Machine Learning Report

Group members: Lo An Guo (1003142), Tan Xiang Hao (1002746), Wong Cheow Fu (1002848)

Part 2

For part 2, we followed the formula given in the project handout to calculate the estimated emissions, where we count the total number of each observation-label pair and divide that by the total number of occurrences of that label.

To find the words that have to be replaced with the special word token #UNK#, we do a count of the number of times that each observation occurred. If it was less than k, the counts of observation-label pairs containing that observation will be added to the #UNK#-label pair count of that label. Similarly, to estimate the emission, the observation-label pair counts are then divided by the number of occurrences of the label.

In the test phase, if the observed word is not in the list of all words in emissions, the observation will also be replaced with #UNK#. The observation-label pair with the highest emission value where the observation occurred will be picked.

For the function that estimates the emission parameters from the training set using Maximum Likelihood Estimation (MLE), please refer to codes.

Results for MLE with smoothing:

AL #Entity in gold data : 8408 #Entity in prediction: 19484 #Correct Entity : 2898 Entity Precision: 0.1487 Entity recall: 0.3447 Entity F: 0.2078 #Correct sentiment : 2457 Sentiment precision: 0.1261 Sentiment recall: 0.2922 Sentiment F: 0.1762	**CN** #Entity in gold data : 1478 #Entity in prediction: 9373 #Correct Entity : 765 Entity Precision: 0.0816 Entity recall: 0.5176 Entity F: 0.1410 #Correct sentiment : 285 Sentiment precision: 0.0304 Sentiment recall: 0.1928 Sentiment F: 0.0525
EN #Entity in gold data : 13179 #Entity in prediction: 19406	**SG** #Entity in gold data : 4537 #Entity in prediction: 18451

#Correct Entity : 9152 Entity Precision: 0.4716 Entity recall: 0.6944 Entity F: 0.5617 #Correct sentiment : 7644 Sentiment precision: 0.3939 Sentiment recall: 0.5800 Sentiment F: 0.4692	#Correct Entity : 2632 Entity Precision: 0.1426 Entity recall: 0.5801 Entity F: 0.2290 #Correct sentiment : 1239 Sentiment precision: 0.0672 Sentiment recall: 0.2731 Sentiment F: 0.1078

Part 3

To obtain the transition parameters, we counted for all the combinations of y1 and y2 by looping through the training set's tweets and then through the observation-label pairs in each tweet. The START and STOP states were added to the start and the end of each tweet respectively. Finally, we take the count for each pair and divide it with the number of times that y1 was counted for. Please refer to codes for the function that estimates the transmission parameters from the training set using MLE.

For the Viterbi algorithm, we implemented it recursively by looking at the index of the observation and the label for that particular observation. When k is 1 (or in our case, when the wordIndex = 0), we use the product of the emission value of the first observation from the label and the transition from START to the label. When k is greater than 1 (or wordIndex > 0), we would then want to find the maximum of the products of a recursive call to itself on k-1 and the previous score, the tag transition value from u to v and the word emission value of the observation at index k from v, for every u in the set of possible labels.

As part of the forward-backward algorithm, we also implemented a backward function which would return the label with the highest probability for the observation at the index k. This function is called starting with k (the length of the tweet) and v (the STOP symbol). Each label with the maximum score is saved into a list of labels and the backward function is called on each previous observation in the tweet. We then compare this backward score to the one that we got from the forward algorithm. Whichever is the highest, we would then assign the corresponding tag.

To solve the underflow issue, we can take the negative log of the parameters. We can modify the Viterbi algorithm such that at each node, we choose the min rather than the max of all the possible scores.

Precision, recall and F scores of all systems using Viterbi:

AL #Entity in gold data: 8408 #Entity in prediction: 8449 #Correct Entity : 6705 Entity precision: 0.7936 Entity recall: 0.7975 Entity F: 0.7955 #Correct Sentiment : 6057 Sentiment precision: 0.7169 Sentiment recall: 0.7204 Sentiment F: 0.7186	**CN** #Entity in gold data: 1478 #Entity in prediction: 714 #Correct Entity : 307 Entity precision: 0.4300 Entity recall: 0.2077 Entity F: 0.2801 #Correct Sentiment : 210 Sentiment precision: 0.2941 Sentiment recall: 0.1421 Sentiment F: 0.1916
EN #Entity in gold data: 13179 #Entity in prediction: 13403 #Correct Entity : 11015 Entity precision: 0.8218 Entity recall: 0.8358 Entity F: 0.8288 #Correct Sentiment : 10385 Sentiment precision: 0.7748 Sentiment recall: 0.7880 Sentiment F: 0.7814	**SG** #Entity in gold data: 4537 #Entity in prediction: 3007 #Correct Entity : 1661 Entity precision: 0.5524 Entity recall: 0.3661 Entity F: 0.4403 #Correct Sentiment : 1035 Sentiment precision: 0.3442 Sentiment recall: 0.2281 Sentiment F: 0.2744

Part 4

For part 4, we implemented an algorithm that was similar to the one used in part 3, where we considered each label for all the observations as well as the previous tag for the observation before it. One key difference is that we store the 7 best paths that leads up to every single observation in a sorted array based on their score. As the algorithm moves forward, all paths are sorted using their scores and the top 7 paths are stored for each node. Hence, to find the 7th best output sequence, we would just need to look for the path with the 7th best score on the final node.

Algorithm:

1. Let S(i,u) denote the array of scores for the node at layer i and state u.
   Let P(i,u) denote the array of paths stored for the node at layer i and state u.
   Base Case: S(0, START)[0] = 1 and P(0, START)[0] = “START”

2. Moving forward recursively, $S(i, v)[0, 1, \cdots, k-1] = \text{arg sort}_{u,t' \in \{0 \ldots k-1\}} S(i-1, u)[t'] \cdot a_{u,v} \cdot b_v(x_i)$ for all $i \in \{1, \cdots, n\}$
3. We will also set the path for each score in the node, $P(i, v)[t] = P(i-1, u)[t'] + (u \rightarrow v)$ where the path $P(i, v)[t]$ has a score $S(i, v)[t]$.
4. We will sort the scores and paths array at each node, and only keep the k-best.
5. Finally, for i = n+1, we have $S(n+1, \text{STOP})[0, 1, \cdots, k-1] = \text{arg sort}_{u,t0 \in \{0,1,\cdots,k-1\}} S(n, u)[t'] \cdot a_{u,\text{STOP}}$
6. To retrieve the 7th best output sequence, we obtain it from P(n+1, STOP)[6]

The results for the algorithm we implemented are as follows:

AL	**EN**
#Entity in gold data: 8408 #Entity in prediction: 8968 #Correct Entity : 5983 Entity precision: 0.6671 Entity recall: 0.7116 Entity F: 0.6887 #Correct Sentiment : 5011 Sentiment precision: 0.5588 Sentiment recall: 0.5960 Sentiment F: 0.5768	#Entity in gold data: 13179 #Entity in prediction: 13684 #Correct Entity : 10448 Entity precision: 0.7600 Entity recall: 0.7891 Entity F: 0.7743 #Correct Sentiment : 9822 Sentiment precision: 0.7178 Sentiment recall: 0.7453 Sentiment F: 0.7313

Part 5

In part 5, to improve the speed and accuracy of the part-of-speech tagging, our group decided to implement the Trigrams'n'Tag (TnT) model. It uses second order Markov models where the states represent the tags and the outputs represent the words, similar to the label and observation that we see in this project. Using unigrams, bigrams and trigrams, we can derive the maximum likelihood probabilities:

$$\text{Unigrams:} \quad \hat{P}(t_3) = \frac{f(t_3)}{N} \tag{2}$$

$$\text{Bigrams:} \quad \hat{P}(t_3|t_2) = \frac{f(t_2, t_3)}{f(t_2)} \tag{3}$$

$$\text{Trigrams:} \quad \hat{P}(t_3|t_1, t_2) = \frac{f(t_1, t_2, t_3)}{f(t_1, t_2)} \tag{4}$$

and estimate the trigram probability as follows:

$$P(t_3|t_1,t_2) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_1,t_2) \quad (6)$$

Based on several tests runs that we had, we found out that the weight for trigrams (lambda3) had to be reduced significantly. This is because the trigram probabilities generated could not be used due to the sparse-data problem. What this means is that there does not exist enough instances of trigrams such that a reliable estimate can be used to find the probability.

The smoothing paradigm that returns the best outcome is using the linear interpolation of unigrams, bigrams and trigrams, as depicted in equation (6) above, with lambda1 + lambda2 + lambda3 = 1, so that P represents a probability distribution. We have also deduced that changing the k value with respect to this model also further improves the results. Optimum F scores can be obtained when the k = 2 for the EN dataset and k = 5 for the AL dataset.

The reason behind this is that for the EN dataset, lowering the k value would mean that words that appear less frequently are to be considered and tagged as well, as opposed to k = 3 for part 2. By reducing the number of words that are being ignored, a more complete dataset can then be used to run the tagging algorithm on, resulting in a more accurate tagging procedure due to the increase in exposure to a wider variety of sentence structures. For the AL dataset, increasing the k value actually yielded better results due to the difference in the lexicon of both languages.

In the Chinese language, one does not need to use as many different characters in order to converse. Different definitions are created by combining words together. For example, if we were to look at the Chinese words “火” (huǒ) and “山” (shān), they mean fire and mountain respectively. However, putting them together gives us a volcano - “火山” (huǒ shān), without the use of any new characters.

In the English language, the opposite is true, where we use a new term “volcano” to describe the geological feature and not just describe it as a “fire mountain”. Hence, a lower k value is needed for the EL dataset in order to take into account all the different, more specific use cases of the words. As for the AL dataset, a higher k value would return better results as we would need to smooth out the characters and find out which words are used more, as the frequency of a conversational Chinese character being used is higher than in English. This would then allow us to sieve out the words that are more essential and improve the accuracy for part-of-speech tagging, as seen from the F scores as follows:

AL	**EN**
#Entity in gold data: 8408 #Entity in prediction: 8660 #Correct Entity : 6845 Entity precision: 0.7904 Entity recall: 0.8141	#Entity in gold data: 13179 #Entity in prediction: 13451 #Correct Entity : 11312 Entity precision: 0.8410 Entity recall: 0.8583

Entity F: 0.8021 #Correct Sentiment : 6221 Sentiment precision: 0.7184 Sentiment recall: 0.7399 Sentiment F: 0.7290	Entity F: 0.8496 #Correct Sentiment : 10898 Sentiment precision: 0.8102 Sentiment recall: 0.8269 Sentiment F: 0.8185

References

Brants, T. (2000, April). TnT: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing* (pp. 224-231). Association for Computational Linguistics.

Link to research paper: http://tagh.de/tom/wp-content/uploads/brants-2000.pdf