Computational Linguistics I *CMSC723, Fall 2010*

P1: Part of Speech Tagging

Hand in at: `http://www.cs.utah.edu/~hal/handin.pl?course=cmsc723`.

Introduction

In this project, we will explore two techniques for part of speech tagging: rule based systems and hidden Markov models. **Big fat warning:** you will be evaluated based on the performance of your systems on *unseen* text, which you will be able to test on *once per day*; so you get a *big advantage* if you start early!

In all parts of this assignment, we will use a stripped down version of the Penn Treebank tag set. Our tags are:

Noun All nouns, *including* proper nouns and pronouns (and posessive pronouns!)

Prep All prepositions

Verb All verbs, but not auxiliaries or gerunds

Det All determiners/articles

Adj All adjectives

Adv All adverbs

Conj All conjunctions

Aux All auxiliary verbs

Ger All gerunds

Relc All question words, usually used to introduce relative clauses

In all cases, we have removed punctuation, and any sentences that contain words not belonging to these part of speech classes.

1 Warm-Up *(10%)*

We will begin by writing a rule-based part of speech tagger for a fixed, small lexicon, and then grow from there. Note that there *is* ambiguity in this lexicon, so you will need some way to deal with that! Here's the lexicon:

fruit: Noun

time: Noun, Verb

flies: Verb, Noun

like: Prep, Verb

eat: Verb

an: Det

arrow: Noun

orange: Noun

For ambiguous terms, the order in which the parts of speech are listed is based on frequency: "time" is more often a Noun than a Verb.

Implementation Note: While you might be tempted to hard-code the lexicon in your code, it will be easier if you read it from a file. We will later replace the above, small lexicon, with a very large lexicon that you probably don't want to hard code. You can find the above lexicon in a friendly format in `lexicon.small`.

Write a program to generate all possible tag sequences for a given input, based on the above lexicon. It should produce the sequences sorted by preference. In other words, less likely tag sequences should come after more likely tag sequences. You may break ties however you wish (i.e., if two different sequences have the same number of less-preferred tags).

(WU1) For each of the following sentences, how many possible tag sequences are there, and what are the best and worst sequences?

1. fruit flies eat an orange
2. fruit flies eat an arrow
3. time flies like an arrow

(Yes, I realize you could do this by hand. But it's to your advantage not to. Besides, since you can do it by hand, you can easily debug your solution!)

2 Rule-based Tagger *(25%)*

If you look in `lexicon.large`, you will find a much larger lexicon, containing 1079 entries (with 988 unique words).

2.1 Most Likely Tagger *(10%)*

Write a program that reads in the lexicon, and for any sentence, assigns the most likely tag from the lexicon to each word. (I.e., the first entry from the lexicon.) If a word does not appear in the lexicon, it should default to "Noun."

Run this program on the data in `allData.de`. It should generate output is exactly the same format as the input. (I.e., word-underscore-tag.) You can evaluate its performance by running "`./evaluate.pl allData.de [your-output-file]`". This will show you the overall accuracy, a confusion matrix showing what common errors were made, and then the 10 worst sentences in the data. The error rate for each of the sentences is shown, then the true output, then your system's output.

If you've implemented this correctly, you should get an accuracy of 84.31%, there should be 666 Adjectives that your tagger accidentally called Nouns, and the worst sentence should have error rate 5.71%.

Save the output of your program in `most-likely-predictions.txt`.

2.2 Making the Tagger Better *(15%)*

Now, you should do whatever you can to make the tagger better! Or really, not *whatever you can* but whatever you can with rules! My suggestion is to look a the confusion matrix, figure out where things are going badly, and try to correct those errors first. For example, you might notice that you're missing a lot of verbs, probably because of the default rule of calling anything unseen a Noun... perhaps you want to at least ensure that every sentence has a Verb in it?

You could of course do this by making the lexicon bigger, but that is going to take a lot of effort (go ahead if you want, though!). But whatever you do, I want you to do *manually*.

My suggestion is to look at rules that talk primarily about the tag sequences, such as: does this sentence have a verb anywhere?

Your score on this part of the assignment will depend on the performance of you tagger on the data in `allData.te`. Note that the true tags have all been replaced with "_X" in this data, so you cannot cheat. All you can do is go to `http://www.cs.utah.edu/~hal/tmp/tagger.pl` to evaluate the performance of your tagger on the test data. However, we will only let you do this *once per 24 hour period*. (More formally, any member of your team can do it once per day; so if you have more teammates, you can evaluate more often!)

Scoring. Your score for this section will be determined based on your performance on the test data according to the following table:

- $\geq$ 85.0%: 2% points
- $\geq$ 87.5%: 5% points
- $\geq$ 90.0%: 10% points
- $\geq$ 92.5%: 12% points
- $\geq$ 95.0%: 15% points
- $\geq$ 100.0%: 0% points – you're cheating!

Note that even if you don't get a great score, we will award up to 10% points for creativity in solutions, so you can still get full credit for a very creative solution that only gets you 87.5% accuracy. (Note that you cannot get more than 15% points here, even if you have a very creative solution that gets 95% accuracy!).

(WU2) What did you do? What is your accuracy now? Have you made certain cells in the confusion matrix much better?

Save your predictions on the development and test data, respectively, as `rule-based-de.txt` and `rule-based-te.txt`.

3 Markov Models *(45%)*

Now, we will get rid of all those annoying rules and just use data! You will implement the Viterbi algorithm for HMMs, as well as estimation for a first order HMM.

3.1 Warm-Up *(20%)*

Implement the crazy coke machine from HW03. Again, I recommend that you don't hard-code the transition and emission probabilities, but instead read them from a file. In this case, I've put an example in the file `coke.hmm`.

Given the specification of an HMM (such as `coke.hmm`) and an input sequence, implement the Viterbi algorithm. It should return *both* the (log) probability of the sequence under the HMM *as well as* the most likely state sequence. Your code should recognize "`<s>`" as the start state and "`</s>`" as the terminal state.

Run your HMM on the following test cases and report both the sequence and probability. The first one is from the homework, so you can check your solution.

1. `<s> coke sprite sprite diet`
2. `<s> coke coke coke coke coke`
3. `<s> sprite sprite code`
4. `<s> coke sprite coke diet`

Implementation hint: You should work with log probabilities rather than probabilities. When you report your answers to the previous section, please report log probabilities, and report them in log-base-e (aka "natural log" aka "ln") format.

(WU3) What are the state sequences and probabilities for these observation sequences?

3.2 Estimating an HMM from Data *(15%)*

Based on the data in `allData.tr`, estimate the parameters of a first order hidden Markov model. You should use add-λ smoothing (aka Laplace smoothing), where you can vary λ. You should use the same λ for both the transition and emission probabilities. Generate the output in the same format as `coke.hmm` with $\lambda = 1$ and call the file `estimate1.txt`; generate it with $\lambda = 0.1$ in the file `estimate0.1.txt`. *Remember:* You'll need to estimate probabilities from the start state and to the end state as well! *Warning:* never allow any transitions directly into the start state, and *do allow* a transition from the start state to the end state (to accept the empty sentence).

As a sanity check, here are some transition and emission values for the case when $\lambda = 0.0$ (these values have been truncated for precision):

```
t <s> Det 0.256
t Adj Noun 0.715
t Adv Verb 0.319
t Conj Adv 0.063
t Noun </s> 0.131
e Conj and 0.692
e Det the 0.585
e Aux to 0.532
e Noun small-business 3.43e-05
e Relc who 0.307
```

And with $\lambda = 1$:

```
e Conj and 0.124
e Det the 0.248
e Aux to 0.100
e Noun small-business 4.95e-05
e Relc who 0.011
```

3.3 Tagging Text *(10%)*

Putting together the HMM you just built with your implementation of Viterbi, we are going to tag raw text. The one thing that your HMM probably can't handle right now is *unseen words*. Modify your implementation so that the *emission probability* of an unseen word *given* any tag is a constant. (Note that this is now not a true probabilistic model, but it will still do something reasonable.) *Note:* you might have to modify your code to correctly handle `<s>` and `</s>`. When you generate output, though, you should *NOT* include these states.

Run this on `allData.tr` and `allData.de` using your `estimate1.txt`, and run the evaluation. You should get *about* 91.5% accuracy on the training data and 88.4% accuracy on the test data. Also run this on the test data and save the output to `estimate1.output.txt`. Do the same for `estimate0.1.txt` to `estimate0.1.output.txt`.

(WU4) Now, experiment with different values of λ. For $\lambda \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$, generate a graph plotting λ in the x-axis and both training performance and development performance on the y-axis. **Hint:** Before you run this experiment, think about what you *expect* to observe. Then you can check that this matches! What do you observe? What's your best value of λ for development data?

4 Making a Better Tagger *(20%)*

Now, have fun. Your job is to get the best performance on the *test data* as possible. You may use whatever techniques you want, so long as you don't use any data that we haven't provided to you. Here are some ideas, but you are definitely encouraged to be creative:

- Use a higher order Markov model
- Use a better smoothing technique
- Handle unseen words better
- Combine your rule-based tagger with your HMM tagger in some interesting way
- Use features of the words (eg., endings) in some way

There are definitely lots of options not listed here. Do something fun, get as high performance as you can, and submit your output on the test data as well as a half-page to full-page description of what you did and how well it worked.

Hint: It's no fun to spend hours implementing, say, a better way to handle unseen words, fix all the unseen words, and see your performance go up 0.001%. I suggest that when you have an idea of something to try, you try to get a sense of an upper bound of how much it could possibly help. For instance, you could look at your output and figure out how many unseen words there are, and how many you're getting wrong. If both of these numbers are high, then this is worth working on. If they're not, then perhaps not.

Scoring. Your score for this section will be determined based on your performance on the test data according to the following table:

- $\geq$ 90.0%: 5% points
- $\geq$ 92.0%: 10% points
- $\geq$ 94.0%: 15% points

- $\geq$ 96.0%: 18% points
- $\geq$ 98.0%: 20% points
- $\geq$ 100.0%: 0% points – you're cheating!

Note that even if you don't get a great score, we will award up to 10% points for creativity in solutions, as before.

Save your predictions on development and test data as `markov.de.txt` and `markov.te.txt`.

(WU5) Describe what you did, and how it worked and how well it worked.

What To Hand In

Only one person from each team should hand in the assignment.

- [`code.zip`] All of your code, zipped up into a single file.
- [`README.txt`] A readme file that tells us how to compile/run your code. It should tell us what to run for each subsection in the assignment. Eg., "For section 1.2, run dots."
- [`partners.txt`] A list of UIDs and last names for everyone in your group, including the person who handed it in.
- [`writeup.pdf`] Please answer all the questions marked "**WU#**" in this assignment.
- [`most-likely-predictions.txt`] Your output of the development data for the most likely tagger.
- [`rule-based-de.txt` and `rule-based-te.txt`] The output of your "better" rule based tagger on development and test data.
- [`estimate1.txt` and `estimate0.1.txt`] Your HMM estimates from data for two different values of λ.
- [`estimate1.output.txt` and `estimate0.1.output.txt`] The output for your HMM on the *test* data.
- [`markov.de.txt` and `markov.te.txt`] Your "better Markov model" tagger on development and test data.