EDAN20 - Assignment 3

Oliver Cosic

September 2021

1 Baseline

The baseline proposed by the CoNLL 2000 task got a decent result considering they only use relative simple methods. But we can also tell by just looking at the table in the description that the machine learning methods gave much better score. In the assignment we did our own baseline and got a score of approxiately 0.77 , evaluating it by using the conlleval script.

2 Using Machine Learning

The feature vector in the chunk program provided in the assignment is similar to the one used by Kudoh and Matsumoto , the difference being the values of the previous chunk tags in the window that we call c 1 and c 2. The chunker gave the F1 score of 91.57. Removing the lexical features decreased the F1 score to 87.4. The classifier is a linear model of logistic regression.

3 Adding the features of Kudoh and Matsumoto

Adding the features used by Kudoh and Matsumoto we did the following: We

```
def extract_features_sent_dyn(sentence, w_size, feature_names):
    start = [{'form': '__bos__', 'pos': '__bos__', 'chunk': '__bos__'}]
    end = [{'form': '__eos__', 'pos': '__eos__', 'chunk': '__eos__'}]
    start *= w_size
    end *= w_size
    padded_sentence = start + sentence
    padded_sentence += end

    # We extract the features and the classes
    # X contains is a list of features, where each feature vector is a dictionary
    # y is the list of classes
    X = list()
    y = list()
    for i in range(len(padded_sentence) - 2 * w_size):
        # x is a row of X
        x = list()
        # The words in lower case
        for j in range(2 * w_size + 1):
            x.append(padded_sentence[i + j]['form'].lower())
        # The POS
        for j in range(2 * w_size + 1):
            x.append(padded_sentence[i + j]['pos'])
        # The chunks (Up to the word)

        for j in range(w_size):
            x.append(padded_sentence[i + j]['chunk'])

        # We represent the feature vector as a dictionary
        X.append(dict(zip(feature_names, x)))
        # The classes are stored in a list
        y.append(padded_sentence[i + w_size]['chunk'])
    return X, y
```

also expanded the feature names with the additional c1 and c2 and retrained the classifier although with the same model used as before, being logistical regression. The only other difference from the ML-program provided was the way we did the actual prediction:

```
from tqdm import tqdm
for sentence in tqdm(test_corpus):
    X1, y1 = extract_features_sent_dyn(sentence, w_size, feature_names_dyn)
    y_test_predicted=[]
    chunk_1='__bos__'
    chunk_2='__bos__'
    for x in X1:
        x['chunk_n1'] = chunk_1
        x['chunk_n2'] = chunk_2
        X2=vec_dyn.transform(x)
        chunk_0=classifier_dyn.predict(X2)
        y_test_predicted_dyn.append(chunk_0[0])
        chunk_2=chunk_1
        chunk_1=chunk_0[0]
```

```
100%|██████████| 2012/2012 [03:22<00:00,  9.94it/s]
```

And this gave us the following result:
Getting a score of 92,65.

```
inx = 0
for sentence in test_corpus:
    for word in sentence:
        word['pchunk'] = y_test_predicted_dyn[inx]
        inx += 1
```

```
save_results(test_corpus, keys, 'out')
```

Evaluation

```
lines = open('out').read().splitlines()
res = conlleval.evaluate(lines)
improved_ml_score = res['overall']['chunks']['evals']['f1']
improved_ml_score
```

```
0.9265266775640221
```

4 Collecting the entities

As the last part of the assignment we collected the entities from the training set defined as NP chunks starting with NNP or a NNPS tag. To test this out we narrowed our set to only containing the ones starting with a letter K. Choosing a couple I think would be in wikipedia:
Kim, Kabul, Kentucky , Kentucky fried Chicken stores.
Choosing a couple I dont think would be in wikipedia: Kwek Hong Png, KTXL, Kajima. Choosing ambiguous:
Kim, Kabul, Kentucky . Judging from the results of the english wikipedia search I was wrong with many of the guesses. Then we tested the swedish wikipedia in the same way and lastly found the entities in both the swedish and english version:

```
confirmed_ne_en_sv=[]
for item in ne_ids_sv:
    if item in ne_ids_en:
        confirmed_ne_en_sv.append(item)
```

```
confirmed_ne_en_sv=list(sorted(confirmed_ne_en_sv))
print(confirmed_ne_en_sv)
```

```
[(('KIM',), 'Q224736'), (('KLM', 'Royal', 'Dutch', 'Airlines'), 'Q181912'), (('Kabul',), 'Q5838'), (('Kansas',), 'Q15
58'), (('Kenton',), 'Q358393'), (('Kentucky',), 'Q1603'), (('Khost',), 'Q386682'), (('Kirin',), 'Q297659'), (('Kodak
',), 'Q486269'), (('Kuala', 'Lumpur'), 'Q1865')]
```

5 Contextual string embeddings for sequence labeling

The main differences between the method in the article and our method as far as I can tell is that they used a recurrent neural network compared to our logistic regression. Furthermore they use what they call contextual string embeddings which according to them combines the best attributes of previously known methods, ours being one of them. The three main attributes they want is 1:pre-train on large unlabeled corpora, 2: capture word meaning in context , which gives different embeddings for the same words based on context and

3: model words and context fundamentally as sequences of characters. In the article they state that they reached an F1-score of 96.72.