600.465: Intro to NLP
Assignment 6: Tagging with a Hidden Markov Model

Sindhuula Selvaraju
sselvar4

April 15, 2016

1. **Spreadsheet:**

   (a) When the first day has just one ice cream:

      i. The probability that day 1 is hot reduces from 0.871 to 0.491. This reduction in probability shows that the consumption of fewer ice creams decreases the probability of the day being hot and increases that of the day being cold. The first 3 days of data 133 show that day 2 and 3 are 3 ice cream days and so strongly favor the probability of the day being hot. Since, the model does not favor rapid change in weather instead of the probability of day 1 being hot going down severely it is only slightly reduced.

      ii. Initially the probability that day2 is hot was 0.977 but after the ice cream on day 1 was set to 1, this probability dropped to 0.918. Cell K28 in the spreadsheet shows this probability.

      iii. After 10 iterations of re-estimation the change in the number of ice creams from 2 to 1 makes a change in the probability of the day being hot for days 1 and 2. When ice cream at day 1 is 1: At the end of the 10th iteration p(H) at day 1 is 0.0, p(h) at day 2 is little above 0.5. When ice cream at day 1 = 2: At the end of the 10th iteration p(H) at day 1 is 1.0 and for day 2 is also close to 1.

   (b) When the bias to interpreting the data is changes to $p(1|H) = 0, p(2|H) = 0.3 and p(3|H) = 0.7$

      i. With the 'never eat just 1 ice cream on a hot day' bias is applied the p(H) for days that have 1 ice cream drops(even if they're surrounded by days with 2 or 3 ice creams). For example as we can see in the graph before changing the bias days 11-13 had high

p(H)(around 0.95 for day 12 and 13 and slightly lesser for day 11) even though day 11 was a 1 ice cream day, but with p(1—H) = 0, day 11's p(H) dropped to 0 and p(H) for days 12-13 also dropped to around 0.5.

ii. After 10 iterations days 11 to 26 are set as cold days. This happens because the 1 ice cream days occur within those days, and since P(1—h) = 0 for days with one ice cream, all the 1 ice cream days are fixed as cold. Given the high probability of those days being cold it also coerces the other days between the 1 ice cream days to cold, of the high C-¿C probability.

iii. P(1—H) remains 0 though out the iterations as we manually changed this in the first iteration. It doesn't get updated. This means the alpha probability at the end state will not consider any path that includes a 'H' state resulting in an observation of '1'. Thus the counts of c(1,H) will remain 0 after 1 estimation, then P(1—H) will be 0 again and so on till the 10th iteration.

(c) The backward algorithm

i. The total probability of a sentence is given by the β probability of the stop state of the sentence. This is because the stop states β probability is the sum of all possible parses(actually parse trees) that can lead to the stop state.

ii. The H constituent in the tree would represent the inside probability of the observations.
H $\rightarrow$ 1 C is the product of P(1—H) i.e. the probability of 'H' giving the observation '1' and P(H$\rightarrow$C) i.e. probability of 'C being followed by 'H'.

4. **Improving the Smoothing:**
I tried to improve the smoothing by implementing the Viterbi tagger with "one count smoothing"(where the parameter addLambda is set to 1). The following results were observed:

(a) vtag ictrain ictest
Tagging accuracy (Viterbi decoding): 90.91% (known: 90.91% novel: 0.00%)
Perplexity per Viterbi-tagged test word: 3.689

(b) vtag ic2train is2test
Tagging accuracy (Viterbi decoding): 90.91% (known: 90.32% novel: 100.00%)
Perplexity per Viterbi-tagged test word: 8.152

(c) vtag entrain entest
Tagging accuracy (Viterbi decoding): 94.15% (known: 96.86% novel:

66.08%)
Perplexity per Viterbi-tagged test word: 973.271

(d) vtag entrain4k entest
Tagging accuracy (Viterbi decoding): 83.29% (known: 96.57% novel: 59.26%)
Perplexity per Viterbi-tagged test word: 1078.848

(e) vtag entrain10k entest
Tagging accuracy (Viterbi decoding): 88.02% (known: 96.46% novel: 62.86%)
Perplexity per Viterbi-tagged test word: 1085.756

(f) vtag entrain25k entest
Tagging accuracy (Viterbi decoding): 91.09% (known: 96.51% novel: 64.22%)
Perplexity per Viterbi-tagged test word: 1073.826

(g) vtag cztrain cztest
Tagging accuracy (Viterbi decoding): 47.43% (known: 92.66% novel: 2.84%)
Perplexity per Viterbi-tagged test word: 24863.423

As we can see as compared to the baseline model this performs much better. For example for entrain and entest the baseline score was 92.48% and the smoothed model gives 94.15%. Improvement can also be seen in the novel words from 56.07% to 66.08%. The perplexity per word also increases from 1577.499 to 973.271

5. **Forward-Backward Algorithm:**

vtag till now consists of both Viterbi and posterior decoders and also the one-count smoothing. Adding the posterior decoder makes the algorithm much slower than before. However the results have much higher accuracy than the baseline. The result of running entrain and entest is:
Tagging accuracy (Viterbi decoding): 94.15% (known: 96.86% novel: 66.08%)
Perplexity per Viterbi-tagged test word: 973.271

6. **EM Algorithm:**
The result of running vtagem on entrain25k entest enraw is:

* * ITERATION 0 * * *
Tagging accuracy (Viterbi decoding): 91.09% (known: 96.52% seen: 62.86% novel: 65.65%)
Perplexity per Viterbi-tagged test word: 1107.52
Perplexity per untagged raw word: 1300.02

* * ITERATION 1 * * *
Tagging accuracy (Viterbi decoding): 91.65% (known: 96.50% seen: 69.64% novel: 65.49%)

Perplexity per Viterbi-tagged test word: 918.82
Perplexity per untagged raw word: 717.09

* * ITERATION 2 * * *
Tagging accuracy (Viterbi decoding): 91.27% (known: 96.35% seen: 67.02% novel: 65.08%)
Perplexity per Viterbi-tagged test word: 904.33
Perplexity per untagged raw word: 679.87

* * ITERATION 3 * * *
Tagging accuracy (Viterbi decoding): 90.73% (known: 96.27% seen: 63.72% novel: 62.74%)
Perplexity per Viterbi-tagged test word: 899.90
Perplexity per untagged raw word: 653.48

Answers to the Questions asked:

(a) The algorithm in Figure 3 initializes $\alpha_{###}(0)$ and $\beta_{###}(n)$ to 1 because the total sum of the probabilities when entering these states is 1. $\alpha_{###}(0)$ holds the sum of the forward path probabilities and $\beta_{###}(n)$ holds the backward path probabilities. Since all paths for the forward algorithm must start from $\alpha_{###}(0)$ which is the start state, the forward probability at this state will be 1. Similarly for $\beta_{###}(n)$

(b) The Viterbi-tagged test word perplexity is calculated from the path with the maximum probability found by the model. The perplexity per untagged raw word, on the other hand, is calculated using the forward probability of the final state i.e. the sum of all path probabilities. Since Viterbi-tagged test word perplexity takes it's value from the maximum probability paths that may not be part of the paths that come in the sum of forward probabilities it has higher values than the perplexity per untagged raw word
The perplexity of the untagged raw word is more important than the perplexity of Viterbi tagged test word because it is more adaptable to the real world scenarios where we may not be able to run the re-estimated values on the test set in every iteration.

(c) V counts the types from train and raw but not from test. This is because we use the test set for the actual evaluation of our algorithm. If the test set is also included then all the words will be accountded for and we will not get possible OOV words making it unnecessary to smooth as novel events will never occur.

(d) The EM re-estimation initially helped the tagging accuracy which improved slightly for the first 2 iterations. But, after that the tagging accuracy dropped.
The 'Known' words percentage declined slightly right from the first.

The 'Seen' vocabulary showed the biggest improvement in the first 2 iterations but then it declined.
The 'Novel' words percentage too decreased slightly with every iteration.

(e) The EM re-estimation procedure helped in increaing the percentage of 'Seen' words in raw data. This way we get the best possible counts for our our raw words and in turn helping in picking the best tag for the word instead of getting a sequence of tags for the raw words.

(f) EM did not always help because it does not use much information from the OOV words. Whereas the Viterbi algorithm picks the tag that will improve the maximum probability of the sequence and makes use of OOV words.
Also using only bi-gram may be another reason why the EM model didn't always help, because instead of looking at a larger phrase it just looked at the word in front to compute the tag.

(g) I ate 4kgs of ice cream along with 4 others(so around 800gms/8scoops each) for an ice cream eating contest(if you finish the ice cream you don't pay for it ,you get a free T-shirt and they put your picture on the wall). It didn't make me sick but i couldn't look at ice cream for weeks after that.