**WINNER OF THE CONTEST:**

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

**brief description about the notebook**

1. First, the text is preprocessed to clean it and prepare it for NER. This involves removing any unnecessary characters, lowercasing the text, and splitting it into sentences.
2. Then, the text is matched with a set of known labels that are already present in a training dataset. These labels might include things like the names of specific diseases or organizations.
3. Next, the text is matched with additional labels that were not present in the training dataset. These additional labels might include things like variations on known labels or labels that were not previously encountered.
4. The text is then split into sentences, and each sentence is encoded using a pre-trained language model (in this case, Electra). The encoded sentences are passed through a neural network that is trained to perform NER. This produces a set of predicted labels for each sentence.
5. The predicted labels are post-processed to remove any labels that do not appear frequently enough in the predictions. This helps to filter out noise in the predictions.
6. Finally, the predicted labels are combined and output in a CSV file in the required submission format.

* The solution uses a two-stage approach: text matching with additional labels in the first stage, and named entity recognition in the second stage.

- The first stage matches the text with known dataset labels and additional labels provided

- while the second stage uses a pre-trained Electra-based model to perform named entity recognition and extract relevant dataset labels.

* The code uses a pre-trained Electra-based model that has been fine-tuned on a labeled dataset of scientific publications. It also uses cross-validation with four folds to improve the model's performance.
* The algorithm used here to implement NER (Named Entity Recognition) is Conditional Random Fields (CRF).

## 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.).

## 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.

## 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.

**summary**

The code consists of four stages:

1. Text Matching with Additional Labels: In this stage, the code reads in the test data and compares the text in the publications to a set of pre-defined known labels and additional labels. It then predicts the labels that match the text and stores them for the next stage.
2. Named Entity Recognition: In this stage, the code uses a pre-trained model for Named Entity Recognition to predict the labels that correspond to datasets. It then combines these labels with those predicted in the first stage.
3. Post Processing: In this stage, the code refines the predicted labels by comparing the similarity of the labels predicted in the first and second stages.
4. Submission: Finally, the code outputs the predicted labels for each publication in a submission file.

**Other notebooks**

# [Coleridge] Matching + MLM 3 32

Steps as in the notebook:

1. Import(libraries)
2. Load data
3. Literal matching
4. Masked Dataset Modeling
5. Transform data to MLM format

Here are the important steps in this code:

1. Loading necessary libraries and data
2. Defining utility functions for text cleaning, model inference, and label filtering
3. Loading the pre-trained NER model and setting up some model parameters
4. Looping over each test paper, cleaning the text, running inference with the NER model, and filtering the labels
5. Saving the filtered labels for each paper as the final prediction for that paper
6. Combining the predictions for all papers into a submission file format and saving it as a CSV file.

The main models used in this code are:

1. **Hugging Face Transformers**: A Python library for natural language processing that provides state-of-the-art pre-trained models for tasks such as text classification, question answering, and summarization.
2. **BERT**: Bidirectional Encoder Representations from Transformers, a pre-trained transformer-based neural network architecture for natural language processing tasks.
3. **BioBERT**: A pre-trained BERT model specifically designed for biomedical text mining tasks.
4. **SciSpacy**: A Python library built on top of spaCy to help with common text processing tasks in scientific documents, such as named entity recognition, sentence segmentation, and entity linking.

The code also uses regular expressions to match and extract certain patterns from the text, and some basic filtering techniques to remove irrelevant results.

## Colaridge-Bert

In summary, this notebook performs the following steps:

1. Load the necessary libraries and set up the input and output file paths.
2. Load and preprocess the training data, including cleaning the text and labeling it using external NER libraries.
3. Define the functions for the exact matching and BERT-based matching methods for labeling the test data.
4. Load and preprocess the test data, including cleaning the text and splitting it into smaller segments for prediction.
5. Run the exact matching and BERT-based matching methods on the test data, and combine the results to generate the final predictions.

- If there is a literal match, it takes that match as the final prediction. Otherwise, it takes the filtered and cleaned BERT predictions as the final prediction.

1. Format the final predictions in the required submission format and save to a file.

Top of Form