#### Part 0:

1. Briefly explain the method you implemented and give an example (such as the E.g in the remove\_stopwords function) in the report.

The method I implemented include includes a series of steps, including remove stopwords, remove punctuation, convert to lowercase, and then stemming.

- Remove stopwords: Stopwords (common words like 'and', 'the', 'is') are often removed from text because they carry little meaningful information about the content of the text.
- Remove Punctuation: Translating each punctuation mark into an empty string using Python's str.translate function.
- Convert to Lowercase: Converting all text to lowercase ensures uniformity and prevents the same words in different cases from being treated as different tokens.
  - Implementation: The string method lower() is applied to the text.
- Stemming: Stemming reduces words to their root form, aiding in generalizing similar forms of a word (e.g., 'running', 'runner' → 'run'). This helps in reducing the complexity of the text data.
  - Implementation:

NLTK's PorterStemmer is used to apply stemming to each token. Tokenizer: A tokenizer splits the text into tokens (words), and stemming is applied to each token.

#### • Example:

Original Text: "Here is a dog."

After Stopwords Removal: "Here dog."
After Punctuation Removal: "Here dog"
After Converting to Lowercase: "here dog"

After Stemming: "here dog"

#### • Part 1:

- 1. Briefly explain the concept of perplexity in report and discuss how it will be influenced.
- Perplexity measures how well a probability distribution predicts a sample. In language models, it quantifies how well the model predicts a sentence. Lower perplexity indicates a better model.
- Influence of Preprocessing: Preprocessing can affect perplexity. For example, removing stopwords might lower perplexity by eliminating common but less informative words, making the model's predictions more focused and accurate.

The perplexity should ideally decrease with effective preprocessing methods as the model becomes better at predicting the next word in the sequence.

# 2. Screenshot the outputs and tell your observations about the differences in the perplexity caused by the preprocessing methods

It's interesting to see that no preprocessing yielded the lowest perplexity in my implementation. Possible reasons for this observation are:

- Stopword Removal Impact:
   Removing stopwords might be removing some contextual cues that help in predicting the sequence of words in the reviews. Stopwords, although often considered "noise," can carry significant syntactic and semantic information crucial for understanding the flow and structure of sentences.
- Punctuation Removal Impact:
   Punctuation marks can also provide important information about sentence structure and boundaries, which can be beneficial for models to predict the next word or understand the context better.
- o Model and Data Fit:

The n-gram model relies heavily on the exact sequence of words to predict the next word. By removing stopwords and punctuation, the sequences that the model trained on might become less representative of the actual language use in the reviews, leading to worse predictions.



F1 score of n-gram with remove stopwords preprocessing:

```
[] !python main.py --model_type ngram --preprocess 1 --part 2

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Perplexity of ngram: 195. 43245350997685
F1 score: 0.6769, Precision: 0.6926, Recall: 0.6815
```

# Part 2:

#### 1. Briefly explain the two pre-training steps in BERT.

Bert is pre-trained on a large corpus using two tasks: masked language modeling and next sentence prediction.

Masked language modeling: Some of the words in a sentence are randomly replaced with a special token [MASK]. The model's objective is to predict the original word that has been masked based on the context provided by the other non-masked words in the sentence.

**Next sentence prediction**: The model is given pairs of sentences and must predict whether the second sentence is connected to the first one.

#### 2. Briefly explain four different BERT application scenarios.

#### Case 1 (Sentiment Analysis):

This case shows how BERT can be used for sentiment analysis. The input is a sequence of text, and the output is a classification (positive or negative sentiment). BERT processes the input sentence and uses a classifier (usually a linear layer) on top of its output embeddings to predict the sentiment.

#### Case 2 (POS Tagging):

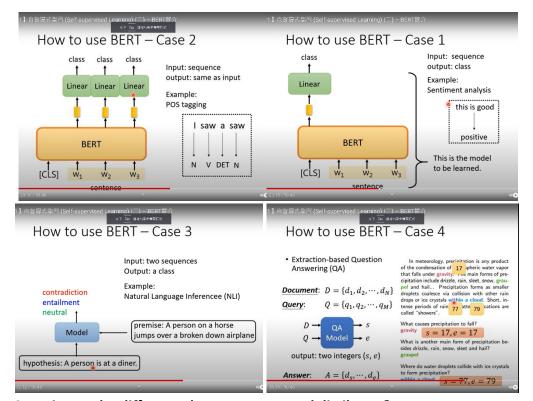
This case demonstrates how BERT can be employed for Part-of-Speech (POS) tagging. The input is a sequence of words, and the output mirrors the input sequence with each word tagged with its corresponding part of speech.

## **Case 3 (Natural Language Inference):**

In natural language inference, BERT helps determine the relationship between two text sequences (a premise and a hypothesis). The model predicts if the hypothesis is entailed by, contradicts, or is neutral to the premise. This type of task is central to understanding text entailment and contradiction in language models.

#### Case 4 (Question Answering):

BERT is also suitable for constricted question answering tasks (which means that the answer must be in the passage). Given a passage (document) and a question (query), BERT can be fine-tuned to predict the answer's start and end positions within the passage. This involves identifying the span of text that correctly answers the question.



#### 3. Discuss the difference between BERT and distilBERT?

BERT is the original model with more layers and parameters, while DistilBERT is a smaller, optimized version that retains most of BERT's performance but is faster and requires less computational resource.

#### 4. 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!

Epoch: 0, F1 score: 0.9275, Precision: 0.9275, Recall: 0.9275, Loss: 0.2657
```

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

BERT is built on the Transformer architecture, which uses self-attention mechanisms to process sequences of data. The core of the Transformer is its ability to handle long-range dependencies and parallel processing of sequences.

#### Part 3:

# 1. Briefly explain the difference between vanilla RNN and LSTM.

LSTM networks are an extension of vanilla RNNs designed to better capture longterm dependencies and solve the vanishing gradient problem inherent in traditional RNNs. LSTMs have more sophisticated structures like cell states and gates, while RNNs only have a simple structure with one tanh layer, which processes one input at a timestep along with the output from the previous timestep.

2. 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.

### Work flow of my code:

Embedding layer->LSTM Layer->Dropout Layer->Fully Connected Layer

#### (1) Embedding Layer:

The input dimension is [batch\_size, sequence\_length], where batch\_size is the number of sequences in a batch and sequence\_length is the length of each sequence.

The output dimension is [batch\_size, sequence\_length, embedding\_dim] after the embedding layer, with embedding\_dim being the size of each embedding vector.

#### (2) LSTM Layer:

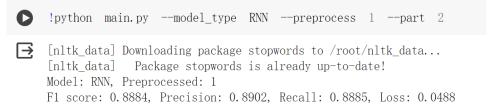
The input to the LSTM layer retains the dimensions from the embedding layer [batch size, sequence length, embedding dim].

The output from the LSTM has the dimensions [batch\_size, sequence\_length, hidden\_dim] for the output at all timesteps, and also outputs the hidden and cell states with dimensions [num\_layers, batch\_size, hidden\_dim] for both hidden state and cell state.

- (3) Dropout Layer: It does not change the dimension.
- (4) Fully Connected Layer:

The fully connected layer maps the LSTM's last hidden state output dimension from [batch\_size, hidden\_dim] to [batch\_size, output\_dim], where output\_dim is typically the number of classes for a classification problem.

3. Screenshot the required test F1-score.



#### • Discussion:

1. Discuss the innovation of the NLP field and your thoughts of why the technique is evolving from n-gram -> LSTM -> BERT.

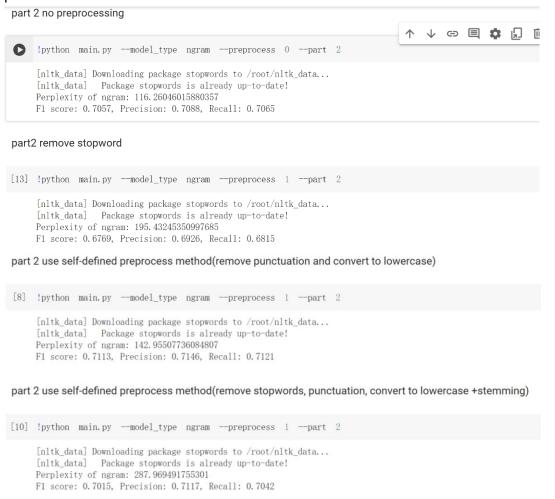
N-gram models are limited by fixed window sizes, LSTMs better capture sequence context, and BERT leverages vast amounts of data and deep learning to understand

language context deeply by using a pre-trained model. The transition from n-gram models to LSTM and then to BERT reflects the NLP field's progress towards models that capture longer-term dependencies and contextual meanings more effectively.

# 2. How preprocessing influences the prediction model:

Based on the results, the self-defined preprocessing model that removed stopwords, punctuation, and converted text to lowercase performed the best (F1: 0.7113), suggesting that these preprocessing steps helped the model to make better predictions.

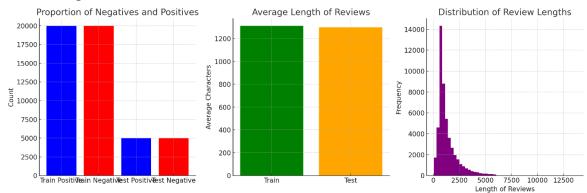
However, unexpectedly, removing only stopwords increased perplexity and decreased F1 score, precision, and recall compared to no preprocessing. This result may due to some uninspected errors in the code, or indicating that some stopwords might be contributing contextual information beneficial for the 2-gram model in making more accurate predictions.



#### 3. Data observations:

It is important to observe the distribution of the labels and the review lengths to make sure our data is not biased too much.

We can see that the proportion of negatives and positives is even among both train and test datasets, and their average length of reviews are close. Also, the distribution of review lengths shows a right-skewed distribution, indicating that most reviews are of shorter length.



#### 4. Performance of different model:

Model	Perplexity	F1 Score	Precision	Recall	Loss
N-gram	195.43	0.6769	0.6926	0.6815	-
BERT	-	0.9275	0.9275	0.9275	0.2657
RNN	-	0.8884	0.8902	0.8885	0.0188

- The BERT model is the best performing in terms of precision and recall, which is consistent with current knowledge that BERT models generally outperform simpler models like n-grams and RNNs in natural language processing tasks.
- The n-gram model's lower performance compared to deep learning models is expected due to its simplicity and the fact that it does not capture long-term dependencies or semantic meaning as effectively as RNN or BERT models.

#### 5. Describe problems you meet and how you solve them.

### (1) Model Performance:

I spent a lot of time trying to improve the model performance. If the model isn't performing well, it might be due to insufficient training, poor model architecture, overfitting to the training data, or not having an appropriate learning rate. Searching among these possibilities is time-consuming.

#### (2) Implementation Details:

When I was implementing the BERT pre-trained model, I didn't know that BERT is expecting certain input formats and was tuck in an error. Thankfully, after reading the documentation for pre-trained models and their expected input and output formats carefully, the problem was solved.

# • In-context learning (BONUS 5%):

# (1) Few-shot:

keview: "It was a quii and poring tiim. I wasted my time watching this." Sentiment: 0

Review: "The plot twist was brilliant and unexpected, truly a masterpiece." Sentiment: 1

Review: "Your turn! Analyze this review and predict the sentiment: "The acting was mediocre and the storyline was predictable."

#### ChatGPT

Based on the pattern observed in the provided movie reviews and their corresponding sentiment labels:

Review: "The acting was mediocre and the storyline was predictable." Sentiment: 0 (Negative sentiment)

The sentiment prediction for this review is negative (0) because it contains negative descriptors like "mediocre" and "predictable."

< 2/2 > (3) (1) (5) (7)







Sentiment: 0

Here's the reasoning behind the prediction:

- The review uses negative words like "mediocre" and "predictable" to describe the acting and storyline.
- These words imply the reviewer did not enjoy these aspects of the movie.
- Based on the previous examples (positive reviews use words like "loved", "fantastic", "brilliant"), negative reviews tend to focus on weaknesses of the film.

Therefore, we can predict the sentiment of this review to be negative (sentiment: 0).

### (2) Chain of Thought:

Chain of thought:

To determine the sentiment of a movie review, we must look for positive or negative language. Let's analyze some examples:

Review: "A total hit, I was on the edge of my seat the entire time!"

Thought Process: The words 'total hit' and 'edge of my seat' are very positive. This suggests a positive sentiment.

Sentiment: 1

Now, it's your turn to apply this process:

Review: "What a disappointment, the movie failed to deliver any excitement."

Thought Process:?

Sentiment:?

#### ChatGPT

Review: "What a disappointment, the movie failed to deliver any excitement."

Thought Process: The phrase 'What a disappointment' and 'failed to deliver any excitement' indicate a negative experience. Therefore, the sentiment expressed in this review is negative.

Sentiment: -1





- "What a disappointment": This phrase expresses clear negativity about the movie experience.
- "failed to deliver any excitement": This reinforces the negativity by highlighting the lack of a key element (excitement) that the reviewer was likely looking for.

Sentiment: -1 (Negative)

# (3) Discussion:

I tried two methods: Few-Shot and Chain of Thought on ChatGPT and Gemini. I observed that their responses were extremely similar, and both focused on the same keywords. For example, in Few-Shot, they both catch the keywords 'mediocre' and 'predictable'. We can say that for these simple tasks, as sophisticated as ChatGPT and Gemini, although very little information is provided in the prompt, they can still predict the sentiments correctly.