11-763 Spring 2018: Homework 3

Due: April 6 2018, 23:59 PM

Instructions

- You can use any programming language for this homework.
- We encourage a healthy discussion on homeworks, but the final proof/solution must be written down individually.
- Cite any references that were used to solve any of the questions.

PS: *Please follow the naming convention described in the last section. It will make grading easier for us.*

1 Introduction

The purpose of this homework is to gain familiarity with dual decomposition as well as inference in generative models. First, you will implement a first-order HMM decoder for part-of-speech tagging and a CYK parser for context-free parsing. Then, you will use dual decomposition to integrate the tagger and the parser in an efficient decoder that maximizes the sum of the two models log-probabilities. A similar integration has been shown to improve the performance of both the tagging and the parsing subproblems [2].

2 Tasks

2.1 POS Tagger

Your first task is to implement a decoder for a first-order hidden Markov model for Arabic part-of-speech tagging. The decoder expects three input files which specify HMM transition probabilities, HMM emission probabilities, and input sentences. Given input sentence $\mathbf{x}$, the job of the tagger is to find the highest scoring tag sequence $\hat{\mathbf{z}} = \text{argmax}_{\mathbf{z}} \log p_{hmm}(\mathbf{z}|\mathbf{x})$. Feel free to modify your HMM decoder from homework 1 for this task. `sentence_boundary` is a special POS tag which is used to mark the beginning and end of a sentence.

You can evaluate the output of your tagger (e.g. `candidate-postags`) against gold standard POS tags of the development set `dev_sents` as follows:

```
./eval.py --reference_postags_filename=dev_postags \
          --candidate_postags_filename=candidate-postags
```

2.2 Context free grammar parser

Your second task is to implement a parser for a Chomsky normal form probabilistic context-free grammar (PCFG) of Arabic syntax. The parser expects two input files which specify the PCFG, and input sentences. Given input sentence $\mathbf{x}$, the job of the tagger is to find the highest scoring derivation $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log p_{pcfg}(\mathbf{y}|\mathbf{x})$.

You can evaluate the output of your parser (e.g. `candidate-parses`) against gold standard parses of the development set `dev_sents` as shown below. The evaluation script provided `eval.py` reports the precision and recall on the binary trees, contrary to the common practice of reporting precision and recall with the original grammar.

```
./eval.py --reference_parses_filename=dev_parses \
          --candidate_parses_filename=candidate-parses
```

2.3 Dual decomposition

Your third task is to perform joint decoding in a combined model by using dual decomposition. Given input sentence $\mathbf{x}$, you will find the highest scoring derivation:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log_{pcfg}(\mathbf{y}|\mathbf{x}) + \log_{hmm}(l(\mathbf{y})|\mathbf{x}) \quad (1)$$

where $l(\mathbf{y})$ maps a derivation $\mathbf{y}$ to the sequence of POS tags in $\mathbf{y}$. Since the PCFG and HMM capture different types of information, combining their scoring mechanisms may yield better predictions (in terms of both parsing and POS tagging accuracy) than with each model in isolation.

We use the same notation and definitions used in Rush and Collins [1]: τ is the set of POS tags, $y(i,t) = 1$ iff parse tree $\mathbf{y}$ has a tag $t \in \tau$ at position i in the sentence, $y(i,t) = 0$ otherwise. $z(i,t) = 1$ iff POS tagging sequence $\mathbf{z}$ has a tag $t \in \tau$ at position i, $z(i,t) = 0$ otherwise. $u(i,t)$ is a Lagrange multiplier enforcing the constraint $y(i,t) = z(i,t)$. ***PS:*** *step size δ_k and K are hyperparameters. These would have to be tuned based on the development set.*

The dual decompostion algorithm for integrating parsing and tagging is as follows:

Algorithm 1 The Dual Decompostion Algorithm

for k = 1, 2, ..., K **do**
  initialization $\leftarrow 0$;
  $\hat{\mathbf{y}} \leftarrow \arg\max_{\mathbf{y}} \log_{pcfg}(\mathbf{y}|\mathbf{x}) + \sum_{i,t} u(i,t)y(i,t)$;
  $\hat{\mathbf{z}} \leftarrow \arg\max_{\mathbf{z}} \log_{hmm}(\mathbf{z}|\mathbf{x}) - \sum_{i,t} u(i,t)z(i,t)$;
  if $\hat{\mathbf{y}}(i,t) = \hat{\mathbf{z}}(i,t) \forall i,t$ **then**
    print $\hat{\mathbf{y}}$;
  else
    $u(i,t) \leftarrow u(i,t) - \delta_k(y(i,t) - z(i,t))$;

Modify your implementation of the POS tagger and the CFG Parser in order to account for the extra Lagrange multiplier terms, without degrading the runtime of the tagger and the parser. Then, implement the dual decomposition algorithm as described in Figure 1. The decoder expects four input files which specify the HMM transitions, HMM emmisions, a PCFG and input sentences. Given an input sentence $\mathbf{x}$, solve the optimization problem (1), finding $\hat{\mathbf{y}}$. The decoder outputs two files: one for the parse trees and another for the POS tag sequences. You can evaluate the two output files of your tagger using the development set as shown earlier.

3 Data Formats

3.1 Parameter files

Three parameter files are provided: `hmm_trans`, `hmm_emits`, `pcfg`. Each file specifies a number of conditional distribution $p(decision|context)$. Each line consists of three tab-separated columns:

```
context     decision     log p(decision | context)
```

In `pcfg`, the start non-terminal is `S` and the *decision* consists of either one terminal symbol (e.g. dog) or two space-separated nonterminal symbols (e.g. ADJ N).

3.2 Plain text files

One plain text file is provided: `dev_sents`. It consists of one tokenized sentence per line. Tokens are space-separated.

3.3 POS tagging files

We describe the format of the provided gold standard POS tags file `dev_postags` as well as your output for the HMM tagging task and the dual decomposition task. Each line consists of a spaceseparated POS tag sequence. The number of tags *must* be equal to the number of tokens in the corresponding sentence (i.e. use `sentence_boundary` tags at the beginning and the end while decoding to find the Viterbi POS tag sequence, but do not write `sentence_boundary` tags to the output file).

3.4 Parse files

We describe the format of the provided gold standard parse trees file `dev_parses` as well as your output for the CFG parser and dual decomposition task. Each line consists of a complete syntactic derivation for the corresponding sentence in a plain text file. For example, in the simple parse below, `S` is the root with two children: `NP` and `V`. `NP` has two children: `ADJ` and `N`. `ADJ`, `N` and `V` each has a single child: `bad`, `tornado` and `coming`, respectively.

```
( S ( NP ( ADJ bad ) ( N tornado ) ) ( V coming ) )
```

3.5 Deliverables

In this homework, your deliverable files should be named as follows: `dev-plain-postags-andrewid`, `dev-plain-parses-andrewid` , `dev-dd-postags-andrewid` , `dev-dd-parses-andrewid`, `proof-andrewid.pdf`.

1. **POS Tagger:** output of your tagger using input files `hmm_trans`, `hmm_emits`, `dev_sents`.
2. **Context free grammer parser:** output of your parser using input files `pcfg`, `dev_sents`.
3. **Dual Decomposition:**
   (a) output files of your dual decomposition decoder with input files `hmm_trans`, `hmm_emits`, `pcfg`, `dev_sents`.
   (b) show that the algorithm given in Algorithm 1 indeed optimizes the objective in Equation 1, when it converges.
4. **Code:** The source code you built to implement the solution with a short readme on how to run it.

3.6 Grading

Our solution code achieves a tagging accuracy of approximately 97% and a parsing F1 of approximately 83 when using independent decoding for each model. For our code, dual decomposition yields an absolute gain of approximately 0.4% for tagging accuracy and 0.4 F1 for parsing. For grading, we will assign full credit for solutions that achieve or improve upon these scores. In general, we are most interested in improvements gained by running dual decomposition. Solutions that do not match these reference scores my still potentially get full or partial credit, though we will not be able to specify these lower cutoffs in advance.

References

[1] A. M. Rush and M. Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 2012.

[2] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics, 2010.