EDAN20 - Lab 4

André Frisk - an8218fr-s

September 2021

The Chunker

This short description will focus on the submission parts of the lab which is the baseline score of prediction precision, how the prediction precision is for the machine learning part and if the entities are correctly resolved from the set which only consist of entities which starts the words with K.

The baseline score is determined quite simple. A list of every part of speech with the chunk distribution is provided which tells the frequency of every chunk for the part of speech. This list is then minimized to get the chunk with the highest frequency for the specific part of speech. This is then used to the prediction which uses the CoNLL technique. This technique uses the two last columns (chunk and predicted chunk) to compute the performance (F1 score) which is around 77% on this test corpus.

To get a better score we need to use a machine learning program which uses CoNLL but also the methods of Kudoh and Matsumoto. They built a feature vector consisting of the values of five words in a window, the values of the five parts of speech in this windows and the two previous chunk tags in the first part of the window. The chunk tags is the important part here. Here a classifier is trained using Logistic Regression and a dynamic prediction. When predicting here we predict a chunk tag for the sequence and this chunk tag that is predicted must be used in the next prediction to properly work and get a score within the Logistic Regression score. A common error is when you predict the test set you use the dynamic features and you can't use the ones from the test file. The two chunks that has been predicted must be used instead of the dynamic features (which was trained with the classifier beforehand). When predicting with the ML-program the accuracy of the prediction in our case was 92,65%.

The entities are collected to a list from the training set which is defined as NP chunks and is starting with a NNP (proper noun) or NNPS (proper noun, plural). They are collected as long as a B-NP is detected and the chunks after the B-NP is I-NP, these are inserted as tuples to the list. We are only interested in the tuples which starts with the letter K. These tuples are then used with the Wikipedia function explained below which gives the entity id for this specific tuple. The same method is used for the Swedish Wikipedia for comparison. In the comparison we take the intersection of the English Wikipedia result (`ne_ids_en`) and the Swedish one (`ne_ids_sv`) which gives the common entities of the results.

Questions from the lab

This lab had many questions and descriptions to answer and instead of answering them in a text description they are answered as single questions (items) here to make sure every question is answered.

- **Read the baseline proposed by the organizers of the CoNLL 2000 shared task (In the Results Sect.). What do you think of it?** It works, they get pretty high score overall of around 91,5 to 92,5 (six out of eleven) which could class as sufficient at a point where maximum precision is not needed.
- **Python translation, evaluation procedure (conlleval.txt):** So what this table shows first is that there exist 961 tokens with 459 phrases where the chunking software found 539 phrases which 371 is correct. This table shows the percentage of the tokens that got the correct chunking tag and the precision, recall and FB1 rates. The FB1 rates are both overall and for each of the chunk types in data.

```
processed 961 tokens with 459 phrases; found: 539 phrases; correct: 371.
accuracy:  87.20%; precision:  68.83%; recall:  80.83%; FB1:  74.35
             ADJP: precision:   0.00%; recall:   0.00%; FB1:   0.00  1
             ADVP: precision:  45.45%; recall:  62.50%; FB1:  52.63  11
               NP: precision:  64.98%; recall:  78.63%; FB1:  71.16  317
               PP: precision:  83.18%; recall:  98.89%; FB1:  90.36  107
             SBAR: precision:  66.67%; recall:  33.33%; FB1:  44.44  3
               VP: precision:  69.00%; recall:  79.31%; FB1:  73.80  100
```

- **ML Program:**
  1. **What is the feature vector that corresponds to the chunking program above? Is it the same Kudoh and Matsumoto used in their experiment?** The feature vector in the program is similar however it have two features missing in comparison with Kudoh and Matsumoto, the values of the two previous chunk tags (c_{i-2} & c_{i-1})
  2. **What is the performance of the chunker?** 91,57%
  3. **Remove the lexical features (the words) from the feature vector and measure the performance. You should observe a decrease.** The accuracy decreased to 87,40%
  4. **What is the classifier used in the program?** Logistic Regression
- **Description of `wikipedia_lookup`:** The lookup is executed in several different steps- Firstly we retrieve the correct URL to the Wikipedia by taking the element in `ne_small_set` and join it with the base URL of Wikipedia. Then we retrieve the text from the URL that we created via a HTML doc which then is used with `Beautiful Soup` to web-scrape the HTML doc. By doing this pull the data out of the HTML doc and creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner. We then find the entity id from this tree and returns it. If nothing of this works we return an UNKNOWN entity id.

```
def wikipedia_lookup(ner, base_url='https://en.wikipedia.org/wiki/'):
try:
    url_en = base_url + ' '.join(ner)
    html_doc = requests.get(url_en).text
    parse_tree = bs4.BeautifulSoup(html_doc, 'html.parser')
    entity_id = parse_tree.find("a", {"accesskey": "g"})['href']
    head_id, entity_id = os.path.split(entity_id)
    return entity_id
```

```
except:
    pass
    # print('Not found in: ', base_url)
entity_id = 'UNK'
return entity_id
```

Contextual String Embeddings for Sequence Labeling

In the Contextual String Embeddings for Sequence Labeling model they used a recurrent neural network instead of logistic regression as we use. We train our model with explicit notion of words or sequence of words and not characters as they do.

The performance they reach on our corpus (CoNLL 2000) using their method gets a score of $96{,}72 \pm 0{,}05\%$.