Import Files

In [1]:
```
!git clone https://github.com/spyysalo/ncbi-disease.git
```

```
fatal: destination path 'ncbi-disease' already exists and is not an empty directory.
```

Importing Libraries

In [2]:
```
import os
import pandas as pd
from collections import Counter
import pycrfsuite
from sklearn.metrics import classification_report
from itertools import chain
```

Problem 1 – Reading the data in CoNLL format

In [3]:
```
def readConllFile(file_path):
    tokens = []
    tags = []

    with open(file_path, 'r') as file:
        currentTokens = []
        currentTags = []
        for line in file:
            line = line.strip()
            if not line:
                tokens.append(currentTokens)
                tags.append(currentTags)
                currentTokens = []
                currentTags = []
            else:
                parts = line.split('\t')
                if len(parts) == 2:
                    token, tag = parts
                    currentTokens.append(token)
                    currentTags.append(tag)
    return tokens, tags
```

In [4]:
```
trainFilePath = "ncbi-disease/conll/train.tsv"
testFilePath = "ncbi-disease/conll/test.tsv"

trainTokens, trainTags = readConllFile(trainFilePath)
testTokens, testTags = readConllFile(testFilePath)

numberTrainSequences = len(trainTokens)
numberTestSequences = len(testTokens)

print("Number of sequences in train:", numberTrainSequences)
print("Number of sequences in test:", numberTestSequences)

print("Train Tokens:", trainTokens[0])
print("Train Tags:", trainTags[0])
print("Test Tokens:", testTokens[0])
print("Test Tags:", testTags[0])
```

```
Number of sequences in train: 5432
Number of sequences in test: 940
Train Tokens: ['Identification', 'of', 'APC2', ',', 'a', 'homologue', 'of', 'the', 'adenomatous', 'polyposis', 'coli', 'tumour', 'suppressor', '.']
Train Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'O', 'O']
Test Tokens: ['Clustering', 'of', 'missense', 'mutations', 'in', 'the', 'ataxia', '-', 'telangiectasia', 'gene', 'in', 'a', 'sporadic', 'T', '-', 'cell', 'l
eukaemia', '.']
Test Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'O', 'O', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'I-Disea
se', 'O']
```

Problem 2 - Data Discovery

In [5]:
```
tagCount = Counter(tag for tagSequence in trainTags for tag in tagSequence)

print("Tag Counts in Training Data:")
for tag, count in tagCount.items():
    print(f"{tag}: {count}")

wordCount = Counter()
for i in range(len(trainTokens)):
    tokenSequence = trainTokens[i]
    tagSequence = trainTags[i]
    for j in range(len(tokenSequence)):
        if tagSequence[j] in ["B-Disease", "I-Disease"]:
            wordCount[tokenSequence[j]] += 1

top20Words = wordCount.most_common(20)

print("\nTop 20 Words Associated with Disease Tags:")
for word, count in top20Words:
    print(f"{word}: {count}")
```

```
Tag Counts in Training Data:
O: 124819
B-Disease: 5145
I-Disease: 6122

Top 20 Words Associated with Disease Tags:
-: 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
```

Problem 3 - Building features

In [6]:
```
def wordToFeatures(tokens, position):
    features = []
    if position >= len(tokens):
        return features

    currentWord = tokens[position].lower()

    suffix = currentWord[-3:]

    previousWord = tokens[position - 1] if position > 0 else "BOS"
    nextWord = tokens[position + 1] if position < len(tokens) - 1 else "EOS"

    wordPrefix = currentWord[:3]

    if currentWord.islower():
        wordShape = "lowercase"
    elif currentWord.isupper():
        wordShape = "uppercase"
    else:
        wordShape = "mixedcase"

    hasHyphen = '-' in currentWord

    char2grams = [currentWord[i:i+2] for i in range(len(currentWord) - 1)]
    char3grams = [currentWord[i:i+3] for i in range(len(currentWord) - 2)]

    features.extend([
        'w0.lower=' + currentWord,
        'w0.suffix3=' + suffix,
        'w-1.word=' + previousWord,
        'w+1.word=' + nextWord,
        'wordPrefix=' + wordPrefix,
        'wordShape=' + wordShape,
        'hasHyphen=' + str(hasHyphen),
    ])

    features.extend(['char2gram=' + gram for gram in char2grams])
    features.extend(['char3gram=' + gram for gram in char3grams])

    return features

for i in range(3):
    features = wordToFeatures(trainTokens[0], i)
    print(features)
```

```
['w0.lower=identification', 'w0.suffix3=ion', 'w-1.word=BOS', 'w+1.word=of', 'wordPrefix=ide', 'wordShape=lowercase', 'hasHyphen=False', 'char2gram=id', 'ch
ar2gram=de', 'char2gram=en', 'char2gram=nt', 'char2gram=ti', 'char2gram=if', 'char2gram=fi', 'char2gram=ic', 'char2gram=ca', 'char2gram=at', 'char2gram=ti',
'char2gram=io', 'char2gram=on', 'char3gram=ide', 'char3gram=den', 'char3gram=ent', 'char3gram=nti', 'char3gram=tif', 'char3gram=ifi', 'char3gram=fic', 'char
3gram=ica', 'char3gram=cat', 'char3gram=ati', 'char3gram=tio', 'char3gram=ion']
['w0.lower=of', 'w0.suffix3=of', 'w-1.word=Identification', 'w+1.word=APC2', 'wordPrefix=of', 'wordShape=lowercase', 'hasHyphen=False', 'char2gram=of']
['w0.lower=apc2', 'w0.suffix3=pc2', 'w-1.word=of', 'w+1.word=,', 'wordPrefix=apc', 'wordShape=lowercase', 'hasHyphen=False', 'char2gram=ap', 'char2gram=pc',
'char2gram=c2', 'char3gram=apc', 'char3gram=pc2']
```

Problem 4 - Training a CRF model

In [7]:
```
def dataToFeatures(tokens, feature_function):
    return [feature_function(tokens, i) for i in range(len(tokens))]

trainFeatures = [dataToFeatures(tokens, wordToFeatures) for tokens in trainTokens]
testFeatures = [dataToFeatures(tokens, wordToFeatures) for tokens in testTokens]

flatTrainTags = list(chain.from_iterable(trainTags))
flatTestTags = list(chain.from_iterable(testTags))

trainer = pycrfsuite.Trainer(verbose=False)
for xseq, yseq in zip(trainFeatures, trainTags):
    trainer.append(xseq, yseq)
trainer.set_params({
    'c1': 1.0,   # L1 penalty
    'c2': 1e-3,  # L2 penalty
    'max_iterations': 50,
    'feature.possible_transitions': True
})
trainer.train('conll2002-esp.crfsuite')

tagger = pycrfsuite.Tagger()
tagger.open('conll2002-esp.crfsuite')

predictedTags = [tagger.tag(xseq) for xseq in testFeatures]
flattenPredictedTags = list(chain.from_iterable(predictedTags))  # Flatten the list of lists

targetNames = ["B-Disease", "I-Disease", "O"]
report = classification_report(flatTestTags, flattenPredictedTags, target_names=targetNames)

print(report)
```

```
              precision    recall  f1-score   support

   B-Disease       0.86      0.72      0.79       960
   I-Disease       0.84      0.76      0.80      1087
           O       0.98      0.99      0.99     22450

    accuracy                           0.97     24497
   macro avg       0.89      0.83      0.86     24497
weighted avg       0.97      0.97      0.97     24497
```

Problem 5 - Inspecting the trained model

In [8]:
```
def displayFeatureWeights(model_path):
    tagger = pycrfsuite.Tagger()
    tagger.open(model_path)

    featureWeights = tagger.info().state_features
    for feature, weight in featureWeights.items():
        print(f"Feature: {feature} | Weight: {weight}")

model_path = 'conll2002-esp.crfsuite'  # Replace with the path to your CRF model
#displayFeatureWeights(model_path)
```

In [9]:
```
feature_names = ["hasHyphen"]
state_features = tagger.info().state_features
print(f"Feature Weights for '{feature_names[0]}' in the model:")
for feature_name in feature_names:
    counter = 0
    for feature, weight in state_features.items():
        if feature_name in feature[0].split("="):
            print(f"Feature: {feature} | Weight: {weight}")
```

```
Feature Weights for 'hasHyphen' in the model:
Feature: ('hasHyphen=False', 'O') | Weight: 0.189586
Feature: ('hasHyphen=False', 'B-Disease') | Weight: -0.267843
Feature: ('hasHyphen=False', 'I-Disease') | Weight: -0.15311
Feature: ('hasHyphen=True', 'O') | Weight: 0.18216
Feature: ('hasHyphen=True', 'I-Disease') | Weight: 0.526131
```

Problem 6 - Document level performance

In [10]:
```
def tokens_to_document_labels(token_tags):
    return 1 if "B-Disease" in token_tags else 0

#print("Predicted Tags:", predictedTags[:5])
#print("Test Tags:", testTags[:5])

trueDocumentLabels = [tokens_to_document_labels(tags) for tags in testTags]
#print("True: ", trueDocumentLabels)

predictedDocumentLabels = [tokens_to_document_labels(tags) for tags in predictedTags]
#print("Predictions: ", predictedDocumentLabels)

truePositive = sum(1 for true_label, predicted_label in zip(trueDocumentLabels, predictedDocumentLabels) if true_label == 1 and predicted_label == 1)
falsePositive = sum(1 for true_label, predicted_label in zip(trueDocumentLabels, predictedDocumentLabels) if true_label == 0 and predicted_label == 1)
falseNegative = sum(1 for true_label, predicted_label in zip(trueDocumentLabels, predictedDocumentLabels) if true_label == 1 and predicted_label == 0)

precision = truePositive / (truePositive + falsePositive) if truePositive + falsePositive > 0 else 0
recall = truePositive / (truePositive + falseNegative) if truePositive + falseNegative > 0 else 0

print("True Positive: ", truePositive)
print("False Positive: ", falsePositive)
print("False Negative: ", falseNegative)
print("Document-Level Precision:", precision)
print("Document-Level Recall:", recall)
```

```
True Positive:  476
False Positive:  16
False Negative:  63
Document-Level Precision: 0.9674796747967548
Document-Level Recall: 0.8831168831168831
```

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.

Answer: An example of a tag-to-tag transition that is absent in the training data is the transition from "O" to "I-Disease." In the tagging scheme for the data, "B" denotes the first word/token in a disease mention, "I" indicates subsequent words/tokens within the same mention, and "O" signifies that a token is not part of a disease mention. This means that "I-Disease" can only occur after "B-Disease," and shows that the transition from "O" to "I-Disease" is never observed. Setting the parameter to True or False depends on the characteristics of your training data. If the training data is comprehensive and covers most tag transitions, it would be best to set the parameter to False. But, if there is a limited amount of training data or unseen transitions, it would be best to set the parameter to True. Because the data is from NBCI Disease Corpus, we can assume that the data is in fact mostly correct and we will not run into a lot of unseen transitions, because of this and because our dataset is comprehensive and cover most tag transitions, I believe it would be beest to set the parameter to false.