Part of Speech Tagging Using HMM Model and Viterbi Decoder

Tricia Lee

October 30, 2013

Table 1: Pseudo Words Mapping

Category	Example
twoDigitNum	90
fourDigitNum	1990
containsDigitAndAlpha	A2343-9789
containsDigitAndDash	90-23
containsDigitAndSlash	10/01/2012
containsDigitAndComma	10,202,234
containsDigitAndPeriod	10.23
allDigit	234364
allCaps	BBC
capPeriod	C.
initCap	Sally
lowercase	swine
nonAlphaDigit	@#$#@
other	none of the above

Building a Sequence Model

Learning/Training

BetterTagScorer class is a better scorer for POSTaggerTester. Here is how the logic works.

Keeping counts of Word-Tag relationship:

- When a word is read, check whether it is in seenWord and if the count is less than 5.
- If less than 5, add the word along with its tag to the unknownWordsToTag counter and increment the count.
- If not, check if the word is already existing in wordToTag counter.
  - If so, increment the count.
  - If not, copy the previous word-tag counts data from unknownWordsToTag and increment one for the current count.
- Validate function would go through all the low-frequncy words stored in unknownWordsToTag counter, turn them into entities listed below (ref: http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf) and recount them into WordsToTag count.

Keeping counts of Tags relationship:

- When a tag is read, the counts for bigram, trigram and unigram are stored whether the word is a low-frequency word or not.
- For smoothing, λvalues are chosen to be $\frac{1}{3}$ for each. I have experimented it by adding more weight on $\lambda_1$but it didn't improve on the accuracy score.

Smoothing

1. For q number calculation which is for the tags, λvalues are chosen as $\frac{1}{3}$ since I didn't get a chance to implement. Otherwise, I would want to find better λvalue just like in homework 1 by experimenting different ones using heldout set and choose the best ones.
2. For e number calculation which is the probability of word seen with tag given the tag, pseudo word logic is used to serve as a smoothing algorithm. If a word is not found in the training vocabulary, it will be turned into pseudo word which surely will be found in the set.

LogScoreCounter

HiddenMarkovModel parameters are used in this to find the log score of each tag given a word in sentence. The equation has two parts: qNumber and eNumber.

The qNumber calculation is the same as what I did in homework 1. The equation is as below.

$$q(y_i|y_{i-2}, y_{i-1}) = \lambda_1 * \frac{Count(y_i, y_{i-1}, y_{i-2})}{Count(y_{i-1}, y_{i-2})} + \lambda_2 * \frac{Count(y_i, y_{i-1})}{Count(y_{i-1})} + \lambda_3 * \frac{Count(y_i)}{Count(AllTags)}$$

However, if any of the denominator is 0, it's considered undefined and the segment won't be participating in the equation. For instance, if $\frac{Count(y_i,y_{i-1},y_{i-2})}{Count(y_{i-1},y_{i-2})}$ has zero denominator, it won't be calculated as part of qNumber equation.

The eNumber calculation is as below.

$$e(w|y_i) = \frac{Count(w, y_i)}{Count(y_i)}$$

This will be calculated for pseudo-words if input words are not found in the list.

The qNumber and eNumber are multiplied and convered the result into the log number form.

$$p(w_1, ..., w_n, y_1, ..., y_n) = \log(q(y_i|y_{i-2}y_{i-1}) * e(w|y_i))$$

Viterbi Decoder

Viterbi decoder, replacement of Greedy Decoder, utilizes the scores stored in Trellis to find the score value of the test sentence up to the current position and choose the best tag for it. The score value is created using the equation below.

$$\pi(0, y_0) = 1$$

for start state.

$$\pi(i, y_i) = \max_{y_{i-1}}(\log(q(y_i|y_{i-1}) * e(w|y_i)) + \pi(i-1, y_{i-1}))$$

for the consequence states.

In this problem, the logic chose the best scoring tag looking forward by choosing the max of possible current score which is the log of qNumber and eNumber multiplied. Then it looks back to see what other previous states can reach to this state. From all those states that can reach to this state, it looks for the best path to it, which is what the π inside the equation is doing. That will assure that the logic not only choose the best possible next tag, it also ensure that the path to this tag is also optimal. Dynamic programming part comes in when it starts storing the path that has been seen and calculated in the past. Instead of going through the same logic to find the path that was already found, the goal is to keep those found path along with score and the state in a datastructure so that we can do a quick lookup. Whenever a score is calculated for a position and a tag, the information is stored in the HashMaps for later use. Without the help of dynamic programming, finding the optimal path would be a hard problem because every path needs to be memorized.

Result and Analysis

In the learning/training part, it is the best if the algorithm has a way to do a good smoothing and has a good way to categorize unknown into some forms. In my logic, I didn't notice much change in tag accuracy. I think it's because I don't have a good smoothing algorithm. I went through only 3 variations for this and couldn't achieve more than 93.7% for tag accuracy. However unknown accuracy went up from 40% to about 50% when I added 6 pseudo-words and has gone up to 75.8% after adding all 14 pseudo-words mentioned in table 0.1. I believe unknown accuracy can go up more if I can carefully choose pseudo-words. Figure 0.1 shows that lambda value with validation data set has better accuracy on unknowns compared to others. In my experiment, since I don't utilize validation set to choose λ, the validation set doesn't act more than as another test set. Please see the data of Figure 0.1 in Table 0.2.

Viterbi decoder implementation wasn't fortunate enough to show the result. It is suffering from IndexOutOfBounds Exception. I believe I messed it up somewhere in calling prevous and next state values iteration. Nevertheless it is an impressive decoder algorithm I have learned through this homework on how to utilize dynamic programming in a hard math problem to make it much easier.

Figure 1: Result Run for different λ values over Test and Validation sets

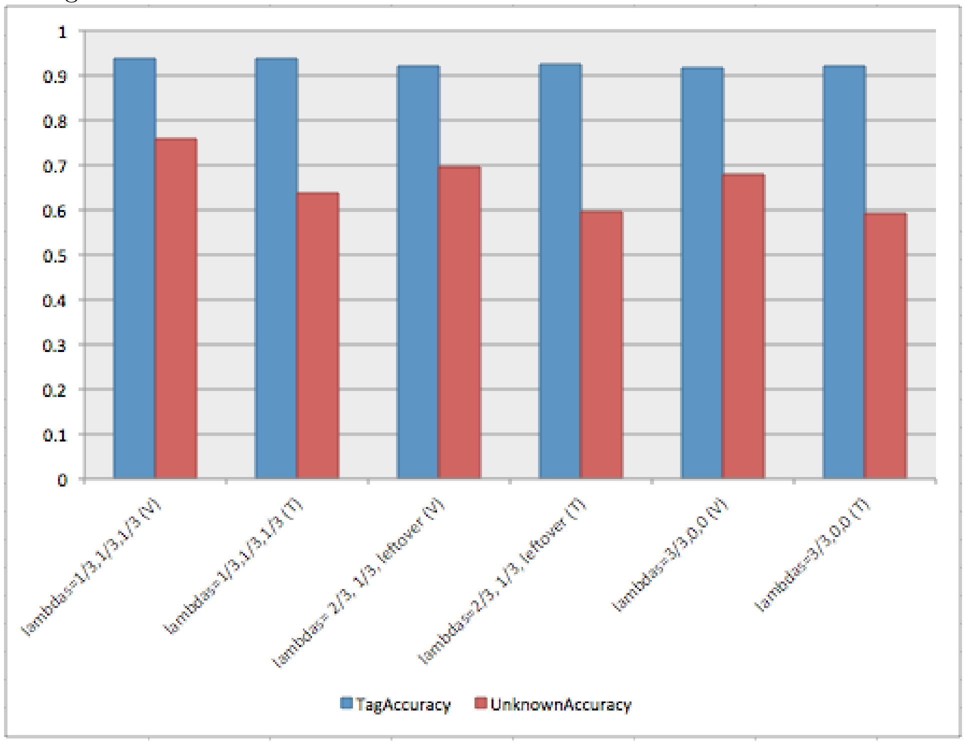

Table 2: Collected data for different λ runs with Greedy Decoder

	TagAccuracy	UnknownAccuracy	Decoder Suboptimality
$\lambda_{ValidationData} = \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	0.937918	.757932	311
$\lambda_{TestData} = \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	0.939190	.638849	496
$\lambda_{ValidationData} = \frac{2}{3}, \frac{1}{3}, \sim$	0.920972	.696283	0
$\lambda_{TestData} = \frac{2}{3}, \frac{1}{3}, \sim$	0.924136	.594245	0
$\lambda_{ValidationData} = \frac{3}{3}, \frac{0}{3}, \frac{0}{3}$	0.917797	.679057	0
$\lambda_{TestData} = \frac{3}{3}, \frac{0}{3}, \frac{0}{3}$	0.920303	.592086	0