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

python-crfsuite library.

PROBLEM 1 – Reading the data in CoNLL format

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

- 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**.

PROBLEM 2 – Data Discovery

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.)

PROBLEM 3 – Building features

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).

PROBLEM 4 – Training a CRF model

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

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

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 one or more disease mentions (at least one "B Disease" tag) and y=0 to a sequence with no 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.