SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

01.112 Machine Learning Design Project

Angelia Lau Kah Mun (1002417)

Kimberlyn Loh (1002221)

Rayson Lim Jun Kai (1002026)

Contents

1 Part 2: Estimating Emission Parameters of HMM

1.1 Approach

The estimation of emission parameters $b_i(o)$ of a Hidden Markov Model (HMM) can be calculated using the Maximum Likelihood Estimate (MLE):

$$b_i(o) = \frac{count(i \to o)}{count(i)}$$

where $count(i \to o)$ is the number of times the observation o was emitted from the tag i and $count(i)$ is the number of times the tag i appears in the training data.

To account for words that appear in the test set but not in the training set, a special word token $\#UNK\#$ is used to replace words that are not in the training set. The emissions probability then becomes:

$$b_i(o) = \begin{cases} \frac{count(i \to o)}{count(i)+k}, & \text{if word o appears in training set} \\ \frac{k}{count(i)+k}, & \text{otherwise} \end{cases}$$

where k is a value that we can choose. This is a smoothing technique. We are essentially assuming that for any tag y there is a certain chance of emitting $\#UNK\#$ as a rare event. We will be using $k = 1$ for this project.

Given a training file, the python function `estEmission(file,k)` will calculate $b_i(o)$ for every i, o pair. It utilises the formula above and returns a dictionary in the form of {`i:{o:emission probability}`}.

Given a test file, we perform tag generation for output o by selecting the tag i with the maximum probability from the dictionary. If the word is not found in the dictionary, we replace the word with the $\#UNK\#$ token.

1.2 Results

As seen in Figure 1 below, the F score for the EN dataset is the highest at 0.469 while the F scores for the remaining datasets are pretty low at an average of 0.133.

```
EN
#Entity in gold data: 802
#Entity in prediction: 1016

#Correct Entity : 544
Entity  precision: 0.5354
Entity  recall: 0.6783
Entity  F: 0.5985

#Correct Entity Type : 427
Entity Type  precision: 0.4203
Entity Type  recall: 0.5324
Entity Type  F: 0.4697
```

```
CN
#Entity in gold data: 1081
#Entity in prediction: 5161

#Correct Entity : 602
Entity  precision: 0.1166
Entity  recall: 0.5569
Entity  F: 0.1929

#Correct Entity Type : 354
Entity Type  precision: 0.0686
Entity Type  recall: 0.3275
Entity Type  F: 0.1134
```

```
FR
#Entity in gold data: 238
#Entity in prediction: 1114

#Correct Entity : 186
Entity  precision: 0.1670
Entity  recall: 0.7815
Entity  F: 0.2751

#Correct Entity Type : 79
Entity Type  precision: 0.0709
Entity Type  recall: 0.3319
Entity Type  F: 0.1169
```

```
SG
#Entity in gold data: 4092
#Entity in prediction: 11588

#Correct Entity : 2291
Entity  precision: 0.1977
Entity  recall: 0.5599
Entity  F: 0.2922

#Correct Entity Type : 1318
Entity Type  precision: 0.1137
Entity Type  recall: 0.3221
Entity Type  F: 0.1681
```

Figure 1: Part 2 Results

2 Part 3: First-Order Hidden Markov Model

2.1 Approach

As observed in the previous section, utilising only emission parameters leads to low accuracy in the test set as we are assuming that every tag is independent of the tags before it. This is not a valid assumption in languages. We can improve the performance by including the transition parameters of the HMM, similarly obtained by using MLE:

$$a_{i,j} = \frac{count(i,j)}{count(i)}$$

where $count(i,j)$ is the number of times tag j is observed to follow after tag i. The joint probability of a sequence of length n under the first-order HMM assumptions can be defined as follows:

$$P(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{n+1} a_{y_{i-1}, y_i} \prod_{i=1}^{n} b_{y_i}(x_i)$$

To get the predictions, brute force enumeration of every transition is computationally expensive with a time complexity of $O(|T|^n)$, where T is the set of possible tags. Instead, we can use the Viterbi Algorithm, a dynamic programming algorithm with time complexity of $O(n|T|^2)$.

Implementing the traditional Viterbi algorithm results in the numerical underflow problem as we have fairly long sentences in our testing data. The product of many probabilities tend to be very small. To resolve that, we use log probabilities instead. The joint probability of a sequence then becomes:

$$\log P(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n+1} \log a_{y_{i-1}, y_i} + \sum_{i=1}^{n} \log b_{y_i}(x_i)$$

We define $\pi(k, v)$ as the maximum log probability of generating a sequence of length k with $y_k = v$. We implement the Viterbi algorithm as follows:

1. Base case:
$$\pi(0, START) = log(1) = 0$$

2. Recursive case for layers $i = 1, 2, \ldots, n$:
$$\pi(i, v) = \max_{u \in T} \{\pi(i-1, u) + log(a_{u,v}) + log(b_v(x_i))\} \; \forall \, v \in T$$

3. End case:
$$\pi(n+1, STOP) = \max_{v \in T} \{\pi(n, v) + log(a_{v, STOP})\}$$

4. Backtrack to get optimal tag sequence:
   At every recursive step, we also store
$$u_i^* = \arg\max_{u \in T} \{\pi(i-1, u) + log(a_{u,v}) + log(b_v(x_i))\}$$
   where u_i^* is the parent of v that gives the highest log probability of getting a sequence of length i for the given sentence such that the i^th tag is v. To get the most likely sequence, we traverse parents starting from the STOP node all the way to the START node. The reverse of this traversal will give the most likely tag sequence.

2.2 Result

As compared to part 2 (Figure 1) which only used emission probabilities, precision increased while recall decreased for all datasets. However, the increase in precision is large enough to compensate for the decrease in recall such that we get higher F scores overall.

```
EN                                      CN
#Entity in gold data: 802               #Entity in gold data: 1081
#Entity in prediction: 844              #Entity in prediction: 1406

#Correct Entity : 542                   #Correct Entity : 449
Entity  precision: 0.6422               Entity  precision: 0.3193
Entity  recall: 0.6758                  Entity  recall: 0.4154
Entity  F: 0.6586                       Entity  F: 0.3611

#Correct Entity Type : 483              #Correct Entity Type : 305
Entity Type  precision: 0.5723          Entity Type  precision: 0.2169
Entity Type  recall: 0.6022             Entity Type  recall: 0.2821
Entity Type  F: 0.5869                  Entity Type  F: 0.2453
________________________________________ ________________________________________


FR                                      SG
#Entity in gold data: 238               #Entity in gold data: 4092
#Entity in prediction: 425              #Entity in prediction: 3949

#Correct Entity : 133                   #Correct Entity : 1680
Entity  precision: 0.3129               Entity  precision: 0.4254
Entity  recall: 0.5588                  Entity  recall: 0.4106
Entity  F: 0.4012                       Entity  F: 0.4179

#Correct Entity Type : 82               #Correct Entity Type : 1008
Entity Type  precision: 0.1929          Entity Type  precision: 0.2553
Entity Type  recall: 0.3445             Entity Type  recall: 0.2463
Entity Type  F: 0.2474                  Entity Type  F: 0.2507
________________________________________ ________________________________________
```

Figure 2: Part 3 Results

3 Part 4: Second-Order Hidden Markov Model

3.1 Approach

3.1.1 Model

The joint probability of a sequence under the second-order HMM assumptions can be defined as follows[1]:

$$P(x_1, \ldots, x_n, y_1, \ldots, y_n) = P(y_1|START, START)P(x_1|y_1)P(y_2|y_1, START)P(x_2|y_2)$$
$$\times \prod_{i=3}^{K} P(y_i|y_{i-1}, y_{i-2})P(x_i|y_i)$$

where x_i, y_i refer to the i^{th} word and tag in a sentence respectively. The joint probability can be derived from the following Bayesian Network in Figure 3.

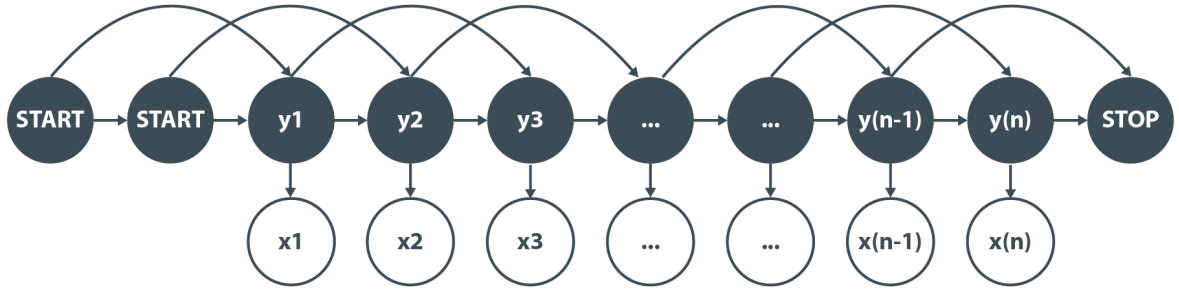

Figure 3: Second-Order Hidden Markov Model

We define the following terms for our algorithm[1]:

- Sequence length is denoted as K
- Adapted from $\pi(k, v)$ used in our first-order HMM, $d_k(m, n)$ is the maximum log joint probability $P(X_k, Y_k)$ of sequence length k given the current node n and its parent node m
- Adapted from the previous transition probability equation, $a_{l,m,n}$ is the conditional probability of a child tag n being generated given a prior (grandparent, parent) tag sequence (l,m)

$$a_{l,m,n} = \frac{count(l, m, n)}{count(l, m)}$$

- The emission probability $b_i(o)$ remains unchanged whereby it is the conditional probability of getting word o given the tag i

$$b_i(o) = \begin{cases} \frac{count(i \to o)}{count(i)+1}, & \text{if word o appears in training set} \\ \frac{1}{count(i)+1}, & \text{otherwise} \end{cases}$$

Hence, the joint probability equation becomes:

$$P(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{K+1} a_{y_{i-2}, y_{i-1}, y_i} \prod_{i=1}^{K} b_{y_i}(x_i)$$

where y_0 and y_{-1} as START tags. This allows us to come up with a Viterbi algorithm that is very similar to the one used in the first-order HMM.

Similar to the first-order HMM, we will use log probabilities to prevent the numerical underflow problem when implementing the algorithm in python. The joint probability equation then becomes:

$$\log P(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{K+1} \log(a_{y_{i-2}, y_{i-1}, y_i}) + \sum_{i=1}^{K} \log(b_{y_i}(x_i))$$

3.1.2 Algorithm

Taking into account all of the above, the adapted Viterbi algorithm is as follows[1]:

1. Base case:

$$\begin{aligned} d_0(none, \text{START}, \text{START}) &= \log(1) = 0 \\ d_1(\text{START}, \text{START}, l) &= \log(a_{\text{START},\text{START},l}) + \log(b_l(x_1)) \; \forall\, l \in T \\ d_2(\text{START}, l, m) &= d_1(\text{START}, \text{START}, l) + \\ &\quad \log(a_{\text{START},l,m}) + \log(b_m(x_2)) \qquad \forall\, l, m \in T \end{aligned}$$

2. Recursive case for layers i = 3 to K:

$$d_i(m, n) = \max_{l \in T} \{d_{i-1}(l, m) + log(a_{l,m,n}) + log(b_n(x_i))\} \; \forall m, n \in T$$

3. Determine last two tags (m,n) of the sequence:

$$\begin{aligned} d_{K+1}(n, STOP) &= \max_{m \in T} \{d_K(m, n) + log(a_{m,n,STOP})\} \; \forall n \in T \\ m &= arg_m \max_{m,n} \{d_{K+1}(n, STOP)\} \\ n &= arg_n \max_{m,n} \{d_{K+1}(n, STOP)\} \end{aligned}$$

4. Backtrack to get optimal tag sequence:
   At every recursive step, we also store

$$l^*_{m,n,i} = \arg\max_{l \in T} \{d_{i-1}(l, m) + log(a_{l,m,n}) + log(b_n(x_i))\} \; \forall m, n \in T$$

   where $l^*_{m,n,i}$ is the grandparent of (m, n) pair that gives the highest log probability for the sequence up to the i^{th} word such that the i^{th} tag is n and the $(i-1)^{th}$ tag is m. To get the most likely sequence, we traverse grandparents starting from the (m, n) pair we get from step (3) all the way to the START node. The reverse of this traversal will give the most likely tag sequence.

The time complexity of the Second Order Viterbi Algorithm is $O(n|T|^3)$ as the sequence is n words long and at each word, we have to try $|T|^3$ number of permutations of l, m and n.

3.2 Results

As seen in Figure 4, entity type precision for EN and CN datasets decreased from first-order HMM to second-order HMM while the precision for FR and SG increased. Entity type recall decreased for EN, maintained for FR and increased for CN and SG. Overall, F score decreased for EN and CN while it increased for SG and FR.

```
EN                                      CN
#Entity in gold data: 802               #Entity in gold data: 1081
#Entity in prediction: 839              #Entity in prediction: 1707

#Correct Entity : 541                   #Correct Entity : 501
Entity  precision: 0.6448               Entity  precision: 0.2935
Entity  recall: 0.6746                  Entity  recall: 0.4635
Entity  F: 0.6594                       Entity  F: 0.3594

#Correct Entity Type : 475              #Correct Entity Type : 340
Entity Type  precision: 0.5662          Entity Type  precision: 0.1992
Entity Type  recall: 0.5923             Entity Type  recall: 0.3145
Entity Type  F: 0.5789                  Entity Type  F: 0.2439
----------------------------------------  ----------------------------------------


FR                                      SG
#Entity in gold data: 238               #Entity in gold data: 4092
#Entity in prediction: 272             #Entity in prediction: 4483

#Correct Entity : 132                   #Correct Entity : 1967
Entity  precision: 0.4853               Entity  precision: 0.4388
Entity  recall: 0.5546                  Entity  recall: 0.4807
Entity  F: 0.5176                       Entity  F: 0.4588

#Correct Entity Type : 82               #Correct Entity Type : 1219
Entity Type  precision: 0.3015          Entity Type  precision: 0.2719
Entity Type  recall: 0.3445             Entity Type  recall: 0.2979
Entity Type  F: 0.3216                  Entity Type  F: 0.2843
----------------------------------------  ----------------------------------------
```

Figure 4: Part 4 Results

4 Part 5: Design Challenge

4.1 Approach

We developed a learning algorithm based on the Perceptron with a logistic regression activation function and a decaying learning rate. The idea is simple. We have achieved decent F scores for the EN dataset using the second-order HMM but not with the FR dataset. Upon studying the FR dataset, we noticed that most words in the FR dataset are labelled with the 'O' tag and most words labelled otherwise are words that can be found in English (eg. cuisine, ambiance, pizza, etc). Hence, we concluded that the FR dataset is most likely labelled by someone who does not speak native French and who is selectively reading only the English looking words from the sentences. Therefore, it is reasonable to model the FR dataset independent of the order of the sequence (unlike in the HMM).

4.1.1 Model

We have thus decided to:

1. Predict each tag v based on only (1) the previous tag u and (2) the current word ignoring the probability of getting the entire sequence
2. Use logistic regression, a discriminative model, to learn $P(y|x)$ directly. This contrasts with HMM, a generative model which learns $P(y|x)$ but is used to predict $P(x, y)$
3. Make a first-order assumption
4. Learn emission and transition scores using a Perceptron algorithm as it is fast and simple to implement
5. Use a learning rate that decreases with each epoch so that we can prevent overfitting during the learning process

The aforementioned design considerations 1, 2 and 3 will help us model the conditional probability for each tag as follows:

$$P(y_i|x_i, y_{i-1}) = \frac{1}{Z(x_i, y_{i-1})} exp\{A_{y_{i-1},y_i} + B_{y_i}(x_i)\}$$
$$Z(x_i, y_{i-1}) = \sum_{y' \in T} exp\{A_{y_{i-1},y'} + B_{y'}(x_i)\}$$

where x_i, y_i refer to the i^{th} word and tag in a sentence respectively, $A_{u,v}$ is the transition score from u to v and $B_v(o)$ is the emission score of o being emitted from tag v.

To tag each word in a sentence, we simply do:

$$\begin{aligned} y_i^* = \arg\max_{y_i \in T} P(y_i|x_i, y_{i-1}) &= \arg\max_{y_i \in T}(\frac{1}{Z(x_i, y_{i-1})} exp\{A_{y_{i-1},y_i} + B_{y_i}(x_i)\}) \\ &= \arg\max_{y_i \in T}(exp\{A_{y_{i-1},y_i} + B_{y_i}(x_i)\}) \\ &= \arg\max_{y_i \in T}(A_{y_{i-1},y_i} + B_{y_i}(x_i)) \end{aligned}$$

4.1.2 Learning

We will be using a Perceptron Algorithm to learn the transition and emission scores, $A_{u,v}$ and $B_v(o)$. The algorithm is as follows:

1. From training set S^n with set of possible tags T and set of possible words W, initialise the transition score $A_{u,v}$ and emission score $B_v(o)$ as such:

$$A_{u,v} = 0 \quad \forall u, v \in T$$
$$B_v(o) = 0 \quad \forall v \in T, \forall o \in W \cup \{\#UNK\#\}$$

2. Repeat for k number of epochs:

- Learning rate $\eta_k = \frac{1}{k+1}$
- Do for each input sequence $X^{(j)}$ in $X = \{X^{(1)}, X^{(2)}, \ldots, X^{(n)}\}$, and for each word $X_i^{(j)}$ in $X^{(j)} = \{X_1^{(j)}, X_2^{(j)}, \ldots, X_m^{(j)}\}$:
  (a) Make a prediction based on the previous tag u and the emitted response $X_i^{(j)}$:

$$\text{predicted\_tag} = \arg\max_{v \in T} \{A_{u,v} + B_v(X_i^{(j)})\}$$

  (b) If the prediction is incorrect, we update the weights like such:

$$A_{u,\text{actual\_tag}} \leftarrow A_{u,\text{actual\_tag}} + \eta_k$$
$$B_{\text{actual\_tag}}(X_i^{(j)}) \leftarrow B_{\text{actual\_tag}}(X_i^{(j)}) + \eta_k$$
$$A_{u,\text{predicted\_tag}} \leftarrow A_{u,\text{predicted\_tag}} - \eta_k$$
$$B_{\text{predicted\_tag}}(X_i^{(j)}) \leftarrow B_{\text{predicted\_tag}}(X_i^{(j)}) - \eta_k$$

4.1.3 Optimisation

The number of epochs k is chosen by using dev.out as the validation dataset. We predict for dev using different number of epochs and plot the curve of F-scores against number of epochs and choose the epoch value that gives us a good F-score.

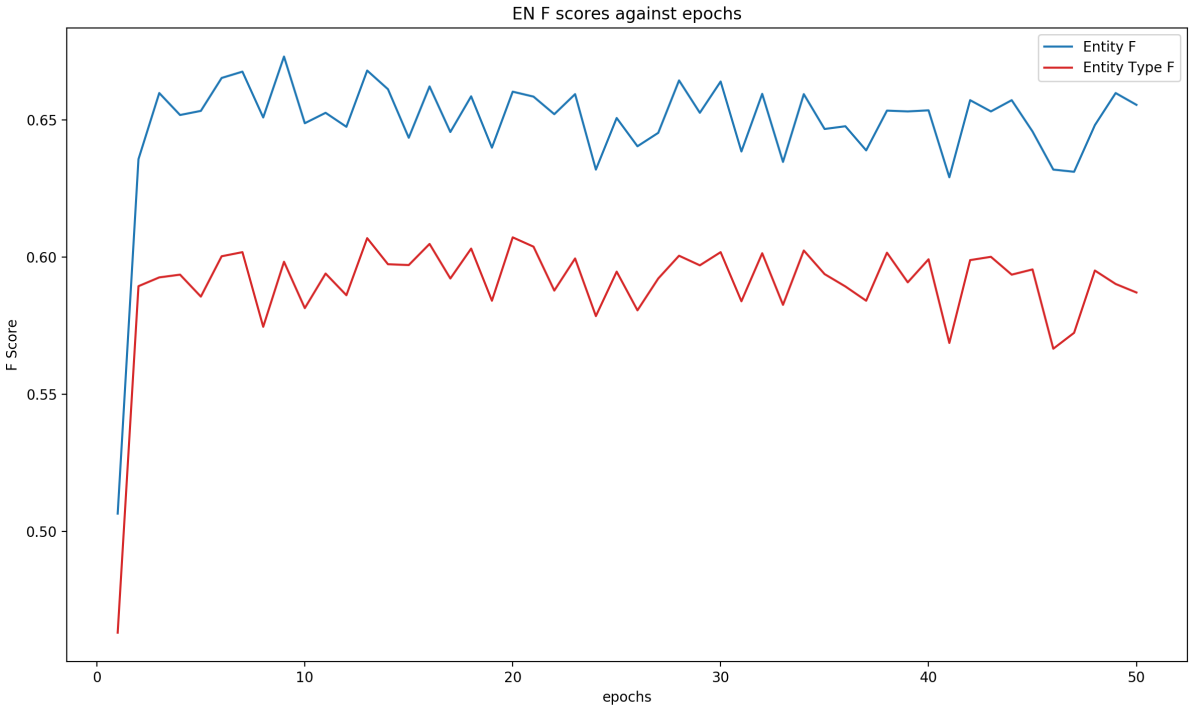

Figure 5: F scores of EN dev predictions against number of epochs

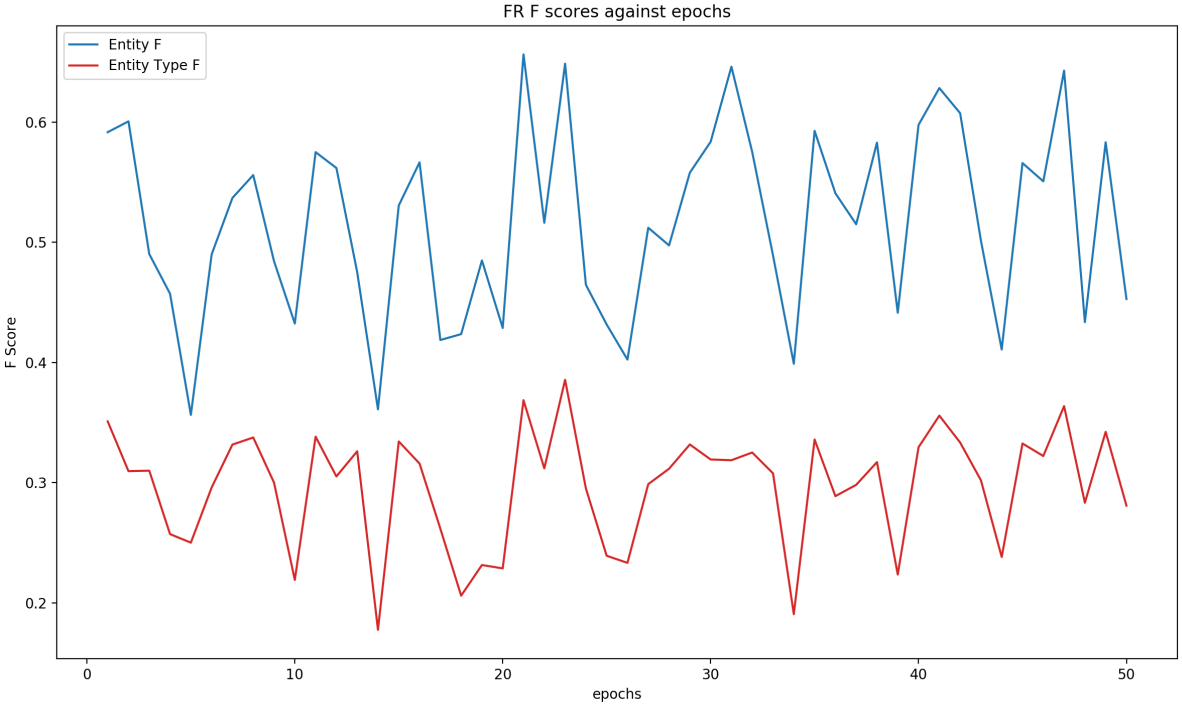

Figure 6: F scores of FR dev predictions against number of epochs

From the plots above, it seems that the algorithm does converge pretty well for EN dataset. We decided to maximize Entity Type F scores for both EN and FR. Hence we chose epochs 20 and 23 for EN and FR datasets respectively.

4.2 Results

Figure 7 below shows the precision, recall and F scores from using our custom algorithm for all the datasets. Table 1 compares the Entity Type F scores generated by all models used in this Design Project. We see that the custom algorithm outperforms all previous models, for all datasets. For the FR dataset, the Entity Type F score improved significantly, agreeing with our initial hypothesis. The improvement over all other datasets may be due to the fact that we are using tweets. Twitter users tend to write in inconsistent and improper grammar. As a result, a model that takes into account entire sequences does not perform as well as a model that does not. We hereby conclude that we are awesome. You're welcome.

```
EN
#Entity in gold data: 802
#Entity in prediction: 667

#Correct Entity : 485
Entity  precision: 0.7271
Entity  recall: 0.6047
Entity  F: 0.6603

#Correct Entity Type : 446
Entity Type  precision: 0.6687
Entity Type  recall: 0.5561
Entity Type  F: 0.6072
```

```
CN
#Entity in gold data: 1081
#Entity in prediction: 1026

#Correct Entity : 396
Entity  precision: 0.3860
Entity  recall: 0.3663
Entity  F: 0.3759

#Correct Entity Type : 275
Entity Type  precision: 0.2680
Entity Type  recall: 0.2544
Entity Type  F: 0.2610
```

```
FR
#Entity in gold data: 238
#Entity in prediction: 203

#Correct Entity : 143
Entity  precision: 0.7044
Entity  recall: 0.6008
Entity  F: 0.6485

#Correct Entity Type : 85
Entity Type  precision: 0.4187
Entity Type  recall: 0.3571
Entity Type  F: 0.3855
```

```
SG
#Entity in gold data: 4092
#Entity in prediction: 3740

#Correct Entity : 1991
Entity  precision: 0.5324
Entity  recall: 0.4866
Entity  F: 0.5084

#Correct Entity Type : 1265
Entity Type  precision: 0.3382
Entity Type  recall: 0.3091
Entity Type  F: 0.3230
```

Figure 7: Part 5 Results

	Part 2	Part 3	Part 4	Part 5
EN	0.4902	0.5869	0.5789	0.6072
FR	0.1404	0.2474	0.3216	0.3855
CN	0.1132	0.2453	0.2439	0.2610
SG	0.1561	0.2507	0.2843	0.3230

Table 1: Comparison of Entity Type F Scores

Bibliography

[1] Yang He. Extended viterbi algorithm for second order hidden markov process. pages 718 – 720, 1988.