Lab 4: Chunker

Felicia Segui, fe7744se-s

October 2020

1 Introduction

The aim of this assignment is to create a program that extracts syntactic groups from a text. To train and test the program, the CoNLL-2000 dataset will be used. A small dataset will be extracted in the end of the assignment, and the connection of the entities and Wikipedia will be investigated.

2 CoNLL-2000

The goal for the CoNLL-2000 shared task was to create a program that divided a text into syntactically correlated parts of words. To evaluate the accuracy of the chunker, the F rate was used, see equation 1.

$$\mathrm{F} = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{1}$$

Both the test set and the training set have the same structure where every line consists of three columns. In the first column the word is found, in the second column the word's part-of-speech tag is found and in the third column its chunk tag is found.

Of the eleven systems applied to the program, two of them had results that stood out. One of them was the system *Support Vector Machines* used by *Kudoh* and *Matsumoto* in which a classifier known for its good generalization formula for text categorization, without over-fitting, was used. The classifier in question has large margins between the different text categories. The other system that performed very well was *Weighted Probability Distribution Voting* used by *Van Halteren* which used five different base chunkers which step by step improved the the text chunking. All the systems had better F-rate than the baseline, which was expected as the baseline was calculated only by extracting the chunk tag which was most frequently associated with the current part-of-speech tag, apart from the aforementioned eleven systems that used machine learning techniques or repeated improved chunking methods.

3 Baseline chunker

Before creating functions for extracting information from the corpus files, the test- and train-files were downloaded and modified into structures more convenient. A function called func_dis() was created which returned a dictionary with the part-of-speech tags as keys and dictionaries with chunk tags appearing on the same words, as the part-of-speech tag in question, as values. In the chunk tags dictionaries the chunk tags were keys and their raw count were values. From the resulting dictionary a new dictionary was created with the most frequent combination for each part-of-speech tag and chunk tag. With this information a prediction was made where the predicted chunk tag was appended as an extra column.

4 Evaluation of results with Python

When an evaluation rate was calculated with the predefined function eval(), the value 0.7729 was presented. It was extracted by simply calculating the percentage of correct chunk predictions. The corresponding value when using the conlleval library was 0.77067 and it differs from the eval() value, because the eval() value is biased by the most frequent tags, apart from the conlleval value as it uses weights to avoid the biases. Both values are close to the baseline values presented in the CoNLL-2000 chunking documentation.

5 The ML program

In the machine learning part of the assignment a program was predefined where the information used for training the model was the values of the surrounding words and part-of speech tags (in total five words and five part-of-speech tags). For extracting the features from a sentence and further creating a feature vector a function called extract_features_sent_static() was used. The function returned a list of dictionaries where the features for the input sentence were stored as strings. Every dictionary represented a word in the sentence and every sentence that was sent into the function was concatenated with two fabricated dictionaries in the beginning and two in the end. The function also returned a list with chunk tags. A function called extract_features_static() made it possible to send in the whole training corpus instead of iterating through it outside a function. From the latter mentioned function a matrix was returned, where each row contained the features of a sentence, as well as a list with the chunk tags. By transforming the matrix a feature vector was extracted.

A classifier with name Logistic Regression was created and used throughout the assignment. This classifier is linear, apart from the MLP classifier that uses a neural network and therefore can handle non-linear relations.

The classifier was fitted with the feature vector and the list of chunk tags. Thereafter the feature vector for the test set was extracted and then the chunk tag predictions were obtained by using the classifier's predict function. The predicted chunk tags were appended to the test corpus and the conlleval evaluation function was used to see how good the model was. The resulting F rate had the value 0.9158 which is a big improvement from the F rate of the baseline.

To improve the model two more features was added to the program: c_{n-2} and c_{n-1}, which represent the chunk tags for the previous words. The same method as described above was used, with a couple of small adjustments in the functions so that also the chunk tags are extracted as features. It was also necessary to add the start chunk tag *BOS* in the beginning of every new sentence, as the first two words could not extract both chunk tags from previous words. The resulting F rate for this program got the value 0.9265, an improvement since the last program version.

To analyze the importance of the w_{n-x} features the same model was trained with only part-of-speech tag features and the F rate then got the value 0.8741.
The system *Support Vector Machines*, mentioned in the CoNLL-2000 section, used same features as the features used in the last machine learning task in this assignment, i.e. [w_{n-2}, w_{n-1}, w_n, w_{n+1}, w_{n+2}, t_{n-2}, t_{n-1}, t_n, t_{n+1}, t_{n+2}, c_{n-2}, c_{n-1}] where w_{n-x} represent the word x steps from the word w_n, t_{n-x} represent the part-of-speech tag x steps from the part-of-speech tag t_n and same concept for c_{n-x} where c is the

chunk tag.

6 A function to look up entities

Before investigating the existence on Wikipedia for a set of entities all NP chunks starting with a proper noun or a plural proper noun were extracted from the training corpus. To create a small set, all entities starting with the letter K were extracted.

The entities were sent one at the time into the function wikipedia_lookup() which returned the Wikipedia entity id if the entity had a Wikipedia website, and UNK if the entity did not have a Wikipedia website. The function has two input parameters: the entity and the predefined Wikipedia base url. On the first line in the try section in the function a new url was created by concatenating the base url with the strings in the entity, by using the join function. Thereafter the function tries to get the web page by using the request.get() function. If it does not succeed the entity id is assigned the string UNK, but if it does succeed BeautifulSoup is used to parse the web page. In the last lines the functions find and split are used to extract the entity id, which is returned from the wikipedia_lookup() function.

A list with two-element-tuples was created with help from the wikipedia_lookup() function. The tuples contained the entities that existed in Wikipedia and their corresponding entity id. All ambiguous entities such as *KIM* and *Ky* were found and got an entity id, together with the more straight forward entities such as *Kuala Lumpur* and *Kurt Hager*. A majority of the entities were not found, for instance *Kathie Roberts* and *Kansas and Texas*. Fewer entities could be found on the Swedish Wikipedia, only about 42% of the entities found on English Wikipedia. This is although not that surprising as the words, apart from the names, were in English, and as the corpus is of American origin.

7 Contextual String Embeddings for Sequence Labeling

The system described in the article *Contextual String Embeddings for Sequence Labeling* had the aim to create a new type of word embedding called *contextual string* *embedding.* Alike our system, the same word has different embeddings depending on the surrounding context it is found in. A difference from our system, the contextual string embedding system uses a method where it handles the words together with their contexts as sequences of characters. This simplifies the handling of misspelled or rare words as well as enables modelling of subword structures such as prefixes and endings. Another difference between the systems is that the contextual string embedding system uses neural networks instead of the logistic regression classifier used in this assignment. In addition to tagging with part-of-speech tags and chunking tags, the system described in the article also uses a type of tagging called NER tagging. NER stands for named-entity recognition and the tagging method locates named entities and then classifies them into predefined categories such as person names, organizations and locations. The contextual string embedding system reached the F rate value 0.9785 for part-of-speech tags, 0.9672 for chunk tags, 0.9309 for NER English tags and 0.8832 for NER German tags on the CoNLL-2000 corpus. All tests except the test with NER German tagging reached higher F rate values than the model used in this assignment.