The parser is modular: it can use several learning algorithms: Maximum Entropy, SVM, Winnow, Voted Perceptron, Memory Based Learning, as well as combinations thereof. The submitted runs used Maximum Entropy and I present accuracy and performance comparisons with other learning algorithms.

No additional resources are used.

No pre-processing or post-processing is used, except stemming for Danish, German and Swedish.

2 Features

Columns from input data were used as follows.

LEMMA was used in features whenever available, otherwise the FORM was used. For Danish, German and Swedish the Snowball stemmer (Porter 2001) was used to generate a value for LEMMA. This use of stemming slightly improved both accuracy and performance.

Only CPOSTAG were used. PHEAD/PDEPREL were not used.

FEATS were used to extract a single token combining gender, number, person and case, through a language specific algorithm.

The selection of features to be used in the parser is controlled by a number of parameters. For example, the parameter `PosFeatures` determines for which tokens the POS tag will be included in the context, `PosLeftChildren` determines how many left outermost children of a token to consider, `PastActions` tells how many previous actions to include as features.

The settings used in the submitted runs are listed below and configure the parser for not using any word forms. Positive numbers refer to input tokens, negative ones to token on the stack.

```
LemmaFeatures          -2 -1 0 1 2 3
PosFeatures            -2 -1 0 1 2 3
MorphoFeatures         -1 0 1 2
DepFeatures            -1 0
PosLeftChildren        2
PosLeftChild           -1 0
DepLeftChild           -1 0
PosRightChildren       2
PosRightChild          -1 0
DepRightChild          -1
PastActions            1
```

The context for POS tags consisted of 1 token left and 3 tokens to the right of the focus words, except for Czech and Chinese were 2 tokens to the left and 4 tokens to the right were used. These values were chosen by performing experiments on the training data, using 10% of the sentences as held-out data for development.

3 Inductive Deterministic Parsing

The parser constructs dependency trees employing a deterministic bottom-up algorithm which performs Shift/Reduce actions while analyzing input sentences in left-to-right order.

Using a notation similar to (Nivre and Scholz, 2003), the state of the parser is represented by a quadruple $\langle S, I, T, A\rangle$, where S is the stack, I is the list of (remaining) input tokens, T is a stack of temporary tokens and A is the arc relation for the dependency graph.

Given an input string W, the parser is initialized to $\langle (), W, (), ()\rangle$, and terminates when it reaches a configuration $\langle S, (), (), A\rangle$.

The parser by Yamada and Matsumoto (2003) used the following actions:

`Shift` in a configuration $\langle S, n|I, T, A\rangle$, pushes n to the stack, producing the configuration $\langle n|S, I, T, A\rangle$.

`Right`[1] in a configuration $\langle s_1|S, n|I, T, A\rangle$, adds an arc from s_1 to n and pops s_1 from the stack, producing the configuration $\langle S, n|I, T, A\cup\{(s_1, r, n)\}\rangle$.

`Left` in a configuration $\langle s_1|S, n|I, T, A\rangle$, adds an arc from n to s_1, pops n from input, pops s_1 from the stack and moves it back to I, producing the configuration $\langle S, s_1|I, T, A\cup\{(n, r, s_1)\}\rangle$.

At each step the parser uses classifiers trained on treebank data in order to predict which action to perform and which dependency label to assign given the current configuration.

4 Non-Projective Relations

For handling non-projective relations, Nivre and Nilsson (2005) suggested applying a pre-processing step to a dependency parser, which consists in lifting non-projective arcs to their head repeatedly, until the tree becomes pseudo-projective. A post-processing step is then required to restore the arcs to the proper heads.

[1] Nivre and Scholz reverse the direction, while I follow here the terminology in Yamada and Matsumoto (2003).