01.112 Machine Learning Project Report

Group Members:

Lee Hang Wee (1002727)

Billio Jeverson (1002939)

Khairunnisa Bte Kunhimohamed N (1002508)

PART 2

1) The first method created input lines and output a dictionary.

- Lines refer to a readlines() of the training data set
- The output the dictionary has key of (x,y) x being the words and key being the tag and value of the emission score of that particular word to tag.

2) The second function created is called the smoothEmission where it takes in line1, line2, k as inputs and output Line1 and Line2:

- The line1 refers to line from r.readlines() of the training datasets while line2 refers to the lines from the test (dev.in) data sets
- The function will check for existence(frequency) of the words in the training set if it appeared less than k-times, it will be replaced with “unk”. It will also check the entry of the test (Dev.in) set and check if the word exist in the training set. It will replace the word with “unk” as well if it does not exist in the training set.
- The function will then output this new cleaned list of words of train set called Line1 and test set of Line2 with a few words being the place with unk.

3) The last function returns a dictionary with the given word as key and label as valu

4) Main function takes in path1,path2, path3.

- This function integrate the all the three function with path1 being the path of the train set, path 2 being the test set path and path 3 being the output of our predicted model.

Results for Part 2:

In the file name dev.p2.out

Accuracy :

EN

```
#Entity in gold data: 13179
#Entity in prediction: 19406

#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
```

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
```

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
```

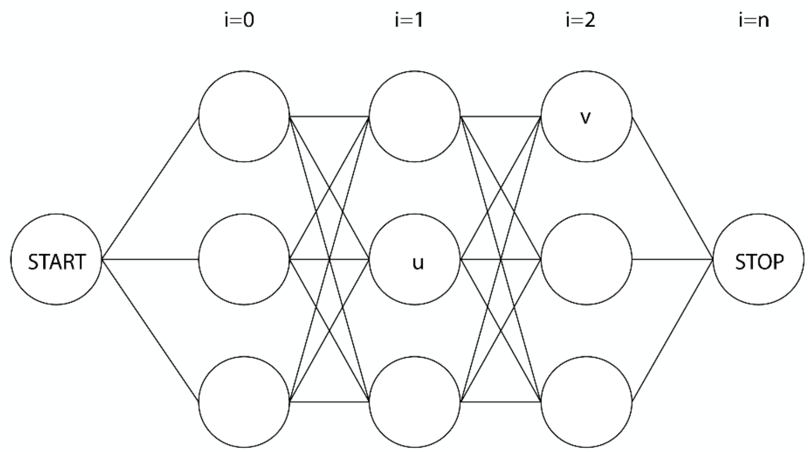
SG

```
#Entity in gold data: 4537
#Entity in prediction: 18451

#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



Legend:
i = Layer Index
n = length of sequence
T = total number of states

Transition values *b(u, v)* were obtained by going through the dataset and retrieving the number of instances whereby a transition from state u to state v *(count(u,v))* was observed, and also the number of times each state u *count(u)* was observed. The transition values are then computed as *(count(u,v)) / count(u)*.

The standard Viterbi algorithm was implemented in Part 3. Due to an underflow issue that was observed, the Viterbi algorithm was slightly modified to ensure underflow does not occur.

Underflow is a concern with a dynamic programming problem of this form, since we compute products of probabilities. Fortunately, underflow is easily avoided by simply taking logarithms. The Following is the underflow-resistant version of the DP algorithm.

1. Let $\hat{\delta}_0(i) = \log[\pi_i b_i(\mathcal{O}_0)]$, for $i = 0, 1, \ldots, N-1$.
2. For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, compute

$$\hat{\delta}_t(i) = \max_{j \in \{0,1,\ldots,N-1\}} \left\{ \hat{\delta}_{t-1}(j) + \log[a_{ji}] + \log[b_i(\mathcal{O}_t)] \right\}.$$

In this case, the optimal score is

$$\max_{j \in \{0,1,\ldots,N-1\}} [\hat{\delta}_{T-1}(j)].$$

Of course, additional bookkeeping is still required to find the optimal path.

The figure above shows how we modified the algorithm:

1. For each node u in first layer (i=0):

   Score = log(a(START, u)) + log(b(u->O))

2. For each node u in layers i=1...n-1:

   Score = $\max_{j \text{ in } \{0,\ldots,T-1\}}$ {score(i-1)(j) + log(a(j,u)) + log(b(u->O))

3. For STOP node (i=n):

   Score = $\max_{j \text{ in } \{0,\ldots,T-1\}}$ {score(n-1)(j) + log(a(j,STOP))

In simpler terms, the only modification made was to log the emission and transition values when computing the scores in each node. The log of a very small positive number would translate to an integer. This helps the model avoid underflow.

Results for Part 3:

EN

```
#Entity in gold data: 13179
#Entity in prediction: 13156

#Correct Entity : 11087
Entity  precision: 0.8427
Entity  recall: 0.8413
Entity  F: 0.8420

#Correct Sentiment : 10616
Sentiment  precision: 0.8069
Sentiment  recall: 0.8055
Sentiment  F: 0.8062
```

AL

```
#Entity in gold data: 8408
#Entity in prediction: 8465

#Correct Entity : 6696
Entity  precision: 0.7910
Entity  recall: 0.7964
Entity  F: 0.7937

#Correct Sentiment : 6048
Sentiment  precision: 0.7145
Sentiment  recall: 0.7193
Sentiment  F: 0.7169
```

SG

```
#Entity in gold data: 4537
#Entity in prediction: 2958

#Correct Entity : 1638
Entity  precision: 0.5538
Entity  recall: 0.3610
Entity  F: 0.4371

#Correct Sentiment : 1019
Sentiment  precision: 0.3445
Sentiment  recall: 0.2246
Sentiment  F: 0.2719
```

CN

```
#Entity in gold data: 1478
#Entity in prediction: 682

#Correct Entity : 303
Entity  precision: 0.4443
Entity  recall: 0.2050
Entity  F: 0.2806

#Correct Sentiment : 210
Sentiment  precision: 0.3079
Sentiment  recall: 0.1421
Sentiment  F: 0.1944
```

PART 4

Explanation of algorithm:

Part 4 was a modification of the Part 3 algorithm as shown above. To compute the 7-th best path, some modifications were made as shown below:

- In each node, instead of storing the maximum score, we stored the top 7 scores in an array.
- At the end of the Viterbi, the STOP node will contain an array with 7 best scores in descending order.
- Taking the 7th score, we backtrack as per usual to get the optimal path. This is possible as we stored the 7 best scores in each node and their corresponding nodes and array position they were from. For example:

    Node u : [(score1, parent_state, parent_state_position), … ,(score7, parent_state, parent_state_position)]

    *parent_state is the state where this score was calculated from

    *parent_state_position is an integer from 0-6 which tells us the position in the array in the parent_state that it came from

This recursive algorithm allowed more efficient computation as compared to a naïve method of computing all possible paths and taking the 7th best path.

Results for Part 4:

EN

```
#Entity in gold data: 13179
#Entity in prediction: 13432

#Correct Entity : 10520
Entity  precision: 0.7832
Entity  recall: 0.7982
Entity  F: 0.7907

#Correct Sentiment : 9967
Sentiment  precision: 0.7420
Sentiment  recall: 0.7563
Sentiment  F: 0.7491
```

AL

```
#Entity in gold data: 8408
#Entity in prediction: 8885

#Correct Entity : 5938
Entity  precision: 0.6683
Entity  recall: 0.7062
Entity  F: 0.6868

#Correct Sentiment : 4969
Sentiment  precision: 0.5593
Sentiment  recall: 0.5910
Sentiment  F: 0.5747
```

PART 5

Model used: **Structured Perceptron**[1]

The model used is based on the averaged or voted perceptron algorithm. The model also relies on Viterbi decoding of training examples, combined with simple additive updates to its feature set. In the (bigram) HMM tagger as implemented from Part 1 to Part 3, each bigram of tags (u,v) is associated with its transition parameter: $a_{u,v} = P(v \mid u)$ and each tag/word pair (u,o) is associated with its emission parameter $b_{u,o} = P(o \mid u)$.

As an alternative to maximum likelihood parameter estimates (as implemented in Part 1 to Part 3), the Structured Perceptron algorithm proposes the following estimation algorithm to calculate the transition and emission parameters (or feature set in the case of structured perceptron) for decoding:

The training set consist of n tagged sentences, the i^{th} sentence being of length $n_{i.}$

Each sentence has sequence of observations/words $w^{(i)}$ and its corresponding (ground truth) sequence of tags $t^{(i)}$.

The transition parameter $a_{u,v}$ is slightly modified to $\alpha_{u,v} = log\ (a_{u,v})$

Likewise, the emission parameter $b_{u,o}$ is slightly modified to $\beta_{u,o} = log\ (\ b_{u,o}\ )$

Inputs: Training set of n tagged sentences, the i^{th} sentence being of length $n_{i.}$ A parameter T defining the number of iterations over the training set.

Initialization: Set all parameters $a_{u,v}$ *and* $b_{u,o}$ to be zero.

Algorithm:

1. For $t = 1...T,\ i = 1....n,$
   - Apply the Viterbi algorithm to find the best tagged sequence for sentence $w^{(i)}$ under the existing parameter settings. The predicted tag sequence will be $z^{(i)}$
   - For every tag transition (u,v) seen d_1 times in $t^{(i)}$ and d_2 times in $z^{(i)}$ where d_1 =/= d_2, set $\alpha_{u,v} = \alpha_{u,v} + d_1 - d_2$

[1] Collins, M. (2002). Discriminative training methods for hidden Markov models. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1–9. doi: 10.3115/1118693.1118694

- For every tag/word pair *(u,o)* seen d_1 times in ($w^{(i)}$, $t^{(i)}$) and d_2 times in ($w^{(i)}$, $z^{(i)}$) where d_1 =/= d_2, set $\beta_{u,o} = \beta_{u,o} + d_1 - d_2$
  - Intuitively this has the effect of increasing the parameter values for the features[2] which were missing from the predicted sequence $z^{(i)}$, and reduce values for "incorrect" features in the sequence $z^{(i)}$. Hence, if $t^{(i)}$ sequence is equal to $z^{(i)}$ sequence, no changes are made to the parameter values because the tag sequence is correct.
- Let $\alpha^{i}_{u,v}$ and $\beta^{i}_{u,o}$ be the parameters after the i^{th} training example has been processed in iteration t over the training data. Calculate the sum of the each $\alpha^{i}_{u,v}$ and $\beta^{i}_{u,o}$ respectively through:

$$\boldsymbol{\alpha}_{sum} = \sum_{t=1}^{T} \sum_{i=1}^{n} \boldsymbol{\alpha}_{u,v}^{t,i}$$

$$\beta_{sum} = \sum_{t=1}^{T} \sum_{i=1}^{n} \beta_{u,o}^{t,i}$$

2. To better generalize the model, the final parameter values will be the averaged transmission and emission parameters to be used for decoding and tag sequence prediction through Viterbi algorithm as implemented in Part 3:

$$\boldsymbol{\alpha}_{avg} = \frac{\boldsymbol{\alpha}_{sum}}{nT}$$

$$\beta_{avg} = \frac{\beta_{sum}}{nT}$$

However, this implementation did not result in a higher F-score than the HMM algorithm implemented in Part 1 – Part 3. This could be due to the usage of only 2 features of bigram tag transitions and current tag/word emission (and their corresponding parameters). In the original paper that inspired this implementation, there were other features that could be accounted, as shown in the Figure 1.

[2] The features in our implementation of the Structured Perceptron refers to every bigram of tags *(u, v)* and every tag/word pair *(u, o)*

Current word	w_i	$\& \ t_i$
Previous word	w_{i-1}	$\& \ t_i$
Word two back	w_{i-2}	$\& \ t_i$
Next word	w_{i+1}	$\& \ t_i$
Word two ahead	w_{i+2}	$\& \ t_i$
Bigram features	w_{i-2}, w_{i-1}	$\& \ t_i$
	w_{i-1}, w_i	$\& \ t_i$
	w_i, w_{i+1}	$\& \ t_i$
	w_{i+1}, w_{i+2}	$\& \ t_i$
Current tag	p_i	$\& \ t_i$
Previous tag	p_{i-1}	$\& \ t_i$
Tag two back	p_{i-2}	$\& \ t_i$
Next tag	p_{i+1}	$\& \ t_i$
Tag two ahead	p_{i+2}	$\& \ t_i$
Bigram tag features	p_{i-2}, p_{i-1}	$\& \ t_i$
	p_{i-1}, p_i	$\& \ t_i$
	p_i, p_{i+1}	$\& \ t_i$
	p_{i+1}, p_{i+2}	$\& \ t_i$
Trigram tag features	p_{i-2}, p_{i-1}, p_i	$\& \ t_i$
	p_{i-1}, p_i, p_{i+1}	$\& \ t_i$
	p_i, p_{i+1}, p_{i+2}	$\& \ t_i$

Figure 1 [3]

Hence, we have further implemented another feature: next tag (as indicated in Figure 1) which indicates the relation between the current tag and the next tag in a sentence.

In the file part5.py, the tag sequences predicted for the test set are based on additional parameters from this feature.

Although this helps to account for bi-directional relationship between the tags (that represents the word emitted), the F-score performed slightly better than having just 2 features as mentioned above (from 0.70 to 0.75 for EN dataset) but not better than the HMM method. For future implementation, we aim to account for other features in Figure 1 to experiment with and find out the best generalized model with Structured Perceptron method.

[3] Feature templates used in POS-tagging task. w_i is the current word, and w_1 … w_n is the entire sentence of words. p_i is tag/label for the current word and p_1 … p_n is the tag sequence for the sentence. t_i is the tag that emitted the ith word.

Results for Part 5:

Best result with iterations T = 3 for dev.in set for EN dataset:

```
#Entity in gold data: 13179
#Entity in prediction: 14643

#Correct Entity : 10424
Entity  precision: 0.7119
Entity  recall: 0.7910
Entity  F: 0.7493

#Correct Sentiment : 9853
Sentiment  precision: 0.6729
Sentiment  recall: 0.7476
Sentiment  F: 0.7083
```

Best result with iterations T = 2 for dev.in set for AL dataset:

```
#Entity in gold data: 8408
#Entity in prediction: 10817

#Correct Entity : 5893
Entity  precision: 0.5448
Entity  recall: 0.7009
Entity  F: 0.6131

#Correct Sentiment : 4902
Sentiment  precision: 0.4532
Sentiment  recall: 0.5830
Sentiment  F: 0.5100
```