Programming Assignment 3: Sequence Tagging

Shihan Ran, A53313589
Fall 2019, CSE 256: Statistical NLP

I. Baseline

A. Design rare word classes

TABLE I: Statistical Information About Dataset

	Train	Dev	Test
Total	399995	15229	15595

We replace infrequent words ($Count(x) < Threshold$) in the original training data file with a common symbol _RARE_. We first analyze how the number of rare words will change according to the threshold. The statistical information is in Table I and Table II.

TABLE II: Number of Rare Words V.S. Threshold

Threshold	**3**	**5**	**7**
# of rare words	26315(6.6%)	37513(9.4%)	45615(11.4%)

To design some informative word classes to replace rare/unseen words. We take a look at some rare words first. The distribution and the format of them may give us some insights. Some examples of rare words and their format are in Table III.

TABLE III: Examples of Rare Words

Format	**Rare Words**	**Counts**
Capitalized first letter	*Beta, Takayasu, Erythromycin*	6931
Capitalized last letter	*apoAI, dNTP, hGCSFR*	703
Capitalized all letters	*HI, SGOT, GGTP, TBG*	4109
All punctuation	*(', '), .–, ?], -]*	196
Contain numeric	*B5, FT4I, FT3, 0acteriuria*	5157
All numeric	*1966, 1973, 479, 1977, 255*	1180
Other rare words	*polyarteritis, erythrocytes*	19237

Our intuition is to let the designed word classes can represent the true distribution of rare words as close as possible. Hence, according to the results of Table III, we design the following classes:

- _ALL_PUNCTUATION_: The word consists of punctuations
- _ALL_NUMERIC_: The word consists of numerics
- _CONTAIN_NUMERIC_: The word contains at least one numeric
- _ALL_CAP_: The word consists of capitalized letters
- _FIRST_CAP_: The first letter of the word is capitalized
- _LAST_CAP_: The last letter of the word is capitalized
- _RARE_: Any other word with the frequency lower than the threshold

B. Performances

We evaluate and compare the new baseline models on train and dev sets. We can see from the results Table IV the following points:

- Adding single classes to _RARE_ isn't helpful. But using some combinations of rare words classes will indeed increase the performance compared with only replace infrequent words with a common symbol.
- Classes _ALL_PUNCTUATION_ and _ALL_CAP_ aren't helpful. Adding them to any of the combinations will not affect the performances.
- We will achieve the best performances if and only if using the combinations of _RARE_ + _ALL_NUMERIC_ + _CON_NUMERIC_ + _FIRST_CAP_ + _LAST_CAP_. Adding another class will not help increase the performance while removing any classes will decrease the performance.

II. Trigram HMM

A. Purpose of the Viterbi algorithm

Viterbi is a kind of algorithm that makes uses of a dynamic programming trellis. The idea is to process the observation sequence left to right,

TABLE IV: Performances of baseline model on different rare classes

Models	Precision	Recall	F1-Score
RARE	**0.159**	**0.660**	**0.256**
RARE + _ALL_PUNCTUATION_	0.159	0.660	0.256
RARE + _ALL_NUMERIC_	0.160	0.660	0.258
RARE + _CONTAIN_NUMERIC_	0.159	0.660	0.256
RARE + _ALL_CAP_	0.159	0.660	0.256
RARE + _FIRST_CAP_	0.159	0.660	0.256
RARE + _LAST_CAP_	0.159	0.660	0.256
RARE + _FIRST_CAP_ + _LAST_CAP_	0.159	0.660	0.256
RARE + _ALL_PUN_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.159	0.660	0.256
RARE + _ALL_PUN_ + _ALL_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.159	0.660	0.256
RARE + _ALL_PUN_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.187	0.623	0.288
RARE + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.190	0.623	0.291
RARE + _ALL_PUN_ + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.190	0.623	0.291
RARE + _ALL_NUM_ + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	**0.190**	**0.623**	**0.291**
RARE + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	0.187	0.623	0.288

Algorithm 1: The Viterbi Algorithm with backpointers

Input: a sequence $x_1, \cdots, x_n$, set of possible tags $\mathcal{K}$, parameters $q(s|u,v)$ and $e(x|s)$
Output: the tag sequence $y_1, \cdots, y_n$

Set $\pi(0, *, *) = 1$;
Initialize $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$ and $\mathcal{K}_k = \mathcal{K}$;
for $k = 1$ to n **do**
  for $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$ **do**
    $\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$;
    $bp(k, u, v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$;
Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n, u, v) \times q(\text{STOP}|u, v))$;
for $k = (n-1)$ to 1 **do**
  $y_k = bp(k+2, y_{k+1}, y_{k+2})$;
return $y_1, \cdots, y_n$;

filling out the trellis. Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $s_0, s_1, \cdots, s_{t1}$, given the automaton λ. The value of each cell $v_t(j)$ is computed by **recursively** taking the most probable path that could lead us to this cell. Note that we represent the most probable path by taking the maximum over all possible previous state sequences. Given that we had already computed the probability of being in every state at time $t-1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell.

Compared with Viterbi, Brute-force is prohibitively computationally expensive. A problem with $S = 10$ and sequence length $T = 80$. The number of all possible sequences in this problem is $O(S^T) = 10^{80}$, which is how many atoms we have in the observable universe! Instead, the time complexity of Viterbi is only $O(T|S|^3)$.

Greedy decoder tags a sequence from left to right, making a hard decision on each word in order and we greedily choose the best tag for each word. However, compared with Viterbi, by making a hard decision on each word before moving to the next word, the greedy tagger can not use the evidence from future decisions.

B. Implementation

Given a new sentence $x_1, \ldots, x_n$, and parameters q and e that we have estimated from a training corpus, we can find the highest probability tag sequence for $x_1, \ldots, x_n$ using the Algorithm 1.

TABLE V: Performances of Trigram HMM on different rare classes

Models	Precision	Recall	F1-Score
RARE	**0.542**	**0.315**	**0.398**
RARE + _ALL_PUNCTUATION_	0.543	0.314	0.398
RARE + _ALL_NUMERIC_	0.537	0.313	0.396
RARE + _CONTAIN_NUMERIC_	0.553	0.347	0.427
RARE + _ALL_CAP_	0.526	0.318	0.396
RARE + _FIRST_CAP_	0.518	0.312	0.389
RARE + _LAST_CAP_	0.534	0.321	0.401
RARE + _CONTAIN_NUMERIC_ +_ALL_PUNCTUATION_	**0.555**	**0.347**	**0.427**
RARE + _FIRST_CAP_ + _LAST_CAP_	0.518	0.316	0.393
RARE + _ALL_PUN_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.513	0.316	0.391
RARE + _ALL_PUN_ + _ALL_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.511	0.316	0.391
RARE + _ALL_PUN_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.530	0.340	0.414
RARE + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.521	0.343	0.414
RARE + _ALL_PUN_ + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.521	0.343	0.414
RARE + _ALL_NUM_ + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	0.532	0.346	0.419
RARE + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	0.542	0.341	0.419

Some specific details are explained in the following subsections:

*1) **Base Case**:* $\pi(0, *, *) = 1$.

*2) **Recursive formulation**:* For any $k \in \{1, \ldots, n\}$, for any $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$, we store the we can define the recursive formulation as: $\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$ where $\mathcal{K}_k$ is the set of possible tags at position k.

*3) **Obtain the joint probability of word sequence and tag sequence**:* If a trigram HMM has parameters $q(s|u, v)$ and $e(x|s)$. Given a training corpus from which we can derive counts, the maximum likelihood estimates for the parameters are $q(s|u, v) = \frac{c(u,v,s)}{c(u,v)}$ and $e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$. Then we can define the joint probability of word sequences $x_1, \ldots, x_n$ paired with a tag sequence $y_1, \ldots, y_{n+1}$ where $y_{n+1} = STOP$ as $p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$.

*4) **Use backpointers to generate the final tag sequence**:* For any $k \in \{1, \ldots, n\}$, for any $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$, we store the element maximizing $\pi(k, u, v)$, that is $bp(k, u, v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$. Using this backpointer table, we can generate the final tag sequence by $y_k = bp(k+2, y_{k+1}, y_{k+2})$.

Some of the key challenges and issues are how to deal with the boundary case: $q(y_1|*, *)$, $q(y_2|*, y_1)$, $q(\text{STOP}|y_{n-1}, y_n)$. We solve this problem by adding $[*, *]$ to the word list when counting frequencies and use $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n, u, v) \times q(\text{STOP}|u, v))$ to generate the last backpointer.

C. Performances

We evaluate and compare the new baseline models on train and dev sets. We can see from the results Table V the following points:

- The F-1 score exceeds the baseline model enormously (from 0.291 to 0.427) while the recall score decreases a lot.
- Adding classes to _RARE_ will affect the performances of trigram HMM. Some rare words classes (e.g. _ALL_PUNCTUATION_ and _CONTAIN_NUMERIC_) will help increase the performance compared with only replace infrequent words with a common symbol _RARE_ while others (e.g. _ALL_NUMERIC_, _FIRST_CAP_, _LAST_CAP_ and _ALL_CAP_) may decrease the performances.
- We will achieve the best performances if and only if using the combinations of _RARE_ + _ALL_PUNCTUATION_ + _CONTAIN_NUMERIC_. Adding another class will not help increase the performance while removing any classes will decrease the performance.

III. Extensions

A. Descriptions of extensions

*1) **Add λ smoothing**:* Considering the HMM we have is not a fully connected one, in which

Algorithm 2: The Viterbi Algorithm of 4-grams HMM

Input: a sequence $x_1, \cdots, x_n$, set of possible tags $\mathcal{K}$, parameters $q(s|u,v,z)$ and $e(x|s)$
Output: the tag sequence $y_1, \cdots, y_n$

Set $\pi(0,*,*,*) = 1$;
Initialize $\mathcal{K}_{-2} = \mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$ and $\mathcal{K}_k = \mathcal{K}$;
for $k = 1$ to n **do**
  for $u \in \mathcal{K}_{k-2}, v \in \mathcal{K}_{k-1}, z \in \mathcal{K}_k$ **do**
    $\pi(k,u,v,z) = \max_{w \in \mathcal{K}_{k-3}} (\pi(k-1,w,u,v) \times q(z|w,u,v) \times e(x_k|z))$;
    $bp(k,u,v,z) = \arg\max_{w \in \mathcal{K}_{k-3}} (\pi(k-1,w,u,v) \times q(z|w,u,v) \times e(x_k|z))$;
Set $(y_{n-2}, y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-2}, v \in \mathcal{K}_{n-1}, z \in \mathcal{K}_n} (\pi(n,u,v,z) \times q(\text{STOP}|u,v,z))$;
for $k = (n-1)$ to 1 **do**
  $y_k = bp(k+3, y_{k+1}, y_{k+2}, y_{k+3})$;
return $y_1, \cdots, y_n$;

many of the transitions between states have zero probability. Hence the probability calculation isn't simply doing $P = \frac{n}{N}$, we should smooth the parameters. We choose **add λ smoothing** methods.

After using smoothing, the mamximum likelihood estimates for the parameters can be adapted into:

$$q(s|u,v) = \frac{c(u,v,s) + \lambda}{c(u,v) + \lambda * |S|}$$

$$e(x|s) = \frac{c(s \rightsquigarrow x) + \lambda}{c(s) + \lambda * |S|}$$

*2) **Moving to 4-grams**:* The implementation details are in the Algorithm 2. Compared with Trigram HMM, key changes and issues lie in dealing with the boundary case. Still, we solve this problem by adding $[*,*,*]$ to the word list when counting frequencies and use STOP sign to generate the last backpointer.

B. Performances

*1) **Add λ smoothing**:* By changing λ, I find the performance varies enormously.

We can see from Table VI, the convergence was reached at $\lambda = 10^{-3}$ with a F1-score **0.427203**, and then the decrease of λ doesn't affect the performance. I assume this phenomenon is due to the limitation of **add λ smooth**. The best performances was achieved at $\lambda = 0.3$ & 0.2 with a F1-score **0.427613**, which is an improvement compared with the previous best results from Trigram HMM.

TABLE VI: Performance of different lambda

λ	Precision	Recall	F1-Score
1	0.526738	0.306854	0.387795
0.9	0.535433	0.317757	0.398827
0.8	0.539062	0.322430	0.403509
0.7	0.550251	0.341121	0.421154
0.6	0.555000	0.345794	0.426104
0.5	0.556110	0.347352	0.427613
0.4	0.556110	0.347352	0.427613
0.3	**0.558603**	**0.348910**	**0.429530**
0.2	**0.558603**	**0.348910**	**0.429530**
0.1	0.553350	0.347352	0.426794
10^{-2}	0.553350	0.347352	0.426794
10^{-3}	0.554726	0.347352	0.427203
10^{-4}	0.554726	0.347352	0.427203
10^{-5}	0.554726	0.347352	0.427203
0	0.554726	0.347352	0.427203

*2) **Moving to 4-grams**:* We evaluate and compare the new baseline models on train and dev sets. We can see from the results Table VII the following points:

- Overall speaking, Trigram HMM outperforms four-gram HMM in almost every combination of rare classes. This may due to the reason for overfitting. Again, the recall score decreases a lot.
- Adding classes to _RARE_ will affect the performances of four-gram HMM. Some rare words classes (e.g. _ALL_PUNCTUATION_ and _CONTAIN_NUMERIC_) will help increase the precision compared with only replace infrequent words with a

TABLE VII: Performances of 4-grams HMM on different rare classes

Models	Precision	Recall	F1-Score
RARE	**0.516**	**0.249**	**0.336**
RARE + _ALL_PUNCTUATION_	0.521	0.268	0.354
RARE + _ALL_NUMERIC_	0.515	0.266	0.351
RARE + _CONTAIN_NUMERIC_	**0.536**	**0.285**	**0.372**
RARE + _ALL_CAP_	0.510	0.283	0.364
RARE + _FIRST_CAP_	0.513	0.268	0.352
RARE + _LAST_CAP_	0.501	0.277	0.357
RARE + _CONTAIN_NUMERIC_ +_ALL_PUNCTUATION_	0.534	0.285	0.372
RARE + _FIRST_CAP_ + _LAST_CAP_	0.519	0.255	0.342
RARE + _ALL_PUN_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.254	0.340
RARE + _ALL_PUN_ + _ALL_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.511	0.316	0.391
RARE + _ALL_PUN_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.249	0.336
RARE + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.249	0.336
RARE + _ALL_PUN_ + _ALL_NUM_ + _CON_NUM_ + _ALL_CAP_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.249	0.336
RARE + _ALL_NUM_ + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.249	0.336
RARE + _CON_NUM_ + _FIRST_CAP_ + _LAST_CAP_	0.516	0.249	0.336

common symbol _RARE_ while others (e.g. _ALL_NUMERIC_, _FIRST_CAP_, _LAST_CAP_ and _ALL_CAP_) may decrease the precision. BUT adding any of the classes will increase the recall score and F1-score.

- We will achieve the best performances if and only if using the combinations of _RARE_ + _CONTAIN_NUMERIC_. Adding another class will not help increase the performance while removing any classes will decrease the performance.