Homework 2 Report

109611066 吳典謀

Part 0

The text go through the following process:

- `remove_stopwords`, the default function provided by TA.
- `remove_tags`, my own function using `re.compile('<.*?>')` and `re.sub()` to remove the html tags.
- `WordNetLemmatizer()` to lemmatize the text.
- Element function `lower()` of `str` to convert the text to lowercase.
- `re.sub(r'[^a-zA-Z]', ' ', text)` to remove all the non-alphabetic characters.
- `tokenizer.tokenize()` and `strip()` to remove the leading and trailing whitespaces of each word.

An example of the text after the process:

Original text:

```
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."
```

Processed text:

```
'one reviewer mentioned watching oz episode hooked right exactly happened me the first
thing struck oz brutality unflinching scene violence set right word go trust show faint
hearted timid show pull punch regard drug sex violence hardcore classic use word it
called oz nickname given oswald maximum security state penitentary focus mainly emerald
city experimental section prison cell glass front face inwards privacy high agenda em
city home many aryans muslims gangsta latinos christians italians irish scuffle death
stare dodgy dealing shady agreement never far away i would say main appeal show due fact
go show dare forget pretty picture painted mainstream audience forget charm forget
romance oz mess around first episode ever saw struck nasty surreal say ready watched
developed taste oz got accustomed high level graphic violence violence injustice crooked
guard sold nickel inmate kill order get away well mannered middle class inmate turned
prison bitch due lack street skill prison experience watching oz may become comfortable
uncomfortable viewing thats get touch darker side'
```

Problem 1: list `stopwords` not found

The `remove_stopwords` function use `ToktokTokenizer` to tokenize the text and then remove the stopwords according to the `stopwords` list provided by `nltk.corpus`.

When testing the `remove_stopwords`, I stumbled upon an error about `Resource wordnet not found.`

After some research, I found this stackoverflow post: Resource corpora/wordnet not found on Heroku and followed its instructions for fixing it. Later I found that google colab somehow fixed the problem, but the solution is still necessary if you want to run the code locally.

Note 1: Why I removed non-alphabetic characters

After the lemmatization process, I found that there are some apostrophes left in the text. For example, the word `I'm` is lemmatized to `'`. I afraid that there might be more problems like this, so I decided to remove all the non-alphabetic characters.

Part 1

Question 1: Concept of perplexity

Perplexity is a measure of how well a model predicts. When the perplexity is low, the model is good at predicting the next word.

If the training data contains noise or is poorly annotated, the model will have a high perplexity.

Result 1: Outputs of the model

1. without preprocess:

2. with `remove_stopwords`:

```
littled3092@littled3092-Latitude-7390:~/Documents/My-Vault/Artificial Intelligence/HW2$ python3 main.py --model_type ngram --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/littled3092/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Reading data...
100%|| 40000/40000 [00:08<00:00, 4605.21it/s]
Computing the co-occurrence of each pair...
100%|| 40000/40000 [00:11<00:00, 3587.91it/s]
Computing perplexity...
100%|| 10000/10000 [01:35<00:00, 104.61it/s]
Perplexity of ngram: 1.1022500494398426
Converting training data to embedding...
100%|| 40000/40000 [01:15<00:00, 530.16it/s]
Converting testing data to embedding...
100%|| 10000/10000 [00:18<00:00, 534.51it/s]
F1 score: 0.6745, Precision: 0.6902, Recall: 0.6792
```

3. with custom preprocess:

```
littled3092@littled3092-Latitude-7390:~/Documents/My-Vault/Artificial Intelligence/HW2$ python3 main.py --model_type ngram --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/littled3092/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Preprocessing data...
Reading data...
100%|| 40000/40000 [00:06<00:00, 6549.32it/s]
Computing the co-occurrence of each pair...
100%|| 40000/40000 [00:08<00:00, 4602.77it/s]
Computing perplexity...
100%|| 10000/10000 [00:22<00:00, 441.60it/s]
Perplexity of ngram: 1.0501755841534741
Converting training data to embedding...
100%|| 40000/40000 [01:01<00:00, 650.35it/s]
Converting testing data to embedding...
100%|| 10000/10000 [00:14<00:00, 672.94it/s]
F1 score: 0.7034, Precision: 0.7135, Recall: 0.706
```

In the second case, the perplexity is slightly higher, and the model is not as good as the first case. I think this is because the `remove_stopwords` function remove words and cause the ngram model unable to predict the next word.

In the third case, the perplexity and accuracy are slightly worse than the first case, but better than the second case. Compared to first case, I think the reason it is worse is the same as the second case. Compared to the second case, the reason it is better is that the custom preprocess function remove some noise in the text.

Part 2

Question 1: Two pre-training steps in BERT

The two pre-training steps in BERT are:

1. Masked Language Model (MLM): Randomly mask some words in the input and predict the masked words.
2. Next Sentence Prediction (NSP): Predicting whether the second sentence is the next sentence of the first sentence.

Question 2: Four different BERT application scenarios

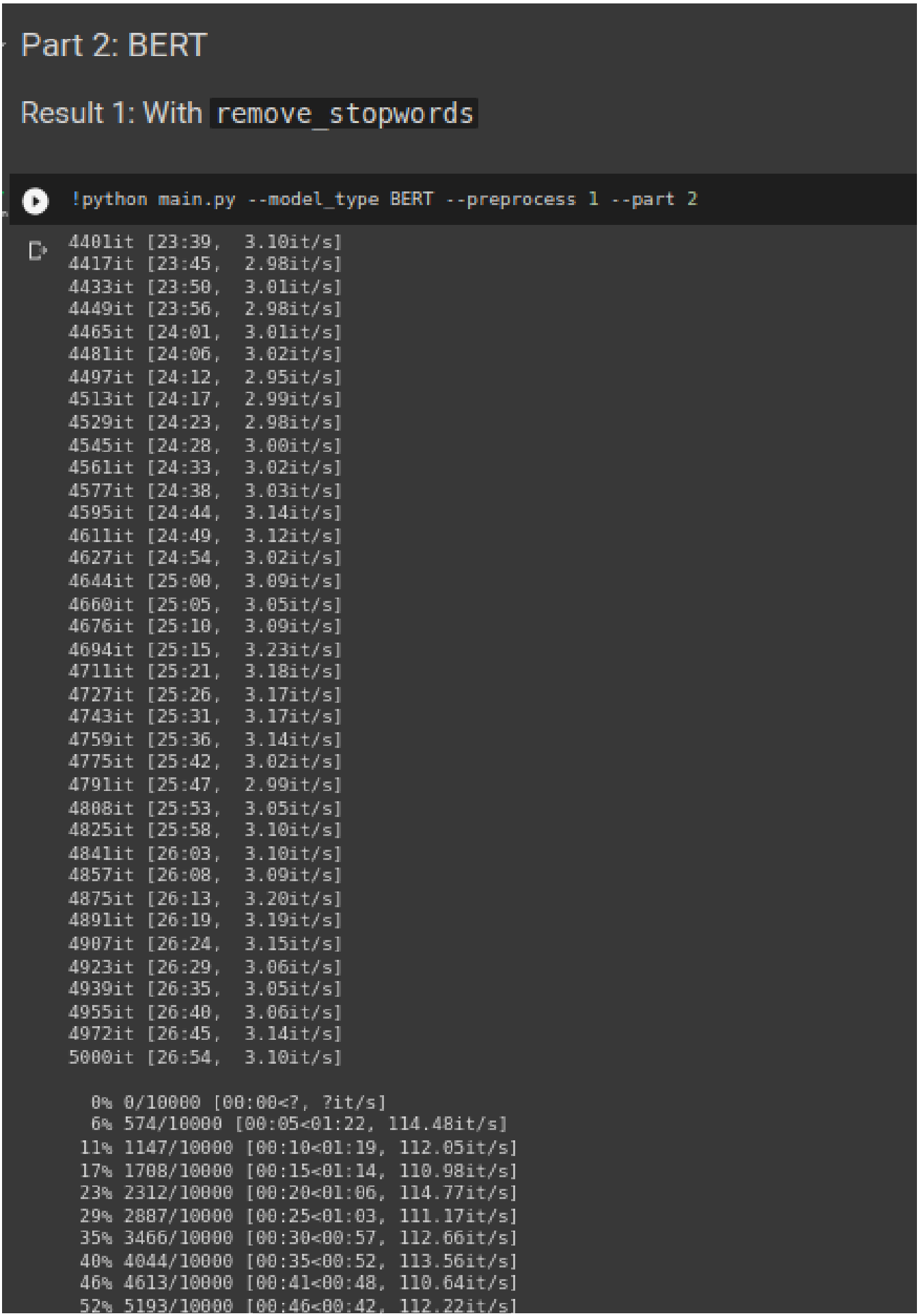
Some of the BERT application scenarios are:

1. Text classification: Classify the text into different categories.
2. Question answering: Answer the question based on the context.
3. Named entity recognition: Identify and classify names in the text, such as person names, organization names, location names, etc.
4. Language generation: Generate text based on the input, such as text completion, text summarization, etc.

Question 3: Difference between BERT and distilBERT

DistilBERT is a model that is smaller and faster than BERT. It is a variant of the original BERT model, with fewer layers and parameters.

Result 1: F1 score



```
 58% 5781/10000 [00:51<00:37, 113.81it/s]
 64% 6351/10000 [00:56<00:33, 109.06it/s]
 69% 6899/10000 [01:02<00:29, 105.63it/s]
 75% 7489/10000 [01:07<00:23, 109.15it/s]
 81% 8072/10000 [01:12<00:17, 111.27it/s]
 86% 8631/10000 [01:17<00:12, 109.99it/s]
 92% 9226/10000 [01:22<00:06, 112.60it/s]
100% 10000/10000 [01:30<00:00, 110.96it/s]
Epoch: 0, F1 score: 0.9229, Precision: 0.9231, Recall: 0.9229, Loss: 0.2662
100% 1/1 [28:24<00:00, 1704.44s/it]
```

Question 4: Relation of the Transformer and BERT and the core of the Transformer

Transformer is a neural network architecture. The core of the Transformer is the attention mechanism, which allows the model to compute the representation of each token by considering the other tokens.

BERT is a pretrained model based on the Transformer.

Part 3

Question 1: Difference between vanilla RNN and LSTM

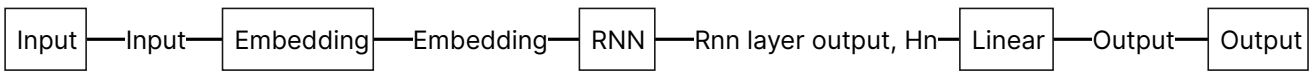
RNN uses hidden state to store the information of the previous time step, while LSTM uses a more sophisticated gating mechanism to select the information to be stored in cell state.

Question 2: Meaning of each dimension of the input and output for each layer in the model

The dimensions of the input and output for each layer in the model are:

- Input: `(seq_len, batch_size)`
- Embedding: `(seq_len, batch_size, embedding_dim)`
- Rnn layer output: `(seq_len, batch_size, hidden_size*2)
- Hn (hidden layer): `((bidirectional ? 2 : 1) * num_layers, batch_size, hidden_size)`
- Output: `(batch_size, num_classes)`

The graph below shows the process of the model. Each layer is represented by a box, and the arrows represent the flow of data.



The parameters used in the dimension:

- `seq_len`: The length of the sequence. May vary depend on different sentence length.
- `batch_size`: The number of comments in a batch.
- `embedding_dim`: The dimension of the embedding vector.
- `hidden_size`: The dimension of the hidden state.
- `num_layers`: The number of layers in the RNN. If `bidirectional` is `True`, the number of layers in the RNN is `num_layers * 2`.
- `num_classes`: The number of classes. In our case, we have 2 classes: positive and negative.

Result 1: F1 score

Part 3: RNN

Result 1: With `remove_stopwords`

```
!python main.py --model_type RNN --preprocess 1 --part 2
```

```
4491it [02:41, 27.61it/s]
4630it [02:46, 27.60it/s]
4769it [02:51, 27.61it/s]
5000it [02:59, 27.79it/s]

  0% 0/10000 [00:00<?, ?it/s]
 17% 1668/10000 [00:05<00:24, 333.39it/s]
 36% 3563/10000 [00:10<00:17, 360.17it/s]
 54% 5417/10000 [00:15<00:12, 364.97it/s]
 72% 7242/10000 [00:20<00:07, 355.50it/s]
100% 10000/10000 [00:27<00:00, 363.21it/s]
Epoch: 8, F1 score: 0.8956, Precision: 0.8956, Recall: 0.8956, Loss: 0.0061
 90% 9/10 [31:09<03:27, 207.67s/it]
0it [00:00, ?it/s]
140it [00:05, 27.80it/s]
281it [00:10, 27.96it/s]
421it [00:15, 27.87it/s]
561it [00:20, 27.54it/s]
699it [00:25, 27.53it/s]
838it [00:30, 27.61it/s]
978it [00:35, 27.70it/s]
1117it [00:40, 27.67it/s]
1259it [00:45, 27.85it/s]
1399it [00:50, 27.86it/s]
1539it [00:55, 27.86it/s]
1680it [01:00, 27.93it/s]
1820it [01:05, 27.85it/s]
1960it [01:10, 27.78it/s]
2099it [01:15, 27.77it/s]
2238it [01:20, 27.67it/s]
2381it [01:25, 27.88it/s]
2522it [01:30, 27.95it/s]
2662it [01:35, 27.76it/s]
2804it [01:40, 27.89it/s]
2945it [01:45, 27.94it/s]
3085it [01:50, 27.90it/s]
3226it [01:55, 27.95it/s]
3366it [02:01, 27.64it/s]
3506it [02:06, 27.73it/s]
3645it [02:11, 27.73it/s]
3784it [02:16, 27.67it/s]
3924it [02:21, 27.72it/s]
4065it [02:26, 27.84it/s]
4205it [02:31, 27.85it/s]
4345it [02:36, 27.84it/s]
4487it [02:41, 27.95it/s]
4627it [02:46, 27.91it/s]
4769it [02:51, 28.02it/s]
5000it [02:59, 27.81it/s]

  0% 0/10000 [00:00<?, ?it/s]
 17% 1728/10000 [00:05<00:23, 345.50it/s]
 36% 3633/10000 [00:10<00:17, 366.35it/s]
 55% 5465/10000 [00:15<00:12, 354.46it/s]
```

```
 74% 7358/10000 [00:20<00:07, 363.74it/s]
100% 10000/10000 [00:27<00:00, 363.64it/s]
Epoch: 9, F1 score: 0.9004, Precision: 0.9004, Recall: 0.9004, Loss: 0.0044
100% 10/10 [34:36<00:00, 207.68s/it]
```

Discussion

Question 1: Innovation of NLP field

The evolution from ngram to LSTM and BERT reflects a trend toward more complex models.

Ngram is a simple model that only considers adjacent text. Note that this method is not suitable for long sentences because previous words are not considered.

LSTM is a more complex model that uses cell state and hidden state to store information. It is suitable for long sentences because it can consider the information of previous words.

BERT is a recent model that is more complex than LSTM. It is a pretrained model based on the Transformer. It uses self-attention mechanism to consider different parts of the text. Because it is a pretrained model, it can be fine-tuned on a relatively small dataset and achieve good results.

The evolution of these models means that we use NLP to analyze more complex text, also we wish to obtain better results without having to collect a large amount of data.

Question 2: Problems I meet

One of the major problems I met is the syntax of the modules used. For example, the `torch` module has a lot of functions and it is hard to find the right one and implement it correctly.

When facing this problem, I usually search the internet for implementations that solve similar problems. I find it hard to figure out the dimensions of the LSTM model, so I searched the internet and come across this video: L15.7 An RNN Sentiment Classifier in PyTorch. Starting from 21 minutes, the author shows the implementation of the model and explains the dimensions of each layer. This video helps me a lot.

Other than that, I also met some `nltk` problems when running code locally. This problem is mentioned in Part 0. I solved this problem by searching similar problems on the internet and I find the solution in a stackoverflow post.