Introduction

We hope to complete a lyrics recommendation system. The user enters a paragraph of text, and we recommend the most matching lyrics to the user through the text. At first, we imagined starting from emotions, classifying sentences and matching them with the most similar emotion lyrics. Later During the process of the project, we found that the performance of the classifier we got was very poor due to the lack of labeled data (for the five classifications, a large amount of labeled data is required for each category), so we hope not only to match lyrics based on sentiment classification, but also we hope to further match related lyrics by calculating sentence similarity. Here we learn from the initial data set and use the BERT model to complete the task. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-train BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks[1], we add an output here Layer can get the result of sentiment classification. For sentence similarity, we use Google Sentence Encoder to calculate. The overall performance of the model meets our expectations.

Related work

Here we mainly introduce two aspects, firstly related work on sentiment classification, and secondly about the sentence similarity.

Sentiment classification is an important subtask of natural language processing, and there are many methods to solve sentiment classification, including supervised and unsupervised methods. Among the supervised methods, early papers used all supervised machine learning methods (such as support vector machines, maximum entropy, naive Bayes, etc.) and feature combinations. Unsupervised methods include different methods using sentiment dictionaries, grammatical analysis, and syntactic patterns.

We have selected several methods based on machine learning: RNN, LSTM, CNN, BERT

RNN：

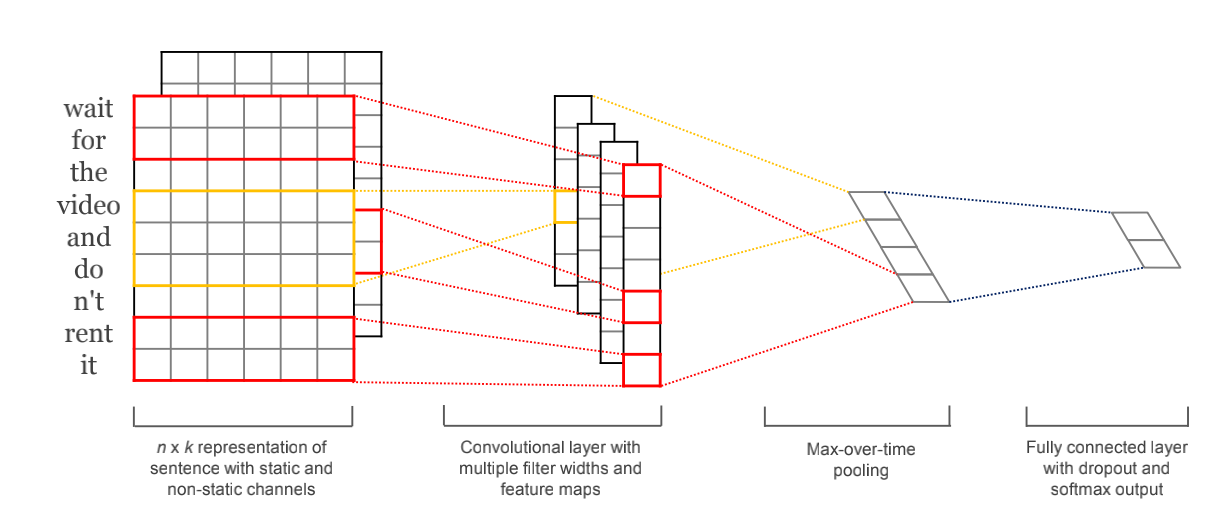
Generally, a word (one-hot representation) is first mapped to its word embedding representation, and then as the input x of the recurrent neural network at each moment. In addition, you can extract the state of the hidden layer at the last moment as a sentence representation to build a classification model

LSTM：

For longer sequence data, the gradient disappears or explodes easily during the training of the recurrent neural network. LSTM overcome this by having an extra recurrent state called a cell c which can be thought of as the "memory" of the LSTM - and the use use multiple gates which control the flow of information into and out of the memory.

CNN：

Convolutional neural networks can extract a variety of local features, and abstract them to obtain higher-level feature representations. Convolutional neural networks are mainly composed of convolution and pooling operations. Features obtained by convolution kernels The stitching together is the fixed-length vector representation of the text, and finally the text classification result is obtained through a softmax fully connected layer.



BERT:

少个BERT的相关工作

Then, it is the calculation of sentence similarity. Similarity calculation is one of the subtasks of natural language processing. There have been many solutions proposed by predecessors:

Using the average of the word vector represents a phrase, and this can be baseline.

Word Mover’s Distance:

A novel distance function between text documents.[2] Record the shortest distance between each word in one sentence and the word in another sentence, and use this distance as a measure of the similarity between the two sentences

Smooth Inverse Frequency:

SIF as the weight to modify and average the sentence vector, that is, the word with lower frequency is considered to be caused by itself in the sentence. The projection on the first principal component (principal component) of the matrix is to erase the common information of all sentences and the distance between each sentence vector.

[InferSent](https://github.com/facebookresearch/InferSent):

An encoder based on a bi-directional LSTM architecture with max pooling, trained on the Stanford Natural Language Inference (SNLI) dataset [5],yields state-of-the-art sentence embeddings compared to all existing alternative unsupervised approaches like SkipThought or FastSent, while being much faster to train. (Supervised Learning of Universal Sentence Representations from Natural Language Inference Data)

[Google Sentence Encoder](https://www.tensorflow.org/hub/modules/google/universal-sentence-encoder/1)：

Limited amounts of training data are available for many NLP tasks. This presents a challenge for data hungry deep learning methods. Given the high cost of annotating supervised training data, very large training sets are usually not available for most research or industry NLP tasks. This sentence embeddings can be used to obtain surprisingly good task performance with remarkably little task specific training data[6].

Since it is difficult to obtain a large amount of annotated data, we chose Google Sentence Encoder to judge sentence similarity, and we can achieve better results on the basis of a small amount of labeled data.

Data

Our project is divided into two phases:

In the first stage, we use the convolutional neural network to classify emotions. Here, the data we use to train the network is TREC. A question dataset task involves classifying a question into 6 question types. The 6 labels (for the non-fine-grained case) correspond to the 6 types of questions in the dataset:

HUM for questions about humans

ENTY for questions about entities

DESC for questions asking you for a description

NUM for questions where the answer is numerical

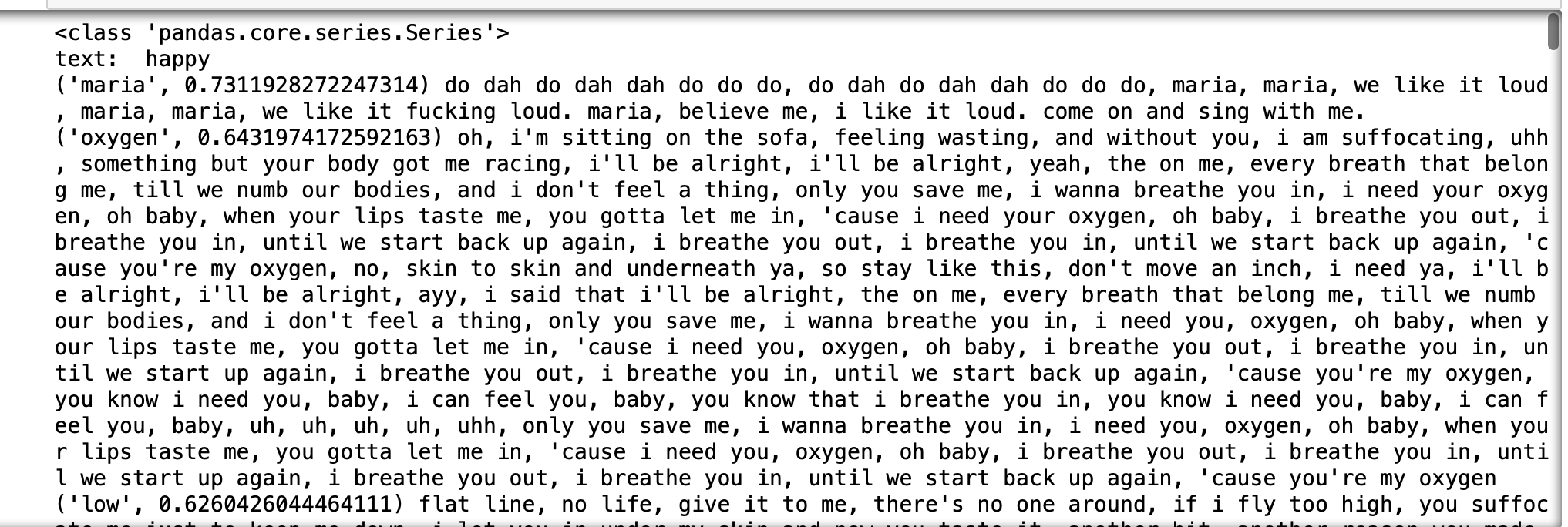
LOC for questions where the answer is a location

ABBR for questions where the answer is a location

Since the training sample is a single question and the label does not match, the result of using this classifier is very unsatisfactory.

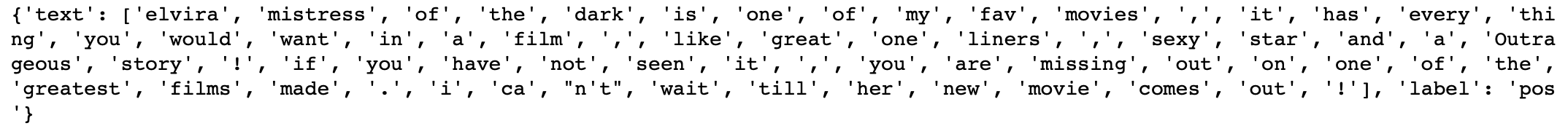
In addition, the dataset(SMILE Twitter Emotion dataset) we found in the preparation phase cannot be used because of too little labeled data.It only contains 3,085 tweets, with 5 emotions namely anger, disgust, happiness, surprise and sadness.

This is the model result we got at the beginning. When the input is a positive word or sentence, the result is often repeated and cannot be classified.



In the second stage, we found that we can use BERT to deal with sentiment classification. Because there is a pre-training model, the dataset we use to train the classifier can be selected more widely. The pre-training model we load here is bert-base-uncase , The dataset uses IMDB.

This is an example in the dataset：



Method

Describe which method you used to answer your question / solve the problem, in other words, HOW did you do it. If you used existing libraries, mention them. Make sure anyone with basic NLP

knowledge can follow what you did. Refer to the literature where relevant. (It’s perfectly OK if you

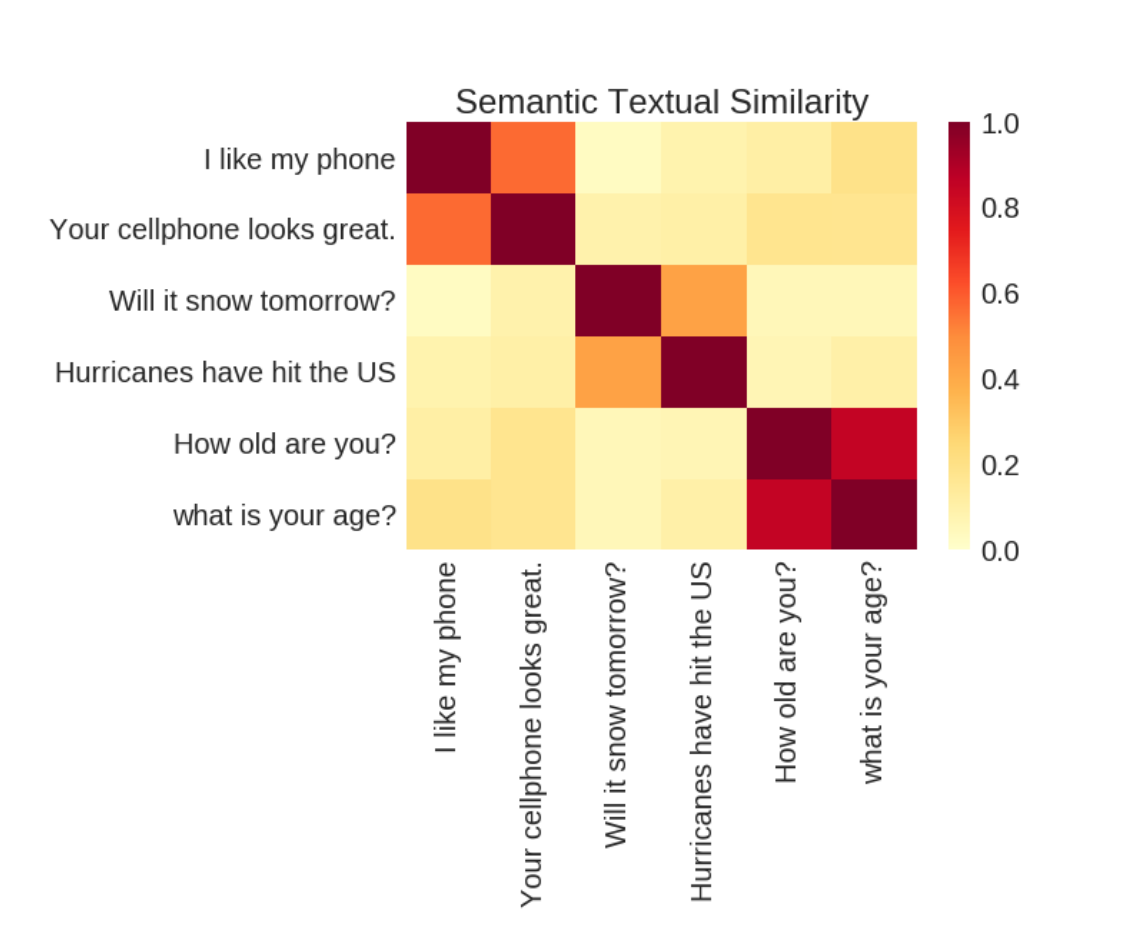
resort to (much) simpler methods than used in the literature – after all, you have less experience and less time.)

Also describe if you have done any preprocessing of the dataset and how you divided it into training

and test sets (if applicable).

First of all, we want to calculate the similarity between the input text and a specific lyrics. Let's first look at how to calculate the similarity, and then introduce how to preprocess the lyrics dataset.

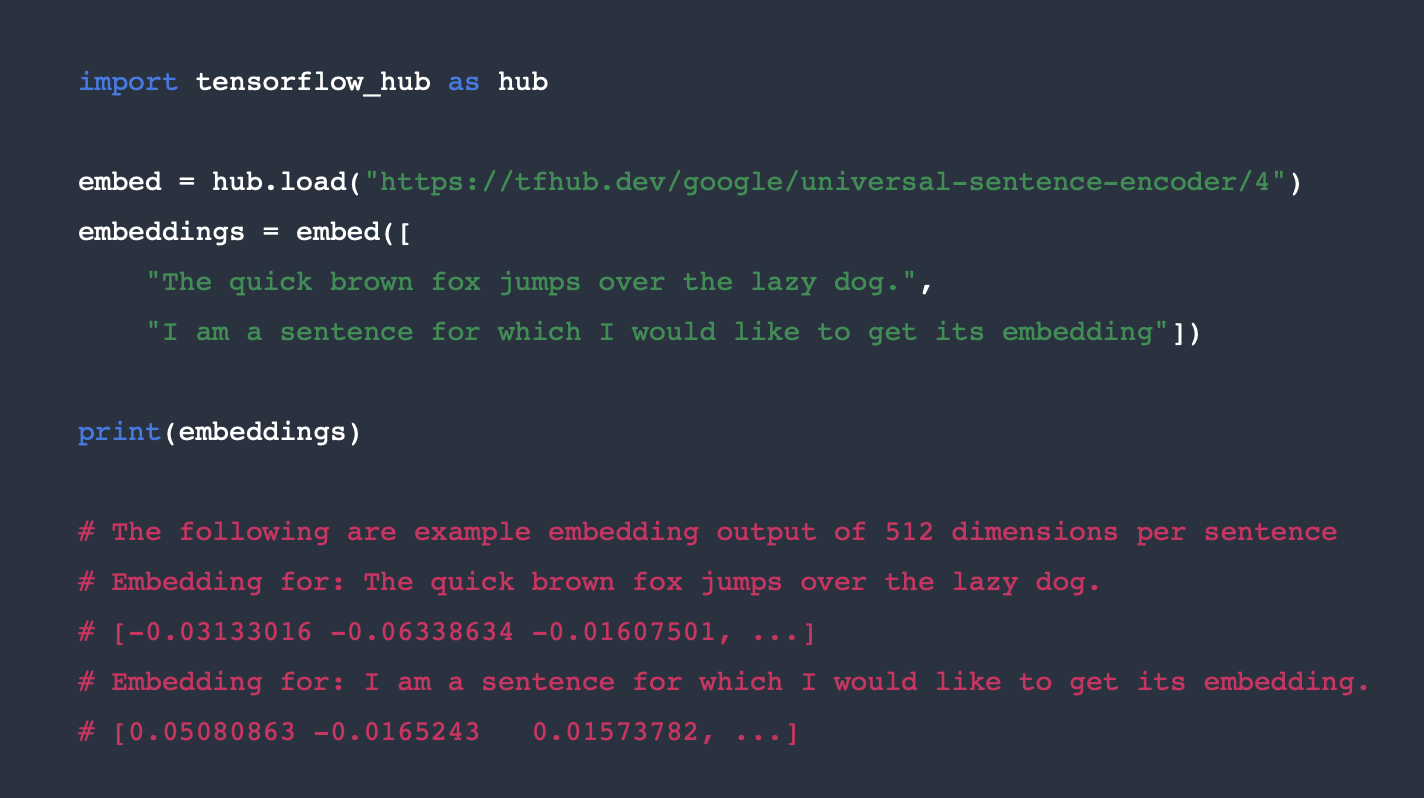
Our method of calculating sentence similarity comes from paper: Universal Sentence Encoder



There are two models for encoding sentences into embedding vectors. One makes use

of the transformer[7] (Vaswani et al., 2017) architecture, while the other is formulated as a deep averaging network (DAN) (Iyyer et al., 2015).

The models take as input English strings and produce as output a fifixed dimensional embedding representation of the string.



Python example code for using the universal sentence encoder.

Then how do we deal with the lyrics data. The song dataset is from AZLyrics song lyrics of kaggle, More than 147k song lyrics from AZLyrics.com.

Processing data, for the lyrics dataset, its format is csv and contains the following fields:

ARTIST\_NAME,ARTIST\_URL,SONG\_NAME,SONG\_URL,LYRICS

Represents artist name, artist url, song name, song url, lyrics.

We mainly use three fields: artist name, song name and lyrics. Lyrics are the most important part. We need to preprocess the data before using it.

First of all, we have to read the data correctly. We discarded the data with irregular format, such as the lyrics are too short, the song name or the artist name is missing.

Secondly, the songs in this dataset are in multiple languages. We only use English songs this time, so we also have a language analysis of all songs. Fortunately, python provides us with third-party packages for detection of many languages, here we use the detect function in the langdatect package to determine which language it is and filter out the en

The third point is that we need to segment the lyrics. We have adopted two strategies. First, we can segment the lyrics according to the period in the lyrics. However, we found that there are many songs in which the period rarely leads to the unsatisfactory sentence. If the sentence is less than 10 sentences after the lyrics are subdivided, we need to re-phrase it. In the second sentence, we use commas to subdivide the sentences. However, there is a problem with the use of comma clauses. The comma clause causes the sentence to be too short due to the lyrics. The lyrics are all based on short dramas. The comma segmentation can identify all the short sentences but will cause the sentences to be too short. So here we introduce a compensation mechanism. For too short sentences, we merge them.

Once we have obtained the segmented lyrics, we can get the embedding vectors by putting the text entered by the user and lyrics into the GSE embed.

We use BERT for positive and negative classification, then combine the similarity calculation model (GSE) to get the final result.

**Results**



Describe how you evaluated your work and report the results such as classification accuracy, or

results on other evaluation measures suitable for your project. Also report any other relevant findings.

**Discussion**

Present a short critical discussion of your work. Looking at the results, what kind of errors are made

by your system? Can you explain them? What are the limitations of the method you used? Are there

problems with your dataset? How could you have done things differently (if you had more time, better

resources, etc.)?

Reference：

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

**Convolutional Neural Networks for Sentence Classifification**

**Yoon Kim**

New York University

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

**[2]From Word Embeddings To Document Distances**

**[4]Supervised Learning of Universal Sentence Representations from Natural Language Inference Data**

**[5]Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference.**

**In Proceedings of EMNLP.**

**[6]Universal Sentence Encoder**

[7]Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ﾅ ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NIPS*.

[8]Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daume III. 2015. Deep unordered composition rivals syntactic methods for text classifification. In *Proceedings of ACL/IJCNLP*.