CS5740: Assignment 5

https://github.com/cornell-cs5740-20sp/assignment-5-zc-as

Zi-Ning Choo
zc462

Ananya Shivaditya
aks298

1 Introduction (5pt)

Dependency parsing establishes grammatical relationships between words in a sentence, designating "head" words and words modifying those heads. This report describes an arc-standard parser for unlabelled transition-based dependency parsing that uses a multi-class SVM oracle to find the best transition at each step. The data consisted of gold-standard dependency trees for the first 1M sentences in the 1B word language model benchmark data set [1], and was divided into training, development, and testing sets with 600K, 200K, and 200K sentences, respectively.

Experiments on feature engineering, hyperparameter tuning, and learning curves are described. Parser performance was measured with unlabeled attachment score (UAS), defined as the fraction of words with correctly-assigned heads. The best-performing model was trained on 200K sentences and used a combination of wordform and part-of-speech features, along with features based on valence, head-modifier distance, and intervening content between head and modifier. UAS scores of this model on the development and test sets were 0.8717 and 0.8707.

2 Parser Description (7pt)

The arc-standard parser consists of a stack σ, buffer β, and set of assigned dependency relationships D. For unlabeled assignment, each dependency relationship has structure (h, m), referring to head and modifier respectively. Given a sentence $\langle w_1, ..., w_n \rangle$, the parser is initialized so that $\sigma = [\text{ROOT}]$, $\beta = [w_1, ..., w_n]$, $D = \{\}$. Until β is empty, a transition $t \in \{\text{SHIFT}, \text{RIGHTARC}, \text{LEFTARC}\}$ is selected. The effect of each transition on σ and β is given below. At termination, β is empty and D contains the edges of a projective dependency tree in which each token w_i is assigned at most one head [2].

	SHIFT
Before	σ, $w_i\|\beta$, D
After	$\sigma\|w_i$, β, D
	LEFT ARC
Before	$\sigma\|w_i$, $w_j\|\beta$, D
After	σ, $w_j\|\beta$, $D \cup (w_j, w_i)$
	RIGHT ARC
Before	$\sigma\|w_i$, $w_j\|\beta$, D
After	σ, $w_i\|\beta$, $D \cup (w_i, w_j)$

3 Features (25pt)

Before computing features, we preprocessed the data by making all tokens lower-case. In total, we use 1.3M features, divided according to the categories tabulated below.

Notation In the following table, σ_0 represents the the top of the stack; β_0, β_1, and β_2 represent the top three positions in the buffer; w and p represent the lower-case word form and POS tag. As an example, $\sigma_0.w$ indicates the word form of the token at the top of the stack. In addition, $V = 14762$ represents vocabulary size. This was the number of unique lower-cased tokens with at least 50 occurrences in the training set. $P = 43$ represents the number of POS tags in the training set.

category	features	# feat.
one-word	$\sigma_0.w$, $\sigma_0.p$, $\sigma_0.wp$, $\beta_0.w$ $\beta_0.p$, $\beta_0.wp$, $\beta_1.w$, $\beta_1.p$ $\beta_1.wp$, $\beta_2.w$, $\beta_2.p$, $\beta_2.wp$	$4 \times (V + P + VP))$ $= 597.6K$
two-word	$\sigma_0.w \circ \beta_0.w$, $\sigma_0.w \circ \beta_0.p$, $\sigma_0.p \circ \beta_0.w$, $\sigma_0.p \circ \beta_0.p$	$(P^2 + 2VP + 1e5)$ $= 699.8K$
three-word	$\sigma_0.p \circ \beta_0.p \circ \beta_1.p$, $\beta_0.p \circ \beta_1.p \circ \beta_2.p$, $\sigma_0.p \circ \beta_1.p \circ \beta_2.p$, $\sigma_0.p \circ \beta_0.p \circ \beta_2.p$,	$P^3 = 79.5K$
dependencies	$\sigma_0.dr$, $\sigma_0.dl$ $\beta_0.dr$, $\beta_0.dl$	4
word distance	$\sigma_0.\beta_0.wd$, $\beta_0.\beta_1.wd$	2
intervening content	$\sigma_0.\beta_0.ic$	$(P + V)$ $= 14.7K$

Baseline features We used a set of baseline features based on word form and POS, similar to the baseline features used by [4]. Each of these is a binary feature associated with an integer index; they are represented in the table above as one, two, and three-word features. To limit the number of two-word word form features ($\sigma_0.w \circ \beta_0.w$), only the most frequent 100K pairs were included as features.

Additional features A small number of additional features that empirically improved parser performance were added:

1. dependencies (*dep*): the average number of left (*dl*) and right (*dr*) dependents of a given word in the training data. For each token in the training set, these values were computed by counting the total number of left and right dependents in all gold-standard dependency trees, and dividing by the total occurrences of that token.

2. word distance (wd): the number of tokens between the tokens at the top of the stack and buffer.
3. intervening content (ic): the wordforms and POS tags of tokens found between the word at the top of the stack and the word at the top of the buffer. This was a binary vector, where each position was 1 if the wordform/POS was present, and 0 otherwise.

Examples To illustrate feature vector construction, we provide an example from each feature category. Consider the example sentence "John likes Mary.", corresponding POS tags (NNP, VBZ, NNP, .), and parser configuration

$$\sigma = [\text{ROOT}|\text{John}], \beta = [\text{likes}|\text{Mary}, .], D = \{\}$$

Using this parser state, some sample features in each general category are as follows:

category	feature	example
one-word	$\sigma_0.wp$	John-NNP
two-word	$\sigma_0.w \circ \beta_0.w$	John-likes
three-word	$\sigma_0.p \circ \beta_0.p. \circ \beta_1.p$	NNP-VBZ-NNP
dependencies	$\sigma_0.dl$	0.0643
word distance	$\sigma_0.\beta_0.wd$	1
intervening content	$\sigma_0.\beta_0.wd$	$\emptyset$

Running time analysis To compute the most likely transition at each step, we must compute the feature vector corresponding with parser state, and the dot product of that feature vector with the weight vector of each of SVMs we have trained for our multi-class problem. For this analysis, let n be the length of some input sentence, and c be the number of binary SVMs.

Our features can generally be computed in $O(1)$ by looking up indices in a precomputed hash table, with the exception of intervening content (ic). Because we must examine every word between the tokens at the top of the stack and buffer, this potentially takes $O(n)$. Likewise, the number of nonzero entries in the resulting feature vector and time required to compute dot product with SVM weight vector is also $O(n)$ in the worst case. Since we need to compute c dot products, the per-transition complexity is $O(nc)$. For a sentence of length n the parser can perform a maximum of $2n$ transitions before terminating [2]. As a result the time required to parse one sentence is $O(cn^2)$.

4 Learning (10pt)

Finding gold-standard transitions The following algorithm takes as input a sentence $W = \langle w_1, ..., w_n \rangle$ and a set of gold-standard dependencies D_g and constructs a list of intermediate parser states S and transitions T that could have been used to produce D_g. These transitions were used to train the SVM oracle described in the next section.

$\sigma \leftarrow [ROOT], \beta \leftarrow [w_1, ..., w_n], D \leftarrow \{\}$
$T \leftarrow [], S \leftarrow []$
while not $\beta = \emptyset$ **do**
  $s \leftarrow (\sigma_0, \beta_0, \beta_1, \beta_2)$
  if $(\beta_0, \sigma_0) \in D_g$ **then**
    $t \leftarrow$ LEFTARC
  else if $(\sigma_0, \beta_0) \in D_g$ **and** $(\beta_0, w_i) \in D \; \forall (\beta_0, w_i) \in D_g$ **then**
    $t \leftarrow$ RIGHTARC
  else
    $t \leftarrow$ SHIFT
  end if
  $T \leftarrow T|t, S \leftarrow S|s$

Multi-class linear SVM To perform multi-class prediction with SVM, three SVMs were trained for one-vs-rest prediction, which we denote SVM_{LA}, SVM_{RA}, and SVM_{SHIFT}. The data used to train each binary SVM classifier consisted of a set of (sparse) feature vectors $X = [x_1, x_2, ..., x_T]$ computed from parser state as previously described, and labels $y = [y_1, y_2, ..., y_T]$. For SVM_k $y_i = +1$ if the i^{th} transition was k and -1 otherwise. The linear SVM objective function can be written as:

$$w^* = \frac{\lambda}{2}\min||w||_2^2 + \sum_{i=1}^{T}\max(0, 1 - y_i(w^T x_i + b_i))$$

This loss function was optimised iteratively with subgradient descent; optimisation was terminated when successive epochs failed to decrease training error by more than $\delta = 0.001$.

Prediction To predict parser transitions on test data, the same features used to generate the training data were computed using the current parser configuration. A score was generated by each of the SVMs and the transition with highest score within a set of valid transitions for current parser state was selected.

5 Data (10pt)

The data consisted of 1M tokenized sentences from the 1B word language model benchmark data set, with gold-standard part-of-speech tags and dependency relationships, divided into a training, development, and test set. Shallow statistics of raw data are reported; in the table below, voc. size refers to the number of unique tokens in each data set without any preprocessing. We also identify the number of non-projective sentences in each data set and show that the number is small ($\approx$ 0.1% of sentences). Nevertheless, because arc-standard parsing cannot produce non-projective dependency trees, we removed these examples when training the SVM.

Dataset	# sent.	# non-proj.	# tokens	voc size
Train	600K	639	15.1M	273K
Dev	200K	218	5.03M	150K
Test	200K	192	5.03M	150K

Preprocessing The data was preprocessed by making all characters lowercase to decrease vocabulary size (and hence number of features). Although capitalization can indicate proper nouns, we compensate for this loss by using part-of-speech tags as features. In addition, we drop rare words with < 50 occurrences in the training set, resulting in a final voc. size of 14762. We show the percentage of OOV tokens in the development and test sets with

and without removing removing rare words. While number of OOV words increases after dropping rare words, this step made training much faster. Given more time, we would have liked to retrain the model with a larger vocabulary size.

Data set	% OOV w/ rare words	% OOV w/o rare words
Dev	0.880%	6.12%
Test	0.879%	6.11%

6 Implementation Details (3pt)

This model used the `sklearn` implementation of linear SVM trained with SGD. We describe tuning of L2-regularization hyperparameter λ with grid search below. For all experiments, learning rate was $\frac{1}{\lambda t}$ where t is the current training iteration, with convergence conditions as specified in the model description above.

7 Experiments and Results

Test Results (3pt) Test results were produced by training three one-vs-rest SVMs on the first 200K sentences of the training data, using all feature categories described previously and reg. hyperparameter λ = 3e-6. UAS on the test set was 0.8707.

Development set results In each of the following sections, the reported UAS are generated by running each model on the development data. Additionally, each training subset with n sentences was generated by selecting the first n sentences from the provided full training data of 600K sentences.

# sent.	1w	2w	3w	dep	wd	ic	UAS
1K	+	-	-	-	-	-	0.6218
1K	+	+	-	-	-	-	0.6462
1K	+	+	-	-	-	-	0.6862
1K	+	+	+	-	-	-	0.7442
1K	+	+	+	+	-	-	0.7689
1K	+	+	+	+	+	-	0.7782
1K	+	+	+	+	+	+	0.7972
4K	+	+	+	-	-	-	0.7852
4K	+	+	+	+	+	+	0.8346
10K	+	+	+	-	-	-	0.7931
10K	+	+	+	+	+	+	0.8433

Feature ablations (15pt) To quickly assess the effects of feature choices, feature ablation studies were performed on a small training set of 1K sentences (shown above). The regularization hyperparameter was $\lambda = 0.0001$ for each entry in the table above. We show similar performance trends for slightly larger training sets with 4K and 10K sentences.

Hyperparameter tuning For each experiment in this section, we vary λ and use the full set of features and the first 200K sentences in the training set to train our classifier.

λ	1e-4	1e-5	3e-6	1e-6
UAS	0.8515	0.8653	0.8717	0.8589

Learning curves We vary the number of sentences used to train the SVM. For each model, optimal λ was found with grid search using the final 1K sentences in the training set for validation. We see that UAS begins to plateau after $\approx 60K$ sentences; although using full 600K sentences may have provided marginally better UAS, it is likely that far larger improvement would be made by refining the feature set rather than adding more data.

# sentences	1K	4K	10K	60K	200K
UAS	0.7972	0.8346	0.8433	0.8619	0.8717

8 Error Analysis (7pt)

The model had poor performance on long sentences and sentences with many OOV words; this is shown quantitatively below. Due to the greedy nature of the arc-standard parsing algorithm, errors tend to accumulate, and one incorrect transition can result in assignment of many incorrect dependencies particularly in long sentences; similar trends were reported by [3]. In addition, in sentences with many OOV words, information on word identity, valence, and bilexical affinity is lost, hurting UAS. Interestingly, we find that UAS for non-projective sentences in the development set was quite high.

Class	Examples	Avg. UAS
Non-proj. sentences	49^{th}, 321^{st}, 729^{th}	0.9352
Sentence length	< 10	0.8974
	10 - 30	0.8930
	30 - 50	0.8595
	> 50	0.7848
# OOV words	< 10	0.8858
	10 - 20	0.8322
	> 20	0.6628

Figures 1 and 2 (next page) compare predicted and ground truth dependencies for two illustrative examples:

1. (124^{th}) sentence with 10 words of which 8 were OOV achieved UAS of 0.5714
2. (660^{th}) sentence with 21 words of which 8 were OOV achieved UAS of 0.3158

For both examples, the head of each unknown word is assigned via a right-arc transition before all of its modifiers are attached. Due to propagation of error, the modifiers of those unknown words are also consequently assigned the incorrect head.

9 Conclusion (3pt)

We describe an arc-standard parser with linear SVM oracle for unlabelled dependency parsing that produces modest UAS of 0.8717 and 0.8707 on development and test data, respectively. Qualitative analysis identifies performance on long sentences and sentences with many OOV words as model weaknesses.

References

[1] Ciprian Chelba et al. "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling". In: *arXiv:1312.3005 [cs]* (Mar. 2014). arXiv: 1312.3005. URL: http://arxiv.org/abs/1312.3005 (visited on 05/12/2020).

[2] Joakim Nivre. "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34.4 (2008), pp. 513–553. DOI: 10.1162/coli.07-056-R1-07-027. URL: https://www.aclweb.org/anthology/J08-4003 (visited on 04/24/2020).

[3] Joakim Nivre and Mario Scholz. "Deterministic Dependency Parsing of English Text". In: *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. Geneva, Switzerland: COLING, Aug. 2004, pp. 64–70. URL: https://www.aclweb.org/anthology/C04-1010 (visited on 05/01/2020).

[4] Yue Zhang and Joakim Nivre. "Transition-based Dependency Parsing with Rich Non-local Features". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 188–193. URL: https://www.aclweb.org/anthology/P11-2033 (visited on 04/23/2020).

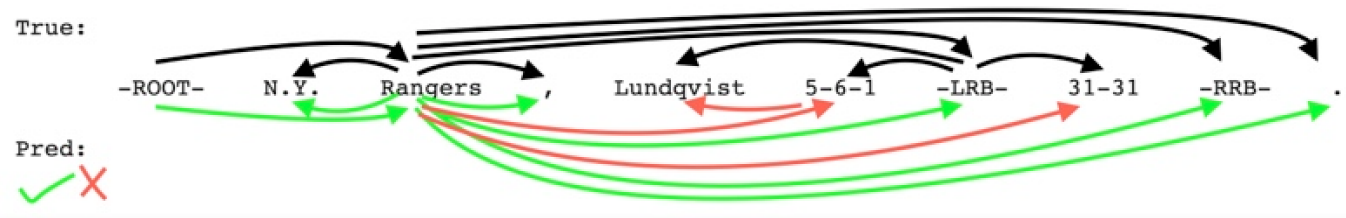

Figure 1: Sentence 124

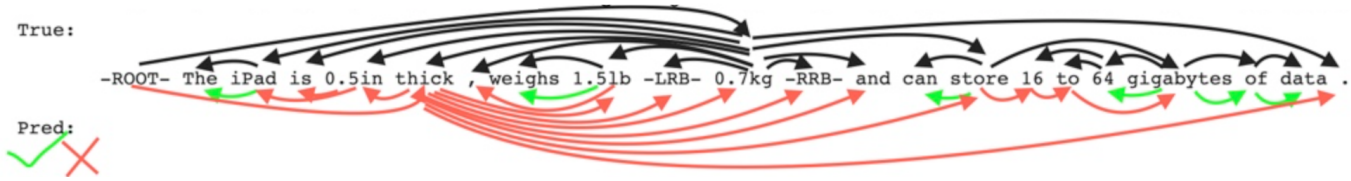

Figure 2: Sentence 660