CS 585 – Fall 2023 – Homework 4 (100 total points)

Due TUESDAY November 7, 11:59pm

GOALS

- Gain experience with **Conditional Random Fields (CRFs)** using an open-source python library
- Perform a **named entity recognition (NER)** experiment framed as sequential labeling problem
- Explore data from the specialized **biomedical** sub-domain of NLP

DATA

In this homework you will work with **biomedical** text data, which is an expanding area of focus within NLP. You will perform named entity recognition where the "entity" of interest is a disease mention.

For this assignment, you will use data from the NCBI Disease Corpus, available on GitHub. In this corpus, disease mentions are tagged in a B-I-O format, where a "B" tag indicates the first word/token in a disease mention, tag "I" indicates subsequent words/tokens, and tag "O" indicates a token is not part of a disease mention.

You will use files posted in the "conll" subfolder: ncbi-disease/conll Note that in these files, each line contains one token or tag, and **a blank line indicates the boundary between sequences.** So, one sequence spans many lines in this file format.

You can read more about the NCBI Disease Corpus, including the annotation process, in this paper.

TOOLS

This assignment requires the **python-crfsuite** library. Refer to: https://pypi.org/project/python-crfsuite

You may find this notebook in the python-crfsuite repository to be a useful reference, although note that you will not follow it exactly for this homework.

You may use other open-source libraries so long as you follow homework directions. Some of the tools you have already used from **scikit-learn** may be helpful. You should use Python to complete this assignment. Other programming languages will not be accepted

WHAT TO SUBMIT

Please upload or submit the following in Blackboard:

- For Problems 1 through 6, please upload to Blackboard:
  - One Jupyter notebook (.ipynb file) with cell output, showing your work for both datasets. Name this file "HW4_[CWID]_[LastName].ipynb"
  - A PDF copy of the exact same notebook (same code and same output)
  - You will lose points if you do not submit both of these above files. Please do not zip or compress files before uploading files to Blackboard.
- For Problem 7: Enter your written answers in Blackboard with your HW submission.

PROBLEM 1 – Reading the data in CoNLL format (20pts)

Note that the NCBI Disease Corpus (See section **DATA** above) is already split into train, development, and test datasets. You will use the **train** and **test** datasets in this homework.

As noted above, you should use files in the "ncbi-disease/conll" subfolder. In this file format, a blank line indicates the start of a new sequence.

- Write a **function** that reads a .tsv files in the CoNLL format and returns two "list of lists" as output:
  - A list of sequences of **tokens**, where a single token may be a word or punctuation.
  - A list of sequences of **tags,** representing token-level annotation. You should see these 3 tags in your data ("B-Disease", "I-Disease", "O")
- Apply your function to train.tsv and test.tsv. To show you have read in the data correctly, show the following in your notebook output:
  - The number of sequences in train and test. (You should see 5432 sequences in train and 940 sequences in test.)
  - The tokens and tags of the first sequence in the **training dataset**. Your output should look like this:

```
['Identification', 'of', 'APC2', ',', 'a', 'homologue', 'of', 'the', 'adenomatous',
'polyposis', 'coli', 'tumour', 'suppressor', '.']
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'I-
Disease', 'O', 'O']
```

PROBLEM 2 – Data Discovery (5 pts)

In this problem you will examine the data that you read into memory in the previous problem. Using the training dataset for analysis, show the following in your notebook output:

- The count of each of the 3 tags in the training data: "B-Disease", "I-Disease", and "O". Note that the most frequent token is "O", since most words are not part of a disease mention.
- The 20 most common words/tokens that appear with the tags "B-Disease" or "I-Disease". That is, show words that often appear disease mentions. (You may show frequent "B-Disease" and "I-Disease" words separately, or you may combine them into a single list.)
- OPTIONAL: Any other data exploration you would like to perform. For example, you may want to print and read a small sample of token sequences, to become familiar with the data.

Review the list of words that commonly appear in disease mentions. Do you see any patterns? (You do not need to answer in writing, but it may be helpful in Problem 3 where you design a feature.)

PROBLEM 3 – Building features (20 pts)

In this problem, you will build the features that you will use in your CRF model. You may find it helpful to refer to this demo notebook, to understand how to work with the python-crfsuite library.

- Write a function that takes two inputs:
    - A sequence of tokens
    - An integer position, pointing to one token in that sequence.

  and returns a list of features, represented as a **list of strings.** At minimum, include these features:

    - The **current word/token** in lower case
    - The **suffix** (last 3 characters) of the current word
    - The **previous word/token** (position i-1) or "BOS" if at the beginning of the sequence
    - The **next word/token** (position i+1), or "EOS" if at the beginning of the sequence
    - **At least one other feature of your choice**
- Apply your function your train and test token sequences (from output of Problem 1).
- To show that you have completed this step, apply your output to the first 3 words in the first sequence of the training set. Your output should look something like this (note the names and order of your features in your notebook do not need to match this output):

```
['w0.lower=identification', 'w0.suffix3=ion', <other features not shown...>, BOS ]
['w0.lower=of', 'w0.suffix3=of', <other features not shown...>]
['w0.lower=apc2', 'w0.suffix3=PC2', <other features not shown...>]
```

PROBLEM 4 – Training a CRF model (20 pts)

In this problem, you will train a CRF model and evaluate it using metrics computed over individual tags.

- Using the python-crfsuite library, train a CRF sequential tagging model using feature sequences that you built in the previous step. Using your training data as input.
- Apply your model to your test dataset to generate predicted tag sequences.
- For each of the 3 labels ("B-Disease", "I-Disease", and "O") show precision, recall, f1-score. [You may use the sckit-learn function classification_report to complete this step. You may also want to "flatten" both the true and predicted tags into a single list of tags to apply this function.]

PROBLEM 5 – Inspecting the trained model (10 pts)

In this problem you will examine parameter weights assigned by your model. You can do this by calling "tagger.info().transitions" and "tagger.info().state_features" on your trained model object.

- In your notebook, show parameter weights given to transitions between the 3 tag types ("B-Disease", "I-Disease", and "O").
- Refer back to the feature you designed in Problem 3 (the feature "of your choice"). Show the parameter weights assigned to this feature. You may truncate this list if it is very long. [This may happen if you included a word from the sequence in the feature name, so your feature was expanded to become a larger set of features that grows with your vocabulary]
- *IF* your feature was dropped during model training (that is, there is nothing to show in the previous step) then return to Problem 4 and design a new feature that is used in your model.

PROBLEM 6 – Document level performance (10 pts)

Tag-level accuracy is easy to compute, but it is not very easy to understand. In particular, one disease reference may cover both "B-Disease" and "I-Disease" tokens. To give another view of model performance, compute **document-level precision and recall** on your experiment output. To do this:

- Write a **function** that aggregates token-level tags to a document-level label. For example, convert a tag sequence like ["O", "B-Disease", "I-Disease", "O", "O"] to a single label y=1. Your function should assign y=1 to a sequence with <u>one or more</u> disease mentions (at least one "B-Disease" tag) and y=0 to a sequence with <u>no</u> disease mentions.
- Apply your function to both true and predicted document-level labels from your test set. Use the output to compute **document level precision and recall** of your model. Show your results in your notebook.

PROBLEM 7 – State Transitions (10 pts – Answer in Blackboard)

The python-crfsuite library allows you to set a Boolean hyper-parameter called "feature.possible_transitions". If this parameter is True, then the model may output tag-to-tag transitions that were never seen in training data. [You do not need to apply this parameter in your code to answer this question]

- What is an example of one tag-to-tag transition that never occurred in the training data?
- For this particular experiment, do you think it makes sense to set this parameter to True or False? That is, should you allow transitions that never occurred in the training data? Explain your answer briefly.

OPTIONAL Exercise (Not graded)

- Run a grid search to tune your CRF model, testing multiple combinations of L1 and L2 regularization parameters ("c1"and "c2"). You may also want to consider different features sets in your grid search, for example, including word-prefix as well as word-suffix.
- Use the development set (“devel”) that is shared with the NCBI corpus to select the best set of hyperparameters. Rerun your experiments with your selected grid point. Did you improve on your test set metrics?

CODE ORGANIZATION AND READABILITY (5 pts)

The receive full credit, please ensure that:

- You have included **both** notebook and PDF files, as described in "What to Submit". Please do **not** upload compressed .zip or .tar files, these file types cannot be viewed in Blackboard by graders.
- Your notebook includes cell output, and does NOT contain error messages
- It is easy to match a problem with its code solution in your notebook via markdown or comments (e.g., “Problem 2”)
- You re-ran your notebook before submitting, and cells are numbered sequentially starting at [1]