**[Coleridge] Text Matching and NER cnt**

**Preprocessing Steps**

1. Importing libraries
2. Define some variables and read in a CSV file using pandas.
3. define several helper functions that will be used later in the code.

* **url\_regex** used to remove URLs starting with "http://" or "https://" from the text of articles.
* **www\_regex** used to remove URLs starting with www. from the text of articles.
* **get\_article :** A function that takes a filename and directory path as input and returns the text of the article as a string. The function first reads in a JSON file with the specified filename from the specified directory, then extracts the text and section titles from the file and concatenates them into a single string. The function also removes URLs from the string using the **url\_regex** and **www\_regex** regular expressions.
* **clean\_text :** A function that takes a string as input and returns a cleaned version of the string. The function removes all non-alphanumeric characters from the string and converts it to lowercase.
* **jaccard\_similarity :** A function that takes two strings as input and returns their Jaccard similarity coefficient as a float.

1. tqdm.pandas() to display a progress bar while applying the **get\_article** function to each row of the **Id** column of the **sample\_df** DataFrame.
2. These lines of code read in the **train.csv** file, preprocess the data and extract a set of unique dataset labels and titles that will be used to match the labels in the training data to the text of the articles in the test data.

* Read train.csv
* **temp\_1 :** A set that extracts unique dataset labels from the **dataset\_label** column of the **train\_df** DataFrame, preprocesses them by removing text inside parentheses using the **bracket\_regex** pattern, converts them to lowercase, and adds a space to the end of any label that consists of a single word. The resulting set is stored in the **temp\_1** variable.
* **temp\_2 :** A set that extracts unique dataset titles from the **dataset\_title** column of the **train\_df** DataFrame, preprocesses them in the same way as **temp\_1**, and stores the resulting set in the **temp\_2** variable.
* **existing\_labels :** combines the sets created in **temp\_1** and **temp\_2**. The resulting set contains unique, preprocessed dataset labels and titles that will be used to match the labels in the training data to the text of the articles in the test data.

1. defines three sets of words: **TOPIC\_WORDS, STOP\_WORDS, existing\_labels**

* **TOPIC\_WORDS** contains a set of words that are likely to be present in the dataset labels,
* **STOP\_WORDS** contains a set of words that are likely to be present in the dataset labels but do not contain any meaningful information
* **existing\_labels** contains all the unique dataset labels found in the training data after cleaning.
* Then, the function **get\_additional\_labels** is defined.

The function then checks whether a dataset label meets the following conditions:

* It appears in at least **df\_thres** training data entries.
* It contains at least one word from the **topic\_words** set.
* It does not contain any words from the **stop\_words** set.

If a dataset label meets these conditions, and if its Jaccard similarity to any of the existing dataset labels in **existing\_labels** is less than 0.5, it is added to the **addtional\_labels** set.

The resulting **addtional\_sets** set contains the additional dataset labels that satisfy the conditions set by the function.

7. The code above is reading in a JSON file containing extracted results, and then processing these results to identify any additional labels that can be added to **addtional\_sets**

8. This will ensure that **additional\_labels** contains a sorted list of strings.

9. This code sets the maximum length of the input sequence to 128 tokens, the batch size for training to 128, and the directory where the pre-trained Electra model weights are located.

10. This code defines a tokenizer using the **ElectraTokenizerFast** class from the Hugging Face library, which is used to tokenize text. The **from\_pretrained()** method loads a pre-trained tokenizer from the directory specified in **ENCODER\_DIR**.

11. This code defines a custom model for Named Entity Recognition (NER) task that includes a Conditional Random Field (CRF) layer on top of a bidirectional LSTM layer. The model takes as input tokenized text and outputs the predicted NER tags for each token.

12. This function takes in a string of text, splits it into sentences based on the period symbol, and returns a set of sentences that have at least six words. It is used to filter out short sentences in the text data to ensure that the model has enough context to make accurate predictions.

13. This function takes in the inputs **x** (a batch of encoded text data), **y** (a batch of predicted labels), **tokenizer** (a tokenizer object used to convert encoded data back to text), and **label2id** (a dictionary mapping labels to their corresponding integer IDs).

The function returns the set of decoded predictions for the entire batch.

## 14.1st Stage: Text Matching with Additional Labels

* For text matching with additional labels in the first stage, no specific algorithm is used, just string matching and text preprocessing techniques (lowercasing, cleaning, etc.).

## 15.2nd Stage: Named Entity Recognition

* For named entity recognition in the second stage, the code uses an Electra transformer model for pretraining, which is fine-tuned on the task at hand using the CRF (Conditional Random Field) algorithm as the loss function during training. The **decode\_prediction()** function applies the BIO (Beginning, Inside, Outside) tagging scheme to the model's output and uses the CRF algorithm to decode the predicted labels.

## 16**.** Post Processing

* For post-processing, the code uses a simple Jaccard similarity threshold to filter out false positives from the second stage that do not overlap sufficiently with the labels obtained in the first stage.

In [16]:

In [15]: