Assignment 4

CMSC 473/673 — Introduction to Natural Language Processing

Due Monday December 11, 2017, 11:59 AM

Item	Summary
Assigned	Tuesday November 21st, 2017
Due	Monday December 11th, 2017
Topic	Syntax
Points	160

In this assignment you will understand and gain experience with syntactic models (formalisms).

As with the previous assignments, you are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4
Points	30	30	50	50

As before, the first page provides a **task list**. This task list captures the questions, and details (without other explanatory text) the tasks you are to do and what your completed assignment should answer. Following the task list are the **full questions**. The full questions do *not* require you to answer additional questions.

The task list enumerates what you must do, but it does not necessarily specify *how*—that's where the full questions come in. Therefore, you **should still read and reference** the full questions.

What To Turn In Turn in a writeup in PDF format that answer the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment using the `submit` utility on GL. The course id is `cs473_ferraro`. This assignment's id is `a4`.

Please run your code on GL before you submit it. In particular, make sure that your code does not use any local *copies* of data sets.

Task List

1. Provide a polynomial time and space algorithm to give the number of derivations a sentence of N words has under a (P)CFG grammar $\mathcal{G}$; you may assume $\mathcal{G}$ is in Chomsky Normal Form. In giving the algorithm, be sure to provide some justification for why your algorithm is correct, including its polynomial time and space bounds.

2. Read and write a half page summary[1] and review of Charniak and Johnson (2005).

3. Write a sentence generator (one that randomly generates sentences from a WCFG) and develop a grammar of English that addresses at least three different types of linguistic phenomena. Turn in your weighted CFG, your sentence generator, a writeup of how you modified the grammar. In your writeup, be sure to analyze the rules you added, why you added them, and how you arrived at the weights. Discuss any difficulties you had in setting grammar weights (e.g., were you suprised or frustrated with the Viterbi parse of certain sentences?). Also describe how your generation program works (e.g., how do you choose which non-terminals to expand, how did you determine what symbols were non-terminals, etc.). Be sure to cite any sources you consult.

4. Compare (syntactically) two languages using the Universal Dependencies. This comparison must use syntax (labeled or unlabeled dependency relations), but it is up to you on how to use the syntax *and* how to do the comparison. You may use any code *you* have written for this class, as well as external versions of models covered in this class, e.g., sklearn's, of Naïve Bayes and logistic regression/maxent classification.

   Turn in a PDF writeup and any code you write to complete this question. Clearly identify what languages (and specific treebanks) you chose, the approach(es) you used, and the results of that/those approach(es). Discuss and analyze these results and limitations of your approach(es); identify at least one future idea to expand your approach.

[1] Single spaced, regular font, and one column is fine.

Full Questions

1. (**30 points**) Provide a polynomial time and space algorithm to give the number of derivations a sentence of N words has under a (P)CFG grammar $\mathcal{G}$; you may assume $\mathcal{G}$ is in Chomsky Normal Form. In giving the algorithm, be sure to provide some justification for why your algorithm is correct, including its polynomial time and space bounds. (You do not have to give a full, formal proof.)

   In terms of the length N of a sentence, algorithms that are (asymptotically) worse than cubic time and quadratic space will receive partial (but not full) credit.

2. (**30 points**) Read and write a half page summary[2] and review of Charniak and Johnson (2005). It is available at

   `http://aclweb.org/anthology/P/P05/P05-1022.pdf`.

   In addition to discussing the basic methodology and findings of this paper, identify findings you found interesting, surprising, or confusing. What is the overall takeaway (for you) from this paper?

   The full Bibtex citation is

```
@InProceedings{charniak-johnson:2005:ACL,
  author    = {Charniak, Eugene  and  Johnson, Mark},
  title     = {Coarse-to-Fine n-Best Parsing and
     MaxEnt Discriminative Reranking},
  booktitle = {Proceedings of the 43rd Annual Meeting of
     the Association for Computational Linguistics
     (ACL'05)},
  month     = {June},
  year      = {2005},
  address   = {Ann Arbor, Michigan},
  publisher = {Association for Computational Linguistics},
  pages     = {173--180},
  url       = {http://www.aclweb.org/anthology/P05-1022},
  doi       = {10.3115/1219840.1219862}
}
```

3. (**50 points**) In this question you will develop a grammar of English.[3] You grammar will need to address at least three different types of linguistic phenomena (see **Linguistic Phenomena** below). To help you write your grammar, it'll be helpful to have both a (sentence) generator and a parser. We'll provide you the parser; you have to write the generator.

   What to turn in Turn in

   (a) your weighted CFG;

   (b) your sentence generator; and

   (c) a writeup of how you modified the grammar.

[2] Single spaced, regular font, and one column is fine.

[3] Really, a very, very, very small subset of English.

In your writeup, be sure to analyze the rules you added, why you added them, and how you arrived at the weights. You should provide example output from your generator to help explain your grammar changes (and to show that your changes work). Discuss any difficulties you had in setting grammar weights (e.g., were you suprised or frustrated with the Viterbi parse of certain sentences?). Also describe how your generation program works (e.g., how do you choose which non-terminals to expand, how did you determine what symbols were non-terminals, etc.).

In capturing the linguistic phenomena, you may examine real life sentences as examples. Be sure to cite any sources you consult.

A Weighted CFG The grammar you write should be a (non-negatively) *weighted* context free grammar (WCFG). The weights you assign should be non-negative, but they may be unnormalized probabilities. The parser we provide will renormalize the rules once the full grammar has been read in; your generator should do the same.

The grammar you write does *not* have to be in CNF.

Each grammar rule should be on its own line, of the form

$$w \quad X \texttt{ - - > } \quad Y_1 \dots Y_m$$

where $w > 0$ is the weight of the $X \rightarrow Y_1 \dots Y_m$ rule. The Y_is form the right hand side (RHS), while the X forms the left hand side (LHS). Each of the Y_i may be a terminal or non-terminal.[4] The following grammar fragment shows an example. **You may use this fragment as a starting point.**

```
1 ROOT --> S .
1 ROOT --> S !
1 ROOT --> is it true that S ?
1 S --> NP VP
1 VP --> Verb NP
1 NP --> Det NPish
1 NP --> NP PP
1 PP --> Prep NP
1 NPish --> Adj NPish
1 NPish --> Noun
```

Note: the root symbol is defined as the very first LHS non-terminal listed in the grammar. The rewrite symbol must be two dashes followed by a right angled bracket.

Minimum grammar requirements and lexicon generation: What words you add to the lexicon is up to you. Do not try to be overly complicated; as you consider more phenomena, this could cause you trouble. At a minimum though, your grammar needs to handle verbs, nouns, adjectives, and prepositions.

A WCFG Sentence Generator Write a generation program **gen-sent**. This program must generate random word sequences from a provided grammar. This should help you debug your grammar as you're writing it.

Your program should have one required argument and three optional flags/arguments:

```
gen-sent [-t] [-r ROOT-SYMBOL] [-n NUM-TO-GEN] grammar-file
```

[4] For the parser, there's a practical limit of 128 RHS symbols *per rule*. Please only approach this limit with excellent justification.

The `grammar-file` argument should be the path to a WCFG; the WCFG must have the form as described above.

By default, your program should generate a single sequence (`NUM-TO-GEN = 1`), from a specified root node (by default, set `ROOT-SYMBOL` to the first LHS symbol found in your grammar); by default, only the words should be printed. Add a `-t` flag (by default off/false) to display the tree that generates the word sequence.

A WCFG Parser We're providing you a WCFG parser; access it on GL at

`/afs/umbc.edu/users/f/e/ferraro/pub/473-f17/a4/cky`

It is a CKY Viterbi parser written in C.[5] As mentioned above, in order to parse, it will automatically convert weighted (non-negative, but not necessarily probabilistic) general CFG rules to a probabilistic CNF CFG; it will then automatically reverse the conversion when showing the Viterbi tree.

The basic usage is

```
$ ./cky input-sentences grammar-file
```

Here, `input-sentences` is a file with a single sentence per ('\n'-separated) line, and `grammar-file` is a WCFG of the form above. In typical Unix fashion, this will read sentences from stdin (the console) when `input-sentences` is the single character `-`. For example, the following execution requires you to type the sentence "the president ate a sandwich ." at the terminal:

```
$ ./cky - grammar
the president ate a sandwich .
-9.875088(ROOT (S (NP (Det the) (Noun president)) \
   (VP (Verb ate) (NP (Det a) (Noun sandwich)))) .)
^D
```

(The `^D` is the sequence "Ctrl D.")

Linguistic Phenomena You must write a grammar to capture at least three linguistic phenomena. Below are five examples; you may select from them, or you may choose others.

Note: The three you capture must work together. That is, your solution for "adjective order" must not break any of the rules you wrote for "a" vs. "an."

(a) noun-verb agreement (singular vs. plural) for present tense verbs : Have your grammar handle number agreement between nouns and verbs. The verbs will need to be in the present tense, as English past (and future) tense verbs do not change form for agreement.

(b) transitive vs. di-transitive vs. intransitive verbs *and some* of their direct/indirect object alternation patterns: Certain verbs are transitive, meaning they can take direct objects ("Chris ran the marathon"); others are di-transitive, meaning they can take direct objects and indirect objects ("Chris gave Pat the book"); and others are intransitive, meaning they do not take any objects ("Chris ran toward the hills"). These may take certain modifiers; both these modifiers and objects exhibit a certain amount of movement capability, e.g., "Chris ran toward the hills

[5] It was originally written by Mark Johnson; then edited by Matt Post; it has been (very slightly) modified for this class. You can copy the source from `/afs/umbc.edu/users/f/e/ferraro/pub/473-f17/a4/mjp-cky`; follow the `Makefile` recipe for `llncky`.

with vigor," "Toward the hills Chris ran with vigor," "Chris ran the marathon with vigor," and "Chris ran the marathon with vigor." We can also say "Chris gave the book to Pat." Extend your grammar to handle some object and modifier alternation patterns among transitive, di-transitive, and intransitive verbs.

(c) verb tenses: English verbs commonly used can generally be analyzed on two dimensions: a temporal one, and a continuation (called *aspect*). The temporal notion is broken into three categories: present, past, and future. Aspect can be broken into three categories as well: simple, perfect, and progressive. For example, the simple present is used in "Chris works [on the assignment]," the perfect past in "Chris had worked [on the assignment]," and the progressive future in "Chris will be working [on the assignment]." Extend your grammar to handle the nine temporal-aspect tense pairs. Be sure your grammar can handle stacking the tenses together; as in the future perfect progressive, "Chris will have been working [on the assignment]."

(d) "a" vs. "an": Generally, we use "a" before nominals that do not begin with a vowel (sound) and "an" before nominals that do. For example, we say "a banana," and "an apple," but we also say "a red apple." Make your grammar handle this alternation.

(e) adjective order: There are many different ways that adjectives can modify nominals. Some modify the number ("two"), others modify a subjective value ("great"), and other modify physical/observable attributes (e.g., size, color and shape). With these modifiers, it is more natural to say "two great big green houses" than it is to say "*great green two big houses." Write rules to capture adjective order.

4. (**50 points**) Using the Universal Dependency treebanks—available at

   `http://universaldependencies.org/`—compare (quantitatively and qualitatively) two different languages.[6]

   You **must** compare the languages using syntax (labeled or unlabeled dependency relations); *how* you do so is up to you. Your approach could be lexicalized, i.e., combining syntactic relations with the observed words; it could use part of speech information; or it could be purely based on the relations themselves. You could compare common sequences of directed dependency relations, or you could acquire statistics on how often a token (or part-of-speech tag) was a (labeled) governor of another. Having acquired any of these statistics, you could apply a Church and Hanks (1989) style analysis (recall that approach from Assignment 1); you could perform HMM-based sequence modeling on dependency subtrees (where the observed words are the words or parts of speech, and the states are directed and labeled dependency relations); or you could perform a language ID classification task, where you predict the language given dependency relations between parts of speech (if you do this, do not use the words themselves; don't make any analysis or classification too trivial). Or, you can do some other experiment/analysis. Have fun with it!

   You may use any code *you* have written for this class. You may also use external versions of models covered in this class, e.g., sklearn's, of Naïve Bayes and logistic regression/maxent classification.

   What to turn in Turn in a PDF writeup and any code you write to complete this question (e.g., analysis, classification, counting, summarization, plotting, etc.). In your writeup, clearly identify what languages (and specific treebanks) you chose, the approach(es) you used, and the results of that/those approach(es). Discuss and analyze these results: did your techniques highlight any commonalities between the languages? Finally, discuss limitations of your approach(es) and at least one future idea to expand your approach.

[6] Different languages here means languages under different country flags. For example, the "Original English" and "LinES" treebanks both appear under the Great Britain flag; they do *not* count as separate languages.