Assignment 5

CMSC 473/673 — Introduction to Natural Language Processing

Due Monday December 10, 2018, 11:59 AM

Item	**Summary**
Assigned	Monday November 26th, 2018
Due	Monday December 10th, 2018
Topic	EM & Syntax
Points	120

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	25	25	40	30

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 on the submission site:

https://www.csee.umbc.edu/courses/undergraduate/473/f18/submit.

Be sure to select "`Assignment 5`."

Full Questions

1. (**25 points**) In this question, you will get familiar with EM for HMMs using the ice cream spreadsheet created by Jason Eisner, available at `http://www.cs.jhu.edu/~jason/465/hw-hmm/lect24-hmm.xls`. You will answer questions related to different initial parameters (the red cells) and try to find out how the initial transition probabilities, emission probabilities and observations affect the end results. This will give you a chance to play around with a working implementation of EM, and to see what happens from iteration to iteration.

   Turn in your responses for the following questions. In answering the following, **always** consider how the various graphs, reconstructed weather (per day and pair of days), and final perplexity can change.

   (a) What will happen if you strongly believe that one day's weather should not be the same as the day after it? This corresponds to $p(H|C)$ and $p(C|H)$ being high.

   (b) What will happen if you strongly believe that one day's weather has nothing to do with the day after it, i.e., $p(H|C) = p(C|H) = p(C|C) = p(H|H)$?

   (c) What will happen if you strongly believe that the colder the weather gets the more ice cream Jason wants to eat, e.g., $p(1|C) = 0.1$ and $p(2|C) = 0.2$ while $p(1|H) = 0.7$ and $p(2|H) = 0.2$?

   (d) What will happen if you strongly believe Jason just wants to eat ice cream and it has nothing to do with weather, i.e., $p(1|C) = p(2|C) = p(1|H) = p(2|H) = 0$?

   (e) What will happen if there are actually some additional, unmodeled factors that affect the number of ice cream Jason eats? For example, all of a sudden ice cream gets too expensive. To be specific, consider the case where

   - For the first 11 days he can eat whatever amount of ice cream he want.
   - Then for the next 11 days things become tight and he can only buy at most 2 ice creams per day.
   - And finally he's starting to run out of money, so he can only afford 1 ice cream per day.

   (f) Try out the new sequence of ice creams eaten: [2, 2, 1, 3, 1, 2, 2, 1, 3, 3, 1, 2, 1, 2, 3, 3, 3, 3, 1, 1, 2, 2, 2, 2, 3, 3, 3, 1, 3, 2, 1, 2, 1, 3] Try different initial probabilities to estimate the actual probabilities behind that observation and submit the weather you predicted. Explain why you think that is the distribution.

   - How will initial reconstruction of the weather (the leftmost graph) change?
   - How will the final graph after 10 iterations change?
   - What is the $p(1|H)$ after 10 iterations? Explain why this happened and what happened in each re-estimation step.

2. (**25 points**) Read and write a half page summary[1] 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

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

```
@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. (**40 points**) In this question you will develop a grammar of English.[2] 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.

   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

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

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

where $w > 0$ is the weight of the $X \rightarrow Y_1 \ldots Y_m$ 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.[3] 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
```

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-f18/a5/cky`

It is a CKY Viterbi parser written in C.[4] 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
```

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

[4] 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-f18/a5/mjp-cky`; follow the `Makefile` recipe for `llncky`.

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 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. (**30 points**) This question asks you to do a thought experiment: you do *not* have to write any code, but what you describe in your PDF writeup must be easily implementable. As a diagnostic: if one of your classmates provided you with a description similar to what you write, would you be able to implement it?

   Task: In dependency parsing, given two words (tokens) w_i and w_j, we need to predict a relationship r such that either the tuple $r(w_i, w_j)$ is likely, or that no relation, with w_i as governor and w_j as dependent, holds between them. For this question provide clear descriptions for two different classifier models for doing so. One model should be a baseline model (e.g., very simple to implement and requiring minimal-to-no annotated language data or fancy statistical models); the other should be a more complex model. The baseline model could be a simple count-based model (counting what?), while the complex model could define features based off of other linguistic annotations.

   Formally, describe two different classifiers that, given a potential governor w_i and potential dependent w_j, predicts the relationship r such that $r(w_i, w_j)$ is in that sentence's dependency parse, or that no valid relation exists between them. Mathematically, this can be thought of as formulating a MAP classifier $p(r \mid \text{gov}{=}w_i, \text{dep}{=}w_j)$. Your classifiers may rely on **lexical, word embedding, part-of-speech, and/or morphological features** of w_i and w_j. These classifiers must be position aware: it should matter whether w_i is given as the potential governor or as the potential dependent. (Hint: any features you define should be defined with respect to the relation under consideration.)

   What to Turn in: Turn in a description of what your two models would be and what features you would define (what parts of the data would you use?). Try to justify why you think those features would be useful. Describe how you would encourage generalization in the models (e.g., feature design, smoothing, regularization, etc.). State what data you looked at to formulate these features, how those data were helpful, and how you would properly develop and evaluate your classifiers.

   Example Data: You may assume that you would be building these classifiers from the Universal Dependency data. In this case, given any two pairs of words w_i and w_j, you would be predicting the dependency relationship that exists between them, if any. This dependency relationship is given by the eighth column (the `DEPREL` label[5], in orange below) from the first six columns. You should ignore any columns after the eighth (denoted in gray, below).

```
1   A     a     DET   DT   Definite=Ind|PronType=Art  2  det            2:det               _
2   team  team  NOUN  NN          Number=Sing         6  nsubj  6:nsubj|8:nsubj:xsubj   _
3   from  from  ADP   IN               _              5  case           5:case              _
4   the   the   DET   DT   Definite=Def|PronType=Art  5  det            5:det               _
```

   The seventh column for a token w_j gives the token index of the token w_i that is the head of the relation between w_i and w_j. For example, in the first two lines, the "2" in the seventh column of the first line indicates that the second token ("team") is the syntactic head/governor of the `det` relationship from "team" to "A." That is, there is a relationship `det(governor=team, dependent=A)` and *not* `det(governor=A, dependent=team)`.

   The above example shows the first four tokens from a particular sentence from the EWT training set. The two systems you describe (one as a baseline, one that is linguistically aware) must be able to take any two tokens and predict the relationship between them. For example, your systems should try to predict that, for the first two tokens when given as the pair `(governor=team, dependent=A)`, there is a `det` relationship from the second to the first (e.g., `det(team, A)`).[6] Note the above UD example does not include word embeddings, though those are generally available as external/supplementary resources. You *may* use word embeddings in your thought-experiment.

[5] `http://universaldependencies.org/format.html`

[6] However, if provided the pair `(governor=A, dependent=team)`, it would need to predict "no relation."