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

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

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

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\_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 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\_assertion\_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")