and lyrical features define the underlying structure of any song and can be used to predict its chart topping behavior.
- Widely researched topic in Music Retrieval as it can be used to identify key musical elements pivotal for success and predict commercial popularity of the song.
- Identify Musical and Lyrical trends over different genres and years.
- Can be used for automatic generation of musical piece or help artist to incorporate popular melodies.

PREVIOUS WORK

Demetriou et al. studied effect of vocals and melodies on listeners

Some studies utilised microblog trends to predict upcoming popular song [9]

Researchers have either used deep learning methods with low level audio features such rhythm, spectral information, chords, etc. or lyrics alone [6].

Zangerele et al. combined using high level features- year, rank, danceability and low-level audio features- BPM, chords, keys etc.

Our approach is similar to Zangrele et al. However, for high level features we use lyrics. This combined technique has not been used in any previous studies.

OUR CONTRIBUTION

- ➢ We develop a data pipeline to build and process Hit Song Dataset. Additionally, we design hit song prediction task as a statistical computational modeling problem.
- ➢ Introduce a new dataset comprising of 6938 commercial English audio songs from 1960-2019 (last 6 decades) in Top-100 annual billboard charts as well as non-hits.

Features in our dataset-

- ❖ **Metadata:** Release Year, Title, Artists, Ranking, Hit/Non-hit
- ❖ **Lyrical:** Textual Lyrics
- ❖ **Audio:** Zero Crossing rate, danceability, BPM, Beat count, Chord scale, Keys, etc + 30 second audio preview.

OUR CONTRIBUTION

- Generated lyrics based contextual embedding model developed with BERT. Further, using lyrics embedding for hit song classification task.
- Network architecture utilising both low-level audio descriptors and high-level features such as lyrics to classify hit song.
- Comparison study of three models to predict Billboard hit songs-
  - Low-level (Audio descriptor based)
  - High-level (Lyrics embedding based)
  - Low-level + High-level (Both lyrics and audio descriptors)
- Analysis showing joint learning improves model classification performance.
- Design a software system with Spotify API, Keras, Essentia Framework, Pandas etc.

CHALLENGES

- Unavailability of non-commercial datasets for HSS task.
- Missing metadata, incomplete audio features and no lyrics in current openly available datasets
- Accurately collecting metadata and other song features requires use of several different sources:
  - Internal sources
  - External sources
- Broad academic literature identifying various aspects of music and pivotal features but not converging on specific key factors.
- No single software platform to get lyrics, audio descriptors, metadata, etc for Top-100 Annual Billboard songs. We have to combine several services to extract data.
- No previous studies identifying lyrics as well as audio descriptors for Hit Song classification.

Existing Datasets

- **MSD (Million Song Dataset)**: 1 million songs that are representative for western commercial music released between 1922 and 2011. No billboard filter.

- **KKBOX dataset**: leading music streaming service provider in Taiwan and East Asia. Asia-Pop music library with over 30 million tracks. User listening logs with no billboard filter.

- **Frieler et al. dataset**: 266 pop western songs from earwormery site with midi features and metadata.

OUR DATASET

6938 commercial English audio songs from 1960-2019 in Top-100 annual billboard charts.

Features in our dataset-

- **Metadata:** Release Year, Title, Artists, Ranking, Hit/Non-hit
- **Lyrical:** Textual Lyrics
- **Audio:** Zero Crossing rate, danceability, BPM, Beat count, Chord scale, Keys, etc + 30 second audio preview.

FEATURES IN OUR DATA

tuning_diatonic_strength, spectral_kurtosis_mean, spectral_kurtosis_stdev, ency, chords_strength_mean, chords_strength_stdev, danceability, beats_loudness_mean, beats_loudness_stdev, chords_number_rate, om
e, beats_count
ist, lyrics

DATA COLLECTION STRATEGY

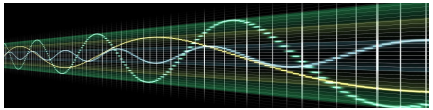
- All software tools and scripts developed in Python 3.6

WIKIPEDIA
The Free Encyclopedia

→ List of songs 1960-2019 in Top-100 annual billboard charts.

Spotify

→ 30 second audio preview.

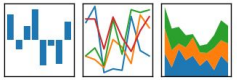


→ Extract audio descriptors.

DATA COLLECTION STRATEGY

- All software tools and scripts developed in Python 3.6

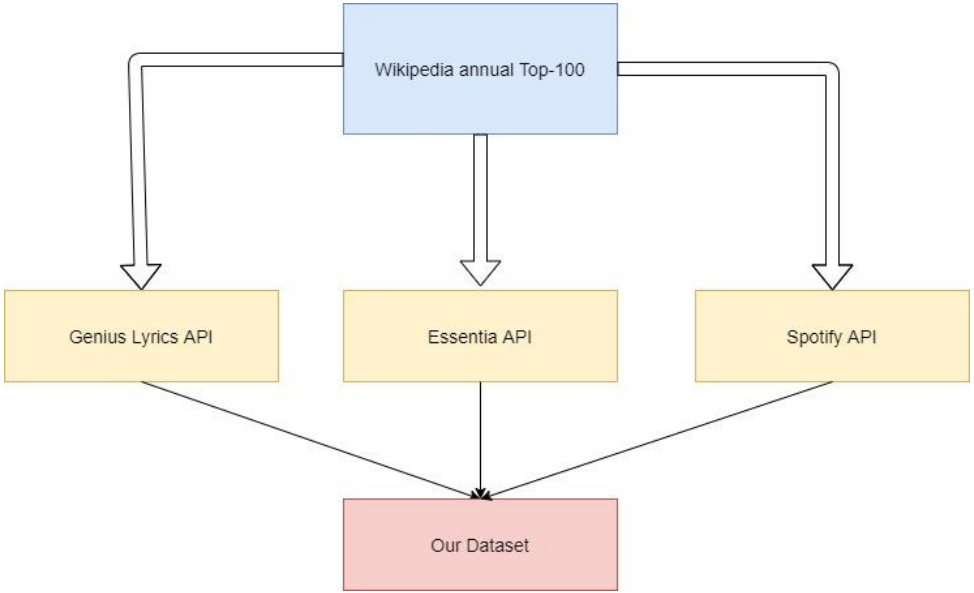
GENIUS → Lyrics for all hit and non-hit songs.

pandas  → Text preprocessing / Data Modeling / Data visualization.

spaCy K

NVIDIA CUDA

DATA COLLECTION STRATEGY



DATA PREPROCESSING

- ★ Remove any missing data and only samples with full data are kept
- ★ All numerical features are standardized and categorical features are encoded with category label.
- ★ Define two types of features- Low-level and High-level. Low-level features corresponds to all audio features which define underlying structure of the song, whereas we use lyrics of the song as a high-level feature representing context of the audio.
- ★ Experimented with min-max and standard scaling for numerical variables. We chose standard scaling based on our experiments.

DATA PREPROCESSING

- ★ Year is also concatenated as a low-level feature.
- ★ Lyrics is a text based feature therefore we apply text normalisation stop word removal and regex cleaning.
- ★ We define a label- 1 (hit-song) or 0 (non-hit). If a song ranking falls within Top-100 Billboard charts it is categorized as hit-song otherwise a non-hit.
- ★ All missing lyrics are imputed with the phrase- '<Title> released by <Artist> in <Year>'

zerocrossingrate_stdev	danceability	...	bpm	chords_key	chords_scale	beats_count	name	year	ranking	artist	lyrics	label
-1.084370	-0.397708	...	-1.219666	9	0	13	theme from a summer place	1965	1	percy faith	summer place \n rain storm \n safe warm \n sum...	1
-1.463847	-0.582061	...	-1.285930	2	0	12	tonight's the night (gonna be alright)	1982	1	rod stewart	Verse 1 \n stay away window \n stay away door ...	1
-0.924485	-1.020842	...	1.119001	3	0	42	stranger on the shore	1967	1	acker bilk	stand watch tide blue dream dream watch ship s...	1
-0.844275	-0.759281	...	0.868882	6	0	37	the way we were	1979	1	barbra streisand	Intro \n hmmmmm hmmmmm \n Hmm hmm hmmmmmm hmmm...	1

Lyrics based Contextual Embedding

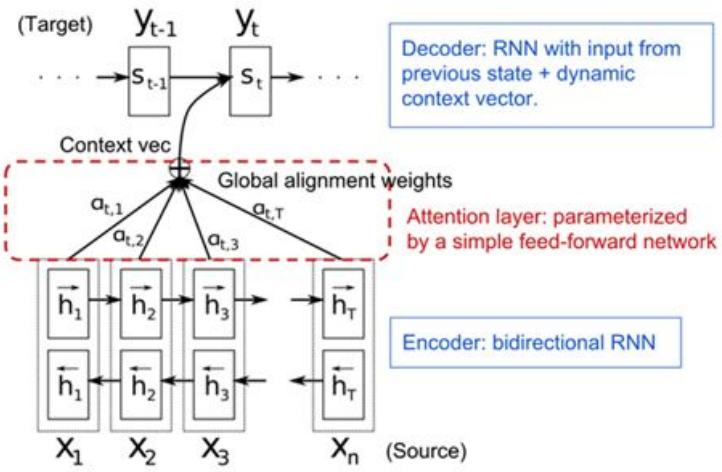
Contextual embeddings assign each word a representation based on its context, thereby capturing uses of words across varied contexts and encoding knowledge that transfers across languages [10]

Method	Architecture	Encoder	Decoder	Objective	Dataset
ELMo	LSTM	✗	✓	LM	1B Word Benchmark
GPT	Transformer	✗	✓	LM	BookCorpus
GPT2	Transformer	✗	✓	LM	Web pages starting from Reddit
BERT	Transformer	✓	✗	MLM & NSP	BookCorpus & Wiki
RoBERTa	Transformer	✓	✗	MLM	BookCorpus, Wiki, CC-News, OpenWebText, Stories
ALBERT	Transformer	✓	✗	MLM & SOP	Same as RoBERTa and XLNet
UniLM	Transformer	✓	✗	LM, MLM, seq2seq LM	Same as BERT
ELECTRA	Transformer	✓	✗	Discriminator (o/r)	Same as XLNet
XLNet	Transformer	✗	✓	PLM	BookCorpus, Wiki, Giga5, ClueWeb, Common Crawl
XLM	Transformer	✓	✓	CLM, MLM, TLM	Wiki, parellel corpora (e.g. MultiUN)
MASS	Transformer	✓	✓	Span Mask	WMT News Crawl
T5	Transformer	✓	✓	Text Infilling	Colossal Clean Crawled Corpus
BART	Transformer	✓	✓	Text Infilling & Sent Shuffling	Same as RoBERTa

PreTrained models available 2020 .Table adapted from [10]

TRANSFORMERS

- Scaled Dot-Product Attention
- Self-Attention
- Multi-head Self-Attention
- Positional Encodings
- Residual Connection



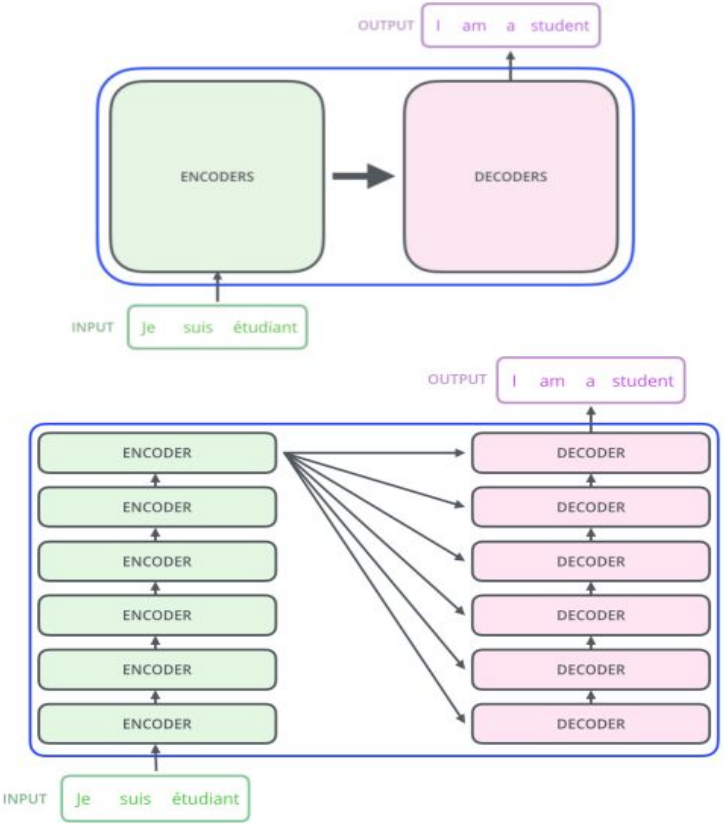
TRANSFORMERS

- Sequential computation inhibits parallelization.
- Both convolution & recurrent models process words in a particular order.
- No explicit modeling of long and short range dependencies.
- We want to model hierarchy and temporal information.

More general definition of attention:

Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

Transformer: a seq2seq model



Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

$$N = 6$$
$$d_{\text{model}} = 512$$

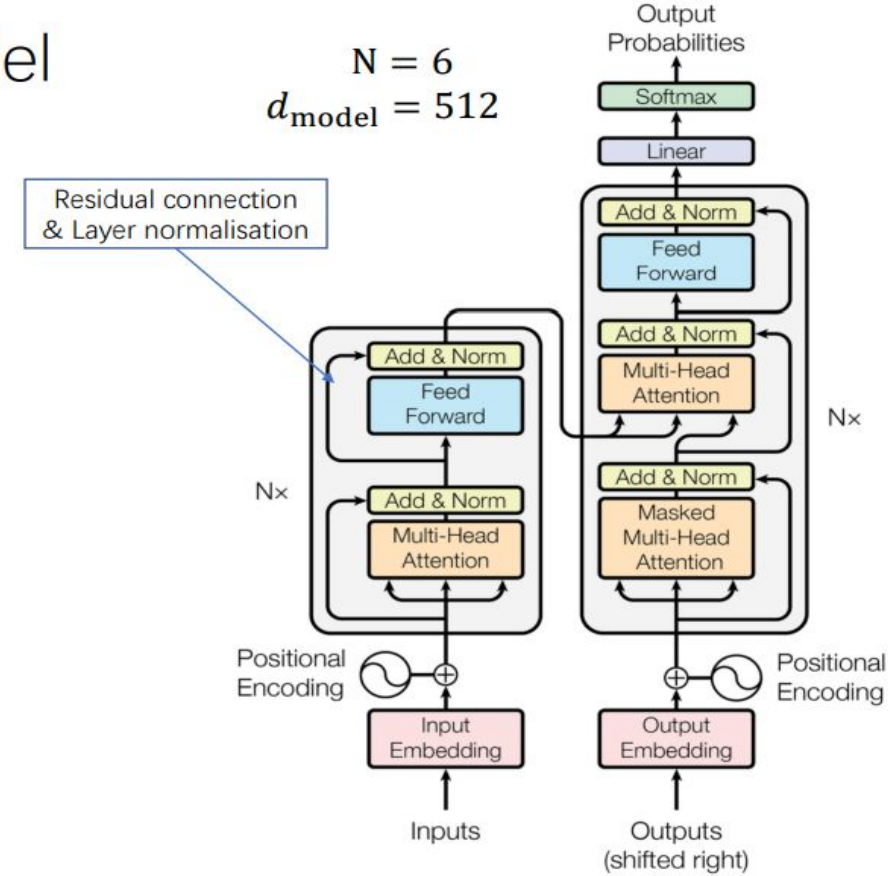
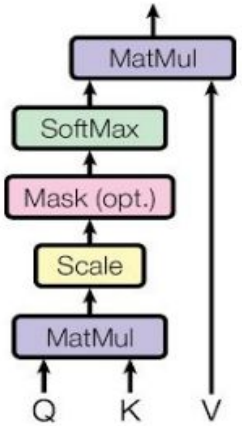
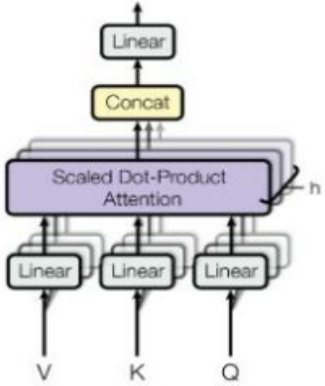


Figure 1: The Transformer - model architecture.

Figure in (Vaswani *et al.*, 2017)

Scaled Dot-Product Attention





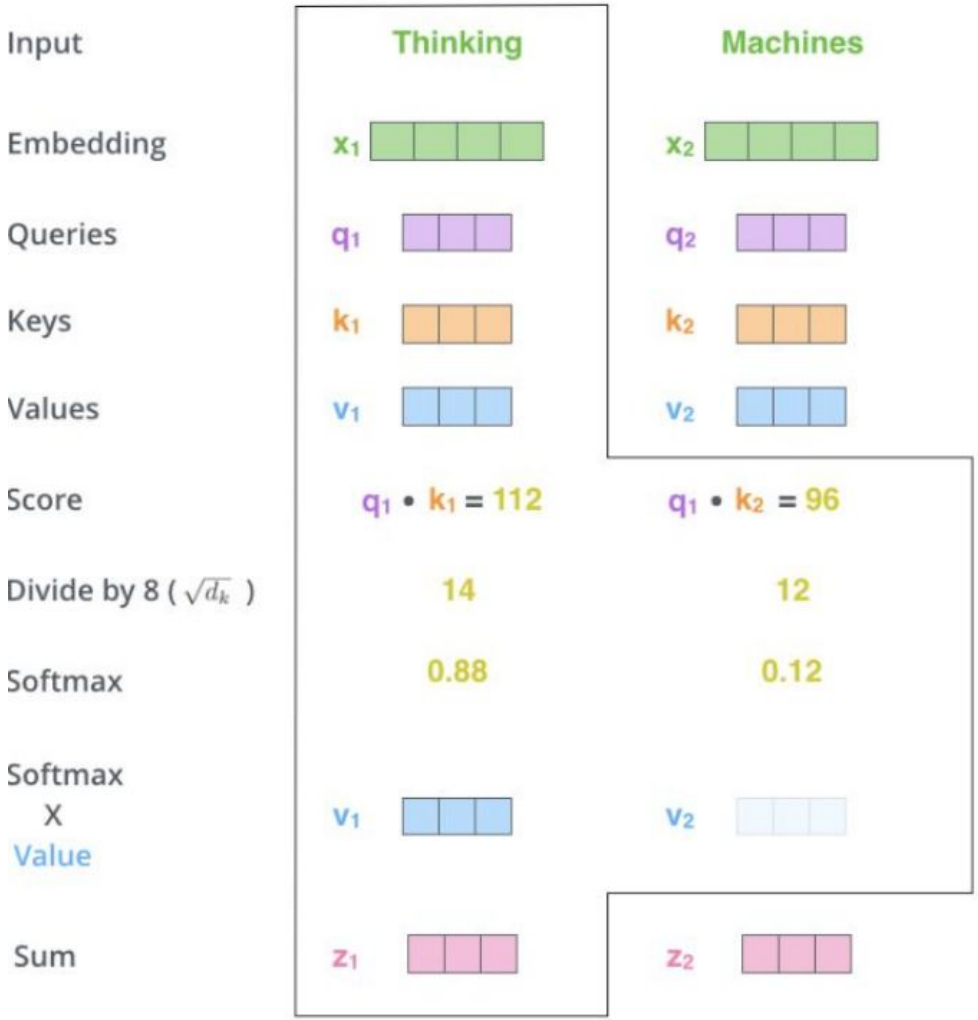
Vaswani, Ashish, et al. "Attention is all you need." *Advances in Neural Information Processing Systems*. 2017.

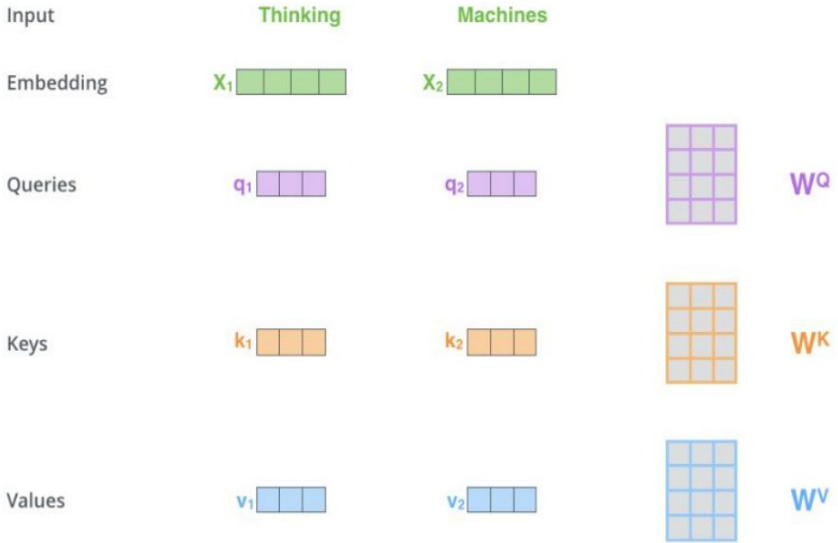
$$Q \in \mathbb{R}^{q \times k} \; K \in \mathbb{R}^{k \times l} \; V \in \mathbb{R}^{l \times v}$$

We choose the dimensions q, k, and v. l is the number of elements to attend to.

https://www.cs.ubc.ca/~lsigal/532S_2018W2/3c.pdf

SELF ATTENTION





https://www.cc.gatech.edu/classes/AY2020/cs7643_fall/slides/L16_attention_transformers.pdf

Dot product attention-

$$Attention(Q, K, V) = softmax\left(\frac{QK}{\sqrt{k}}\right)V$$

- Weighted sum over elements to attend to.
- Scaling to counteract saturating the softmax function, leading to small gradients.

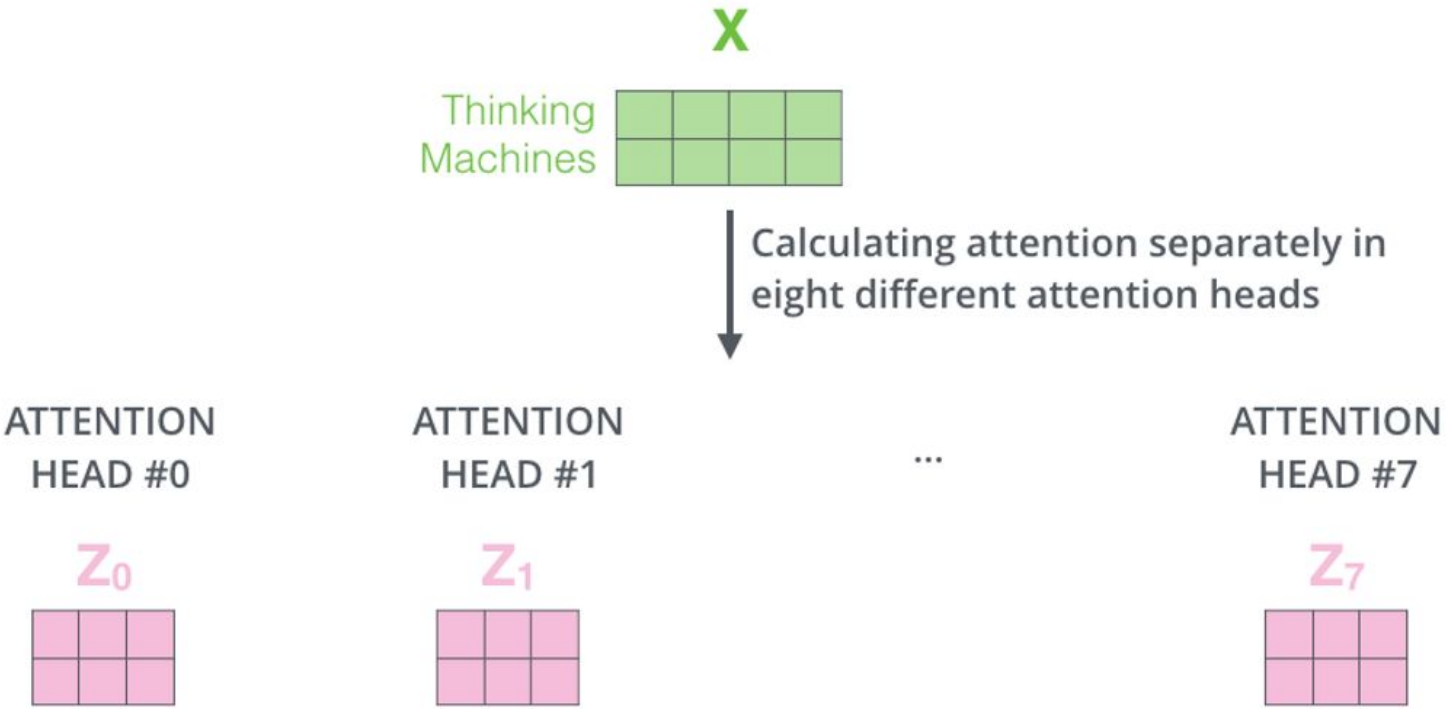
Multi-Head Attention-

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

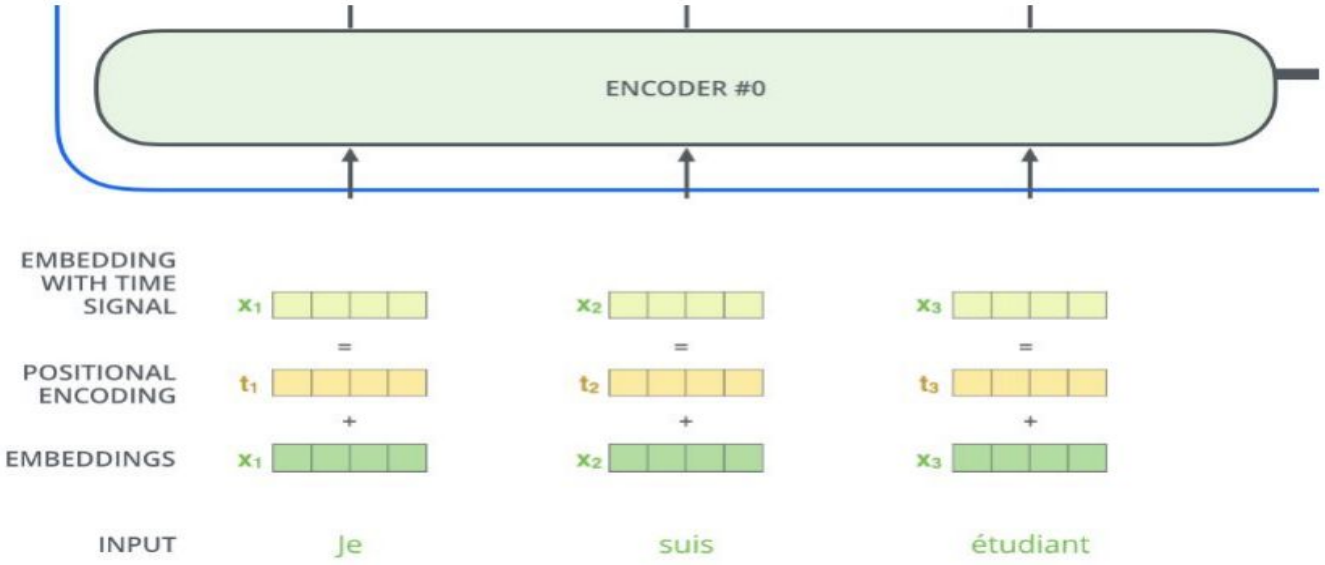
$$head_i = Attention(QW_i^Q, W_i^K K, VW_i^V)$$

- Projection into smaller dimensions instead of the dimensional output from previous layers.
- We can consider these as intermediate embeddings
- Choose parallel attention layers

https://www.cs.ubc.ca/~lsigal/532S_2018W2/3c.pdf


X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7

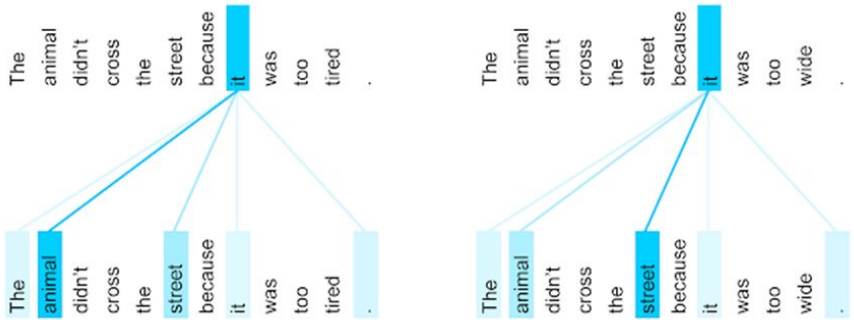
We want to inject some temporal information so that the model knows the relationships between words based on their context (distance & word order is important)



– represent periodicity of positions: a continuous way of binary encoding of position

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

TRANSFORMERS



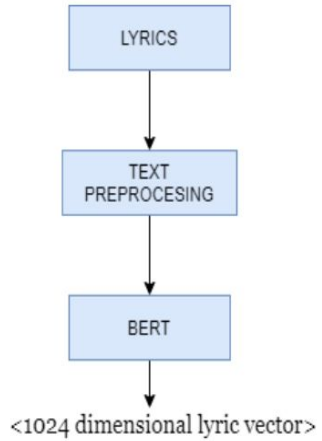
The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Why BERT?

1. SOTA model for downstream NLP tasks.
2. Provides contextual embedding for OOV words.
3. Doesn’t require specific encoding of input (unlike RNNs)
4. Encodes short and long term temporal dependencies.
5. Cheap computation with pretrained model weights.
6. Encodes Segment embeddings

Lyrics based Contextual Embedding

Python library "Bert-as-service".
For every song lyric we generate an Embedding.



'bert-large-uncased' model which has -

24-layer, 1024-hidden, 16-heads (340M parameters)

1024 dimensional output embedding for each lyric our our song.

https://arxiv.org/abs/1810.04805

METHODS

Batch Norm:

$$x_{new} = \frac{x - \mu}{\sigma}$$

- Batch Normalization (BN) is a normalization method/layer for neural networks.
- Usually inputs to neural networks are normalized to either the range of [0, 1] or [-1, 1] or to mean=0 and variance=1
- BN essentially performs Whitening to the intermediate layers of the networks.
- BN allows higher learning rates. (Because of less danger of exploding/vanishing gradients.)

METHODS

Step Decay:

```
lr = lr0 * drop^floor(epoch / epochs_drop)
```

Drop learning rate by a factor every nth step.

Exponential Decay:

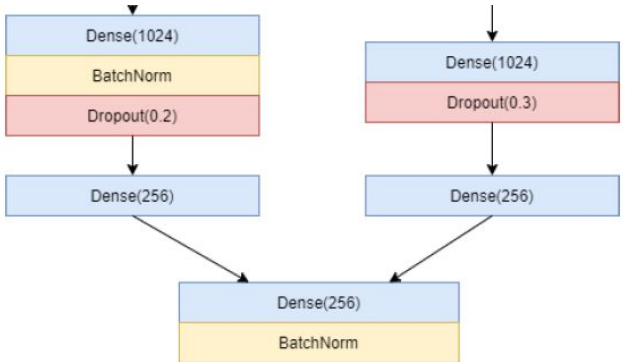
```
lr = lr0 * e^(-kt)
```

Drop learning rate exponentially after nth iteration with condition.

METHODS

Feature Aggregation Layer:

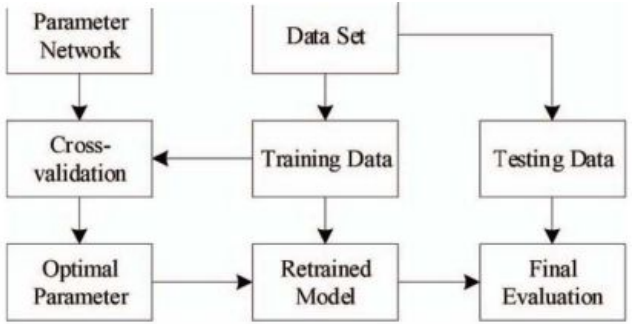
```
tf.keras.layers.Concatenate(axis=-1, **kwargs)
```




```
>>> x = np.arange(20).reshape(2, 2, 5)
>>> print(x)
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]
 [[10 11 12 13 14]
  [15 16 17 18 19]]]
>>> y = np.arange(20, 30).reshape(2, 1, 5)
>>> print(y)
[[[20 21 22 23 24]]
 [[25 26 27 28 29]]]
>>> tf.keras.layers.Concatenate(axis=1)([x, y])
<tf.Tensor: shape=(2, 3, 5), dtype=int64, numpy=
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [20, 21, 22, 23, 24]],
       [[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19],
        [25, 26, 27, 28, 29]]])>
```

GRID SEARCH CV

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR


gsc = GridSearchCV(
        estimator=SVR(kernel='rbf'),
        param_grid={
            'C': [0.1, 1, 100, 1000],
            'epsilon': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05,
0.1, 0.5, 1, 5, 10],
            'gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5]
        },
        cv=5, scoring='neg_mean_squared_error', verbose=0, n_jobs=-1)
```



MODELS

1. **Audio model:**
    a. Low-level abstraction of song.
    b. Fully connected network with audio descriptors as input.
2. **Lyrics model:**
    a. High level abstraction of song.
    b. Fine Tuned BERT embedding.
3. **Audio + Lyrics model:**
    a. Using joint objective learning with both high and low-level abstraction.
    b. Feature Aggregation Layer concatenation.

All three models developed using neural network architectures.

AUDIO MODEL

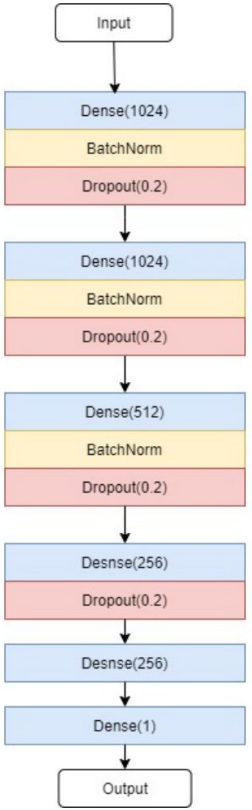
Input: 19 low-level audio features

Output: Label 1 (Hit) or 0 (Non-hit)

Batch size: 1024

Objective: Binary Cross Entropy Loss

- Parameter selection based on skopt (hyperparameter optimization library)
- Continuous variables- Standard scaling
- Categorical variables- Categorical scaling



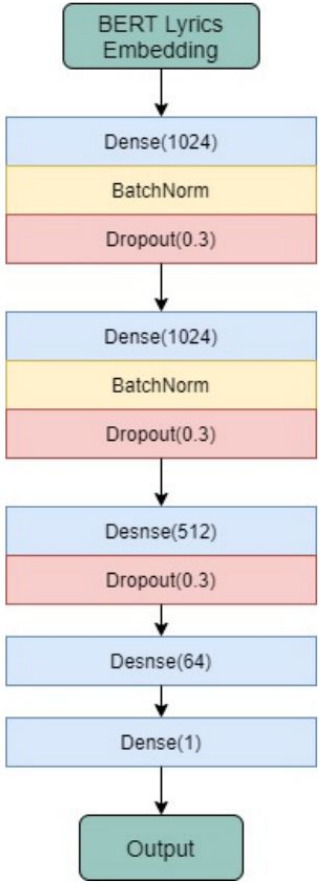
LYRICS MODEL

Input: 1024 dimensional BERT embeddings

Output: Label 1 (Hit) or 0 (Non-hit)

Batch size: 128

Objective: Binary Cross Entropy Loss



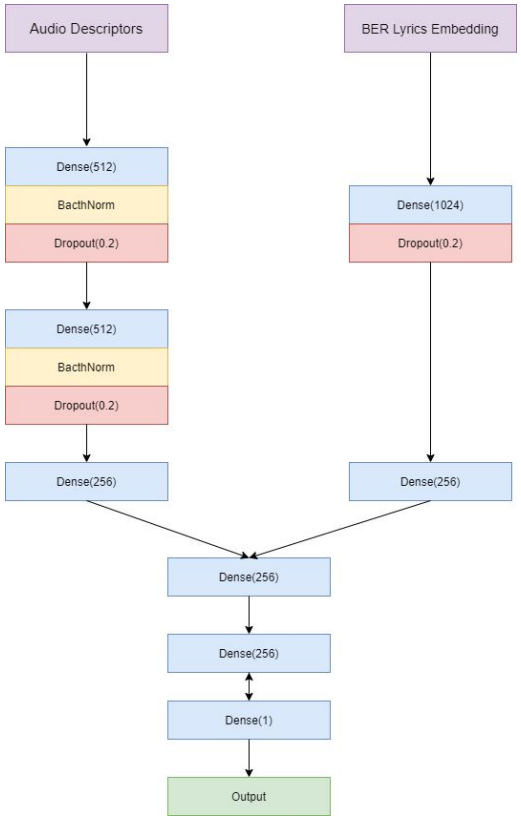
AUDIO + LYRIC MODEL

Input: 19 low-level audio features + 1024 dimensional lyrics embedding

Output: Label 1 (Hit) or 0 (Non-hit)

Batch size: 512

Objective: Binary Cross Entropy Loss



- Feature aggregation dense layer (concatenation)
- Additional fully connected network to train joint objective.

Example code for Audio+Lyrics model:

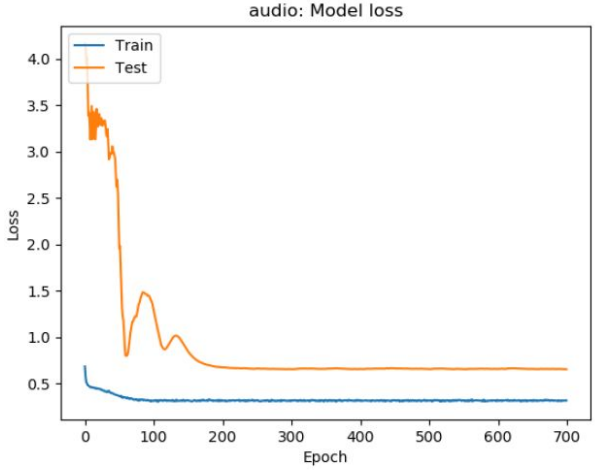
```
def lyrics_audio_model():
    # lowlevel
    input1 = Input(shape=(19,))
    low1 = Dense(512, activation='relu', kernel_initializer='random_uniform')(input1)
    bacthnorm1 = BatchNormalization()(low1)
    dp1 = Dropout(0.3)(bacthnorm1)
    low2 = Dense(512, activation='relu', kernel_initializer='random_uniform')(dp1)
    batchnorm2 = BatchNormalization()(low2)
    dp2 = Dropout(0.3)(batchnorm2)
    low3 = Dense(256, activation='relu', kernel_initializer='random_uniform')(dp2)

    # highlevel
    high1 = Input(shape=(1024,))  # 1024 bert
    dphigh = Dropout(0.2)(high1)
    high2 = Dense(256, activation='relu', kernel_initializer='random_uniform')(dphigh)

    # shared layer
    shared = Dense(256, activation='relu', kernel_initializer='random_uniform')
    op1 = shared(low3) # lowlevel
    op2 = shared(high2) # highlevel

    merge_layer = Concatenate()([op1, op2])

    f1 = Dense(256, activation='relu', kernel_initializer='random_uniform')(merge_layer)
    f2 = Dense(256, activation='relu', kernel_initializer='random_uniform')(f1)
    predictions = Dense(1, activation='sigmoid')(f2)

    adam = Adam(learning_rate=0.01)
    model = Model(inputs=[input1, high1], outputs=predictions)
    model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy',  f1_m,precision_m, recall_m])

    return model
```

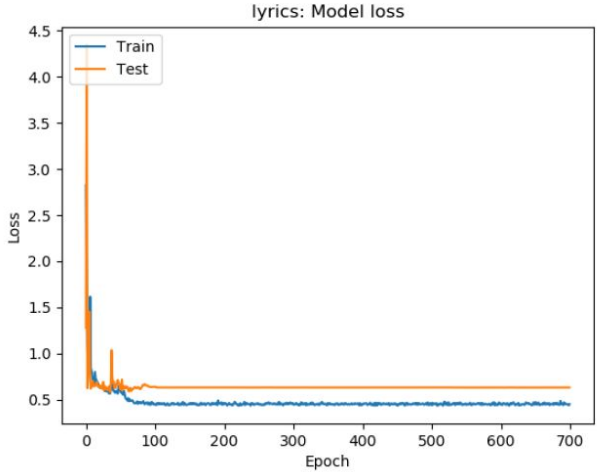
MODELING STRATEGY

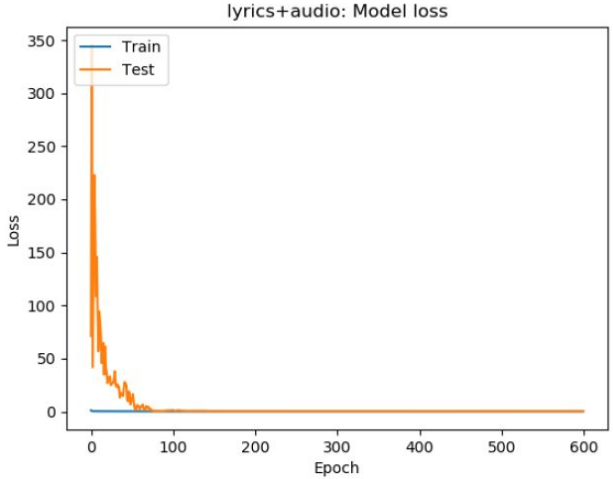
- Train/Validation/Test split: 52.5% - 22.5% - 25%
- Keras framework for model development.
- GPU based model training (NVIDIA Tesla K40 GPU)
- Develop model training/testing pipeline for GPU based environment.
- Test set consisting of 1735 samples.
- Skopt for hyperparameter optimisation.

https://github.com/scikit-optimize/scikit-optimize

RESULTS


audio: Model loss
Train
Test
Loss
Epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
100
200
300
400
500
600
700


lyrics: Model loss
Train
Test
Loss
Epoch
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
100
200
300
400
500
600
700


lyrics+audio: Model loss
Train
Test
Loss
Epoch
350
300
250
200
150
100
50
0
0
100
200
300
400
500
600

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

	Relevant	Not Relevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Sensitivity of a classifier is the ratio between how much were correctly identified as positive to how much were actually positive. (Recall)

Specificity of a classifier is the ratio between how much were correctly classified as negative to how much was actually negative.

Model Performances

	Accuracy	F1	Precision	Recall
audio	0.7043	0.6454	0.7318	0.5943
lyrics	0.7360	0.7176	0.7070	0.7375
audio+lyrics	0.7689	0.7490	0.7629	0.7530

RESULTS

- All the three approaches successfully disambiguate hit or non-hit songs. Our lyrics based sentence embedding network outperform audio descriptor model with better sensitivity.
- Lyrics embedding derived from BERT are effective for Hit Song Classification task and provides a high level abstraction of song features.
- Our fine tuned lyrics embedding generated from BERT is better for depicting a song likability in comparison to our selected audio descriptors.
- Joint learning approach with a shared layer and common objective for song classification, proves to be effective technique in achieving high accuracy with moderate training cost and modifications to existing architectures.
- Both audio and lyrical features can improve generalization capabilities and accuracy for hit song classification task. It performs better than other two models.

FUTURE WORK

1. Experiment with different model architectures utilising feature aggregation layers.
2. Analyse lyrics based contextual embedding for better interpretability of model.
3. Analyse multiple audio descriptors to explain musical trends.
4. Compute or engineer audio or lyrics based features for Hit Song Science and musical score generation.
5. Generate better lyrics based embeddings with efficient training procedures and architecture design.
6. Integrate social media contexts, sale trends, reviews etc with joint objective learning or multitask learning for similar task.
7. Examine lyrics based trends expanding over several years.

Thank you

References

1. https://nlp.stanford.edu/seminar/details/jdevlin.pdf
2. https://cgi.csc.liv.ac.uk/~hang/ppt/BERT%20and%20Transformer.pdf
3. https://scikit-optimize.github.io/stable/
4. https://keras.io/
5. https://www.researchgate.net/publication/315795317_Revisiting_the_problem_of_audio-based_hit_song_prediction_using_convolutional_neural_networks
6. A. Singhi and D. G. Brown, “Hit song detection using lyric features alone,” Proceedings of International Society for Music Information Retrieval, 2014.
7. Andrew Demetriou, Andreas Jansson, Aparna Kumar, and Rachel Bittner. Vocals in music matter: The relevance of vocals in the minds of listeners. In Proc. International Society for Music Information Retrieval Conference, 2018.

References

8. Eva Zangerle, Martin Pichl, Benedikt Hupfauf, and Günther Specht. Can microblogs predict music charts? an analysis of the relationship between #nowplaying tweets and music charts. In Proc. International Society for Music Information Retrieval Conference, pages 365–371, 2016.
9. Kim, Yekyung, Bongwon Suh, and Kyogu Lee. "# nowplaying the future Billboard: mining music listening behaviors of Twitter users for hit song prediction." Proceedings of the first international workshop on Social media retrieval and analysis. 2014.
10. Liu, Qi, Matt J. Kusner, and Phil Blunsom. "A Survey on Contextual Embeddings." arXiv preprint arXiv:2003.07278 (2020).