HW2 Report

Part 0

`remove_stopwords`

The provided method is aim to remove stopword which is a basic preprocessing step used in text mining and NLP. The purpose is to eliminate common words that appear frequently in a language but do not carry significant semantic importance for the analysis.

`preprocessing_function()`

My preprocessing is including four major steps:

1. **Convert text to lowercase**, which ensures that the same words are not counted as different tokens.
2. **Replace newline symbol with spaces**, which can maintain sentence continuity and readability
3. **Remove punctuation,** which lower the noise in text data processing.
4. **Stemming**, a process of reducing words to their word stem, base, or root form, which can simplify text data and improve the match between words during text processing.

Example

- text:

```
This is species already hatching into a beautiful model (Mathilda May). A smashing baby wi
th an urge to kiss and kill!<br /><br />The movie begins with a strong launch, and infecte
d by a bore-virus throughout the middle to end.<br /><br />
```

- remove_stopwords:

```
species already hatching beautiful model ( Mathilda May ) . smashing baby urge kiss kill !
<br / ><br / >The movie begins strong launch , infected bore-virus throughout middle end.<
br / ><br / >
```

- preprocessed_text:

```
speci alreadi hatch beauti model mathilda may smash babi urg kiss kill the movi begin stro
ng launch infect boreviru throughout middl end
```

Part 1

Concept of perplexity

Perplexity is a statistical measure used in NLP to evaluate how well a probability model predicts a sample. It essentially measures the uncertainty of a language model in predicting the next token in the sequence. For a language model that assigns a probability P to a sequence of N words ($w_1, w_2, \ldots, w_N$), the perplexity (PP) of the model is given by:

$$PP(W) = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_1,w_2,\ldots,w_{i-1})}$$

The preprocessing steps applied to the data before training or testing a language model can significantly influence its perplexity. Here's how different preprocessing methods might impact this metric:

1. **No Preprocessing**:

   Keeping the text as-is preserves all linguistic cues and context, which might help the model in making more informed predictions. Thus, this typically results in lower perplexity, indicating better predictive performance.

2. **Removal of Stopwords**:

   Stopwords are often removed to focus the model on more meaningful words. However, this can remove necessary context and grammatical structure, potentially increasing perplexity.

3. **Lowercase Conversion, Removing Newlines, and Punctuation**:

   These steps simplify the text and standardize its format, which can help in some cases by reducing the variety of token types the model needs to handle. However, important cues like sentence boundaries (denoted by punctuation) and proper nouns (indicated by capitalization) are lost, which might increase perplexity as the model struggles with incomplete information.

4. **Stemming**:

   Reducing words to their base or root form simplifies the vocabulary, potentially aiding the model in recognizing patterns between similar words. However, this also causes different words with distinct meanings to converge, possibly confusing the model and increasing perplexity due to a loss of semantic precision.

Testing Result

- Without any preprocess:
  - Preplexity: `116`
  - F1-score: `0.4948`
  - Precision: `0.5003`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type ngram --preprocess 0 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Perplexity of ngram: 116.26046015880357
F1 score: 0.4948, Precision: 0.5003, Recall: 0.5003
```

- With remove stopwords:
  - Preplexity: `195`
  - F1-score: `0.4266`
  - Precision: `0.5092`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type ngram --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Perplexity of ngram: 195.43245350997685
F1 score: 0.4266, Precision: 0.5092, Recall: 0.5042
```

- With my method:
  - Preplexity: `287`
  - F1-score: `0.4467`
  - Precision: `0.499`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type ngram --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Perplexity of ngram: 287.969491755301
F1 score: 0.4467, Precision: 0.499, Recall: 0.4994
```

Observations and Analysis

Perplexity

- **Without preprocessing**, the perplexity is the lowest (116). This suggests that the model finds the text data relatively more predictable when the text is in its original form. This might be due to maintaining the full context and content of the data, including all stylistic and content-based cues that help in language modeling.
- **Removing stopwords** increases the perplexity to 195. This increase indicates that the model finds the text slightly harder to predict. Stopwords, though often seen as 'noise', actually carry significant contextual information that helps in predicting the flow of the text and the structure of sentences.
- **Custom preprocessing method** results in the highest perplexity (287). This significant increase can be attributed to several factors:
  - **Lowercasing all text** might lead to a loss of information (e.g., 'US' vs 'us').
  - **Removing punctuation** can alter the meaning or structure of sentences.
  - **Stemming** reduces words to their roots, which might cause different words with distinct meanings to be treated as the same (e.g., "university" and "universal" might both be stemmed to "univers").

F1-score and Precision

- The F1-score and Precision metrics also vary with preprocessing. The highest F1-score and precision are observed without any preprocessing, which aligns with the lower perplexity, indicating better overall model performance.
- Removing stopwords and applying the custom method both lead to decreases in F1-score, despite the slight increase in precision in the stopwords-removed scenario. This might suggest that while the model becomes slightly better at predicting relevant tokens correctly (precision), it does so at the expense of overall accuracy and recall, leading to a lower F1-score.

Part 2

Pre-training steps

BERT is pre-trained using two main self-supervised methods:

1. **Masked Language Model (MLM)**:

   In this pre-training task, 15% of the words in each sentence are replaced with a special token `[MASK]`. BERT then attempts to predict the original value of the masked words, based only on the context provided by the other non-masked words in the sequence. This allows BERT to effectively learn a deep bidirectional representation. For instance, in the sentence "The quick brown fox jumps over the lazy dog," words like "brown" or "dog" might be masked and BERT would predict their identity during training.

2. **Next Sentence Prediction (NSP)**:

   Here, BERT is given pairs of sentences as input and learns to predict whether the second sentence in the pair is the subsequent sentence in the original document. This task helps BERT capture relationships between consecutive sentences, which is useful in tasks that require understanding the relationship between multiple sentences, like question answering and natural language inference.

Application scenarios

1. Input: a sequence, Output: a class.
   - **Example**: *Sentiment analysis.*

- **Scenario**: BERT can be fine-tuned for the task of sentiment analysis, where the input is a text sequence such as a review or a tweet, and the output is a class label indicating the sentiment expressed in the text (e.g., positive, negative, neutral).

2. Input: sequence, Output: same as input.
    - **Example**: *POS tagging.*
    - **Scenario**: In POS tagging, each word in an input sequence of words is tagged with its corresponding part of speech (e.g., noun, verb, adjective), effectively mirroring the input sequence with added annotations.
3. Input: two sequences, Output: a class.
    - **Example**: *Natural Language Interface (NLI).*
    - **Scenario**: BERT is utilized to determine the relationship between two input text sequences. The task is to predict whether the second sentence logically follows from the first, contradicts it, or is neutral (i.e., neither following nor contradicting).
4. Extraction-based Question Answering (QA)
    - **Scenario**: In extraction-based QA, the model predicts the exact answer to a question from a given text passage. BERT looks at the question and the passage to find the text span within the passage that best answers the question.

Difference between BERT and distilBERT

BERT and DistilBERT are related but have key differences:

- **Model Size and Speed**:

  DistilBERT is essentially a smaller, faster version of BERT. It has fewer parameters than BERT-base and runs faster, while retaining BERT's language understanding capabilities and being cheaper to train.

- **Training Process**:

  DistilBERT is trained through a process called distillation, which involves training a smaller model (the "student") to reproduce the behavior of a larger model (the "teacher"). In this case, the teacher would be BERT. The student model learns to mimic the teacher's probabilities output for different tasks, allowing DistilBERT to maintain much of BERT's performance despite its reduced size.

- **Performance**:

  While DistilBERT generally performs very well and is more efficient, it may slightly underperform compared to BERT in more complex or sensitive tasks due to its reduced model complexity and parameter count.

Testing Result

- Without any preprocess:
    - F1-score: `0.9347`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type BERT --preprocess 0 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/1: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.2319, Precision: 0.9347, Recall: 0.9347, F1: 0.9347
```

- With remove stopwords:
    - F1-score: `0.9214`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type BERT --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/1: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.2686, Precision: 0.9219, Recall: 0.9214, F1: 0.9214
```

- With my preprocess method
    - F1-score: `0.8897`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type BERT --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/1: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.3194, Precision: 0.8927, Recall: 0.8899, F1: 0.8897
```

Bonus

- BERT uses the Transformer's encoder mechanism to create pre-trained representations of natural language. It further innovates by pre-training on a large corpus of text with a masked language model (MLM) objective, which allows it to capture bidirectional context and learn deep representations of language that are transferable to a wide variety of NLP tasks.
- The core of the Transformer architecture is the **self-attention mechanism**, which allows for modeling dependencies without regard to their distance in the input or output sequences.

Part3

Difference Between Vanilla RNN and LSTM

Vanilla RNN:

- Vanilla RNN is a basic form of RNN that processes sequential data by maintaining a 'hidden state' that captures information about the sequence processed so far. At each time step, the hidden state is updated based on the current input and the previous hidden state.
- The main limitation of vanilla RNNs is their inability to effectively learn long-range dependencies in a sequence due to the vanishing gradient problem, where the gradients tend to either explode or vanish during the backpropagation process, making it difficult for the RNN to learn weights that capture long-term dependencies.

LSTM:

- LSTM units are a type of RNN that are designed to avoid the vanishing gradient problem and better capture long-range dependencies. LSTMs maintain a separate cell state alongside the hidden state and use gates to control the flow of information. These gates decide what information to keep, discard, or pass through, which helps LSTMs to preserve information over many time steps.

Dimensions of Input and Output for Each Layer in LSTM

Embedding Layer

- **Input**: A batch of token IDs (integer values representing each token), typically with a size of `[batch_size, sequence_length]`.
- **Output**: Embedded vectors corresponding to these token IDs with a size of `[batch_size, sequence_length, embedding_dim]`.

LSTM Layer

- **Input**: Embedded token vectors of size `[batch_size, sequence_length, embedding_dim]`.
- **Output**: The LSTM layer outputs two things:
  - **Output of all hidden states** across all time steps, with size `[batch_size, sequence_length, hidden_dim]`.
  - **(hidden state, cell state)** of the last layer at the final time step, each with a size of `[num_layers, batch_size, hidden_dim]`.

Linear Layer

- **Input**: Typically the output from the last time step of the LSTM layer or a pooled representation of all time steps, depending on the specific task, with size `[batch_size, hidden_dim]`.
- **Output**: Transformed output with a size of `[batch_size, output_dim]`, which can be used for tasks like classification or regression depending on the desired output size (`output_dim`).

Testing Result

- Without any preprocess:
  - F1-score: `0.825`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type RNN --preprocess 0 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/10: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.6934, Precision: 0.5718, Recall: 0.5689, F1: 0.5646
Epoch 2/10: 100%|
Evaluating: 100%|
Epoch: 2, Loss: 0.6865, Precision: 0.6461, Recall: 0.6442, F1: 0.6430
Epoch 3/10: 100%|
Evaluating: 100%|
Epoch: 3, Loss: 0.6392, Precision: 0.7471, Recall: 0.7262, F1: 0.7203
Epoch 4/10: 100%|
Evaluating: 100%|
Epoch: 4, Loss: 0.5755, Precision: 0.7674, Recall: 0.7553, F1: 0.7525
Epoch 5/10: 100%|
Evaluating: 100%|
Epoch: 5, Loss: 0.5370, Precision: 0.7795, Recall: 0.7773, F1: 0.7769
Epoch 6/10: 100%|
Evaluating: 100%|
Epoch: 6, Loss: 0.4971, Precision: 0.8118, Recall: 0.8085, F1: 0.8080
Epoch 7/10: 100%|
Evaluating: 100%|
Epoch: 7, Loss: 0.4674, Precision: 0.8097, Recall: 0.8087, F1: 0.8085
Epoch 8/10: 100%|
Evaluating: 100%|
Epoch: 8, Loss: 0.4543, Precision: 0.8252, Recall: 0.8252, F1: 0.8252
Epoch 9/10: 100%|
Evaluating: 100%|
Epoch: 9, Loss: 0.4477, Precision: 0.8295, Recall: 0.8285, F1: 0.8284
Epoch 10/10: 100%|
Evaluating: 100%|
Epoch: 10, Loss: 0.4583, Precision: 0.8252, Recall: 0.8250, F1: 0.8250
```

- With remove stopwords:
  - F1-score: `0.8562`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type RNN --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/10: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.6926, Precision: 0.6046, Recall: 0.6045, F1: 0.6044
Epoch 2/10: 100%|
Evaluating: 100%|
Epoch: 2, Loss: 0.6146, Precision: 0.7885, Recall: 0.7798, F1: 0.7781
Epoch 3/10: 100%|
Evaluating: 100%|
Epoch: 3, Loss: 0.5022, Precision: 0.8169, Recall: 0.8164, F1: 0.8163
Epoch 4/10: 100%|
Evaluating: 100%|
Epoch: 4, Loss: 0.4630, Precision: 0.8163, Recall: 0.8154, F1: 0.8153
Epoch 5/10: 100%|
Evaluating: 100%|
Epoch: 5, Loss: 0.4500, Precision: 0.8378, Recall: 0.8374, F1: 0.8374
Epoch 6/10: 100%|
Evaluating: 100%|
Epoch: 6, Loss: 0.4352, Precision: 0.8466, Recall: 0.8463, F1: 0.8463
Epoch 7/10: 100%|
Evaluating: 100%|
Epoch: 7, Loss: 0.4112, Precision: 0.8518, Recall: 0.8515, F1: 0.8515
Epoch 8/10: 100%|
Evaluating: 100%|
Epoch: 8, Loss: 0.4030, Precision: 0.8452, Recall: 0.8445, F1: 0.8444
Epoch 9/10: 100%|
Evaluating: 100%|
Epoch: 9, Loss: 0.3960, Precision: 0.8489, Recall: 0.8447, F1: 0.8442
Epoch 10/10: 100%|
Evaluating: 100%|
Epoch: 10, Loss: 0.3867, Precision: 0.8572, Recall: 0.8563, F1: 0.8562
```

- With my preprocess method
  - F1-score: `0.8619`

```
PS C:\Users\ChuEating\Intro-to-AI\HW2> python main.py --model_type RNN --preprocess 1 --part 2
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ChuEating\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Epoch 1/10: 100%|
Evaluating: 100%|
Epoch: 1, Loss: 0.6888, Precision: 0.7422, Recall: 0.7333, F1: 0.7308
Epoch 2/10: 100%|
Evaluating: 100%|
Epoch: 2, Loss: 0.5996, Precision: 0.8106, Recall: 0.8054, F1: 0.8046
Epoch 3/10: 100%|
Evaluating: 100%|
Epoch: 3, Loss: 0.4762, Precision: 0.8315, Recall: 0.8283, F1: 0.8279
Epoch 4/10: 100%|
Evaluating: 100%|
Epoch: 4, Loss: 0.4510, Precision: 0.8245, Recall: 0.8035, F1: 0.8003
Epoch 5/10: 100%|
Evaluating: 100%|
Epoch: 5, Loss: 0.4308, Precision: 0.8488, Recall: 0.8488, F1: 0.8488
Epoch 6/10: 100%|
Evaluating: 100%|
Epoch: 6, Loss: 0.4177, Precision: 0.8551, Recall: 0.8549, F1: 0.8549
Epoch 7/10: 100%|
Evaluating: 100%|
Epoch: 7, Loss: 0.4204, Precision: 0.8571, Recall: 0.8534, F1: 0.8530
Epoch 8/10: 100%|
Evaluating: 100%|
Epoch: 8, Loss: 0.3988, Precision: 0.8558, Recall: 0.8554, F1: 0.8554
Epoch 9/10: 100%|
Evaluating: 100%|
Epoch: 9, Loss: 0.3945, Precision: 0.8541, Recall: 0.8490, F1: 0.8485
Epoch 10/10: 100%|
Evaluating: 100%|
Epoch: 10, Loss: 0.3862, Precision: 0.8653, Recall: 0.8622, F1: 0.8619
```

Discussion

Innovation of the NLP

N-gram

- Simple and based on statistical probabilities
- Limitations: Only captures limited context and suffers from sparsity and scalability issues.

LSTM

- Can remember information for long periods and are better at capturing context over longer sequences
- Addresses vanishing gradient problem in RNNs, allowing them to learn from longer input sequences.

BERT

- Uses a bidirectional approach to understand the context of a word based on all its surrounding words (not just the words that preceded it).
- It introduces the attention mechanism that allows it to consider the importance of each word in a sentence, regardless of their position, enabling a deeper understanding of language.

Due to computational advances and deep learning innovations, more powerful computational resources have allowed for the development and training of more complex models like LSTM and BERT. And also facilitate the handling of long-range dependencies and context much more effectively.

Problems and Solution

1. Training Time Too Long
    - **Problem:** As models get more complex and datasets larger, the time to train a model increases significantly.
    - **Solution:** Utilizing CUDA drivers to leverage GPU acceleration can drastically reduce training times. This approach offloads intensive computation tasks to GPUs, which are better suited for parallel processing necessary in deep learning.
2. Unfamiliarity with PyTorch
    - **Problem:** Challenges in using PyTorch effectively due to a lack of experience with the framework, which can hinder progress in model development and experimentation.
    - **Solution**: Read the official PyTorch documentation for functions and modules. PyTorch documentation is well-maintained and provides a good combination of theoretical background and practical examples.