6.864, Fall 2005: Problem Set 5

Total points: 140 regular points
Due date: 5pm, 29 November 2005
Late policy: 5 points off for every day late, 0 points if handed in after 5pm on 3 December 2005

Question 1a (25 points)

Say that we have used IBM model 2 to estimate a model of the form

$$P_{M2}(\mathbf{f}, \mathbf{a}|\mathbf{e}, m) = \prod_{j=1}^{m} T(f_j|e_{a_j}) D(a_j|j, l, m)$$

where $\mathbf{f}$ is a French sequence of words $f_1, f_2, \ldots f_m$, $\mathbf{a}$ is a sequence of alignment variables $a_1, a_2, \ldots a_m$, and $\mathbf{e}$ is an English sequence of words $e_1, e_2, \ldots e_l$. (Note that the probability P_{M2} is conditioned on the identity of the English sentence, $\mathbf{e}$, as well as the length of the French sentence, m.)

Question 1b (10 points) Give pseudo-code for an efficient algorithm that takes an input an English string $\mathbf{e}$, and an integer m, and returns

$$\arg\max_{\mathbf{f},\mathbf{a}} P_{M2}(\mathbf{f}, \mathbf{a}|\mathbf{e}, m)$$

where the $\arg\max$ is taken over all $\mathbf{f}, \mathbf{a}$ pairs whose length is m.

Question 1c (10 points) Give pseudo-code for an efficient algorithm that takes an input an English string $\mathbf{e}$, and an integer m, and returns

$$\arg\max_{\mathbf{f}} P_{M2}(\mathbf{f}|\mathbf{e}, m)$$

where the $\arg\max$ is taken over all $\mathbf{f}$ strings whose length is m. Note that

$$P(\mathbf{f}|\mathbf{e}, m) = \sum_{\mathbf{a}:|\mathbf{a}|=\mathbf{m}} \prod_{j=1}^{m} T(f_j|e_{a_j}) D(a_j|j, l, m)$$

Question (5 points) Given that it is possible to efficiently find

$$\arg\max_{\mathbf{f}} P_{M2}(\mathbf{f}|\mathbf{e})$$

when P_{M2} takes the above form, why is it preferable to search for

$$\arg\max_{\mathbf{e}} P_{M2}(\mathbf{f}|\mathbf{e}) P_{LM}(\mathbf{e})$$

rather than

$$\arg\max_{\mathbf{e}} P_{M2}(\mathbf{e}|\mathbf{f})$$

when translating from French to English? (Note: P_{LM} is a language model, for example a trigram language model)

Question 2 (15 points)

Consider the following transduction PCFG:

S	⇒	[NP	VP]	0.7
S	⇒	⟨NP	VP⟩	0.3
VP	⇒	Vi		0.4
VP	⇒	[Vt	NP]	0.01
VP	⇒	⟨Vt	NP⟩	0.79
VP	⇒	[VP	PP]	0.2
NP	⇒	[DT	NN]	0.55
NP	⇒	⟨DT	NN⟩	0.15
NP	⇒	[NP	PP]	0.7
PP	⇒	⟨P	NP⟩	1.0

Vi	⇒	sleeps/ϵ	0.4
Vi	⇒	sleeps/asleeps	0.6
Vt	⇒	saw/asaw	1.0
NN	⇒	man/aman	0.7
NN	⇒	woman/awoman	0.2
NN	⇒	telescope/atelescope	0.1
DT	⇒	the/athe	1.0
IN	⇒	with/awith	0.5
IN	⇒	in/ain	0.5

Assume we'd like to use this grammar to translate from English to "pseudo-English" (for example, we might translate *the man sleeps* to *athe aman asleeps*).

Question (15 points) Show 4 possible translations under the above transduction PCFG for

the woman saw the man

For each translation you should show the underlying parse tree, and how to calculate the probability of the translation under the transduction PCFG.

Your task is to design a method for coreference resolution. For the purpose of this assignment, we will treat coreference resolution as a binary classification task: given a noun phrase and a pronoun, we want to predict whether they are coreferent or not. We define an *instance* in this framework as a pronoun and a potential antecedent, and we represent each instance with a feature vector, as described below. You will use the BoosTexter classifier to make coreference prediction by classifying these feature vectors as coreferent or not. You can download BoosTexter from

```
http://www.research.att.com/cgi-bin/access.cgi/as/vt/ext-software/
                               www-ne-license.cgi?table.BoosTexter.binary
```

(This link is also posted on the assignment section.)

Data format: For simplicity, we will assume that an antecedent is located either within the same sentence as the corresponding pronoun, or within the previous sentence. The provided training and testing data consist of pairs of consecutive sentences, one sentence per line, where each pair of sentences is separated from the other pairs by blank lines.

Words are annotated with POS tags using underscores: `<word>_<tag>`. Noun phrases are marked with double square brackets: `[[ <noun phrase> ]]`. Note that noun phrases can be embedded within other noun phrases; embedded bracketing reflects this relation.

Pronouns that need to be resolved are marked by ID tags appended to their closing brackets: `[[ <pronoun> ]]_###<ID>###`. The corresponding antecedents are marked with matching ID tags: `[[ <antecedent> ]]_##<ID>##`. Note that antecedents may contain embedded noun phrases or be embedded in larger noun phrases.

Generation of training and test instances: A pronoun can be resolved to any noun phrase from the corresponding sentence pair. For each pronoun marked with an ID tag, and for every noun phrase within the corresponding sentence pair, you must create an instance that pairs the given pronoun and noun phrase. Make sure you also consider embedded noun phrases; for instance, the noun phrase `[[ [[ labor_NN ]] unions_NNS ]]` should yield two instances: one for "labor unions", and one for "labor".

Features: Each instance is represented by a feature vector that captures various properties of the pronoun and potential antecedent. For every instance in the training and test sets, you should generate a feature vector containing the following 7 features:

- **Number agreement** indicates whether the pronoun and potential antecedent agree in number. You can compute number information from the POS tags; a listing of the definitions of the part of speech tags will be linked from the homework page. In English, the head noun typically appears at the end of a noun phrase, so you should define the number of a noun phrase as the number of its rightmost word.
- **Sentence match** records whether the pronoun and the potential antecedent appear in the same sentence.
- **Distance** captures the number of *non-embedded* noun phrases between a pronoun and its potential antecedent. If the antecedent is embedded within another noun phrase, then you should measure from the non-embedded noun phrase containing the antecedent.

- **Antecedent position** records the number of non-embedded noun phrases between the potential antecedent and the beginning of the sentence to which it belongs. As before, if the antecedent is embedded, then measure from the non-embedded noun phrase containing it.
- **Pronoun position** records the number of non-embedded noun phrases between the pronoun and the beginning of the sentence to which it belongs.
- **Relative position** indicates whether the antecedent appears before the pronoun or not.
- **Embedding level** of an antecedent within its enclosing noun phrase. This should be 0 for a non-embedded antecedent.

The feature vectors you generate should be usable as input to the BoosTexter classifier (please consult the `README` file for BoosTexter for more information). We will provide a `coref.names` file that defines the specific format of these feature vectors. **Your task is to create `coref.data` and `coref.test` files that give feature vectors corresponding to the training and test data sets. These files should contain one feature vector per line, and blank lines should be used to separate sets of feature vectors from different sentence pairs.**

NB: The training and test sentence pairs may contain pronouns whose corresponding antecedents do *not* appear in the sentence pair. This is not an error; these correspond to instances of coreference where the antecedent appears in a sentence earlier than the previous sentence. In these cases, the correct behavior is to generate all negative instances.

Training and testing the classifier: Using the `coref.data` training set you generated above, train a classifier using 50 iterations of the BoosTexter program. Test your classifier on the `coref.test` test set and compute its precision and recall on the recovery of coreference relations. **You should submit a `coref.results` file that contains the precision and recall of your classifier, formatted as two lines with the first line giving precision and the second giving recall**.