PROJECT REPORT

Topic: Sentiment Analysis

On SST dataset

By: Majji Bhavana

Abstract

Sentiment classification is an important process in understanding people's perception towards a product, service, or topic. Many natural language processing models have been proposed to solve the sentiment classification problem. In this project, I have used a deep learning model called BERT (Bidirectional Encoder Representations from Transformers) to solve the fine-grained sentiment classification task on the Stanford Sentiment Treebank (SST-5) dataset and also binary sentiment classification on SST-2 (binary) dataset.

Introduction

Sentiment classification is a form of text classification in which a piece of text has to be classified into one of the predefined sentiment classes. It is a supervised machine learning problem. In binary sentiment classification, the possible classes are positive and negative. In fine-grained sentiment classification, there are five classes (very negative, negative, neutral, positive, and very positive).

Firstly, we need to convert a text sequence of words represented into a fixed sized vector that encodes the meaningful information of the text using any of the available deep learning NLP models. In this project, we use the pre-trained BERT model and fine tune it for the fine-grained sentiment classification task on the Stanford Sentiment Treebank (SST) dataset.

Dataset

Stanford Sentiment Treebank (SST) is one of the most popular publicly available datasets for fine-grained sentiment classification task. I have taken this SST dataset from pytree bank python package which has movie reviews given by various users. It has a training set of 8544 sentences and a testing set of 2210 sentences with corresponding sentiment value. There are five sentiment labels in SST: 0 (very negative), 1 (negative), 2 (neutral), 3 (positive), and 4 (very positive). If we only consider positivity and negativity, we get the binary SST-2 dataset. If we consider all five labels, we get SST-5.

Methodology

The most important task in sentiment analysis is converting a text into a fixed size vector known as embeddings. There are various methods for getting embeddings.

1. Bert

In this project, I have used a pre-trained bert model which basically gives a sentence vector. To get the sentence embedding, I have used the sentence transformer package. In bert, the transformer encoder directly reads the entire sequence of words at once. BERT uses mainly two training strategies:

a) Masked LM: It basically replaces 15% of tokens with [MASK] token and model then predicts original value of the masked words with the help of non-masked words based upon the context.

b) Next Sentence Prediction: It basically gets the pair of sentences and then tries predicting whether they are subsequent to each other or not. 50% of the training data is joined with their succeeding sentence and rest 50% with random sentences from the corpus and then the model tries to predict the next statement.

There are two variants in BERT models one is BERTlarge and BERTbase.

	BERTbase	**BERTlarge**
No. of layers	12	24
No. of hidden units	768	1024
No. of self-attentive leads	12	16
Total trainable parameters	110M	340M

Bert requires first token of every sequence to be [CLS] (classification token) and a [SEP] (separation token) after every sentence. The output embedding corresponding to the [CLS] token is the sequence embedding that can be used for classifying the whole sequence.
Before the text reviews are fed to the model, the text is preprocessed by removing all the digits, punctuation symbols and accent marks, and converting to lowercase. Next, we tokenize the text using the Word-Piece tokenizer and add [CLS] and [SEP] tokens at the appropriate positions.
After we computed the sequence embedding from BERT, we apply dropout with a probability factor of 0.1 to regularize and prevent overfitting. Finally, the softmax classification layer will output the probabilities of the input text belonging to each of the class labels such that the sum of the probabilities is 1. The output node with the highest probability is then chosen as the predicted label for the input.

Glove, word2vec etc. gives word embeddings instead of sentence embeddings.

2. GloVe

I have used LSTM (and also Bi-LSTM), taken word embeddings and learned the relations among the word sequences, which is an RNN(helps in classifying, processing and making predictions using the time series data). I have used a pre-trained glove.42B.300D (Common crawl, 42B tokens, 1.9M vocab, 300d vectors) model which has word embeddings of various words from the dataset provided by common crawl. After pre-processing, the tokens are sent to the embedding layer and LSTM will combine these word embeddings and interpret the relation among them. Embedding layer can be used in two ways:
a) Supplying the embedding matrix which consists of word embeddings (taken from GloVe) of the words in the vocabulary. These are used for getting the embeddings of input sequences.
b) In this case, it generates its own word embeddings of input sequences. Next is a dense layer and then the softmax layer which predicts the class label. I have used a dropout of 0.2 for avoiding overfitting of the model.

Other Models

I used ‘Textblob’ and ‘Vader lexicon’ pre-trained models which have sentiments associated with particular sentences.
I have also used Pytorch for generating sentence embedding. It basically considers the number of layers, number of hidden units (of BERT model) and sentence id. According to the results given by the experts the more accurate embedding is

produced by concatenating the output of the last 4 layers (among the 12 layers of BERTbase model).

I have also checked the accuracy for few other machine learning classifiers like logistic regression, k-nearest neighbors, gradient boost etc.

Experiments and Results

Method	**Accuracy**	
	SST-2	**SST-5**
LSTM(with glove embeddings)	76.96	34.58
BI-LSTM(with glove embeddings)	77.51	39.23
LSTM(with random embeddings)	75.06	39.95
BI-LSTM(with random embeddings)	73.44	39.77
TEXT BLOB	20.45	28.37
VADER LEXICON	20.95	31.54
BERT-NN	84.16	**49.32**
BERT-LOGISTIC REGRESSION	**85.11**	47.47
BERT-GB	84.03	48.51
BERT-XGB	84.30	48.37

BERT: Bi-Directional Encoder Representations from Transformers.
NN: Neural Network.
GB: Gradient Boost.
xGB: Extreme Gradient Boosting.
LSTM: Long Short Time Memory.
Bi-LSTM: Bi Directional Long Short Time Memory.