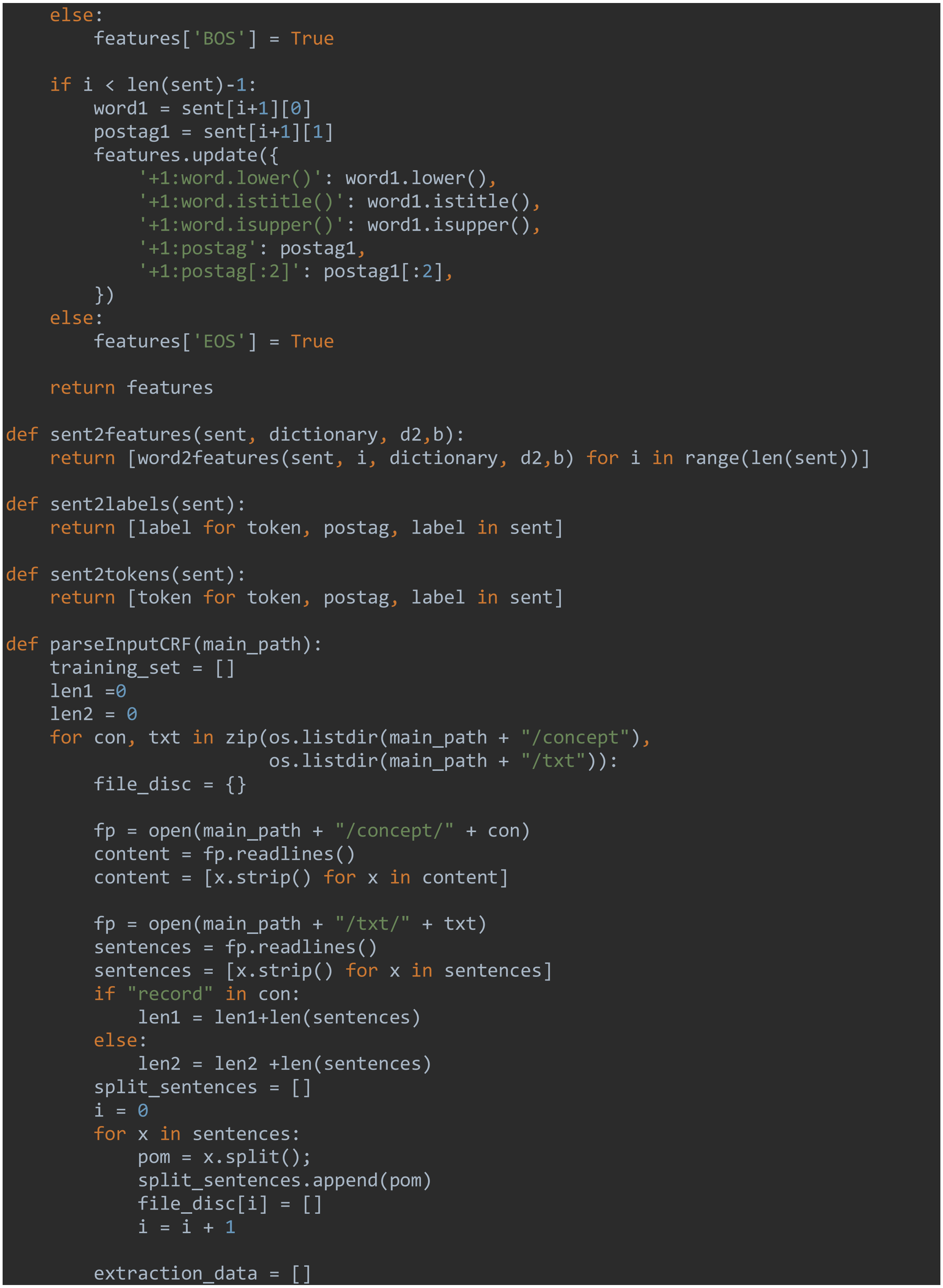
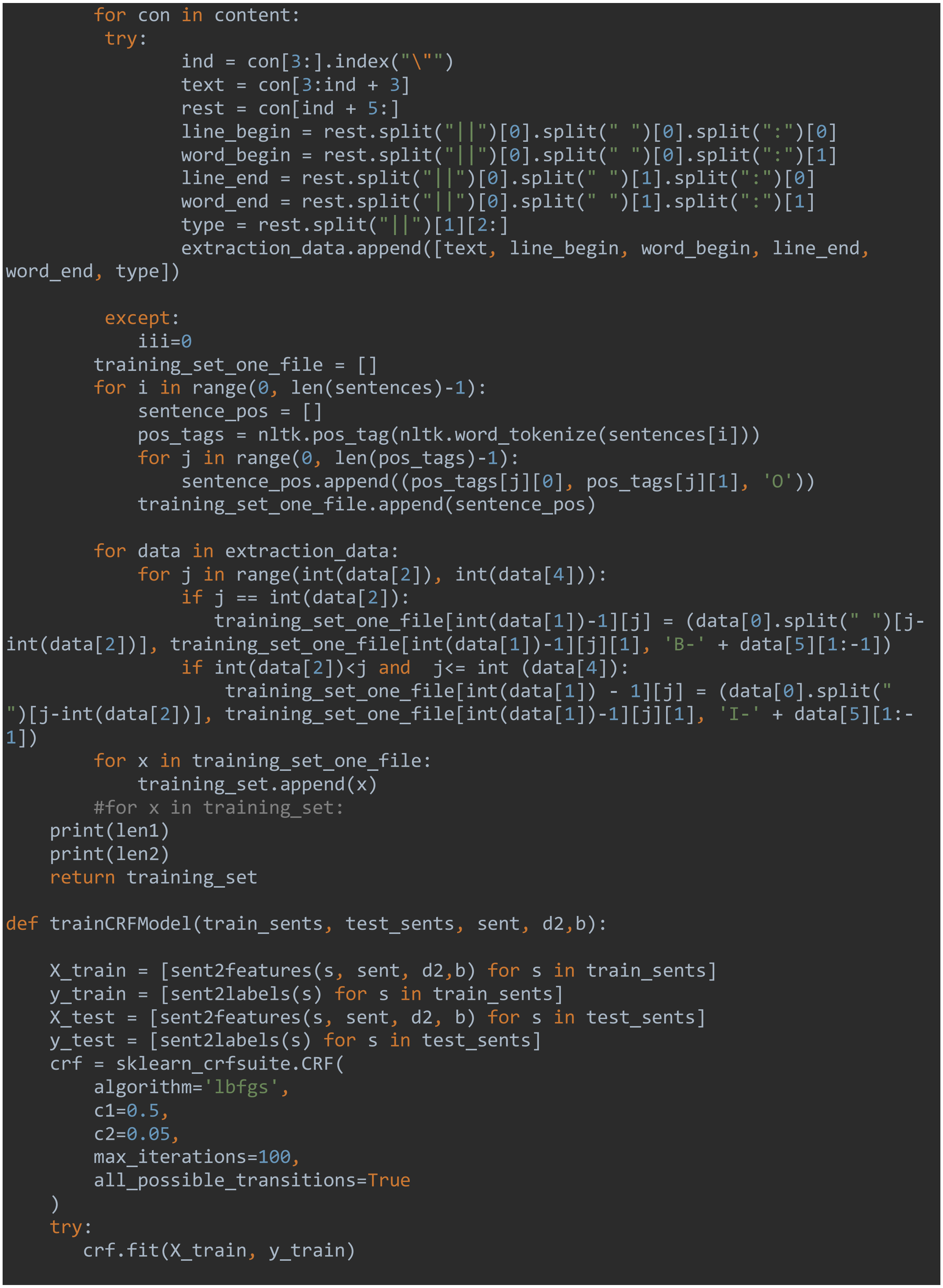
CRFModel.py

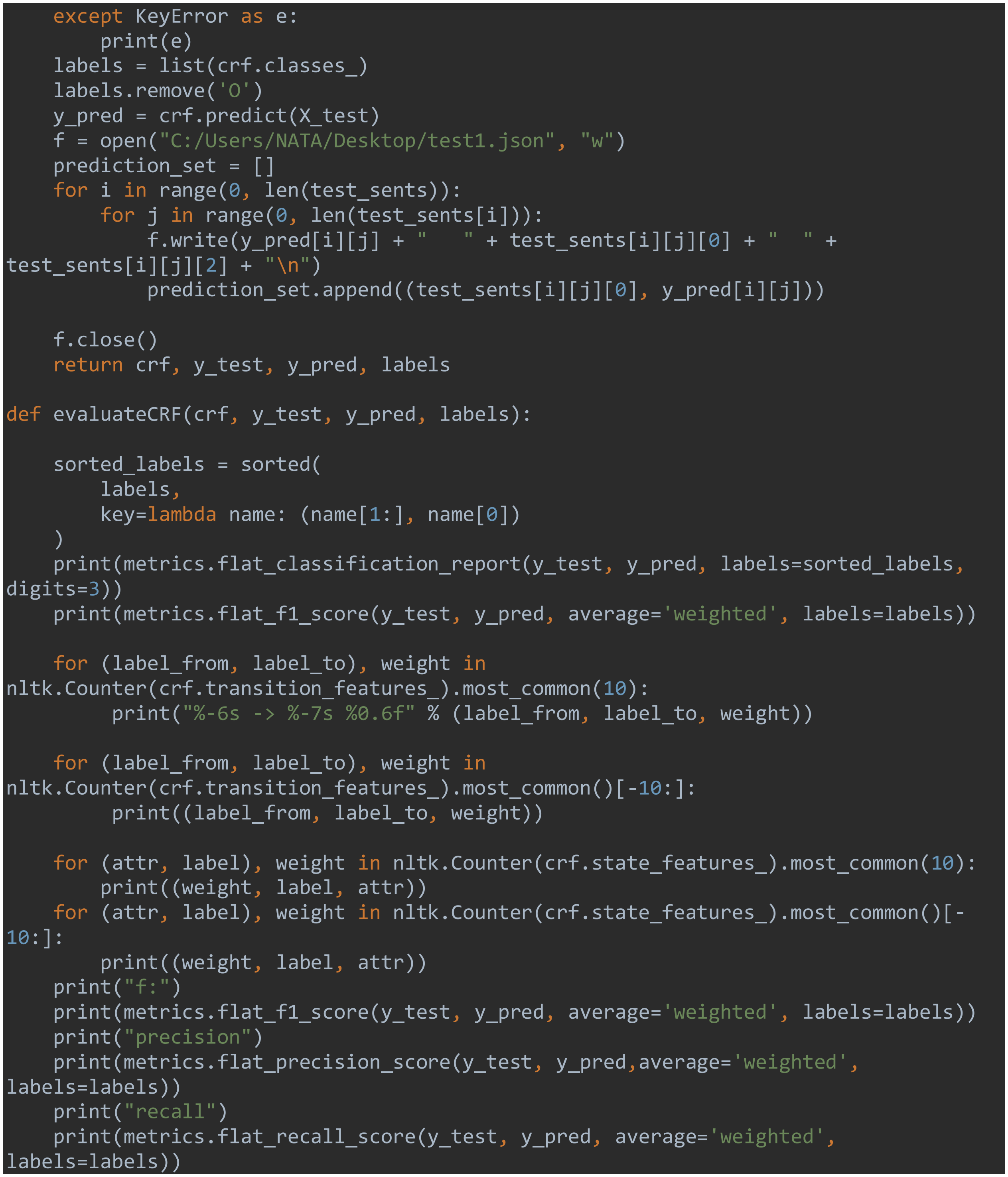
```
import os
import nltk
import scipy
import sklearn_crfsuite
from sklearn.metrics import make_scorer
from sklearn.model_selection import RandomizedSearchCV
from sklearn_crfsuite import metrics
def transformLexicon():
    fp = open("C:/Users/NATA/Desktop/master rad/New folder/text mining/DM.DB.txt")
    fp1 = open("C:/Users/NATA/Desktop/recnik.txt", "a")
    content = fp.readlines()
    for x in content:
        fp1.write(x.split("|")[0].lower()+"\n"+x.split("|")[3].lower()+"\n")

def word2features(sent, i, dictionary, d2, b):

    word = sent[i][0]
    postag = sent[i][1]
    rt = False
    stemmer = nltk.SnowballStemmer('english')
    root = stemmer.stem(word)
    if root in dictionary:
        rt = True
    lex = False
    if word.lower() in d2:
         lex = True

    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word[-3:]': word[-3:],
        'word[-2:]': word[-2:],
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
        'postag': postag,
        'postag[:2]': postag[:2],

    }
    if b and lex :
        features.update({
            'dics': lex,
        })
    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
            '-1:postag': postag1,
            '-1:postag[:2]': postag1[:2],
        })
```



```
    else:
        features['BOS'] = True

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
            '+1:postag': postag1,
            '+1:postag[:2]': postag1[:2],
        })
    else:
        features['EOS'] = True

    return features

def sent2features(sent, dictionary, d2,b):
    return [word2features(sent, i, dictionary, d2,b) for i in range(len(sent))]

def sent2labels(sent):
    return [label for token, postag, label in sent]

def sent2tokens(sent):
    return [token for token, postag, label in sent]

def parseInputCRF(main_path):
    training_set = []
    len1 =0
    len2 = 0
    for con, txt in zip(os.listdir(main_path + "/concept"),
                        os.listdir(main_path + "/txt")):
        file_disc = {}

        fp = open(main_path + "/concept/" + con)
        content = fp.readlines()
        content = [x.strip() for x in content]

        fp = open(main_path + "/txt/" + txt)
        sentences = fp.readlines()
        sentences = [x.strip() for x in sentences]
        if "record" in con:
            len1 = len1+len(sentences)
        else:
            len2 = len2 +len(sentences)
        split_sentences = []
        i = 0
        for x in sentences:
            pom = x.split();
            split_sentences.append(pom)
            file_disc[i] = []
            i = i + 1

        extraction_data = []
```



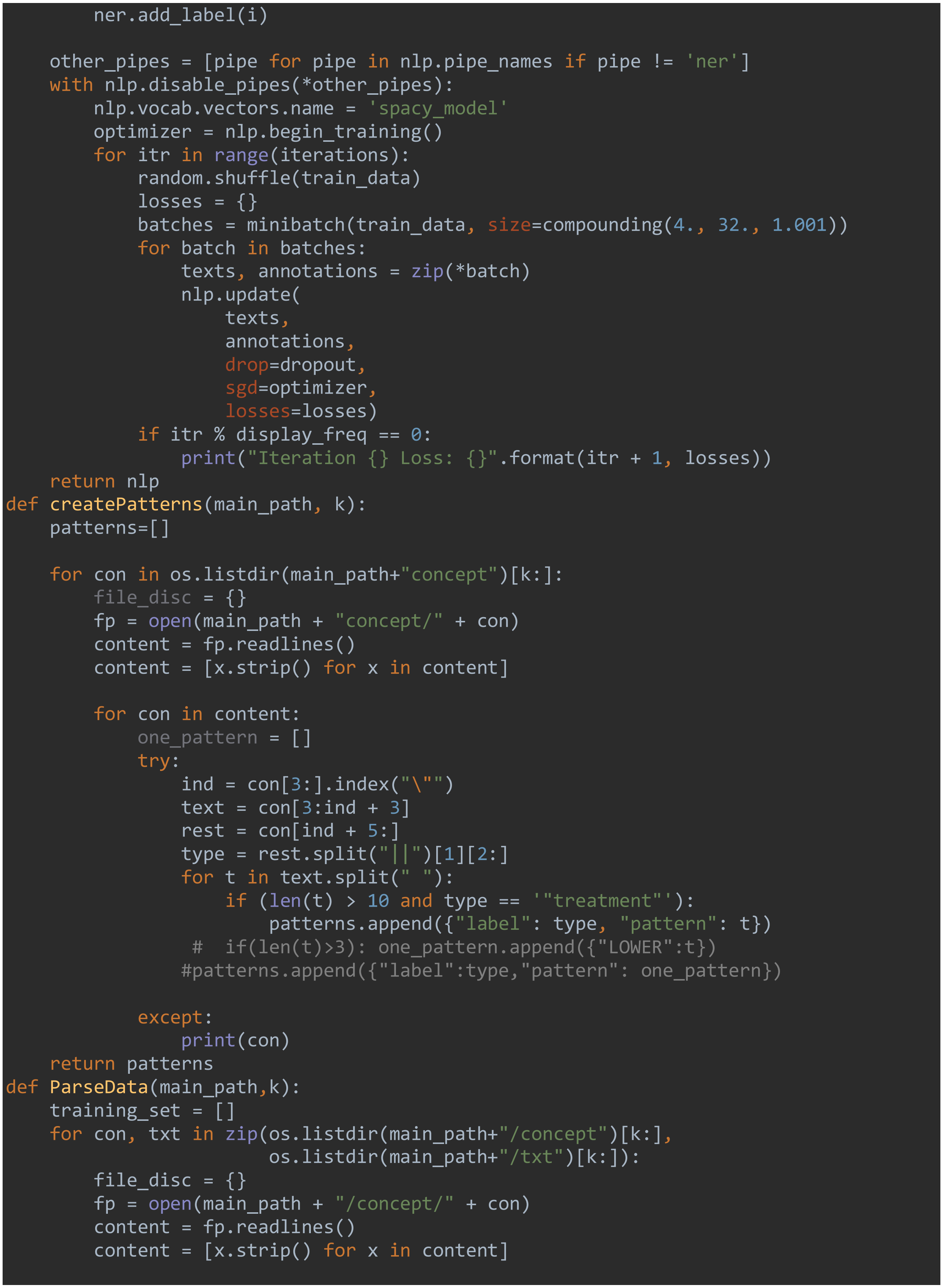
```
        for con in content:
         try:
                ind = con[3:].index("\"")
                text = con[3:ind + 3]
                rest = con[ind + 5:]
                line_begin = rest.split("||")[0].split(" ")[0].split(":")[0]
                word_begin = rest.split("||")[0].split(" ")[0].split(":")[1]
                line_end = rest.split("||")[0].split(" ")[1].split(":")[0]
                word_end = rest.split("||")[0].split(" ")[1].split(":")[1]
                type = rest.split("||")[1][2:]
                extraction_data.append([text, line_begin, word_begin, line_end,
word_end, type])

         except:
            iii=0
        training_set_one_file = []
        for i in range(0, len(sentences)-1):
            sentence_pos = []
            pos_tags = nltk.pos_tag(nltk.word_tokenize(sentences[i]))
            for j in range(0, len(pos_tags)-1):
                sentence_pos.append((pos_tags[j][0], pos_tags[j][1], 'O'))
            training_set_one_file.append(sentence_pos)

        for data in extraction_data:
            for j in range(int(data[2]), int(data[4])):
                if j == int(data[2]):
                   training_set_one_file[int(data[1])-1][j] = (data[0].split(" ")[j-
int(data[2])], training_set_one_file[int(data[1])-1][j][1], 'B-' + data[5][1:-1])
                if int(data[2])<j and  j<= int (data[4]):
                    training_set_one_file[int(data[1]) - 1][j] = (data[0].split("
")[j-int(data[2])], training_set_one_file[int(data[1])-1][j][1], 'I-' + data[5][1:-
1])
        for x in training_set_one_file:
            training_set.append(x)
        #for x in training_set:
    print(len1)
    print(len2)
    return training_set

def trainCRFModel(train_sents, test_sents, sent, d2,b):

    X_train = [sent2features(s, sent, d2,b) for s in train_sents]
    y_train = [sent2labels(s) for s in train_sents]
    X_test = [sent2features(s, sent, d2, b) for s in test_sents]
    y_test = [sent2labels(s) for s in test_sents]
    crf = sklearn_crfsuite.CRF(
        algorithm='lbfgs',
        c1=0.5,
        c2=0.05,
        max_iterations=100,
        all_possible_transitions=True
    )
    try:
       crf.fit(X_train, y_train)
```



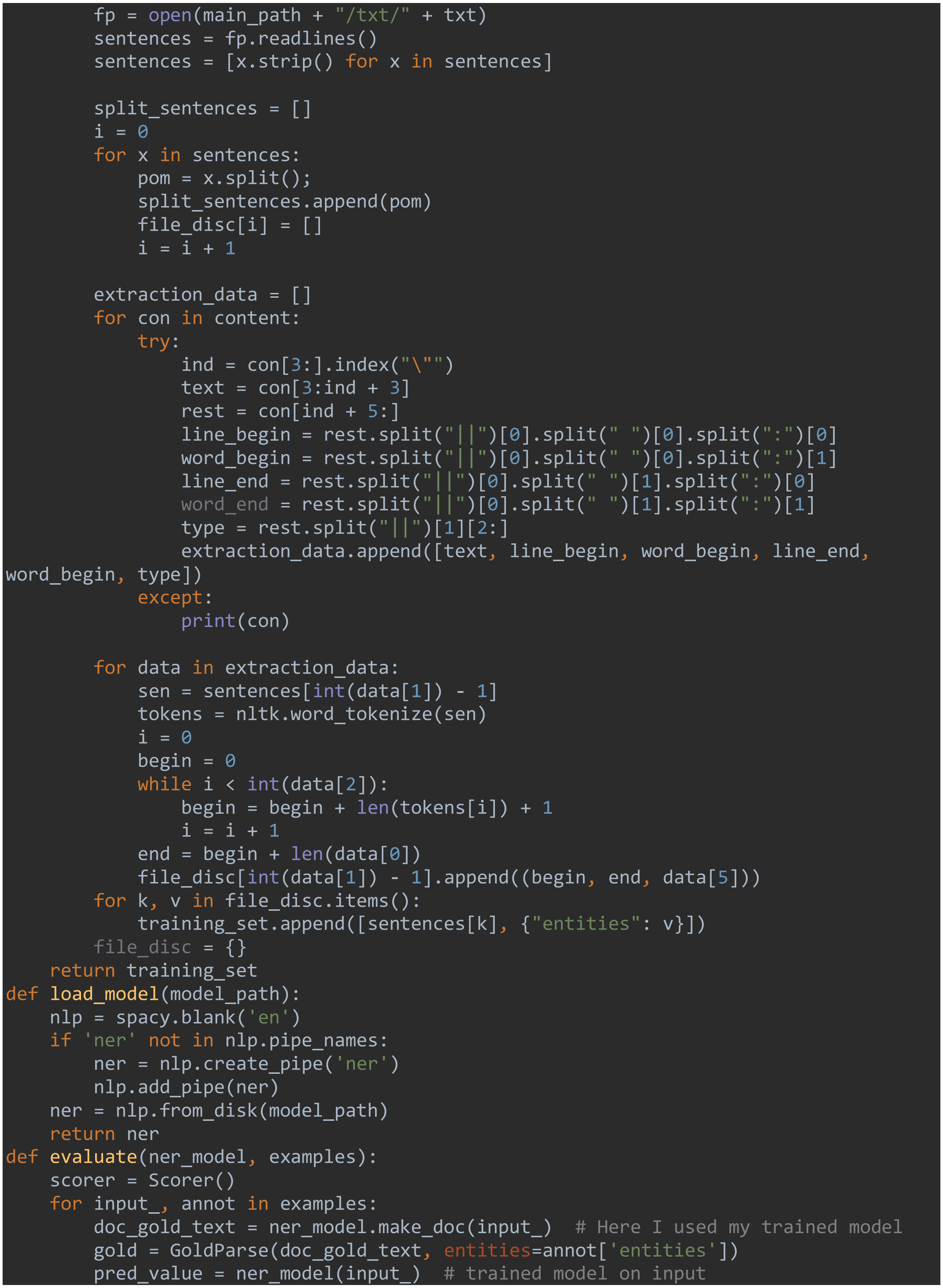
```
    except KeyError as e:
        print(e)
    labels = list(crf.classes_)
    labels.remove('O')
    y_pred = crf.predict(X_test)
    f = open("C:/Users/NATA/Desktop/test1.json", "w")
    prediction_set = []
    for i in range(0, len(test_sents)):
        for j in range(0, len(test_sents[i])):
            f.write(y_pred[i][j] + "   " + test_sents[i][j][0] + "  " +
test_sents[i][j][2] + "\n")
            prediction_set.append((test_sents[i][j][0], y_pred[i][j]))

    f.close()
    return crf, y_test, y_pred, labels

def evaluateCRF(crf, y_test, y_pred, labels):

    sorted_labels = sorted(
        labels,
        key=lambda name: (name[1:], name[0])
    )
    print(metrics.flat_classification_report(y_test, y_pred, labels=sorted_labels,
digits=3))
    print(metrics.flat_f1_score(y_test, y_pred, average='weighted', labels=labels))

    for (label_from, label_to), weight in
nltk.Counter(crf.transition_features_).most_common(10):
         print("%-6s -> %-7s %0.6f" % (label_from, label_to, weight))

    for (label_from, label_to), weight in
nltk.Counter(crf.transition_features_).most_common()[-10:]:
         print((label_from, label_to, weight))

    for (attr, label), weight in nltk.Counter(crf.state_features_).most_common(10):
        print((weight, label, attr))
    for (attr, label), weight in nltk.Counter(crf.state_features_).most_common()[-
10:]:
        print((weight, label, attr))
    print("f:")
    print(metrics.flat_f1_score(y_test, y_pred, average='weighted', labels=labels))
    print("precision")
    print(metrics.flat_precision_score(y_test, y_pred,average='weighted',
labels=labels))
    print("recall")
    print(metrics.flat_recall_score(y_test, y_pred, average='weighted',
labels=labels))
```

SpacyModel.py

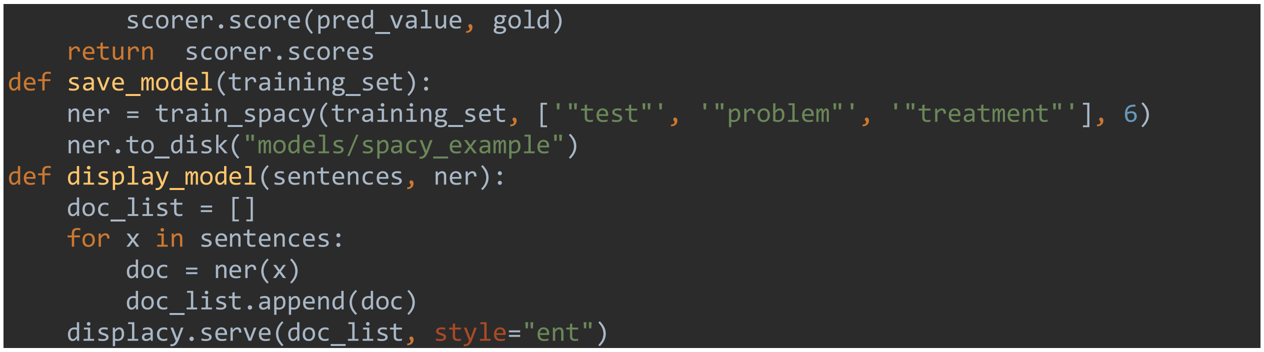
```python
import os
import random

import nltk
import spacy
from spacy import displacy
from spacy.gold import GoldParse
from spacy.scorer import Scorer
from spacy.util import compounding, minibatch

def train_model(TRAIN_DATA, LABELS, model=None, n_iter=30):
    random.seed(0)
    if model is not None:
        nlp = spacy.load(model)  # load existing spaCy model
        print("Loaded model '%s'" % model)
    else:
        nlp = spacy.blank("en")  # create blank Language class
        print("Created blank 'en' model")
    if "ner" not in nlp.pipe_names:
        ner = nlp.create_pipe("ner")
        nlp.add_pipe(ner)
        reset_weights = True
    else:
        ner = nlp.pipe("ner")
    for l in LABELS:
        ner.add_label(l)  # add new entity label to entity recognizer
    # Adding extraneous labels shouldn't mess anything up+5/'.;4rers
    if model is None or reset_weights:
        optimizer = nlp.begin_training()
    else:
        optimizer = nlp.resume_training()
    # other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
    #  with nlp.disable_pipes(*other_pipes):
    sizes = compounding(1.0, 4.0, 1.001)
    for itn in range(n_iter):
        random.shuffle(TRAIN_DATA)
        batches = minibatch(TRAIN_DATA, size=sizes)
        losses = {}
        for batch in batches:
            texts, annotations = zip(*batch)
            nlp.update(texts, annotations, sgd=optimizer, drop=0.35, losses=losses)
        print("Losses", losses)
    return nlp
def train_spacy(train_data, labels, iterations, dropout=0.2, display_freq=1):

    nlp = spacy.blank('en')
    if 'ner' not in nlp.pipe_names:
        ner = nlp.create_pipe('ner')
        nlp.add_pipe(ner)

    for i in labels:
```



```
        ner.add_label(i)

    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'ner']
    with nlp.disable_pipes(*other_pipes):
        nlp.vocab.vectors.name = 'spacy_model'
        optimizer = nlp.begin_training()
        for itr in range(iterations):
            random.shuffle(train_data)
            losses = {}
            batches = minibatch(train_data, size=compounding(4., 32., 1.001))
            for batch in batches:
                texts, annotations = zip(*batch)
                nlp.update(
                    texts,
                    annotations,
                    drop=dropout,
                    sgd=optimizer,
                    losses=losses)
            if itr % display_freq == 0:
                print("Iteration {} Loss: {}".format(itr + 1, losses))
    return nlp
def createPatterns(main_path, k):
    patterns=[]

    for con in os.listdir(main_path+"concept")[k:]:
        file_disc = {}
        fp = open(main_path + "concept/" + con)
        content = fp.readlines()
        content = [x.strip() for x in content]

        for con in content:
            one_pattern = []
            try:
                ind = con[3:].index("\"")
                text = con[3:ind + 3]
                rest = con[ind + 5:]
                type = rest.split("||")[1][2:]
                for t in text.split(" "):
                    if (len(t) > 10 and type == '"treatment"'):
                        patterns.append({"label": type, "pattern": t})
                 #  if(len(t)>3): one_pattern.append({"LOWER":t})
                #patterns.append({"label":type,"pattern": one_pattern})

            except:
                print(con)
    return patterns
def ParseData(main_path,k):
    training_set = []
    for con, txt in zip(os.listdir(main_path+"/concept")[k:],
                        os.listdir(main_path+"/txt")[k:]):
        file_disc = {}
        fp = open(main_path + "/concept/" + con)
        content = fp.readlines()
        content = [x.strip() for x in content]
```



```
        fp = open(main_path + "/txt/" + txt)
        sentences = fp.readlines()
        sentences = [x.strip() for x in sentences]

        split_sentences = []
        i = 0
        for x in sentences:
            pom = x.split();
            split_sentences.append(pom)
            file_disc[i] = []
            i = i + 1

        extraction_data = []
        for con in content:
            try:
                ind = con[3:].index("\"")
                text = con[3:ind + 3]
                rest = con[ind + 5:]
                line_begin = rest.split("||")[0].split(" ")[0].split(":")[0]
                word_begin = rest.split("||")[0].split(" ")[0].split(":")[1]
                line_end = rest.split("||")[0].split(" ")[1].split(":")[0]
                word_end = rest.split("||")[0].split(" ")[1].split(":")[1]
                type = rest.split("||")[1][2:]
                extraction_data.append([text, line_begin, word_begin, line_end,
word_begin, type])
            except:
                print(con)

        for data in extraction_data:
            sen = sentences[int(data[1]) - 1]
            tokens = nltk.word_tokenize(sen)
            i = 0
            begin = 0
            while i < int(data[2]):
                begin = begin + len(tokens[i]) + 1
                i = i + 1
            end = begin + len(data[0])
            file_disc[int(data[1]) - 1].append((begin, end, data[5]))
        for k, v in file_disc.items():
            training_set.append([sentences[k], {"entities": v}])
        file_disc = {}
    return training_set
def load_model(model_path):
    nlp = spacy.blank('en')
    if 'ner' not in nlp.pipe_names:
        ner = nlp.create_pipe('ner')
        nlp.add_pipe(ner)
    ner = nlp.from_disk(model_path)
    return ner
def evaluate(ner_model, examples):
    scorer = Scorer()
    for input_, annot in examples:
        doc_gold_text = ner_model.make_doc(input_)  # Here I used my trained model
        gold = GoldParse(doc_gold_text, entities=annot['entities'])
        pred_value = ner_model(input_)  # trained model on input
```

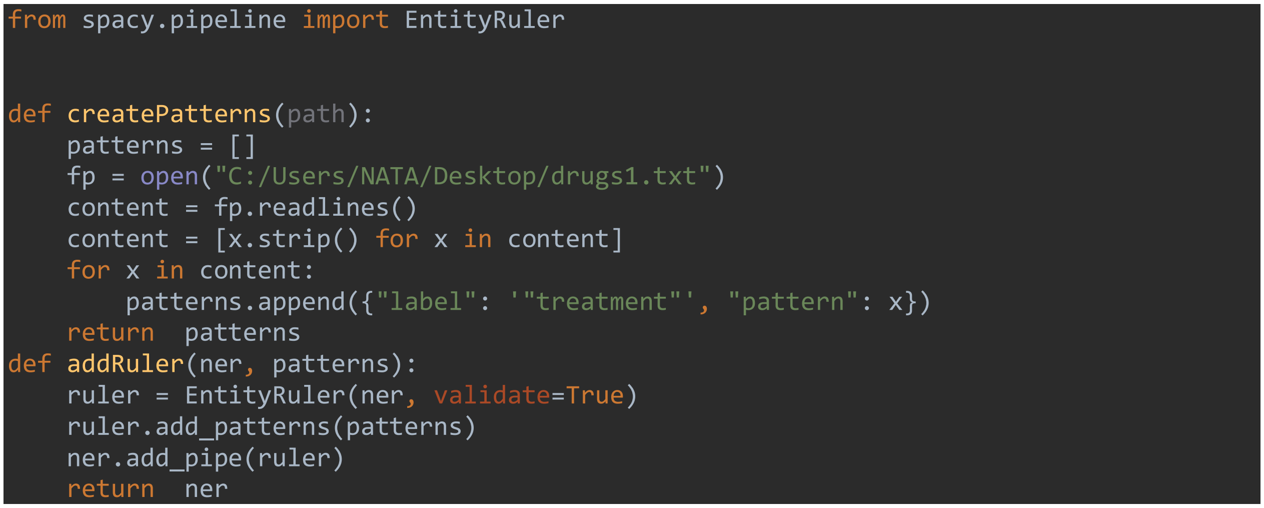


```
        scorer.score(pred_value, gold)
    return  scorer.scores
def save_model(training_set):
    ner = train_spacy(training_set, ['"test"', '"problem"', '"treatment"'], 6)
    ner.to_disk("models/spacy_example")
def display_model(sentences, ner):
    doc_list = []
    for x in sentences:
        doc = ner(x)
        doc_list.append(doc)
    displacy.serve(doc_list, style="ent")
```

Graph.py

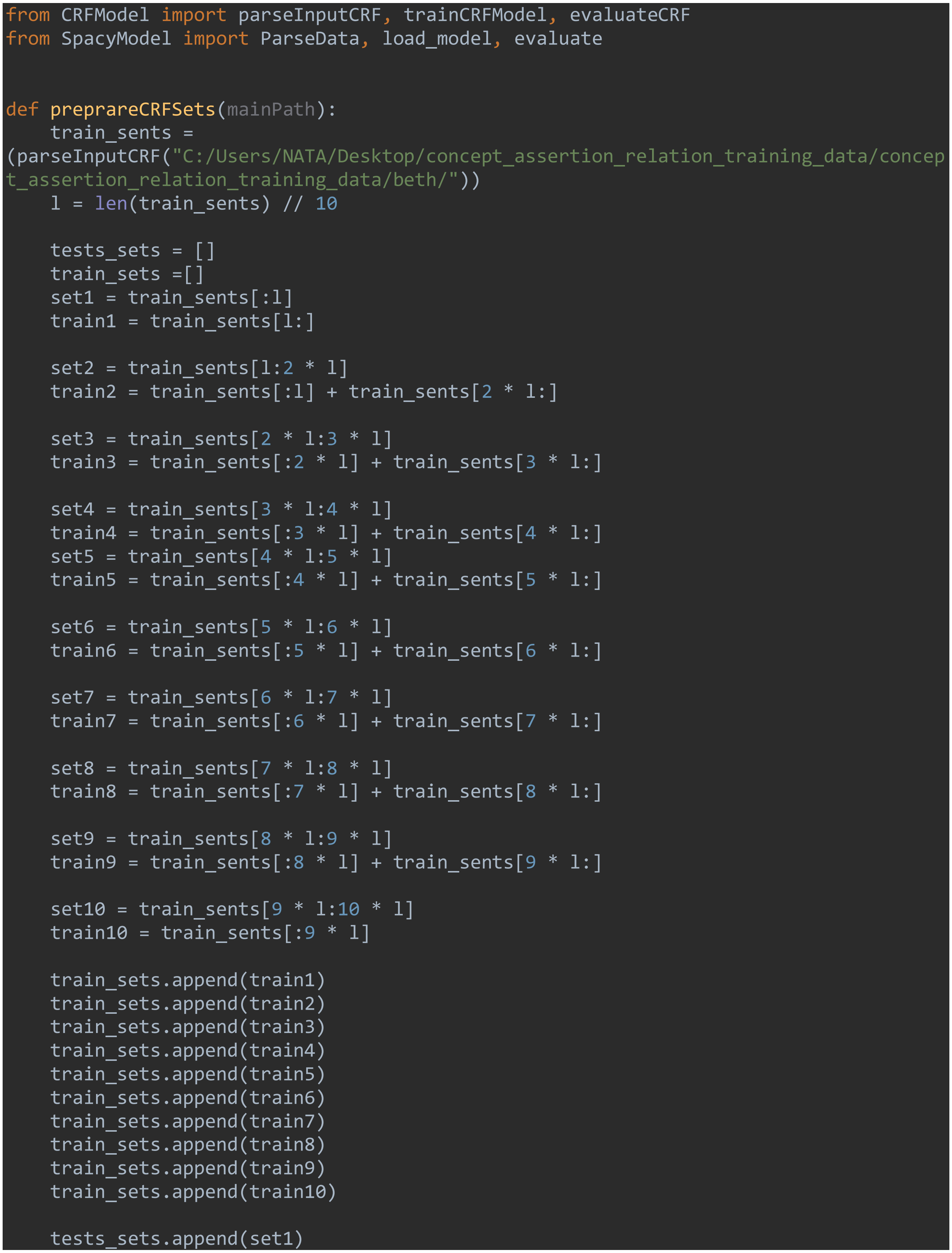
```python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

def prepareGraph(labels, set1,set2):

    x = np.arange(len(labels))  # the label locations
    width = 0.35  # the width of the bars
    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, set1, width, label='CRFSuite')
    rects2 = ax.bar(x + width/2, set2, width, label='Spacy')
    ax.set_ylabel('Scores')
    ax.set_title('Results')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()
    return ax, fig, rects1, rects2

def autolabel(rects, ax):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

def drawGraph(rects1, rects2, fig, ax):
    autolabel(rects1, ax)
    autolabel(rects2, ax)
    fig.tight_layout()
    plt.show()
```

Dictionary.py



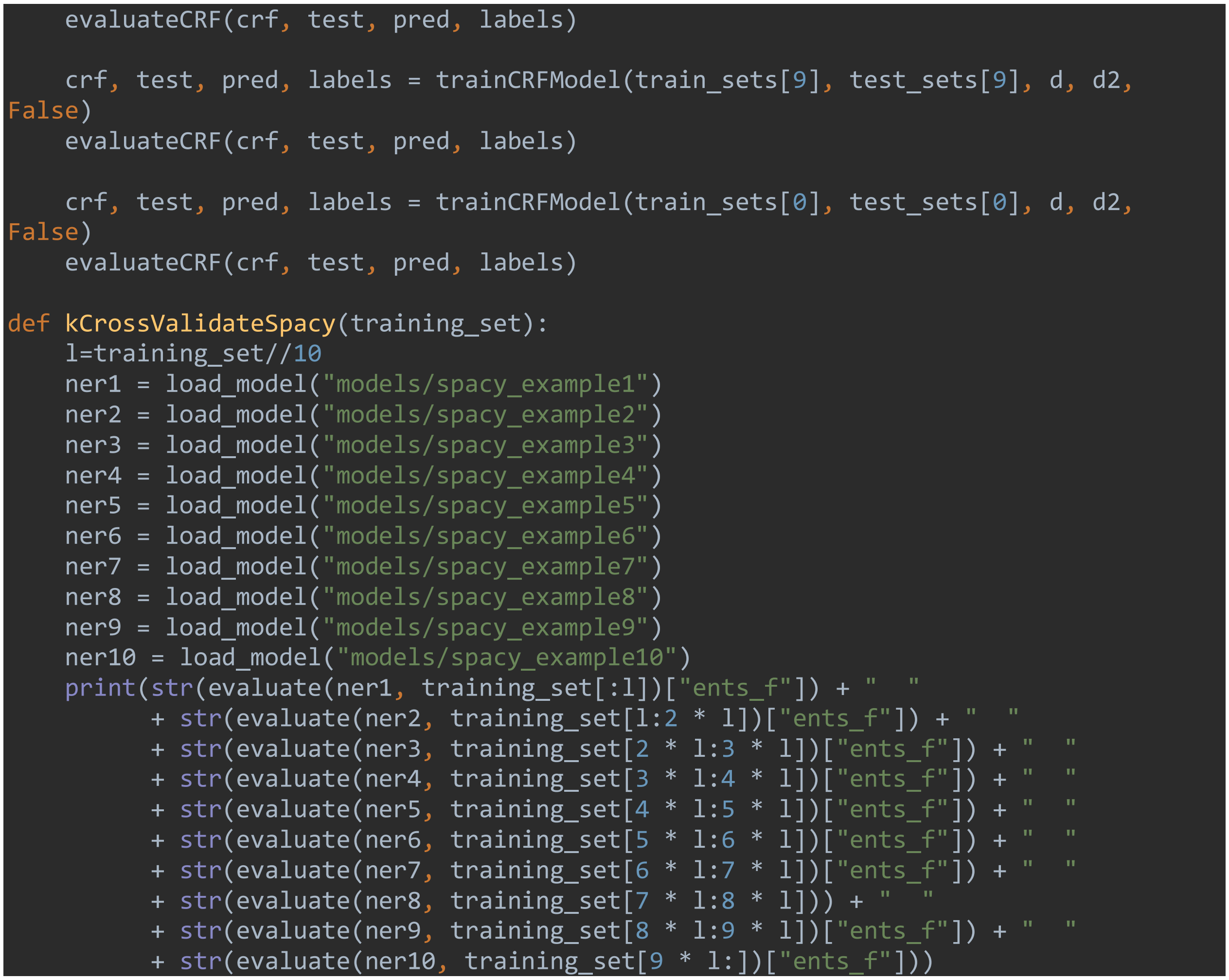
```python
from spacy.pipeline import EntityRuler


def createPatterns(path):
    patterns = []
    fp = open("C:/Users/NATA/Desktop/drugs1.txt")
    content = fp.readlines()
    content = [x.strip() for x in content]
    for x in content:
        patterns.append({"label": '"treatment"', "pattern": x})
    return  patterns
def addRuler(ner, patterns):
    ruler = EntityRuler(ner, validate=True)
    ruler.add_patterns(patterns)
    ner.add_pipe(ruler)
    return  ner
```

CrossValidation.py

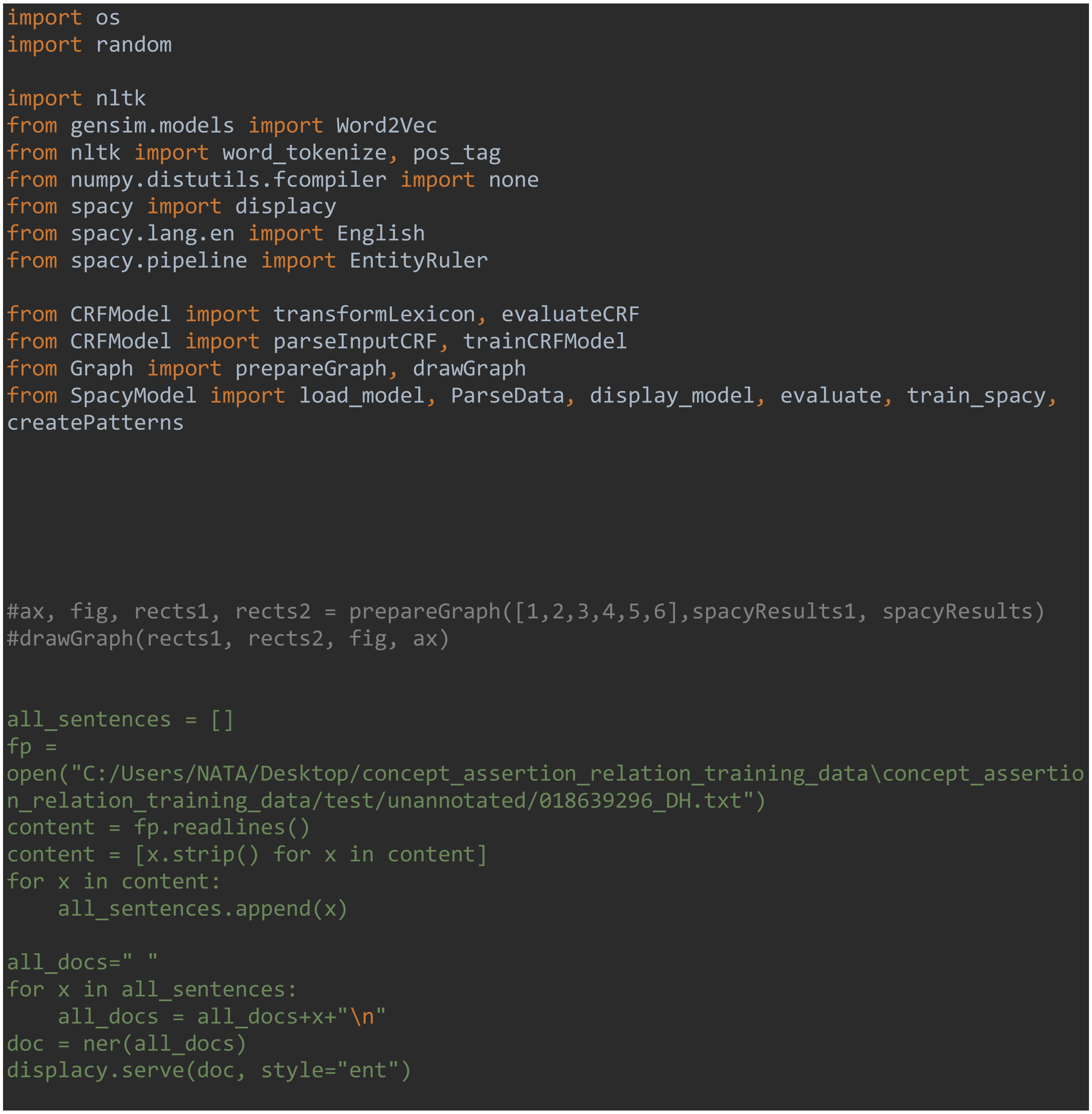


```
from CRFModel import parseInputCRF, trainCRFModel, evaluateCRF
from SpacyModel import ParseData, load_model, evaluate


def preprareCRFSets(mainPath):
    train_sents =
(parseInputCRF("C:/Users/NATA/Desktop/concept_assertion_relation_training_data/concept_assertion_relation_training_data/beth/"))
    l = len(train_sents) // 10

    tests_sets = []
    train_sets =[]
    set1 = train_sents[:l]
    train1 = train_sents[l:]

    set2 = train_sents[l:2 * l]
    train2 = train_sents[:l] + train_sents[2 * l:]

    set3 = train_sents[2 * l:3 * l]
    train3 = train_sents[:2 * l] + train_sents[3 * l:]

    set4 = train_sents[3 * l:4 * l]
    train4 = train_sents[:3 * l] + train_sents[4 * l:]
    set5 = train_sents[4 * l:5 * l]
    train5 = train_sents[:4 * l] + train_sents[5 * l:]

    set6 = train_sents[5 * l:6 * l]
    train6 = train_sents[:5 * l] + train_sents[6 * l:]

    set7 = train_sents[6 * l:7 * l]
    train7 = train_sents[:6 * l] + train_sents[7 * l:]

    set8 = train_sents[7 * l:8 * l]
    train8 = train_sents[:7 * l] + train_sents[8 * l:]

    set9 = train_sents[8 * l:9 * l]
    train9 = train_sents[:8 * l] + train_sents[9 * l:]

    set10 = train_sents[9 * l:10 * l]
    train10 = train_sents[:9 * l]

    train_sets.append(train1)
    train_sets.append(train2)
    train_sets.append(train3)
    train_sets.append(train4)
    train_sets.append(train5)
    train_sets.append(train6)
    train_sets.append(train7)
    train_sets.append(train8)
    train_sets.append(train9)
    train_sets.append(train10)

    tests_sets.append(set1)
```

```
    tests_sets.append(set2)
    tests_sets.append(set3)
    tests_sets.append(set4)
    tests_sets.append(set5)
    tests_sets.append(set6)
    tests_sets.append(set7)
    tests_sets.append(set8)
    tests_sets.append(set9)
    tests_sets.append(set10)


    return train_sets, tests_sets

def prepareSpacySets(mainPath):
    train_sents =
(ParseData("C:/Users/NATA/Desktop/concept_assertion_relation_training_data/concept_as
sertion_relation_training_data/beth/"))
    l = len(train_sents) // 10

    tests_sets = []
    train_sets = []
    set1 = train_sents[:l]
    train1 = train_sents[l:]

    set2 = train_sents[l:2 * l]
    train2 = train_sents[:l] + train_sents[2 * l:]

    set3 = train_sents[2 * l:3 * l]
    train3 = train_sents[:2 * l] + train_sents[3 * l:]

    set4 = train_sents[3 * l:4 * l]
    train4 = train_sents[:3 * l] + train_sents[4 * l:]
    set5 = train_sents[4 * l:5 * l]
    train5 = train_sents[:4 * l] + train_sents[5 * l:]

    set6 = train_sents[5 * l:6 * l]
    train6 = train_sents[:5 * l] + train_sents[6 * l:]

    set7 = train_sents[6 * l:7 * l]
    train7 = train_sents[:6 * l] + train_sents[7 * l:]

    set8 = train_sents[7 * l:8 * l]
    train8 = train_sents[:7 * l] + train_sents[8 * l:]

    set9 = train_sents[8 * l:9 * l]
    train9 = train_sents[:8 * l] + train_sents[9 * l:]

    set10 = train_sents[9 * l:10 * l]
    train10 = train_sents[:9 * l]

    train_sets.append(train1)
    train_sets.append(train2)
    train_sets.append(train3)
    train_sets.append(train4)
    train_sets.append(train5)
    train_sets.append(train6)
    train_sets.append(train7)
    train_sets.append(train8)
    train_sets.append(train9)
    train_sets.append(train10)

    tests_sets.append(set1)
    tests_sets.append(set2)
    tests_sets.append(set3)
    tests_sets.append(set4)
    tests_sets.append(set5)
    tests_sets.append(set6)
    tests_sets.append(set7)
    tests_sets.append(set8)
    tests_sets.append(set9)
    tests_sets.append(set10)

    return train_sets, tests_sets

def kCrossValidateCRF(train_sets, test_sets,d,d2, useDictionary):
    crf_results = []
    crf, test, pred, labels = trainCRFModel(train_sets[0], test_sets[0], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[1], test_sets[1], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[2], test_sets[2], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[3], test_sets[3], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[4], test_sets[4], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[5], test_sets[5], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[6], test_sets[6], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[7], test_sets[7], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[8], test_sets[8], d, d2,
False)
```



```
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[9], test_sets[9], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

    crf, test, pred, labels = trainCRFModel(train_sets[0], test_sets[0], d, d2,
False)
    evaluateCRF(crf, test, pred, labels)

def kCrossValidateSpacy(training_set):
    l=training_set//10
    ner1 = load_model("models/spacy_example1")
    ner2 = load_model("models/spacy_example2")
    ner3 = load_model("models/spacy_example3")
    ner4 = load_model("models/spacy_example4")
    ner5 = load_model("models/spacy_example5")
    ner6 = load_model("models/spacy_example6")
    ner7 = load_model("models/spacy_example7")
    ner8 = load_model("models/spacy_example8")
    ner9 = load_model("models/spacy_example9")
    ner10 = load_model("models/spacy_example10")
    print(str(evaluate(ner1, training_set[:l])["ents_f"]) + "  "
          + str(evaluate(ner2, training_set[l:2 * l])["ents_f"]) + "  "
          + str(evaluate(ner3, training_set[2 * l:3 * l])["ents_f"]) + "  "
          + str(evaluate(ner4, training_set[3 * l:4 * l])["ents_f"]) + "  "
          + str(evaluate(ner5, training_set[4 * l:5 * l])["ents_f"]) + "  "
          + str(evaluate(ner6, training_set[5 * l:6 * l])["ents_f"]) + "  "
          + str(evaluate(ner7, training_set[6 * l:7 * l])["ents_f"]) + "  "
          + str(evaluate(ner8, training_set[7 * l:8 * l])) + "  "
          + str(evaluate(ner9, training_set[8 * l:9 * l])["ents_f"]) + "  "
          + str(evaluate(ner10, training_set[9 * l:])["ents_f"]))
```



```
import os
import random

import nltk
from gensim.models import Word2Vec
from nltk import word_tokenize, pos_tag
from numpy.distutils.fcompiler import none
from spacy import displacy
from spacy.lang.en import English
from spacy.pipeline import EntityRuler

from CRFModel import transformLexicon, evaluateCRF
from CRFModel import parseInputCRF, trainCRFModel
from Graph import prepareGraph, drawGraph
from SpacyModel import load_model, ParseData, display_model, evaluate, train_spacy,
createPatterns






#ax, fig, rects1, rects2 = prepareGraph([1,2,3,4,5,6],spacyResults1, spacyResults)
#drawGraph(rects1, rects2, fig, ax)


all_sentences = []
fp =
open("C:/Users/NATA/Desktop/concept_assertion_relation_training_data\concept_assertio
n_relation_training_data/test/unannotated/018639296_DH.txt")
content = fp.readlines()
content = [x.strip() for x in content]
for x in content:
    all_sentences.append(x)

all_docs=" "
for x in all_sentences:
    all_docs = all_docs+x+"\n"
doc = ner(all_docs)
displacy.serve(doc, style="ent")
```