CS 510 Advanced Information Retrieval

MP 4

Ques 1:

(a) Forward Algorithm:

The α values computed using forward algorithm are as follows:

$\alpha_1(V) = 0.1$
$\alpha_1(N) = 0.05$
$\alpha_2(V) = 0.0085$
$\alpha_2(N) = 0.013$
$\alpha_3(V) = 0.00696$
$\alpha_3(N) = 0.00099$
$\alpha_4(V) = 0.0004671$
$\alpha_4(N) = 0.0019674$

The data likelihood probability is given by:

P ("print line hello world") = P (X | lambda) = $\alpha_4(V) + \alpha_4(N) = 0.0024345$

(b) Backward Algorithm:

The β values computed using forward algorithm are as follows:

$\beta_1(V) = 0.01592$
$\beta_1(N) = 0.01685$
$\beta_2(V) = 0.122$
$\beta_2(N) = 0.1075$
$\beta_3(V) = 0.3$
$\beta_3(N) = 0.35$
$\beta_4(V) = 1.0$
$\beta_4(N) = 1.0$

The data likelihood probability is given by:

P ("print line hello world") = P (X | lambda) = $\alpha_1(V)\beta_1(V) + \alpha_1(N)\beta_1(N) = 0.0024345$

Note that data likelihood using both the algorithms is equal.

Ques 2:

Note that in my implementation, the words are case sensitive i.e. for example, 'This' and 'this' are treated as two separate words in the vocabulary and the datasets.

(a)

Start Probabilities for each of tags are as follows:

Tags	Start Probability
RB	0.009640
NN	0.028738
CC	0.000624
VBN	0.006548
JJ	0.019930
IN	0.017279
VBZ	0.000312
DT	0.344732
NNS	0.011953
NNP	0.560244

Top 10 words with highest output probabilities for each of tags are as follows:

Tags	**Top 10 words with highest Output Probabilities**
RB	['also', 'currently', 'now', 'well', 'as', 'not', 'commonly', 'just', 'approximately', 'only']
NN	['family', 'moth', 'village', 'district', 'state', 'town', 'area', 'station', 'genus', 'film']
CC	['and', 'or', 'but', 'either', 'both', 'And', 'et', 'moth', 'plus', 'But']
VBN	['located', 'known', 'based', 'owned', 'been', 'used', 'situated', 'named', 'operated', 'written']
JJ	['American', 'historic', 'small', 'Indian', 'former', 'geologic', 'civil', 'public', 'national', 'southern']
IN	['of', 'in', 'by', 'on', 'as', 'at', 'for', 'with', 'from', 'near']
VBZ	['is', 'has', 'consists', 'includes', 'serves', 'lies', 'features', 'contains', 'plays', 'provides']
DT	['the', 'a', 'The', 'an', 'This', 'A', 'this', 'both', 'all', 'An']
NNS	['species', 'stars', 'roles', 'forests', 'moths', 'sports', 'areas', 'people', 'services', 'spiders']
NNP	['County', 'New', 'England', 'River', 'India', 'South', 'District', 'US', 'Crambidae', 'School']

(b)

Note that I used log in the Viterbi algorithm while implementation i.e. rather than maximizing the product of terms, I maximized the sum of logarithm terms in the algorithm, since when the sentences are longer, the product of terms is zero to machine precision and results are not very accurate.

As we deal with longer sentences, the parameter values get extremely small very fast. Taking the logarithm, product of these almost zero (very small) values is converted to sum of logarithm of these values which can then be used as the quantity to be maximized in the Viterbi algorithm. In my implementation, I have used the same technique.

The tagging results of the simple Viterbi algorithm (without any smoothing) applied on test_0 file is stored in a new file on Gitlab in the: '**TaggingResults_files/ tag_results_test_0_RK'**

(c)

If we a given a sentence that contains words that are not in the vocabulary, simple Viterbi algorithm used in Part (b) cannot be used since for words that are not in vocabulary, in the Emission Probability Matrix B, probability of generating this unseen word given the tag i.e.

B[tag][unseen word] doesn't exist.

While computing the parameters using the training data set, emission probabilities of generating only seen words (in the vocabulary) given the tag was computed, and given a new unseen word, we do not know the probability of generating it given a POS tag.

Hence, the Viterbi path Probability at time t+1 ending in state i, $VP^{(i)}_{t+1} = \max_j VP^{(j)}_t \ A_{ji} \ B_i(o_{t+1})$ could not be computed for the new unseen word, where A_{ji} = Transition probability from state j to state i and $B_i(o_{t+1})$ = emission probability of generating observation o_{t+1} when at state i.

When we used the training dataset to compute the parameter values, especially the emission probabilities, we used the Maximum likelihood estimates to do so, namely:

$P(w_j | t_i) = \frac{c(w_j | t_i)}{c(t_i)}$ where $c(w_j | t_i)$ is no. of times we see word w_j with POS tag t_i and $c(t_i)$ is no. of times we see tag t_i.

Now, the Maximum likelihood estimates assign zero values to the emission state probability if the word is unseen in the training dataset i.e. when $c(w_j | t_i) = 0$. This clearly is not accurate since there may be some unseen words in the training set which may appear in the test set as with test_1 file.

Hence, the algorithm in part (b) cannot be used in the case.

<u>METHOD 1 to solve this problem: SMOOTHING</u>

We can solve this problem using smoothing techniques. Smoothing is important because maximum likelihood estimates given assign zero values to the emission probability of unseen words in the training set. We may use Laplace smoothing or Absolute Discounting smoothing methods to smooth the parameters and then apply Viterbi Algorithm.

I used Laplace Smoothing and then applied Viterbi Algorithm to the test_1 dataset to generate POS tags file 'tag_results_test_1_RK_LaplaceSmoothed_Viterbi'.

Laplace smoothing (Add 1 smoothing) will avoid assigning zero values of probability for estimated parameters, even when the data is unobserved in the training set. It is implemented as follows:

1. While computing the emission probability i.e. Probability of observing a word w_j from the tag t_i, add 1 to the no. of times w_j is observed from the tag t_i i.e. add 1 to $c(w_j|\ t_i)$ for each instance. Since emission probabilities must add to 1 for each tag t_i, normalize by $c(t_i)+ |V|$ where $|V|$ is the size of vocabulary rather than normalizing by $c(t_i)$.

   Hence, using Laplace smoothing, the emission probabilities are computed as
   $P(w_j|\ t_i) = \frac{c(w_j|\ t_i)+1}{c(t_i)+|V|}$ where $c(w_j|\ t_i)$ is no. of times we see word w_j with POS tag t_i and $c(t_i)$ is no. of times we see tag t_i and $|V|$ is the size of the vocabulary.

2. While computing the transition probability i.e. Probability of moving to state t_j from state t_i, add 1 to the no. of times tag t_i is followed by tag t_j i.e. add 1 to $c(t_i, t_j)$ for each instance. Since transition probabilities must add to 1 for each tag t_i, normalize by $\sum_{j'} c(t_i, t_{j'}) + N$ where N is the no. of states in the system rather than normalizing by $\sum_{j'} c(t_i, t_{j'})$.

   Hence, using Laplace smoothing, the transition probabilities are computed as
   $P(t_j|\ t_i) = \frac{c(t_i,t_j)+1}{\sum_{j'} c(t_i,t_{j'})+ N}$ where $c(t_i, t_j)$ is no. of times we see tag t_i followed by tag t_j and N is the total no. of POS tags or states.

3. While computing the start probabilities, i.e. the probability of starting from state t_i, add 1 to the count of sentences that start with tag t_i i.e. add 1 to c (#s, t_i) for each instance. Since initial probabilities must add to 1 for all tags t_i, normalize by $|D|+ N$, where $|D|$ is the total no. of sentences in training set and N is the total no. of POS tags or states in the system, rather than normalizing by $|D|$.

   Hence, using Laplace smoothing, the start probabilities are computed as

$P(t_i) = \frac{c(\#s, t_i) + 1}{|D| + N}$ where $c(\#s, t_i)$ is no. of sentences we see starting with tag t_i, $|D|$ is the total no. of sentences in the training set and N is the total no. of POS tags or states.

Hence, after applying Laplace smoothing, the parameters estimates are as follows:

Start Probabilities: $P(t_i) = \frac{c(\#s, t_i) + 1}{|D| + N}$

Transition Probabilities: $P(t_j | t_i) = \frac{c(t_i, t_j) + 1}{\sum_{j'} c(t_i, t_{j'}) + N}$

Emission Probabilities: $P(w_j | t_i) = \frac{c(w_j | t_i) + 1}{c(t_i) + |V|}$

Next, Viterbi algorithm is applied after smoothing the parameters to the test_1 dataset to get the POS tags.

The tagging results of the smoothed Viterbi Algorithm applied on test_1 file is stored in a new file on Gitlab in the: **'TaggingResults_files/ tag_results_test_1_RK_LaplaceSmoothed_Viterbi'**

<u>METHOD 2 to solve this problem:</u>

Another method to solve this issue of unseen words in the test file is to modify the Viterbi algorithm in a way that we neglect the emission probability term in the algorithm for the unseen words since $B_i(o_{t+1})$ term is unknown for an unseen observation o_{t+1} not in the training set:

The Viterbi path Probability at time t+1 ending in state i, **$VP^{(i)}_{t+1} = \max_j VP^{(j)}_t \ A_{ji} \ B_i(o_{t+1})$ for the seen words in the vocabulary**, where A_{ji} = Transition probability from state j to state i and

$B_i(o_{t+1})$ = emission probability of generating observation o_{t+1} when at state i, whereas **for new unseen words Viterbi path Probability at time t+1 ending in state i is computed as $VP^{(i)}_{t+1} = \max_j VP^{(j)}_t \ A_{ji}$** and state chosen 'i' is computed at $q_{t+1}^{(i)} = q_t(k).(i)$ where $k = \text{argmax}_j \ VP^{(j)}_t \ A_{ji}$.

Similarly, for the initial Viterbi path probability, **$VP^{(i)}_1 = \max_i \pi_i B_i(o_1)$ for seen words** in the vocabulary and **for the new unseen word, we compute $VP^{(i)}_1 = \max_i \pi_i$** and the state chosen 'i' is computed at $q_1^{(i)} = \text{argmax}_i \ \pi_i$.

Modified Viterbi Algorithm to deal with unseen word problem is therefore as follows:

1.

$$VP^{(i)}_1 = \begin{cases} \max_i \pi_i B_i(o_1), \text{ for seen words in the vocabulary} \\ \max_i \pi_i, \text{ for unseen words in the vocabulary} \end{cases}$$

$$q_1^{(i)} = \begin{cases} \text{argmax}_i \pi_i B_i(o_1), \text{ for seen words in the vocabulary} \\ \text{argmax}_i \pi_i, \text{ for unseen words in the vocabulary} \end{cases}$$

2.

$$VP^{(i)}_{t+1} = \begin{cases} \max_j VP^{(j)}_t \ A_{ji} \ B_i(o_{t+1}), \text{ for seen words in the vocabulary} \\ \max_j VP^{(j)}_t \ A_{ji} \text{ for unseen words in the vocabulary} \end{cases}$$

$$q_{t+1}^{(i)} = \begin{cases} q_t(k).(i) \text{ where } k = \text{argmax}_j VP^{(j)}_t \ A_{ji} \ B_i(o_{t+1}), \text{ for seen words in the vocabulary} \\ q_t(k).(i) \text{ where } k = \text{argmax}_j VP^{(j)}_t \ A_{ji}, \text{ for unseen words in the vocabulary} \end{cases}$$

3.

$q^* = q_T(k)$ where $k = \text{argmax}_i VP^{(i)}_T$

The tagging results of the smoothed Viterbi Algorithm applied on test_1 file is stored in a new file on Gitlab in the: **'TaggingResults_files/tag_results_test_1_RK_Method2_Modified_Viterbi'**

I have implemented both the solving techniques in the Python code and computed the resulted tags file and accuracy.

(d)

We define Accuracy = $\frac{No.of\ correct\ tags}{Total\ no.of\ tags}$ which is computed by matching

1. 'tag_results_test_0_RK' (Viterbi Algorithm) with 'test_0_tagging'
2. 'tag_results_test_1_RK_LaplaceSmoothed_Viterbi' (smoothed Viterbi Algorithm) with 'test_1_tagging'
3. 'tag_results_test_1_RK_Method2_Modified_Viterbi' (Modified Viterbi Algorithm with VP=VP*A for unseen words) with 'test_1_tagging'

Then,

1. Accuracy for tagging in test_0 file using Viterbi Algorithm = 0.9741666
2. Accuracy for tagging in test_1 file using Smoothed using Laplace Viterbi Algorithm = 0.918967
3. Accuracy for tagging in test_1 file using Modified Viterbi Algorithm with VP=VP*A for unseen words = 0.9403383

Note that the accuracy of Modified Viterbi algorithm which is explained as Method 2 in the Part (c) has higher accuracy than Laplace smoothed Viterbi algorithm. But none the less, laplace smoothing makes more sense and is more robust method I believe.

Submitted by:
Rachneet Kaur NET ID: rk4