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

```
def load_conll_data(file_path):
    data_sequences = []
    data_tags = []
    with open(file_path, 'r') as data_file:
        current_sequence = []
        current_tag_sequence = []
        for data_line in data_file:
            data_line = data_line.strip()
            if data_line:
                data_token, data_tag = data_line.split('\t')
                current_sequence.append(data_token)
                current_tag_sequence.append(data_tag)
            else:
                data_sequences.append(current_sequence)
                data_tags.append(current_tag_sequence)
                current_sequence = []
                current_tag_sequence = []
        if current_sequence:
            data_sequences.append(current_sequence)
            data_tags.append(current_tag_sequence)
    return data_sequences, data_tags

# Load data from train.tsv and test.tsv files
train_data_sequences_tokens, train_data_tags = load_conll_data('/content/conll/train.tsv')
test_data_sequences_tokens, test_data_tags = load_conll_data('/content/conll/test.tsv')

# Print the number of sequences in the train and test datasets
print("Number of sequences in the train dataset:", len(train_data_sequences_tokens))
print("Number of sequences in the test dataset:", len(test_data_sequences_tokens))

# Print the tokens and tags of the first sequence in the training dataset
print("\nTokens of the first sequence in the train dataset:")
print(train_data_sequences_tokens[0])
print("\nTags of the first sequence in the train dataset:")
print(train_data_tags[0])
```

```
Number of sequences in the train dataset: 5432
Number of sequences in the test dataset: 940

Tokens of the first sequence in the train dataset:
['Identification', 'of', 'APC2', ',', 'a', 'homologue', 'of', 'the', 'adenomatous', 'polyposis', 'coli', 'tumour', 'suppressor', '.']

Tags of the first sequence in the train dataset:
['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 “IDisease” words separately, or you may combine them into a single list.)

```
from collections import Counter

# Count of each tag in the training data
tag_counts = Counter(tag for tag_sequence in train_data_tags for tag in tag_sequence)
print("Tag counts in train:", tag_counts)

# Word counts for B-Disease and I-Disease tags
all_words = []
bdisease_words = []
idisease_words = []

for sequence, tag_sequence in zip(train_data_sequences_tokens, train_data_tags):
    for token, tag in zip(sequence, tag_sequence):
        all_words.append(token)
        if tag == 'B-Disease':
            bdisease_words.append(token)
        if tag == 'I-Disease':
            idisease_words.append(token)

B_or_I_disease_counts = Counter(all_words)
bdisease_counts = Counter(bdisease_words)
idisease_counts = Counter(idisease_words)

print("\n20 most common words/tokens with either B-Disease or I-Disease tag:")
print(B_or_I_disease_counts.most_common(20))
print("\n20 most common words/tokens with B-Disease tag with their respective counts:")
print(bdisease_counts.most_common(20))
print("\n20 most common words/tokens with I-Disease tag with their respective counts:")
print(idisease_counts.most_common(20))
```

```
Tag counts in train: Counter({'O': 124819, 'I-Disease': 6122, 'B-Disease': 5145})

20 most common words/tokens with either B-Disease or I-Disease tag:
[('.', 6294), ('the', 5703), ('of', 5369), (',', 4585), ('in', 3617), ('-', 3580), ('and', 3176), ('a', 2345), (')', 1964), ('(', 1954), ('to', 1706), ('with', 1526), ('gene', 1148), ('is', 1026),

20 most common words/tokens with B-Disease tag with their respective counts:
[('DM', 120), ('breast', 115), ('DMD', 110), ('APC', 94), ('X', 92), ('ALD', 86), ('PWS', 75), ('G6PD', 68), ('WAS', 63), ('autosomal', 58), ('familial', 58), ('myotonic', 57), ('Duchenne', 56),

20 most common words/tokens with I-Disease tag with their respective counts:
[('-', 636), ('syndrome', 281), ('deficiency', 275), ('disease', 256), ('cancer', 230), ('of', 178), ('dystrophy', 176), ('and', 120), ('disorder', 92), ('ovarian', 86), ('muscular', 84), ('linkec
```

• 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.)

```
#print token sequences having token hyphen '-'
result = []
```

```
for sublist in train_data_sequences_tokens:
    if '-' in sublist:
        result.append(sublist)
print(result[:5])
```

```
[['The', 'adenomatous', 'polyposis', 'coli', '(', 'APC', ')', 'tumour', '-', 'suppressor', 'protein', 'controls', 'the', 'Wnt', 'signalling', 'pathway', 'by', 'forming', 'a', 'complex', 'with', 'g
```

In the context of this dataset, B-Disease and I-Disease tags are annotations used for named entity recognition. They stand for "Beginning" and "Inside" respectively, which are part of the BIO tagging scheme.

In the BIO scheme:

B-Disease: stands for the "beginning" of a disease entity mention in the text. It indicates the first word of a disease mention. I-Disease: stands for "inside" and it is used for all other words of the disease name, which are not the first word. For example, in the phrase "adenomatous polyposis coli", "adenomatous" might be marked as "B-Disease", and "polyposis" and "coli" as "I-Disease"

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.

```
import string

def token_features(tokens, index):
    token = tokens[index]
    features = [
        f'w0.length={len(token)}' ,  # New feature for punctuation
        f'w0.is_punctuation={token in string.punctuation}',  # New feature for punctuation
        f'w0.lower={token.lower()}',
        f'w0.suffix3={token[-3:]}',
        f'w0.is_digit={token.isdigit()}',

    ]
```

```
    if index > 0:
        prev_token = tokens[index - 1]
        features.extend([
            f'w-1={prev_token}',
        ])
    else:
        features.append('BOS')
    if index < len(tokens) - 1:
        next_token = tokens[index + 1]
        features.extend([
            f'w+1={next_token}',
        ])
    else:
        features.append('EOS')
    return features

# Applying the function to the first 4 tokens
sample_features = [token_features(train_data_sequences_tokens[0], i) for i in range(4)]
for i, features in enumerate(sample_features):
    print(f"Token {i} features: {features}")
```

```
Token 0 features: ['w0.length=14', 'w0.is_punctuation=False', 'w0.lower=identification', 'w0.suffix3=ion', 'w0.is_digit=False', 'BOS', 'w+1=of']
Token 1 features: ['w0.length=2', 'w0.is_punctuation=False', 'w0.lower=of', 'w0.suffix3=of', 'w0.is_digit=False', 'w-1=Identification', 'w+1=APC2']
Token 2 features: ['w0.length=4', 'w0.is_punctuation=False', 'w0.lower=apc2', 'w0.suffix3=PC2', 'w0.is_digit=False', 'w-1=of', 'w+1=,']
Token 3 features: ['w0.length=1', 'w0.is_punctuation=True', 'w0.lower=,', 'w0.suffix3=,', 'w0.is_digit=False', 'w-1=APC2', 'w+1=a']
```

Here, I printed first 4 tokens intead of 3 in order to show 'w0:is_punctuation=True' for 4th record

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.]

Answer - I performed this using sklearn-crfsuite and pycrfsuite both, getting the same output both the times.

```
!pip install sklearn-crfsuite
```

```
Requirement already satisfied: sklearn-crfsuite in /usr/local/lib/python3.10/dist-packages (0.3.6)
Requirement already satisfied: python-crfsuite>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from sklearn-crfsuite) (0.9.9)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from sklearn-crfsuite) (1.16.0)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (from sklearn-crfsuite) (0.9.0)
Requirement already satisfied: tqdm>=2.0 in /usr/local/lib/python3.10/dist-packages (from sklearn-crfsuite) (4.66.1)
```

```
from sklearn.metrics import classification_report
import sklearn_crfsuite
```

```
X_train = [[token_features(seq, i) for i in range(len(seq))] for seq in train_data_sequences_tokens]
y_train = train_data_tags
X_test = [[token_features(seq, i) for i in range(len(seq))] for seq in test_data_sequences_tokens]
y_test = test_data_tags

# Training the CRF model
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)
crf.fit(X_train, y_train)
# Prediction on the test set
y_pred = crf.predict(X_test)

# Flattening the prediction and true labels
y_pred_flat = [item for sublist in y_pred for item in sublist]
y_test_flat = [item for sublist in y_test for item in sublist]

# Evaluating the model performance
print(classification_report(y_test_flat, y_pred_flat, labels=["B-Disease", "I-Disease", "O"], digits=3))
```

```
                  precision    recall  f1-score   support

        B-Disease      0.878     0.741     0.803       960
        I-Disease      0.879     0.745     0.807      1087
                O      0.980     0.994     0.987     22450

         accuracy                          0.973     24497
        macro avg      0.912     0.827     0.866     24497
     weighted avg      0.972     0.973     0.972     24497
```

```
!pip install python-crfsuite
```

```
     Requirement already satisfied: python-crfsuite in /usr/local/lib/python3.10/dist-packages (0.9.9)
```

```
import os
import sys
from sklearn.metrics import classification_report
import pycrfsuite

# Create a Trainer instance
trainer = pycrfsuite.Trainer()

# Add data to the trainer
for x_seq, y_seq in zip(X_train, y_train):
    trainer.append(x_seq, y_seq)

# Set training parameters
trainer.set_params({
    'c1': 0.1,
    'c2': 0.1,
    'max_iterations': 100,
```

```
    'feature.possible_transitions': True,
})

# Redirect stdout to hide the iterations during training
stdout_orig = sys.stdout
sys.stdout = open(os.devnull, 'w')

# Train the CRF model
trainer.train('my_disease_crf_model.crfsuite')

# Restore stdout
sys.stdout = stdout_orig

# Create a Tagger instance and open the trained model
tagger = pycrfsuite.Tagger()
tagger.open('my_disease_crf_model.crfsuite')


# Make predictions on the test set
y_pred = [tagger.tag(x_seq) for x_seq in X_test]

# Flatten the predictions and true labels
y_pred_flat = [item for sublist in y_pred for item in sublist]
y_test_flat = [item for sublist in y_test for item in sublist]

# Evaluate the model's performance
print(classification_report(y_test_flat, y_pred_flat, labels=["B-Disease", "I-Disease", "O"], digits=3))
```

```
                  precision    recall  f1-score   support

       B-Disease      0.878     0.741     0.803       960
       I-Disease      0.879     0.745     0.807      1087
               O      0.980     0.994     0.987     22450

        accuracy                          0.973     24497
       macro avg      0.912     0.827     0.866     24497
    weighted avg      0.972     0.973     0.972     24497
```

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 ("BDisease", "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.

Answer- Here I'm showing weights for my preferred features length and punctuation symbols

```
# To inspect the parameter weights assigned by the trained model, we can use the tagger.info().transitions and tagger.info().state_features attributes.
transitions = tagger.info().transitions
state_features = tagger.info().state_features

# Parameter weights for transitions between tag types
print("Parameter weights for transitions:")
for transition in transitions:
    print(f"From '{transition[0]}' to '{transition[1]}': {transitions[transition]}")

# Parameter weights for the feature designed in Problem 3
print("\nParameter weights for my preferred feature length and punctuation are as follows:")
# Filter and print features with 'w0.length'
for key, value in state_features.items():
    if 'w0.length' in key[0]:
        print(f"feature :{key}: {value}")
    elif 'w0.is_punctuation' in key[0]:
        print(f"feature :{key}: {value}")
```

```
Parameter weights for transitions:
From 'O' to 'O': 1.640437
From 'O' to 'B-Disease': -0.474237
From 'O' to 'I-Disease': -8.170689
From 'B-Disease' to 'O': -2.835766
From 'B-Disease' to 'B-Disease': -5.988195
From 'B-Disease' to 'I-Disease': 1.508452
From 'I-Disease' to 'O': -1.353068
From 'I-Disease' to 'B-Disease': -3.395391
From 'I-Disease' to 'I-Disease': 3.165557

Parameter weights for my preferred feature length and punctuation are as follows:
feature :('w0.length=14', 'O'): -0.413239
feature :('w0.length=14', 'B-Disease'): 0.411352
feature :('w0.length=14', 'I-Disease'): 0.208877
feature :('w0.is_punctuation=False', 'O'): 0.41104
feature :('w0.is_punctuation=False', 'B-Disease'): -0.060381
feature :('w0.is_punctuation=False', 'I-Disease'): -1.078765
feature :('w0.length=2', 'O'): 0.324532
feature :('w0.length=2', 'B-Disease'): -0.526723
feature :('w0.length=2', 'I-Disease'): -0.207495
feature :('w0.length=4', 'O'): 0.378566
feature :('w0.length=4', 'B-Disease'): -0.417988
feature :('w0.length=4', 'I-Disease'): -0.228984
feature :('w0.length=1', 'O'): 1.089097
feature :('w0.length=1', 'B-Disease'): -2.670551
feature :('w0.length=1', 'I-Disease'): 0.099382
feature :('w0.is_punctuation=True', 'O'): 2.680262
feature :('w0.is_punctuation=True', 'I-Disease'): -0.039539
feature :('w0.length=9', 'O'): -0.006646
feature :('w0.length=9', 'B-Disease'): -0.077556
feature :('w0.length=9', 'I-Disease'): 0.07761
feature :('w0.length=3', 'O'): -0.015061
feature :('w0.length=3', 'B-Disease'): 0.573182
feature :('w0.length=3', 'I-Disease'): -0.130034
feature :('w0.length=11', 'O'): -0.025141
feature :('w0.length=11', 'B-Disease'): -0.11614
feature :('w0.length=11', 'I-Disease'): 0.400667
feature :('w0.length=6', 'O'): -0.038101
feature :('w0.length=6', 'B-Disease'): -0.489008
feature :('w0.length=6', 'I-Disease'): 0.042327
```

```
feature :('w0.length=10', 'O'): -0.141241
feature :('w0.length=10', 'B-Disease'): -0.08771
feature :('w0.length=10', 'I-Disease'): -0.001477
feature :('w0.length=7', 'O'): 0.079946
feature :('w0.length=7', 'B-Disease'): -0.486808
feature :('w0.length=7', 'I-Disease'): 0.011503
feature :('w0.length=8', 'O'): 0.057342
feature :('w0.length=8', 'B-Disease'): -0.170386
feature :('w0.length=8', 'I-Disease'): -0.198975
feature :('w0.length=5', 'O'): 0.343632
feature :('w0.length=5', 'B-Disease'): -0.709179
feature :('w0.length=5', 'I-Disease'): -0.08968
feature :('w0.length=12', 'O'): -0.163508
feature :('w0.length=12', 'B-Disease'): -0.207498
feature :('w0.length=12', 'I-Disease'): 0.127266
feature :('w0.length=13', 'O'): -0.306874
feature :('w0.length=13', 'B-Disease'): 0.295211
```

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 one or more disease mentions (at least one "BDisease" 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. Show your results in your notebook.

```
def aggr_to_doc_level(tags):
    # Checking here  if the sequence contains at least one "B-Disease" tag
    return 1 if any(tag.startswith("B-Disease") for tag in tags) else 0

# Apply the aggregation function to true and predicted labels
true_document_labels = [aggr_to_doc_level(doc) for doc in y_test]  # Replace 'y_test' with your true labels
predicted_document_labels = [aggr_to_doc_level(doc) for doc in y_pred]  # Replace 'y_pred' with your predicted labels

# Compute document-level precision and recall
def compute_doc_precision_recall(true_labels, predicted_labels):
    tps = sum(1 for true, predicted in zip(true_labels, predicted_labels) if true == 1 and predicted == 1)
    fps = sum(1 for true, predicted in zip(true_labels, predicted_labels) if true == 0 and predicted == 1)
    fns = sum(1 for true, predicted in zip(true_labels, predicted_labels) if true == 1 and predicted == 0)

    precision = tps / (tps + fps) if tps + fps > 0 else 0
    recall = tps / (tps + fns) if tps + fns > 0 else 0

    return precision, recall

document_precision, document_recall = compute_doc_precision_recall(true_document_labels, predicted_document_labels)

print(f"Document-Level Precision value: {document_precision:.3f}")
print(f"Document-Level Recall value : {document_recall:.3f}")
```

```
Document-Level Precision value: 0.979
Document-Level Recall value : 0.883
```

```
def aggr_to_doc_level(tags):
```

```
    def aggr_to_doc_level(tags):
        # Checking here if the sequence contains at least one "B-Disease" tag or "I-Disease" tag or
        if any(tag.startswith("B-Disease") for tag in tags):
            return 1
        elif any(tag.startswith("I-Disease") for tag in tags):
            return 1
        else:
            return 0

    # Apply the aggregation function to true and predicted labels
    true_document_labels = [aggr_to_doc_level(doc) for doc in y_test]  # Replace 'y_test' with your true labels
    predicted_document_labels = [aggr_to_doc_level(doc) for doc in y_pred]  # Replace 'y_pred' with your predicted labels

    # Compute document-level precision and recall
    document_precision, document_recall = compute_doc_precision_recall(true_document_labels, predicted_document_labels)

    print(f"Document-Level Precision value: {document_precision:.3f}")
    print(f"Document-Level Recall value : {document_recall:.3f}")
```

```
Document-Level Precision value: 0.979
Document-Level Recall value : 0.883
```

observation- Getting the same precison-recall value result in both cases