Practice Exam Problems: Structured Prediction

Natural Language Processing (JHU 601.465/665)

Prof. Jason Eisner

1. Some part-of-speech taggers may have trouble deciding whether to tag `to` as a preposition `P` or an auxiliary verb `V`.

   Why? Because it's not enough to know the tag of the previous word. Consider the difference between `walked to` and `wanted to`. The previous word is a verb in both cases, but it makes a difference *which* verb it is!

   A Conditional Random Field (CRF) can take this into account. A CRF tagger defines

   $$p(\vec{t} \mid \vec{w}) = \frac{1}{Z_w} \exp \vec{\theta} \cdot \vec{f}(\vec{t}, \vec{w})$$

   where $\vec{w}$ denotes the input sentence of n words and $\vec{t}$ denotes a possible sequence of n tags for these words.

   Feature number 5 happens to count the number of times that `walked` is immediately followed by a preposition `P`:

   $$f_5(\vec{t}, \vec{w}) = \sum_{i=1}^{n} I(w_{i-1} = \texttt{walked} \textbf{ and } t_i = \texttt{P})$$

   where $I(\ldots)$ returns 1 if its argument is true and 0 otherwise.

   Let's do supervised training of our model on the Brown corpus, where the word `walked` has 159 tokens. 12 of those tokens are immediately followed by a preposition `P`. To be slightly more specific, f_5 takes value 1 on 12 of the 52108 tagged sentences, and takes value 0 on the rest. Thus the empirical expectation of f_5 is $12/52108 \approx 0.00023$.

   (a) Suppose you are training by gradient ascent on the log-likelihood objective function (that is, maximum-likelihood with no regularization). Then you will increase θ_5 if under the current parameters $\vec{\theta}$, the model's expectation of f_5 is BELOW ABOVE 0.00023. *(circle <u>one</u>)*

   (b) To compute the model's expectation of f_5, you must average over all possible taggings of all sentences in the corpus. To average over the taggings of a single sentence, in proportion to their model probabilities, which of the following numerical operations do you need? *(circle <u>all</u> that apply)*
   ADDITION SUBTRACTION MULTIPLICATION DIVISION MAX MIN
   NONE OF THE ABOVE

(c) What is the *maximum possible* value of the model's expectation of f_5 (for any $\vec{\theta}$)?

(d) What is the *minimum possible* value of the model's expectation of f_5 (for any $\vec{\theta}$)?

(e) How many times is it necessary to access the parameter θ_5 while computing this expectation?

(f) There are many instances in the Brown corpus of phrases like `walked up to`, `walked off to`, `walked over to`, `walked me back to`, `walked all the miles back to`, `walked past the entrance to`, `walked from midnight to`, ...

In such phrases, `walked` still seems like a good clue to the tag of `to`, even though it doesn't *immediately* precede `to`.

So you consider modifying feature f_5 to be defined as follows:

$$f_5(\vec{t}, \vec{w}) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} I(w_j = \texttt{walked} \textbf{ and } t_i = \texttt{P})$$

Is it still possible as in question 1b to average over the taggings of a single sentence, in proportion to their model probabilities, in *polynomial time*?

YES NO *(circle one)*

2. (a) The problem of "named entity recognition" is sometimes treated as a tagging problem. You build a tagger to identify all of the place names in a sentence, using the tagging scheme known as BIO (or IOB). Write the correct tags over the tokens of this sentence.

`From San Francisco New York can seem like a museum`

[If we have not covered BIO tagging yet in class, the discussion leaders will show you the answer to this one.]

(b) The score of a tagging of this 10-word sentence is the sum of _____ transition values and _____ emission values. (Assume that there must be a special BOS tag preceding the tagging and a special EOS tag following the tagging.)

(c) Here is a recurrence for finding the score of the best-scoring tag sequence of the sentence $w_1 w_2 \ldots w_n$ (where $n = 10$ in the example above). We have $w_0 = \text{BOS}, w_{n+1} = \text{EOS}$. The β values below have "start position" and "end position" arguments, just as in the CKY algorithm.

$$\beta_{\text{EOS}}(n, n+1) = 0$$
$$\beta_t(j-1, n+1) = \max_{t'} \big(\text{emission\_score}(t, w_j) + \text{transition\_score}(t, t') + \beta_{t'}(j, n+1)\big)$$

i. To carry out the recurrence, you need to carry out two nested "for" loops. Fill in the three blanks.
for $j =$ ____ to ____
for $t \in$ _____
compute $\beta_t(j-1, n+1)$

ii. Write an expression which, once the recurrence has been carried out, will evaluate to the score of the best-scoring tag sequence.

iii. This program can be used to find the best tagging under an HMM. If you want to do this, what value should you assign to `transition("I","O")`? (*Hint:* Your answer should include $p(\cdots)$ notation and mention the symbols `"I"` and `"O"`.)

iv. This program can also be used to find the most probable tagging according to a CRF (conditional random field). TRUE FALSE *(circle <u>one</u>)*

v. This program can also be used to find the highest-scoring tagging according to a structured perceptron. TRUE FALSE *(circle <u>one</u>)*

(d) A more direct approach identifying named entities would be to "tag" the sentence like this:

```
Prep |---Place---| |-Place-| Verb Verb Prep Det N
From San Francisco New  York can  seem like a   museum.
```

In an hidden Markov model, each tag emits exactly one single word. But in our new scheme, a tag may emit a whole sequence of words. For example, the first `Place` tag above emits `San Francisco`, and the second one emits `New York`. All other tags in this example emit exactly 1 word.

Revise the recurrence relation above so that it can cope with this extension. Don't worry about how the emission scores will be defined, only about how your equation will use them.

(e) Perhaps the most natural way to recognize named entities is to parse the sentence first, and then classify each noun phrase in the parse. For example, in the above sentence, the NP `San Francisco` should be classified as a place name, whereas the NP `a museum` should be classified as not a named entity.

Here we consider features that might be useful for such a classifier:

- Suggest two features that tend to indicate that a name (in context) is a place name. In the above sentence, these features should fire on `San Francisco` but not on `a museum`.

- Suggest one feature that tends to indicate that a name (in context) is not a named entity. In the above sentence, this feature should fire on `a museum` but not on `San Francisco`.

(f) We have now considered three approaches to identifying named entities in a sentence. Assume for each approach that you are given an already-trained model. When applying the model to a new sentence, what is the runtime of finding the best tagging, as a function of the sentence length n? Give the smallest runtime that you can guarantee.

- BIO tagging (question 2a): $O($_____$)$
- Phrasal tagging (question 2d): $O($_____$)$
- Parsing + classification (question 2e): $O($_____$)$

(g) Assume that as part of the approach in question 2e, you have trained a log-linear model $p(y \mid x)$ for classifying NPs.

If the score $\vec{\theta} \cdot \vec{f}(x, \texttt{Company})$ is 2 plus the score $\vec{\theta} \cdot \vec{f}(x, \texttt{Place})$, then what can you say about the relationship of $p(\texttt{Company} \mid x)$ and $p(\texttt{Place} \mid x)$?

(h) Company names have typical lengths. An NP of 3 or 4 words is more likely to be a company name than a very short or very long NP. Thus, your classifier for question 2e could include features that are sensitive to the length of the noun phrase x (counted in number of words). Here are three possible templates for such "length features." Each template defines a set of features $f_1, f_2, \ldots, f_8$ by taking $k = 1, 2, \ldots, 8$:

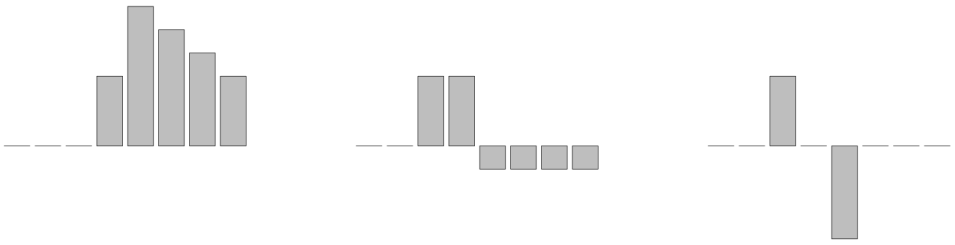
$$\text{Scheme A:} \quad f_k(x, y) = \begin{cases} 1 & \text{if } y = \texttt{Company} \wedge \text{length}(x) = k \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Scheme B:} \quad f_k(x, y) = \begin{cases} 1 & \text{if } y = \texttt{Company} \wedge \text{length}(x) \geq k \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Scheme C:} \quad f_k(x, y) = \begin{cases} \text{length}(x) - k & \text{if } y = \texttt{Company} \wedge \text{length}(x) \geq k \\ 0 & \text{otherwise} \end{cases}$$

After training, you can visualize the effect of the length features, by making a graph that shows how long the classifier thinks company names should be. For each length from 1 to 8 on the horizontal axis, the vertical axis shows how much the classifier increases the score $\vec{\theta} \cdot \vec{f}(x, \texttt{Company})$ when x is an NP of that length.

The graphs for three different classifiers are shown below. Each of these classifiers *could* have been produced by any of the feature schemes. Nonetheless, your choice of features can affect how your training procedure decides to generalize from a small dataset. Match each of the 3 graphs with the single feature scheme (A, B, or C) that probably produced it, assuming that you trained a log-linear model $p(y \mid x)$ with that feature scheme **and using strong ℓ_1 (or L_1) regularization**.



Scheme _____ Scheme _____ Scheme _____