Transition-based Dependency Parsing
Project Report

Li LIN 02.2023

1. Introduction:

Dependency parsing is a fundamental task in Natural Language Processing (NLP), which focuses on analyzing the syntactic structure of sentences by parsing the dependencies between words.

The main goal of this project is to implement a dependency parser based on a transition-based system in statistical methods, trained on English and German data in CoNLL06 format. Finally, the obtained parser and model are used to analyze the English and German test datasets to obtain the dependency relations for each word in each sentence and return the analysis result file.

2. Overview

Decoding algorithm:	Transition-based, ArcStandard.
The feature model:	Based on Niver(2008) and Zhang and Niver(2011).
Machine learning method:	Multi-class perceptron.
Evaluation :	Unlabeled attachment score(UAS).
Programming language:	Python 3.10.

3. Project Specification

3.1. Decoder

1). ArcStandard System:

A. Three types of transitions: LeftArc, RightArc and Shift.

B. Configurations: a configuration includes a stack, a buffer and a arc set.

C. Start state: Root on stack, all tokens on buffer and empty arc set.

The empty arc set is represented by a filled -1 list of length num(root+tokens), with the element index indicating the corresponding token number. The index 0 is for Root.

D. Terminal state: when the buffer is empty.

E. Transition algorithm(Greedy search):

a. Obtain the correct dependencies for each sentence in the training set, and the arc set of the current configuration;

b. Determine the transitions of the configuration by Oracle, in the order of priority LeftArc, RightArc, Shift;

c. Apply the transaction to the configuration to update it;

d. Cycle through the above steps until the configuration reaches the Terminal state.

2). Aims:

Obtain the configurations in the parsing step for each sentence in the training data, and the corresponding sequence of transitions. The total number of transitions for each sentence is 2*num(tokens).

This prepares the training data for feature extraction and training of the machine learning model.

3.2. Feature model

1). Feature extraction:

Based on the configurations obtained above for each sentence, the features of each configuration are extracted.

A. Template:

References are Niver (2008) and Zhang and Niver (2011), with minor additions and deletions.

B. Features elements:

token‘s form, token‘s POS, token‘s lemma, token‘s morphology, distance between two tokens.

C. Self-expanding templates:

S[-1]-pos+distance(B[0]-S[-1]), S[-1]-form+distance(B[0]-S[-1]), S[-1]-pos+B[0]-pos+distance(B[0]-S[-1]), S[-1]-form+B[0]-form+distance(B[0]-S[-1]).

2).Feature Mapping:

For training, all extracted features are stored in a dictionary, with each feature corresponding to an unique integer value.

3).Feature vector representation:

For testing, the features of each configuration during the parsing of the sentence are extracted and represented by the corresponding integer values of the features dictionary obtained above. The aim is to save storage space by not representing it through a sparse matrix.

4).Aims:

Prepare for training machine learning models to train weights and calculate the scores of transitions for individual configurations during testing.

3.3. Machine Learning Method

1). Multi-class perceptron

The weight matrix has three rows, corresponding to three transitions: LeftArc, RightArc and Shift.

For the training data, the scores of the three transitions are calculated for each configuration by using the feature vector*weight vector, with the highest score being the best transition. If the predicted transition is not the same as the correct one, the corresponding weight vectors are modified.

2) .Aims:

The final weight matrix is obtained by training several epochs.

During testing, the best transitions for each configuration are calculated based on the obtained weights matrix to achieve dependency analysis.

4. Results

Unlabeled attachment score(UAS):

Train data results:
Epochs = 10
English: 1336893 / 1350888 * 100 = 98.96 %
German: 790925 / 796314 * 100 = 99.32 %

Dev data results:
English: 21836 / 26595 * 100 = 82.11 %
German: 64160 / 76704 * 100 = 83.65 %
Average: 82.88%

Test data results:
English: 27604 / 33306 * 100 = 82.88 %
German: 73182 / 92004 * 100 = 79.54 %
Average: 81.21%