MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2014

Project 3: Pun With Words Issued: Tues., 4/15 Due: Fri., 4/25 at 9am

Project Submission: Please submit two files—a *single* PDF file containing all your answers, code, and graphs, and a *second* .zip file containing all the code you wrote for this project, to the Stellar web site by 9am, April 25th.

Introduction

Your task is to build a simple mixture model classifier for parts-of-speech tagging. Part of speech tagging is a critical step in most natural language understanding systems. Consider the following sentence:

John went to the bank.

In order to extract information from this sentence, we would like to know that *John* is a proper noun, and that in this instance *bank* is a noun, not a verb *("I would like to bank this check", or even "bank the plane to the left").*

More formally, given a dictionary of english words $\mathcal{W}$, and a set of parts-of-speech tags $\mathcal{T}$, we can consider each sentence as a sequence of words, $w_1, \ldots, w_n$, with a corresponding sequence of parts-of-speech tags, $t_1, \ldots, t_n$. If the sentence has been labeled for us, we are given both the words and the tags; if not, we are given just the words.

Your task is to learn a model to predict the sequence of tags for any new sentence, given a set of already tagged training sentences and a set of unlabeled sentences (without tags), taken from the *Wall Street Journal*. This data set, along with some skeleton Python code, is available for download from Stellar as `project3.zip`.

We use a standard set of 33 possible word tags and 9 possible punctuation tags. This tag set is a little more complex than the one you are used to—rather than nouns and verbs, it includes more specific categories like *NN* (noun, singular or mass), *NNS* (noun, plural), and *VBD* (verb, past tense).

As you have already seen, a given word's part of speech usually depends on more than just the word itself; it often depends on context. This context dependence makes writing an accurate parts-of-speech tagger quite challenging, so we'd like to be able to automatically learn a good tagger.

Models

We will consider here three versions of mixture models that we slide across the sentence to predict each tag. The simplest model will base its predictions about the tag only on the word itself. By contrast, the most involved model will use the information about the previous, current, and the next word to predict the tag for the current word. All the models are composed of (subsets of) the following four types of probabilities.

1. The prior probability distribution for the tags. These probabilities would be the basis for predicting word tags if you could not read the actual word or see how it was used:

$$P(t) = \theta(t), \;\; t \in \mathcal{T}.$$

2. The probability that word w_{i-1} directly precedes a word with tag t_i:

$$P_{-1}(w_{i-1}|t_i) = \theta_{-1}(w_{i-1}|t_i), \;\; w_{i-1} \in \mathcal{W}, \; t_i \in \mathcal{T}.$$

3. The probability of the word w_i, given we know it has tag t_i:

$$P_0(w_i|t_i) = \theta_0(w_i|t_i), \;\; w_i \in \mathcal{W}, \; t_i \in \mathcal{T}.$$

4. The probability that word w_{i+1} directly follows a word with tag t_i:

$$P_{+1}(w_{i+1}|t_i) = \theta_{+1}(w_{i+1}|t_i), \;\; w_{i+1} \in \mathcal{W}, \; t_i \in \mathcal{T}.$$

Each probability table $\theta_{-1}(w|t)$, $\theta_0(w|t)$, $\theta_{+1}(w|t)$ sums to one over the words $w \in \mathcal{W}$ given a tag $t \in \mathcal{T}$. The three models, models A, B, and C, are defined on the basis of these parameters as:

A) $P_A(t_i, w_i) = P(t_i)P_0(w_i|t_i)$

B) $P_B(t_i, w_{i-1}, w_i) = P(t_i)P_{-1}(w_{i-1}|t_i)P_0(w_i|t_i)$

C) $P_C(t_i, w_{i-1}, w_i, w_{i+1}) = P(t_i)P_{-1}(w_{i-1}|t_i)P_0(w_i|t_i)P_{+1}(w_{i+1}|t_i)$.

Given that the models are slid across each sentence to predict the tags, there will be "edge" effects. For instance, consider model C and a four-word sentence w_1, w_2, w_3, w_4 with tags t_1, t_2, t_3, t_4. Model C views the data as broken into pieces where each piece consists of a tag and (at most) three words around the tag. In this example,

$$\begin{aligned} &t_1, w_1, w_2 \\ &t_2, w_1, w_2, w_3 \\ &t_3, w_2, w_3, w_4 \\ &t_4, w_3, w_4 \end{aligned}$$

and four pieces are considered independent instances. If we didn't have tag t_1, we would use:

$$P_C(t_1, w_1, w_2) = P(t_1)P_0(w_1|t_1)P_{+1}(w_2|t_1) = \theta(t_1)\theta_0(w_1|t_1)\theta_{+1}(w_2|t_1)$$

to evaluate the posterior probability $P_C(t_1|w_1, w_2)$. Similarly, the posterior probability over the 2nd tag t_2 would come from

$$\begin{aligned} P_C(t_2, w_1, w_2, w_3) &= P(t_2)P_{-1}(w_1|t_2)P_0(w_2|t_2)P_{+1}(w_3|t_2) \\ &= \theta(t_2)\theta_{-1}(w_1|t_2)\theta_0(w_2|t_2)\theta_{+1}(w_3|t_2), \end{aligned}$$

and so on. The best tag is obtained by maximizing the posterior probability. For example,

$$\hat{t}_1 = \arg\max_{t_1} P_C(t_1|w_1, w_2) = \arg\max_{t_1} \frac{P_C(t_1, w_1, w_2)}{\sum_{t_1'} P_C(t_1', w_1, w_2)} = \arg\max_{t_1} P_C(t_1, w_1, w_2).$$

The problem

We will use both labeled and unlabeled sentences to estimate the parameter tables for the models A, B, and C. When the sentence is labeled, we aim to maximize the joint likelihood of words and tags (broken into pieces as shown above) as the model is slid across each sentence. For unlabeled sentences, we try to maximize the log-likelihood of generating only the words around each tag position. Note that since the models are slide across the sentence, we will repeatedly generate some of the words, i.e., the little tag+words pieces overlap. This is ok because we are not building a language model but rather using the words (always observed) to predict the corresponding tags.

In a typical tagging scenario, we would have a few labeled sentences (expensive to obtain, may require human involvement) together with a large number of unlabeled sentences (easy to come by). In our code, this setting is simulated by randomly removing labels from a fraction of training sentences.

The EM algorithm

Let S_L be the set of labeled (tagged) sentences and S_U the set of unlabeled (untagged) sentences. Each sentence $s \in S_L$ (or $s \in S_U$) is a sequence of words, i.e., $s = (w_1^s, \ldots, w_{|s|}^s)$, where the length varies according to the sentence. When the sentence is labeled, we also have the corresponding sequence of tags $t_1^s, \ldots, t_{|s|}^s$. For example, if we consider model A, the EM algorithm tries to maximize the log-likelihood:

$$l(D;\theta) = c \sum_{s \in S_L} \sum_{i=1}^{|s|} \log \left[\theta_0(w_i^s|t_i^s)\theta(t_i^s) \right] + \sum_{s \in S_U} \sum_{i=1}^{|s|} \log \left[\sum_{t \in \mathcal{T}} \theta_0(w_i^s|t)\theta(t) \right] \quad (1)$$

with respect to the parameters $\theta_0(w|t)$ and $\theta(t)$. The parameter c here can be used to emphasize the labeled data (which is more informative) over the unlabeled data. We have provided you with skeleton Python code that does most of the work. However, there are specific parts missing that you need to fill in.

1. In the M-step, we need to re-estimate the parameters based on word-tag counts. You will complete the function *Mstep()* in the provided framework. Here we have the real counts evaluated directly from labeled data, and expected counts based on the current model (E-step). You can find the definition of the parameters at the beginning of *Mstep*. Your task is to first compute the counts as the weighted sum of the real counts and expected counts. For instance, the count for tag t is defined as:

   $$n(t) = c \cdot r(t) + e(t),$$

   where c is the weight parameter from equation 1, $r(t)$ is the real count and $e(t)$ is the expected count. We compute the following counts in a similar fashion:

   $$n(t),\ n_0(t,w),\ n_{-1}(t,w),\ n_{+1}(t,w),$$

   and then turn them into the parameter estimates. Here, e.g., $n_{-1}(t,w)$ is the number of times that a tag t is preceded by word w. Inspect the code, and place the correct evaluations of the parameter tables $\theta(t)$, $\theta_{-1}(w|t)$, $\theta_0(w|t)$, $\theta_{+1}(w|t)$ in the M-step.

   Turn in: A few paragraphs describing the calculations you made, and the resulting code.

2. In the E-step, we must compute the posterior probability *(be sure to normalize)* of each tag given the information in the words (the number of context words depend on the model). Please fill in the missing steps for evaluating posteriors for each model A, B, and C. Note that there are "edge" effects (as discussed above), and you need to deal with them properly in each case. Your task is to complete the functions *EstepA()*, *EstepB()*, and *EstepC()* in the provided framework by filling in the code at the position marked *"Your code here:"*.

   Turn in: A few paragraphs describing the calculations you made, and the resulting code.

3. Finally, to make predictions, you need to use each model for tagging on a test sentence (which may also be one of the unlabeled sentences). Please fill in this part of the code, separately for each model A, B, or C. The provided code uses these predictions to evaluate the model on the test set of tagged sequences. Your task is to complete the functions *predictA()*, *predictB()*, and *predictC()* in the provided framework by filling in the code at the position marked *"Your code here:"*.

   Turn in: A few paragraphs describing the calculations you made, and the resulting code.

4. Now, after the above steps, we finally have a working implementation of the EM algorithm for any split of labeled/unlabeled sentences. Please run your code for a few EM iterations in the following three scenarios. As a sanity check, make sure that the log-likelihood (which should be a large, negative number) is non-decreasing.

   (a) Training data is split 100% labeled, 0% unlabeled. In other words, we are just training the three models as classifiers with all the training data. The provided code will evaluate the tagging accuracy on the test sentences. You can do this task by calling the function *task1()*.

   (b) Training data is split 50% labeled, 50% unlabeled. Now the EM iterations will alter the model as the parameters are tailored to find "clusters" of word combinations for each tag. How does the test accuracy change as a function of EM iterations? *Note: it may in some cases go down with more iterations.* Do the unlabeled sentences help or hurt? You can do this task and get the rest accuracy and log-likelihood by calling function *task2()*.

   (c) Finally, we split the training data 1% labeled, 99% unlabeled. In this case, the training data are predominantly unlabeled. Compare the test accuracy to the above two scenarios, and to the test accuracy of the model based on the labeled data alone. You can do this task and get the test accuracy and log-likelihood by calling function *task3()*.

   For each scenario, turn in: A collection of clearly labeled graphs and tables describing your results, along with a short summary.

5. Now that we've learned a model, we can use it to answer some interesting questions. Your task in this part is to write some code to extract answers to the following questions from a learned model of your choice:

   (a) What is the word with the highest unconditional probability of directly specifying a tag? In other words, without any information about its neighboring words, which word most clearly specifies a tag? What is that probability? (There may be several, in which case, pick one.)

   (b) Devise a metric for determining which word contains the *least* information about its tag (again, ignoring the words surrounding it). Which word maximizes this metric? (If there is more than one such word, just pick one.) Give a couple of example sentences from the data set where your chosen word has different tags.

   Turn in: The code you wrote to solve this problem, a description of how you went about it, and the results of both queries.

   REMEMBER Project Submission: Please submit two files—a *single* PDF file containing all your answers, code, and graphs, and a *second* .zip file containing all the code you wrote for this project, to the Stellar web site by 9am, April 25th.