**11-763 Homework 3**
**Due: 11/03/2015**

# 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 the dual decomposition method 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 (Rush et al., 2010).

# 2 Tasks

The necessary input files for performing the tasks described below are provided in the tgz bundle: `hw3_files.tgz`.

## 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 $\hat{\mathbf{z}} = \arg\max_{\mathbf{z}} \log p_{hmm}(\mathbf{z} \mid \mathbf{x})$. Feel free to modify your CRF 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
```

**Deliverable**: submit the output of your tagger using input files `hmm_trans`, `hmm_emits`, `test_sents`.

## 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 $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log p_{pcfg}(\mathbf{y} \mid \mathbf{x})$.

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
```

**Deliverable**: submit the output of your parser using input files `pcfg, test_sents`.

## 2.3 Dual decomposition

Given input sentence $\mathbf{x}$, it is required to find the highest scoring derivation:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log p_{pcfg}(\mathbf{y} \mid \mathbf{x}) + \log p_{hmm}(l(\mathbf{y}) \mid \mathbf{x}) \quad (1)$$

where $l(\mathbf{y})$ maps a derivation $\mathbf{y}$ to the sequence of POS tags in $\mathbf{y}$. Since each of the PCFG and the HMM captures different types of information, combining both models may improve both the accuracy of parsing as well as POS tagging, compared to solving the two problems in isolation.

We use the same notation and definitions used in (Rush and Collins, 2012): $\mathcal{T}$ is the set of POS tags, $y(i,t) = 1$ iff parse tree $\mathbf{y}$ has a tag $t \in \mathcal{T}$ at position $i$ in a sentence, $y(i,t) = 0$ otherwise. $z(i,t) = 1$ iff POS tagging sequence $\mathbf{z}$ has a tag $t \in \mathcal{T}$ at position $i$, $z(i,t) = 0$ otherwise. $u(i,t)$ is a Lagrange multiplier enforcing the constraint $y(i,t) = z(i,t)$.

The dual decomposition algorithm for integrating parsing and tagging, adapted from (Rush and Collins, 2012), is as follows:

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

**Algorithm 1:** The dual decomposition algorithm.

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 HMM transitions, HMM emissions, 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 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.

**Deliverables:**

- show that the algorithm given in Figure 1 indeed optimizes the objective in Equation 1, when it converges.
- submit the output files of your dual decomposition decoder with input files `hmm_trans, hmm_emits, pcfg, test_sents`.

## 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 $p(\textit{decision} \mid \textit{context})$. 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

Two plain text files are provided: `dev_sents, test_sents`. Each of the two files consist 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 space-separated 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))
```

## 4 Submission procedures

Please submit your homework to `cmu-instructors-11763-fa2015@googlegroups.com`. In this homework, your deliverable files should be named as follows: `test-plain-postags-andrewid` , `test-plain-parses-andrewid` , `test-dd-postags-andrewid` , `test-dd-parses-andrewid` , `proof-andrewid.pdf`

## References


A. Rush and M. Collins. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. In *Tutorial at ACL.*

A. Rush, D. Sontag, M. Collins, and T. Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. EMNLP.*