PROBLEM 1 – Reading the data in CoNLL format

```
import requests

# Define the URLs of the train and test data on GitHub
train_url = 'https://raw.githubusercontent.com/spyysalo/ncbi-
disease/master/conll/train.tsv'
test_url = 'https://raw.githubusercontent.com/spyysalo/ncbi-
disease/master/conll/test.tsv'

# Download the train and test data from GitHub
train_response = requests.get(train_url)
test_response = requests.get(test_url)

# Check if the downloads were successful
if train_response.status_code == 200 and test_response.status_code == 200:
    # Read the data from the response content
    train_data = train_response.text
    test_data = test_response.text

    # Define the function to read CoNLL format
    def read_conll_data(data):
        token_sequences = []
        tag_sequences = []
        tokens = []
        tags = []
        for line in data.split('\n'):
            line = line.strip()
            if not line:
                if tokens and tags:
                    token_sequences.append(tokens)
                    tag_sequences.append(tags)
                tokens = []
                tags = []
            else:
                parts = line.split('\t')
                if len(parts) == 2:
                    token, tag = parts
                    tokens.append(token)
                    tags.append(tag)
        return token_sequences, tag_sequences

    # Apply the function to the train and test data
    train_tokens, train_tags = read_conll_data(train_data)
    test_tokens, test_tags = read_conll_data(test_data)
```

```
    # Show the requested information
    print(f'Number of sequences in train: {len(train_tokens)}')
    print(f'Number of sequences in test: {len(test_tokens)}')

    print("Tokens and tags of the first sequence in the training
dataset:")
    print(train_tokens[0])
    print(train_tags[0])
else:
    print("Failed to download the data from GitHub. Please check the
URLs.")
```

OUTPUT: -

```
Number of sequences in train: 5432
Number of sequences in test: 940
Tokens and tags of the first sequence in the training dataset:
['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: -

```
from collections import Counter

# Count the occurrences of each tag in the training data
tag_counts = Counter(tag for tag_sequence in train_tags for tag in
tag_sequence)

# Count the 20 most common words/tokens associated with "B-Disease" and
"I-Disease" tags
disease_words_counter = Counter()
for tokens, tags in zip(train_tokens, train_tags):
    for token, tag in zip(tokens, tags):
        if tag in ["B-Disease", "I-Disease"]:
            disease_words_counter[token] += 1

# Sort the disease words by frequency in descending order
common_disease_words = disease_words_counter.most_common(20)

# Print the tag counts
print("Tag Counts in Training Data:")
print(tag_counts)
```

```python
# Print the count of the "O" tag
print(f"Count of 'O' tag: {tag_counts['O']}")

# Print the 20 most common disease-related words
print("\n20 Most Common Disease-Related Words:")
for word, count in common_disease_words:
    print(f"{word}: {count}")

# Optional: Print and read a small sample of token sequences
sample_size = 5
print("\nSample of Token Sequences:")
for i in range(sample_size):
    print(train_tokens[i])
    print(train_tags[i])
    print()
```

```
Tag Counts in Training Data:
Counter({'O': 124819, 'I-Disease': 6122, 'B-Disease': 5145})
Count of 'O' tag: 124819

20 Most Common Disease-Related Words:
-: 636
deficiency: 322
syndrome: 281
cancer: 269
disease: 256
of: 178
dystrophy: 176
breast: 151
ovarian: 132
X: 122
and: 120
DM: 120
ALD: 114
DMD: 110
APC: 100
disorder: 94
muscular: 94
G6PD: 92
linked: 81
the: 78
```

```
Sample of Token Sequences:
['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']

['The', 'adenomatous', 'polyposis', 'coli', '(', 'APC', ')', 'tumour', '-', 'suppressor', 'protein', 'controls', 'the', 'Wnt', 'signalling', 'pathway', 'by',
['O', 'B-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',

['Complex', 'formation', 'induces', 'the', 'rapid', 'degradation', 'of', 'betacatenin', '.']
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

['In', 'colon', 'carcinoma', 'cells', ',', 'loss', 'of', 'APC', 'leads', 'to', 'the', 'accumulation', 'of', 'betacatenin', 'in', 'the', 'nucleus', ',', 'wher
['O', 'B-Disease', 'I-Disease', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',

['Here', ',', 'we', 'report', 'the', 'identification', 'and', 'genomic', 'structure', 'of', 'APC', 'homologues', '.']
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
```

PROBLEM 3 – Building features: -

```
def generate_crf_features(tokens, position):
    features = []

    # Get the current word/token in lowercase
    current_word = tokens[position].lower()
    features.append("word=" + current_word)

    # Get the suffix (last 3 characters) of the current word
    suffix = current_word[-3:]
    features.append("suffix=" + suffix)

    # Get the previous word/token or "BOS" if at the beginning of the
sequence
    if position > 0:
        previous_word = tokens[position - 1].lower()
    else:
        previous_word = "BOS"
    features.append("prev_word=" + previous_word)

    # Get the next word/token or "EOS" if at the end of the sequence
    if position < len(tokens) - 1:
        next_word = tokens[position + 1].lower()
    else:
        next_word = "EOS"
    features.append("next_word=" + next_word)

    # Add at least one other feature of your choice
    # For example, you can add the length of the current word
    word_length = len(current_word)
    features.append("word_length=" + str(word_length))

    return features
# Assuming train_tokens and test_tokens are your sequences of tokens
train_features = []
test_features = []
```

```
# Generate features for the training set
for i in range(len(train_tokens[0])):
    features = generate_crf_features(train_tokens[0], i)
    train_features.append(features)

# Generate features for the test set
for i in range(len(test_tokens[0])):
    features = generate_crf_features(test_tokens[0], i)
    test_features.append(features)
for i in range(3):
    features = generate_crf_features(train_tokens[0], i)
    print(f"Features for word {i + 1}: {features}")
```

```
Features for word 1: ['word=identification', 'suffix=ion', 'prev_word=BOS', 'next_word=of', 'word_length=14']
Features for word 2: ['word=of', 'suffix=of', 'prev_word=identification', 'next_word=apc2', 'word_length=2']
Features for word 3: ['word=apc2', 'suffix=pc2', 'prev_word=of', 'next_word=,', 'word_length=4']
```

PROBLEM 4 – Training a CRF model: -

```
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)
```

```
import pycrfsuite
from sklearn.metrics import classification_report

# Define the function to generate CRF features
def generate_crf_features(tokens, position):
    features = []

    # Get the current word/token in lowercase
    current_word = tokens[position].lower()
    features.append("word=" + current_word)

    # Get the suffix (last 3 characters) of the current word
    suffix = current_word[-3:]
    features.append("suffix=" + suffix)

    # Get the previous word/token or "BOS" if at the beginning of the
sequence
    if position > 0:
        previous_word = tokens[position - 1].lower()
    else:
```

```
        previous_word = "BOS"
    features.append("prev_word=" + previous_word)

    # Get the next word/token or "EOS" if at the end of the sequence
    if position < len(tokens) - 1:
        next_word = tokens[position + 1].lower()
    else:
        next_word = "EOS"
    features.append("next_word=" + next_word)

    # Add at least one other feature of your choice
    # For example, you can add the length of the current word
    word_length = len(current_word)
    features.append("word_length=" + str(word_length))

    return features

# Prepare training and test data
train_data = []  # Training data in the required format

for i in range(len(train_tokens)):
    sequence = []
    for j in range(len(train_tokens[i])):
        features = generate_crf_features(train_tokens[i], j)
        label = train_tags[i][j]
        sequence.append((features, label))
    train_data.append(sequence)

test_data = []  # Test data in the required format

for i in range(len(test_tokens)):
    sequence = []
    for j in range(len(test_tokens[i])):
        features = generate_crf_features(test_tokens[i], j)
        label = test_tags[i][j]
        sequence.append((features, label))
    test_data.append(sequence)

# Train a CRF model
trainer = pycrfsuite.Trainer(verbose=True)

for sequence in train_data:
    features, labels = zip(*sequence)
    trainer.append(features, labels)
```

```
trainer.set_params({
    'c1': 1.0,   # Coefficient for L1 penalty
    'c2': 1e-3,  # Coefficient for L2 penalty
    'max_iterations': 50,  # Maximum number of iterations
    'feature.possible_transitions': True  # Include transitions that are
possible but not observed in the training data
})

model_filename = 'disease_ner_model.crfsuite'
trainer.train(model_filename)

# Apply the trained model to the test data
tagger = pycrfsuite.Tagger()
tagger.open(model_filename)

true_labels = []
predicted_labels = []

for sequence in test_data:
    features, labels = zip(*sequence)
    true_labels.extend(labels)
    predicted_labels.extend(tagger.tag(features))

# Compute precision, recall, and f1-score
report = classification_report(true_labels, predicted_labels,
target_names=["B-Disease", "I-Disease", "O"])
print(report)
```

OUTPUT: -

```
Loss: 6002.229634
Feature norm: 69.473446
Error norm: 120.006403
Active features: 3220
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.044

***** Iteration #48 *****
Loss: 5993.886385
Feature norm: 70.002899
Error norm: 171.255091
Active features: 3173
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.043

***** Iteration #49 *****
Loss: 5986.723460
Feature norm: 70.268821
Error norm: 162.145716
Active features: 3129
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.044

***** Iteration #50 *****
Loss: 5979.140622
Feature norm: 70.797813
Error norm: 144.976701
Active features: 3102
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 0.048

L-BFGS terminated with the maximum number of iterations
Total seconds required for training: 2.380
```

```
Storing the model
Number of active features: 3102 (31427)
Number of active attributes: 2139 (27706)
Number of active labels: 3 (3)
Writing labels
Writing attributes
Writing feature references for transitions
Writing feature references for attributes
Seconds required: 0.003
```

	precision	recall	f1-score	support
B-Disease	0.86	0.70	0.77	960
I-Disease	0.84	0.74	0.79	1087
O	0.98	0.99	0.98	22450
accuracy			0.97	24497
macro avg	0.89	0.81	0.85	24497
weighted avg	0.97	0.97	0.97	24497

PROBLEM 5 – Inspecting the trained model

```
# Show parameter weights for transitions between tag types
transition_weights = tagger.info().transitions
print("Parameter weights for transitions between tag types:")
for label_from, label_to in transition_weights:
    weight = transition_weights[(label_from, label_to)]
    print(f"Transition: {label_from} -> {label_to}, Weight: {weight}")
```

```
# Show parameter weights for the "word_length" feature
state_features = tagger.info().state_features
word_length_weights = {feature: weight for feature, weight in
state_features.items() if "word_length" in feature}
print("\nParameter weights for the 'word_length' feature:")
for feature, weight in word_length_weights.items():
    print(f"Feature: {feature}, Weight: {weight}")

# Corrected feature name
my_feature_name = 'word_length'  # Feature

# Display parameter weights for the specified feature
feature_weights = tagger.info().state_features
print(f"Parameter weights for feature '{my_feature_name}':")
for feature, weight in feature_weights.items():
    feature_str = ' '.join(feature)  # Convert the feature tuple to a
string
    if feature_str.startswith(my_feature_name):
        print(f"{feature_str}: {weight}")
```

OUTPUT: -

```
Parameter weights for transitions between tag types:
Transition: O -> O, Weight: 1.952336
Transition: O -> B-Disease, Weight: -0.455422
Transition: O -> I-Disease, Weight: -8.443748
Transition: B-Disease -> O, Weight: -1.592283
Transition: B-Disease -> B-Disease, Weight: -5.688209
Transition: B-Disease -> I-Disease, Weight: 1.937948
Transition: I-Disease -> O, Weight: -1.72059
Transition: I-Disease -> B-Disease, Weight: -4.353711
Transition: I-Disease -> I-Disease, Weight: 1.911456

Parameter weights for the 'word_length' feature:
Parameter weights for feature 'word_length':
word_length=14 O: -0.075894
word_length=14 B-Disease: 0.156378
word_length=2 O: 0.829588
word_length=2 B-Disease: -0.771092
word_length=2 I-Disease: -0.721985
word_length=4 O: 0.820452
word_length=4 B-Disease: -0.345401
word_length=4 I-Disease: -0.477079
word_length=1 O: 1.774242
word_length=1 B-Disease: -2.535428
word_length=1 I-Disease: 0.000515
word_length=9 O: 0.075153
word_length=9 B-Disease: -0.076107
word_length=9 I-Disease: -0.177396
word_length=3 O: 0.1825
word_length=3 B-Disease: -0.010764
```

```
word_length=3 I-Disease: -0.718703
word_length=11 O: 0.103558
word_length=11 B-Disease: -0.240302
word_length=11 I-Disease: -0.058577
word_length=6 O: 0.261589
word_length=6 B-Disease: -0.467804
word_length=6 I-Disease: -0.260238
word_length=10 O: 0.20253
word_length=10 B-Disease: -0.068965
word_length=10 I-Disease: -0.265982
word_length=7 O: 0.485039
word_length=7 B-Disease: -0.245902
word_length=7 I-Disease: -0.099405
word_length=8 O: 0.548168
word_length=8 B-Disease: 0.023715
word_length=8 I-Disease: -0.097213
word_length=5 O: 0.781591
word_length=5 B-Disease: -0.40354
word_length=5 I-Disease: 0.004052
word_length=12 O: 0.243878
word_length=13 O: 0.014378
word_length=13 B-Disease: 0.163965
word_length=13 I-Disease: -0.145113
word_length=15 O: -0.557724
word_length=15 B-Disease: 0.606896
word_length=15 I-Disease: -0.046079
word_length=16 O: -0.992215
word_length=16 B-Disease: 0.604835
word_length=17 O: -0.276114
```

PROBLEM 6 – Document level performance:-

```
def aggregate_labels(tag_sequence):
    # Check if the tag sequence contains at least one "B-Disease" tag
    return 1 if "B-Disease" in tag_sequence else 0

# Apply the aggregation function to the true and predicted test labels
true_document_labels = [aggregate_labels(tag_sequence) for tag_sequence in
test_tags]
```

```
predicted_document_labels = [aggregate_labels(tag_sequence) for
tag_sequence in y_pred]

# Calculate document-level precision and recall
true_positive = sum(1 for true_label, predicted_label in
zip(true_document_labels, predicted_document_labels) if true_label == 1
and predicted_label == 1)
false_positive = sum(1 for true_label, predicted_label in
zip(true_document_labels, predicted_document_labels) if true_label == 0
and predicted_label == 1)
false_negative = sum(1 for true_label, predicted_label in
zip(true_document_labels, predicted_document_labels) if true_label == 1
and predicted_label == 0)

document_precision = true_positive / (true_positive + false_positive)
document_recall = true_positive / (true_positive + false_negative)

# Print document-level precision and recall
print(f"Document-level Precision: {document_precision:.2f}")
print(f"Document-level Recall: {document_recall:.2f}")
```

OUTPUT: -

```
Document-level Precision: 1.00
Document-level Recall: 1.00
```

PROBLEM 7 – State Transitions: -

The "feature.possible_transitions" parameter in python-crfsuite controls whether the model is allowed to output tag-to-tag transitions that were never seen in the training data.

An example of a tag-to-tag transition that never occurred in the training data could be the transition from "B-Disease" to "I-Disease" for a specific pair of disease entities that were not present in the training data.

Whether it makes sense to set this parameter to True or False depends on the specific task and the nature of the data:

Setting "feature.possible_transitions" to True:

Advantages: Allowing possible transitions that were not seen in the training data can make the model more flexible and potentially better at handling unseen or rare tag transitions.
Disadvantages: It can also introduce noise if the model generates unlikely transitions that do not make sense in the context of the application.
Setting "feature.possible_transitions" to False:

Advantages: Restricting the model to only output transitions observed in the training data can lead to more conservative and reliable predictions. It prevents the model from making unrealistic tag transitions.
Disadvantages: It may limit the model's ability to generalize to unseen cases or adapt to slight variations in the data.
In the context of named entity recognition (NER) tasks like identifying diseases, it might make sense to set "feature.possible_transitions" to True if the dataset is limited and you expect variations in tag transitions. However, you should carefully evaluate the impact on model performance and possibly use techniques like cross-validation to make an informed decision.

Ultimately, the choice of whether to allow possible transitions depends on the trade-off between model flexibility and the risk of introducing noise, and it should be made based on the specific requirements and characteristics of the task and data.