Assignment 4

CS 421: Natural Language Processing

1 Introduction

In this assignment, you will learn to implement the Named Entity Recognition task using the LSTM model. You will also build the TF-IDF word vectorizer and find cosine similarity between two sentences.

Named Entity is defined as any word/term that can be referred to with a proper name: a person, a location, or an organization. The task of **Named Entity Recognition (NER)** is to find spans of text that constitute proper names and tag them with the proper type of entity. NER is a useful first step in a lot of natural language processing tasks like sentiment analysis where we may find consumers' perceptions of certain entities.

You need to use **Long-Short Term Memory (LSTM)** model for the NER task, a type of Recurrent Neural Network which uses a series of 'gates' that control how the information in a sequence of data comes in, is stored and leaves the network. There are three gates in a typical LSTM: *forget gate*, *input gate*, and *output gate*.

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects how important a word is to the document in a corpus. It is often used as a weight in the tasks like information retrieval and text mining. It is the product of two terms: *Term Frequency* and *Inverse Document Frequency*. You will learn to implement TF-IDF in this assignment.

To measure the similarity between two words, we need a metric that takes a vector representation of the words and gives a measure of their similarity. One of the most common metrics is **Cosine Similarity**.

2 Instructions

Code

This section needs to be completed using Python 3.6+. You will also require following packages:

- `pandas`
- `numpy`
- `NLTK` or `SpaCy`
- `scikit-learn`
- `Keras`

If you want to use an external package for any reason, you are required to get approval from the course staff prior to submission.

3 Questions

Q1. TF-IDF and cosine similarity: Code [15]

In this section, you will build the TF-IDF model and calculate the cosine similarity score. Use **conll2003** dataset from Huggingface[1].

1. Load the dataset in a pandas DataFrame. Drop all columns except `tokens` and `ner_tags`. Create a vocabulary set of all words appearing in the column `tokens`. We will refer to each row as a document.
2. Iterating over the vocabulary, create a dictionary containing the word as the key and a unique index as the value. You must assign a unique index to each word which is an integer that will be incremented by one.
3. Iterating over all the vocabulary words, create a dictionary with the word as the key and the number of documents the word occurred in as the value. If the word 'NLP' appeared in 5 rows of the dataset, the entry in the dictionary will be {'NLP':5}.
4. Implement a method `term_frequency(term, document)` to calculate the term frequency $tf_{t,d}$ as follows:
$$tf_{t,d} = \log_{10}(count(t,d) + 1)$$
$count(t,d)$ is the number of times the term appeared in the document.
5. Implement a method `idf(word)` to calculate the Inverse Document Frequency idf_t given as
$$idf_t = \log_{10}(\frac{N}{df_t})$$
Here, df_t is the number of documents the terms appeared in, which you will get from step 3. If the term is not in the dictionary, assume $df_t = 1$ to avoid division by 0.
6. Write a method `tfidf(document)` to calculate the TF-IDF for each document. In the method, create a NumPy array with all zeros of shape `(len(vocab),)`. For each word in the document, calculate TF-IDF by calling the methods `term_frequency` and `idf` and taking a product of the values. Update this value of the array (at the index of the word) with the TF-IDF calculated.
7. Create an empty NumPy array. Follow step 6 for all the documents in the dataset and append them to the array.
8. Cosine similarity metric is calculated as follows:
$$cosine(v,w) = \frac{v.w}{|v||w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

[1] `https://huggingface.co/datasets/conll2003`

For each of the following examples, find the TF-IDF vectors for both sentences and compute cosine similarity. Report your observations.

- sentence 1: "I love football" sentence 2: "I do not love football"
- sentence 1: "I follow cricket" sentence 2: "I follow baseball"

Q2: Compute Positive Pointwise Mutual Information (PPMI) for all pair of words in a sentence.

PPMI is a measure of the association between two words, and is defined as:

$$PPMI(x, y) = max(PMI(x, y), 0)$$

where $PMI(x, y)$ is the pointwise mutual information between x and y, and is defined as:

$$PMI(x, y) = log_2(p(x, y)/(p(x) \cdot p(y)))$$

Here, $p(x)$ is the probability of word x occurring in the text, $p(y)$ is the probability of word y occurring in the text, and $p(x, y)$ is the probability of x and y occurring together in the text.

- You should write a function (calculate_ppmi) that takes as input a list of strings (representing the words in the text) and returns a dictionary that maps tuples of words to their PPMI values. For example, if the input is [‘a’, ‘b’, ‘a’, ‘c’], the output should be a dictionary that maps the tuple (‘a’, ‘b’) to the PPMI value of ‘a’ and ’b’ in the text [‘a’, ‘b’, ‘a’, ‘c’].
- You may assume that the input list of strings is non-empty and contains no duplicates.

Q3. Named Entity Recognition using LSTM: Code [35]

You will learn to build LSTM in Keras and apply it to the task of Named Entity Recognition. You will also learn to utilize word2vec model to obtain word embedding. The **conll2003** dataset follows BIO tagging and consists of 9 NER tags.

- 0 is for no tag.
- Tags 1,2 refer to **Person** tags, **B-PER** and **I-PER** respectively.
- Tags 3,4 refer to **Organization** tags, **B-ORG** and **I-ORG** respectively.
- Tags 5,6 refer to **Location** tags, **B-LOC** and **I-LOC** respectively.
- Tags 7,8 refer to **Miscellaneous** tags, **B-MISC** and **B-MISC** respectively.
- For example, ’John Doe and Jane Doe’ will have (1,2,0,1,2) tags.

Complete each of the following tasks.

1. Install Keras[2]. Using `sklearn`, split the **conll2003** dataset with first 80% samples for training and the remaining 20% samples for testing. For both sets, obtain the word

[2] `https://pypi.org/project/keras/`

embedding from word2vec model[3] trained on Google News corpus.

2. Create a model in Keras using `Sequential`[4], `Embedding`[5], and `LSTM`[6] models. The model must contain 3 LSTM layers, one fully-connected layer, and one last fully-connected layer with `softmax` activation and 9 outputs (one for each tag).
3. Train the model using Cross entropy loss, an optimizer of your choice (use of adam is recommended), and 10 epochs. You are free to vary the number of epochs to boost performance.
4. Calculate the accuracy and macro-average precision, recall, and F1 score on the test set.

[3] https://radimrehurek.com/gensim/models/word2vec.html#pretrained-models
[4] https://keras.io/guides/sequential_model/
[5] https://keras.io/api/layers/core_layers/embedding/
[6] https://keras.io/api/layers/recurrent_layers/lstm/

4 Rubrics

This assignment will be graded according to the rubric below. Partial points may be awarded for rubric items at the discretion of the course staff.

Q1 TF-IDF and cosine similarity: (25 points possible)	
Q 1.1 - Q 1.2	+5
Q 1.3 - Q 1.4	+5
Q 1.5	+5
Q 1.6 - Q 1.7	+5
Q 1.8	+5
Q2 Compute PPMI: (5 points possible)	
Q2 Named Entity Recognition using LSTM: (20 points possible)	
Q 3.1 - Q 3.2	+10
Q 3.3	+5
Q 3.4	+5