10-710/11-763 Homework 5
Due: 12/12/2013

1 Introduction

The main purpose of this homework is to implement an Expectation Maximization algorithm. As a by-product, you will also gain familiarity with dependency parsing and implement the inside outside algorithm for inference. Your task is to learn the parameters of the dependency model with valence (DMV) (Klein, 2005) using a training set of POS-tagged English sentences, then use the model to find the Viterbi dependency parses.

2 Dependency model

We recommend that you use the DMV model for this homework, but feel free to use other (projective or non-projective) models if you so desire. Below is an intuitive description of DMV for your convenience. For a formal description, see section 6.2 in (Klein, 2005).

According to this model, a sentence is generated via a series of probabilistic, and conditionally independent steps. The model generates the sentence in Figure 1 as follows. The local probabilities of those generative steps constitute the model parameters:

- at ROOT:

  go $right$[1]:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{ROOT}, rightchildren = 0)$.

  go $left$:

  I decide to generate stuff with $p(\neg\,\text{STOP} \mid \text{ROOT}, leftchildren = 0)$.

  Then, I generate a VBD with $p(\text{VBD} \mid \text{ROOT}, left)$.

  Then, I decide to stop generating stuff with $p(\text{STOP} \mid \text{ROOT}, leftchildren > 0)$.

- at VBD:

  go $right$:

  I decide to generate stuff with $p(\neg\,\text{STOP} \mid \text{VBD}, rightchildren = 0)$.

  Then, I generate a IN with $p(\text{IN} \mid \text{VBD}, right)$.

  Then, I decide to stop generating stuff with $p(\text{STOP} \mid \text{VBD}, rightchildren > 0)$.

  go $left$:

  I decide to generate stuff with $p(\neg\,\text{STOP} \mid \text{VBD}, leftchildren = 0)$.

  Then, I generate a NNS with $p(\text{NNS} \mid \text{VBD}, left)$.

  Then, I decide to stop generating stuff with $p(\text{STOP} \mid \text{VBD}, leftchildren > 0)$.

- at IN:

  go $right$:

  I decide to generate stuff with $p(\neg\,\text{STOP} \mid \text{IN}, rightchildren = 0)$.

  Then, I generate a NN with $p(\text{NN} \mid \text{IN}, right)$.

[1]For this homework, you do not need to stochastically decide which direction to go first. It does not make a difference as far as our evaluation metric is concerned.

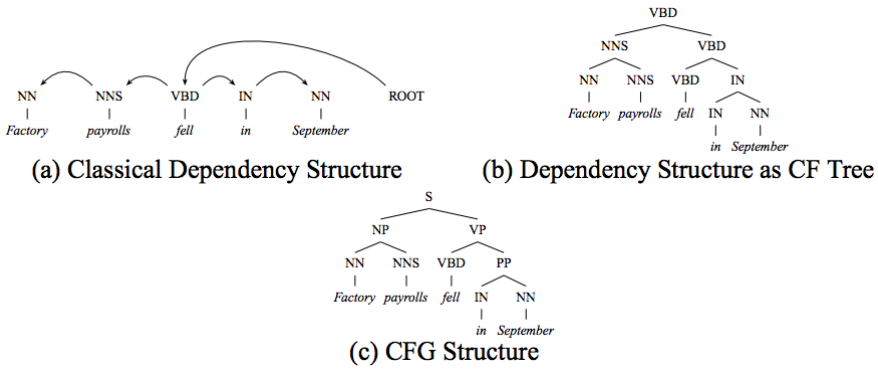

Figure 1: Three kinds of parse structures. The figure is reproduced from (Klein, 2005).

Then, I decide to stop generating stuff with $p(\text{STOP} \mid \text{IN}, rightchildren > 0)$.

go $left$:

I decide to stop generating stuff with $p(\text{STOP} \mid \text{IN}, leftchildren = 0)$.

- at NN[2]:

  go $right$:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{NN}, rightchildren = 0)$.

  go $left$:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{NN}, leftchildren = 0)$.

- at NNS:

  go $right$:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{NNS}, rightchildren = 0)$.

  go $left$:

  I decide to generate stuff with $p(\neg\,\text{STOP} \mid \text{NNS}, leftchildren = 0)$.

  Then, I generate a NN with $p(\text{NN} \mid \text{NNS}, left)$.

  Then, I decide to stop generating stuff with $p(\text{STOP} \mid \text{NNS}, leftchildren > 0)$.

- at NN[3]:

  go $right$:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{NN}, rightchildren = 0)$.

  go $left$:

  I decide to stop generating stuff with $p(\text{STOP} \mid \text{NN}, leftchildren = 0)$.

Note that we replace each token in the sentence with its word class[4] to reduce sparsity.

[2]This is the POS tag of the last token in the sentence

[3]This is the POS tag of the first token in the sentence

[4]The part-of-speech tags provided in `train.conll` and `test.conll` can be used as word classes.

3 Task description

3.1 Data and format

Use `train.conll`[5], a set of 2660 short English sentences, to train your model[6] without using the dependency parse annotations (i.e. unsupervised training). Use `test.conll`[7] for blind testing.

All files described here with the extension `.conll` follow the CoNLL 2007 shared task format[8]. For (unsupervised) training, you are only allowed to consume columns 1 through 6.[9] Columns 7 through 10 of `train.conll` and `test.conll` are the gold dependency parses which you are not allowed to use for training[10]. When you produce your output files, copy all 10 columns for each token, but use column 7 to specify the head of that token according to your output parse.

3.2 Evaluation

Use the evaluation script of CoNLL 2007 shared task, `eval07.pl`[11] to evaluate the output of your parses against `train.conll` and `test.conll`. We are only concerned with the "unlabeled attachement score" in this homework. Hereafter, we use "accuracy" to refer to this score. To help you develop the habit of using a blind test set, we will only reveal `test.conll` 48 hours before the homework is due.

3.3 Left-branching baseline

A simple heuristic for unlabeled dependency parsing is to always attach each token to its right neighbor. This heuristic gives an accuracy of 27.89% on `train.conll`.

3.4 Expectation maximization

Now, consider the DMV model. Had someone given us good parameter values for this model, it would have been easy to find the Viterbi parse of each sentence. Had we known the correct parses at training time, it would have been easy to compute the maximum likelihood estimates of the model parameters. As discussed in class, the expectation maximization framework lends itself to this kind of "chicken and egg" problems. In this homework, it is required to implement the EM algorithm to estimate the parameters of a dependency parsing model such as DMV.

Expectation maximization is a general algorithm for estimating model parameters from incomplete data. Starting from some initialization of the model parameters[12], you run several iterations each consisting of an E-step and an M-step, until you reach convergence. In the E-step, you compute the sufficient statistics required in the M-step, conditioning on the observed data and the old model parameters. In the M-step, you use those sufficient statistics to re-estimate model parameters. Thanks to the categorical distribution parameterization of the DMV model, there is a closed form solution for maximum likelihood estimation of the model parameters in the M-step, which is the normalized (expected)[13] count of corresponding events. To compute these expected counts,

[5] `www.cs.cmu.edu/~wammar/misc/hw5/train.conll`

[6] Feel free to split it into train and development sets if you like.

[7] This data set will be available 48 hours before the due date at `www.cs.cmu.edu/~wammar/misc/hw5/train.conll`

[8] `http://nextens.uvt.nl/depparse-wiki/DataFormat`

[9] To replicate (Klein, 2005), you only need to use columns 4 and 5

[10] Failure to follow this instruction in particular will be strictly treated as cheating.

[11] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage`

[12] Section 3.3 in (Smith, 2006) describes a few easy initializations of the model parameters and how they perform.

[13] Training data only consists of sentences, not parses. So, it is not possible to count the number of times an event such as (a VBD generates an NNS to the left) occurs. Instead, we can use the old model parameters to define a distribution over dependency parses of a sentence, and compute the expected count of an event with respect to this distribution.

you need to implement the inside-outside algorithm[14] which was presented in class. Runtime of the vanilla inside-outside algorithm for dependency parsing is $O(n^5)$ for a sentence of length n. Eisner (2000) and Johnson (2007) discuss faster algorithms with cubic runtime.

We recommend that you work out a tiny example by hand in order to have a clear picture of what quantities need to be inferred in the E-step, how to do inference efficiently, then how to use those quantities to re-estimate model parameters.

3.5 Be competitive!

After having a correct implementation of EM, here are a few ideas to help you compete.

- use better initialization of model parameters (e.g. $Ad-hoc^*$ (Spitkovsky et al., 2009), IBM model 1 (Gimpel and Smith, 2012)), or use multiple random initializations (Smith, 2006).
- use other word classes than provided POS tags (e.g. Brown clusters[15], split-merge (Petrov et al., 2006)).
- use deterministic annealing or structural annealing methods (Smith, 2006).
- learn from shorter examples first (Spitkovsky et al., 2009).
- use a log-linear parameterization of the local conditional probabilities (Berg-Kirkpatrick et al., 2010).

4 Deliverables, grading, and fair use of gold parses:

Use your Google directory to submit the following:

- your code in a compressed tarball, `hw5_code_andrewid.tgz`.
- a brief writeup `hw5_writeup_andrewid.pdf` (1 or 2 pages) describing:
  1. one plot of the training data log-likelihood on the vertical axis and the EM iteration number on the horizontal axis. If you use different initializations, select any one of them.
  2. the criterion you use to determine the convergence of EM.
  3. how you initialize the model parameters.
  4. using the model with the best training set log-likelihood, what is the accuracy of the Viterbi dependency parses when evaluated on the training set, and when evaluated on the test set?
  5. which model achieves the best accuracy when evaluated on the training set? What is that "oracle" accuracy?
  6. which model achieves the best accuracy when evaluated on the test set? What is that "oracle" accuracy?
  7. did you use the provided gold dependency parses for any purpose other than evaluation?
- the Viterbi dependency parses of the training set, using the model with the best log-likelihood on the training set, `hw5_bestll_trainparses_andrewid.conll`.
- the Viterbi dependency parses of the test set, using the model with the best accuracy on the test set, `hw5_oracle_testparses_andrewid.conll`.

[14] A self-contained description can be found at `http://www.cs.columbia.edu/ mcollins/io.pdf`, also at Figures 1.6 and 1.8 at (Goodman, 1998)

[15] an implementation can be found at `https://github.com/mheilman/tan-clustering`

Your correct implementation of EM for the DMV model is worth full marks in this assignment (12 points). Everyone who beats the accuracy of the right-branching baseline on the training set gets an additional bonus point. The three submissions with the highest accuracy on the test set also get two additional bonus points.

You may use the gold dependency parses for evaluation, oracle experiments, and error analysis. You may not use them in any other way. Failure to do so will be strictly treated as cheating. Also, you may not use the data for other purposes than this homework.

For questions, please email Waleed at waleed.ammar@gmail.com

References

T. Berg-Kirkpatrick, A. Bouchard-Cote, J. DeNero, and D. Klein. 2010. Painless unsupervised learning with features. In *Proc. of NAACL*.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*.

K. Gimpel and N. A. Smith. 2012. Concavity and initialization for unsupervised dependency parsing. In *Proc. of NAACL*.

J. T. Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, Harvard University.

M. Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable cfgs with unfold-fold. In *Proc. of ACL*.

D. Klein. 2005. *The Unsupervised Learning of Natural Language Structure*. Ph.D. thesis, Stanford University.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. of ACL*.

N. A. Smith. 2006. *Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text*. Ph.D. thesis, Johns Hopkins University.

V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. 2009. Baby steps: How "less is more" in unsupervised dependency parsing. In *Pro. of NIPS 2009 workshop on Grammar Induction, Representation of Learning and Language Learning*.