Part 0: Implement a different preprocessing method (10%)

■ Briefly explain the method you implemented and give an example (such as the E.g in the remove_stopwords function) in the report.

Ans: In remove_stopwords(), we transpose all characters in the text to lower cases and then eliminate terms which belong to stopwords provided by nltk. In preprocessing_function(), I additionally call BeautifulSoup() to remove HTML labels, re.sub() to remove non-character symbols and lemmatizer.lemmatize() to do lemmatization, a method that is more effective than stemming.

e.g.,
text: “

Headlines warn us of the current campaign to demonize drug users,”

output from remove_stopwords(): “<br / ><br / >Headlines warn us current campaign demonize drug users ,”

output from preprocessing_function(): “Headlines warn us current campaign demonize drug user”

Part 1: Implement the bi-gram language model (20%)

■ Briefly explain the concept of perplexity in report and discuss how it will be influenced.

Ans: In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It is a mathematical concept that measures the uncertainty or the confusion associated with predicting the next symbol in a sequence based on the previous symbols. In general, lower perplexity means that the model can predict the next symbol with higher accuracy.

■ Screenshot the outputs and tell your observations about the differences in the perplexity caused by the preprocessing methods(1. without

preprocess 2. with remove stopwords 3. with your method).

Ans: We can see that perplexity: without preprocess < with remove stopwords < with my method. I at first thought that with applying more preprocessing methods, the perplexity will get lower, but the result is beyond my expectation. A straightforward reason comes to my mind that when applying more preprocessing methods, the sentence will become more fragmented, and thus leads to confusion. It's worth talking about that in this case, higher perplexity does not absolutely imply lower precision, by the result of 2. and 3.

1. without preprocess:

2. with remove stopwords:

3. with my method:

Part 2: Implement BERT model (15%)

■ Briefly explain the two pre-training steps in BERT.

Ans: The two pre-training steps in BERT are Masking Input and Next Sentence Prediction. During the process of Masking Input, some of the input tokens are randomly chosen to be masked, and the model is trained to predict the original tokens from the masked ones. In the step of Next Sentence Prediction, our model is trained to predict whether a sentence follows another sentence. Those two pre-training steps allow the model to learn the relationships between words and sentences, respectively.

■ Briefly explain four different BERT application scenarios.

Ans: The four different BERT application scenarios are:

1. Input: sequence, output: class. E.g. Sentiment analysis
2. Input: sequence, output: same as input. E.g. POS tagging
3. Input: two sequences, output: a class. E.g. Natural Language Inference
4. Extraction-based Question Answering(QA)

■ Discuss the difference between BERT and distilBERT?

Ans: The main difference between BERT and DistilBERT is their model size. BERT has a large number of parameters. DistilBERT, on the other hand, is a smaller and more efficient version of BERT that has fewer parameters. Report shows that it is possible to reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster. (DistilBERT, a distilled version of BERT: smaller)

■ Screenshot the required test F1-score.

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Some weights of the model checkpoint at distilbert-base-uncased were not used w
- This IS expected if you are initializing DistilBertModel from the checkpoint
- This IS NOT expected if you are initializing DistilBertModel from the checkpo
100% 10000/10000 [30:32<00:00,  5.46it/s]
100% 10000/10000 [01:57<00:00, 85.05it/s]
Epoch: 0, F1 score: 0.9316, Precision: 0.9316, Recall: 0.9316, Loss: 0.2302
```

```
!python main.py --model_type BERT --preprocess 0 --part 2
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Some weights of the model checkpoint at distilbert-base-uncased were not used when in
- This IS expected if you are initializing DistilBertModel from the checkpoint of a m
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of
100% 10000/10000 [30:29<00:00,  5.47it/s]
100% 10000/10000 [01:56<00:00, 85.88it/s]
Epoch: 0, F1 score: 0.931, Precision: 0.931, Recall: 0.931, Loss: 0.2305
100% 10000/10000 [30:33<00:00,  5.45it/s]
100% 10000/10000 [01:57<00:00, 85.11it/s]
Epoch: 1, F1 score: 0.9354, Precision: 0.9355, Recall: 0.9354, Loss: 0.1215
```

■ (BONUS 5%) Explain the relation of the Transformer and BERT and the core of the Transformer.

Ans: The core of the Transformer architecture is the self-attention mechanism. It allows the model to look at all the tokens in the input sequence simultaneously and weigh the importance of each token when generating the output. This differs from traditional recurrent neural networks (RNNs), which process the input sequence one token at a time. The self-attention mechanism makes the Transformer more parallelizable and allows it to better capture long-range dependencies in the input sequence.

BERT is a specific implementation of the Transformer architecture. To be specific, in the original Transformer architecture, there are both encoder and decoder layers, and BERT mainly uses the Transformer's encoder layers, as BERT is a language model that generates contextualized representations of words or phrases in a sentence, rather than a model for machine translation or sequence generation.

Part 3: Implement LSTM model (15%)

■ Briefly explain the difference between vanilla RNN and LSTM.

Ans: Vanilla RNN and LSTM are both types of recurrent neural networks used for sequence modeling. Vanilla RNN suffers from the gradient vanishing problem, or in contrast, gradient exploding problem, which means that the gradients used for backpropagation tend to become very small or very large, respectively. LSTM is aimed at solving the problems mentioned earlier by having an additional component, gate, which controls the flow of information through the network, allowing it to selectively forget or remember previous inputs.

■ Please explain the meaning of each dimension of the input and output for each layer in the model. For example, the first dimension of input for LSTM is batch size.

Ans: For each layer in my model, I respectively call nn.Embedding(), nn.LSTM() and nn.Sequential() which includes nn.Dropout(), nn.Linear()

and nn.Sigmoid(). Next I'll list the meaning of each dimension of the input and output of them.

nn.Embedding():
Input: (batch_size, sequence_length)
Output: (batch_size, sequence_length, embedding_dim)

nn.LSTM():
Input: (sequence_length, batch_size, input_size)
Output: (sequence_length, batch_size, hidden_size)

nn.Dropout():
Input: Input can be of any shape.
Output: Output is of the same shape as input.

nn.Linear():
Input: (batch_size, input_features)
Output: (batch_size, output_features)

nn.Sigmoid():
Input: Input can be of any shape.
Output: Output is of the same shape as input.

■ Screenshot the required test F1-score.

```
!python  main.py  --model_type  RNN  --preprocess  0  --part  2
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
  0% 0/15 [00:00<?, ?it/s]Epoch: 0, F1 score: 0.5931, Precision: 0.5931, Recall: 0.5931, Loss: 0.6927
  7% 1/15 [02:33<35:53, 153.86s/it]Epoch: 1, F1 score: 0.7474, Precision: 0.7713, Recall: 0.7519, Loss: 0.6215
 13% 2/15 [05:05<33:03, 152.57s/it]Epoch: 2, F1 score: 0.7746, Precision: 0.7965, Recall: 0.7781, Loss: 0.5106
 20% 3/15 [07:37<30:28, 152.37s/it]Epoch: 3, F1 score: 0.7915, Precision: 0.8088, Recall: 0.794, Loss: 0.4734
 27% 4/15 [10:09<27:54, 152.24s/it]Epoch: 4, F1 score: 0.8244, Precision: 0.8311, Recall: 0.8252, Loss: 0.4492
 33% 5/15 [12:40<25:18, 151.82s/it]Epoch: 5, F1 score: 0.8275, Precision: 0.8292, Recall: 0.8277, Loss: 0.434
 40% 6/15 [15:12<22:47, 151.93s/it]Epoch: 6, F1 score: 0.8337, Precision: 0.8343, Recall: 0.8338, Loss: 0.4233
 47% 7/15 [17:44<20:14, 151.81s/it]Epoch: 7, F1 score: 0.8408, Precision: 0.8469, Recall: 0.8414, Loss: 0.4138
 53% 8/15 [20:17<17:45, 152.17s/it]Epoch: 8, F1 score: 0.8188, Precision: 0.8285, Recall: 0.82, Loss: 0.4154
 60% 9/15 [22:50<15:14, 152.43s/it]Epoch: 9, F1 score: 0.8313, Precision: 0.834, Recall: 0.8316, Loss: 0.4079
 67% 10/15 [25:21<12:40, 152.13s/it]Epoch: 10, F1 score: 0.8472, Precision: 0.8485, Recall: 0.8473, Loss: 0.3965
 73% 11/15 [27:53<10:08, 152.11s/it]Epoch: 11, F1 score: 0.8405, Precision: 0.8406, Recall: 0.8405, Loss: 0.3967
 80% 12/15 [30:25<07:36, 152.02s/it]Epoch: 12, F1 score: 0.8471, Precision: 0.8478, Recall: 0.8472, Loss: 0.3887
 87% 13/15 [32:59<05:04, 152.42s/it]Epoch: 13, F1 score: 0.8464, Precision: 0.8475, Recall: 0.8465, Loss: 0.3864
 93% 14/15 [35:35<02:33, 153.51s/it]Epoch: 14, F1 score: 0.8547, Precision: 0.8547, Recall: 0.8547, Loss: 0.3851
100% 15/15 [38:09<00:00, 152.66s/it]
```

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

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch: 0, F1 score: 0.7748, Precision: 0.776, Recall: 0.775, Loss: 0.6705
Epoch: 1, F1 score: 0.8141, Precision: 0.8312, Recall: 0.8162, Loss: 0.5017
Epoch: 2, F1 score: 0.8547, Precision: 0.855, Recall: 0.8547, Loss: 0.4403
Epoch: 3, F1 score: 0.8621, Precision: 0.8623, Recall: 0.8621, Loss: 0.416
Epoch: 4, F1 score: 0.8665, Precision: 0.8667, Recall: 0.8665, Loss: 0.3996
Epoch: 5, F1 score: 0.8573, Precision: 0.8603, Recall: 0.8576, Loss: 0.388
Epoch: 6, F1 score: 0.864, Precision: 0.8675, Recall: 0.8643, Loss: 0.379
Epoch: 7, F1 score: 0.8701, Precision: 0.8701, Recall: 0.8701, Loss: 0.374
Epoch: 8, F1 score: 0.8706, Precision: 0.8706, Recall: 0.8706, Loss: 0.3708
Epoch: 9, F1 score: 0.8685, Precision: 0.8694, Recall: 0.8686, Loss: 0.3649
```

Discussion:

■ Discuss the innovation of the NLP field and your thoughts of why the technique is evolving from ngram -> LSTM -> BERT.

Ans: The innovation of the NLP field in recent years includes Pre-trained Language Models, Transformer Architecture, Attention Mechanisms, and so on. In my opinion, ngram is the most straightforward technique since it's just based on statistical analysis. That's why it was first applied to the NLP field. In recent years, deep learning techniques have become popular, and computational resources are easier to attach. Hence more sophisticated models such as LSTM, a type of recurrent neural network, and BERT, which has more than 100M trainable parameters, can be applied to solve NLP tasks.

■ Describe problems you meet and how you solve them.

Ans: In this assignment, the most difficult problem is that I didn't know how to start building a model. To be more specific, how could I write functions to evaluate the performance of my model? What's the meaning of the parameter in my model? I find it hard to get an answer if I simply read the spec and tutorial of the official document. Hence I surf on the internet to grab more information. A good reference is like the homework of professor Hung-yi Lee (2019 Spring ML HW6).