01.112 Machine learning, Fall 2019

Design Project

Team member:
Gou Yuanyuan | 1002972
Wang Zijia | 1002885
Wang Cheng | 1002953

Abstract:
In this design project, we would like to design our sequence labelling model for informal texts from social media using the Hidden Markov Model, and we hope this sequence labelling system for informal texts can serve as the very first step towards building a more complex, intelligent language processing system.

Specifically, we will focus on building four Natural Language Processing systems - an *English noun phrase chunking* system, a *Chinese address chunking* system as well as *two sentiment analysis* systems for Tweets/Weibo.

Part2

Objective

In this part, our objective is to estimate the emission parameters from the training set using MLE (maximum likelihood estimation):

$$e(x|y) = \frac{Count(y \rightarrow x)}{Count(y)} \quad (1)$$

Where $Count(y \rightarrow x)$ is the number of input x appears with tag y in the dataset and $Count(y)$ is the number of times that tag y appears in the dataset.

By using these emission parameters, we produce a tag y for each word x in testing set in sequence.

$$y^* = argmax_y e(x|y) \quad (2)$$

Approach

Firstly, we apply **smoothing method** to our training datasets to get a modified training set. During this process, we replace the words that appear less than k times (in our case, k=3) in the training set with #UNK#. We then use this modified training set to train our model.

Secondly, we **estimate the emission parameters** by applying formula (1) to each word x in the training set with all tags iteratively, and store the results in a dictionary with x as the key and a list of two sublists as the value.
For each value in the dictionary, the first sublist consists of all tags, and the second sublist consists of the emission parameters of x corresponding to each tag in the first sublist.
(e.g. emission_dic = {'word1': [[tag_1, ..., tag_k], [parameter_1, ..., parameter_k]], ..., 'wordn': [['tag_1', ..., 'tag_k'][parameter_1, ..., parameter_k]]}).

Thirdly, we use the above parameters to **produce the tag y** for each word x in the testing set (dev.in) by applying formula (2).
For each word of the sequence, if it is one of the keys in the result dictionary, we will return the tag which refers to the largest emission parameter. If not, we will treat that word as #UNK# and repeat the above implementation.

Results

Since the sequence of the tags are produced based on the emission parameters only, the scores will be relatively lower.

The precision, recall and F scores for each dataset are as follows:

Dataset name	AL	EN
Scores info.	#Entity in gold data: 8408 #Entity in prediction: 19484	#Entity in gold data: 13179 #Entity in prediction: 19406

	#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	#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

Dataset name	CN	SG
Scores info.	#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	#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

Part3

Objective

In this part, our objective is to estimate the transition parameters from the training set using MLE (maximum likelihood estimation):

$$q(y_i|y_{i-1}) = \frac{Count(y_{i-1}, y_i)}{Count(y_{i-1})} \quad (3)$$

where $Count(y_{i-1}, y_i)$ is the number of times that tag y_{i-1} transits to tag y_i and $Count(y_{i-1})$ is the number of times that tag y_{i-1} appears in the dataset.

By using the emission parameters from part2 and the transition parameters, we produce a sequence of tags for a sequence of given words in the testing set.

$$y_1^*, ..., y_n^* = argmax_{y_1,...,y_n} p(x_1, ..., x_n, y_1, ..., y_n) \quad (4)$$

Approach

Firstly, we apply **smoothing method** to our training datasets to get a modified training set. This is implemented in the same way as part 2.

Secondly, we **initialize all the emission parameters and transition parameters** to be -1e99 and **store the logarithm of them** in a similar way as part 2.
(e.g. emission_dic = {'word1': [[tag_1, ..., tag_k], [log(parameter_1), ..., log(parameter_k)]], ..., 'wordn': [['tag_1', ..., 'tag_k'][log(parameter_1), ..., log(parameter_k)]]}
transition_dic = {'tag_1': [[tag_1, ...tag_n], [log(parameter_1), ..., log(parameter_k)]], ..., 'tag_n': [['tag_1', ..., 'tag_n'][log(parameter_1), ..., log(parameter_k)]]}).

Since we need to take START and STOP into consideration for estimating the transition parameters, we replace the space between every two sentences with tag STOP. (We can just add a STOP to represent both START and STOP states since the state STOP means the end of the previous sentence and also means the start of the next sentence.)

Thirdly, we use the above parameters to **produce the tag y** for each word x in the testing set (dev.in) by applying the **Viterbi Algorithm**.
During running the Viterbi Algorithm, for preventing underflow, the score is calculated as $\sum_{j=0}^{n}(loga_{y_j,y_{j+1}} + logb_{y_j}(x_j))$ *(i.e. the summation of the logarithm of the parameters)* instead of $\prod_{j=0}^{n} a_{y_j,y_{j+1}}$ x $\prod_{j=0}^{n} b_{y_j}(x_j)$ *(i.e. the multiplication of the parameters).*

Implementation of Viterbi Algorithm:
We create two arrays with the dimension of T*n. One is for storing the highest score from START to each node and the other one is for storing the tag which leads to the score stored in that state.

In the backtrace step, the last layer will sort out the node with the highest dp and trace back based on the values in the path matrix.
The two arrays are **dp** for the highest logarithm probability, **path** for the index of the tag.
(T: number of tags, n: length of sentence)

Results

Since the sequence of the tags are produced based on both emission parameters and transition parameters, the scores are all higher than the ones in part 2.

The precision, recall and F scores for each dataset are as follows:

Dataset name	AL	EN
Scores info.	#Entity in gold data: 8408 #Entity in prediction: 8500 #Correct Entity : 6735 Entity precision: 0.7924 Entity recall: 0.8010 Entity F: 0.7967 #Correct Sentiment : 6079 Sentiment precision: 0.7152 Sentiment recall: 0.7230 Sentiment F: 0.7191	#Entity in gold data: 13179 #Entity in prediction: 12903 #Correct Entity : 10876 Entity precision: 0.8429 Entity recall: 0.8253 Entity F: 0.8340 #Correct Sentiment : 10449 Sentiment precision: 0.8098 Sentiment recall: 0.7929 Sentiment F: 0.8012

Dataset name	CN	SG
Scores info.	#Entity in gold data: 1478 #Entity in prediction: 725 #Correct Entity : 307 Entity precision: 0.4234 Entity recall: 0.2077 Entity F: 0.2787 #Correct Sentiment : 210 Sentiment precision: 0.2897 Sentiment recall: 0.1421 Sentiment F: 0.1906	#Entity in gold data: 4537 #Entity in prediction: 3014 #Correct Entity : 1661 Entity precision: 0.5511 Entity recall: 0.3661 Entity F: 0.4399 #Correct Sentiment : 1035 Sentiment precision: 0.3434 Sentiment recall: 0.2281 Sentiment F: 0.2741

Part4

Objective

In this part, our objective is to use the estimated transition and emission parameters from the previous parts to find the 7-th best output sequences.

By using the emission and transition parameters from the last part, we produce a sequence of tags for the 7-th best output sequence of given words in the testing set.

$$y_1^*, ..., y_n^* = argmax_{y_1,...y_n} p(x_1, ..., x_n, y_1, ..., y_n)$$

Approach

The overall implementation of part4 is similar to part3. In order to find the 7th best output sequences among all possible combinations, our approach is to **find the top 7 sequences with the highest scores first, then pick the 7th**.

Firstly, for convenience, we use some of the techniques as the previous parts like the logarithm smoothing and the parameter format based on Python dictionary.

Secondly, some changes are done on the dynamic programming process in Viterbi algorithm. In each position, instead of taking the maximum result from the last layer, **all the top 7 sequences will be sorted out and stored**. The reason behind this approach is similar to the reason why we use dynamic programming in the original Viterbi algorithm. From START to END, if we can find out the top 7 paths with the highest score at every position, the 7 paths in the last layer can be guaranteed as the top 7.

For the implementation, we create two arrays both with dimension of 7T*n. In each layer except the first and last, the 7 consecutive values at position i*7 to (i+1) represent the top 7 paths and dp respectively for the word state i.

Thirdly, in the first layer, some special setting is implemented to make sure the final results do represent 7 different paths. In the first layer, if we simply assign the same log(transition(STOP,i)) + log(emission(i, word 1) to the 7 values i*7 to (i+1)*7, in the next layer, all the 7 highest scores will be selected under the same state of word. The following layers will repeat the same mistake and the 7 paths selected in the end actually represents the same path. To avoid that, in the first layer, **we only set the first position i*7 as log(transition(STOP,i)) + log(emission(i, word 1) and set the other 6 positions i*7+1 to (i+1)*7 to negative infinity for each word state i**. By doing so, the top 7 paths in the second layers are forced to choose among different word states and the final paths will be absolutely unique.

Lastly, when the dynamic programming process is done, the backtrace steps start from the last layer. The last layer sort out the node with the 7th highest score and trace back based

on the indexes stored in the path matrix. The path will be regenerated similar as part3. **Each value in the path is divided by 7 and rounded down to an integer in order to get the index for the respective state of word**, then the prediction will be done.

Results

The precision, recall and F scores of the 7th best output sequence for AL and EN dataset are as follows:

Dataset name	AL	EN
Scores info.	#Entity in gold data: 8408 #Entity in prediction: 8942 #Correct Entity : 5985 Entity precision: 0.6693 Entity recall: 0.7118 Entity F: 0.6899 #Correct Sentiment : 5013 Sentiment precision: 0.5606 Sentiment recall: 0.5962 Sentiment F: 0.5779	#Entity in gold data: 13179 #Entity in prediction: 13326 #Correct Entity : 10338 Entity precision: 0.7758 Entity recall: 0.7844 Entity F: 0.7801 #Correct Sentiment : 9796 Sentiment precision: 0.7351 Sentiment recall: 0.7433 Sentiment F: 0.7392

Part5

Objective

In this part, instead of using the generative approach, we will be using a discriminative approach -- Perceptron Algorithm to implement the sentiment analysis.

The description of Perceptron Algorithm is as below:

Input:

A training set of tagged sentences $(w^{i}_{[1:n_i]},\ t^{i}_{[1:n_i]})$ for *i = 1 … n*.

A parameter *T* specifying number of iterations over the training set.

A "local representation" Φ which is a function that maps history/tag pairs to d-dimensional feature vectors.

The glocal representation $\Phi_s(w_{[1:n_i]},\ t_{[1:n_i]}) = \Sigma^{n}_{i=1}\Phi_s(h_i, t_i)$, where $h_i < t_{i-1}, t_{i-2},\ w_{[1:n]}, i >$

Algorithm:

Set all parameter vector α =0

For *t = 1 … T, i = i … n*:

Use the viterbi algorithm to find the output of the model on the i'th training sentence with the current parameter settings, i.e.,

$$Z[1 : n_i] = argma_{u_{[1:n_i]} \in T^{n_i}} \Sigma_s \alpha_s \Phi_s(w^{i}_{[1:n_i]},\ u_{[1:n_i]})$$

where T^{n_i} is the set of all tag sequences of length n_i.

If $Z[1 : n_i] \neq t^{i}_{[1:n_i]}$ then update the parameters

$$\alpha_s = \alpha_s + \Phi_s(w^{i}_{[1:n_i]},\ t^{i}_{[1:n_i]}) - \Phi_s(w^{i}_{[1:n_i]},\ z_{[1:n_i]})$$

Output:

Parameter vector α

Approach

Firstly, we convert all the characters in the words in the training set to **lowercase** as the capital or lowercase letters do not affect the sentiment prediction in many cases, followed by **smoothing method** which is implemented in the same way as part 2.

Secondly, we **initialize all the emission parameters to be 0**, and store these into a dictionary with the tag $t^{i} \in \{t^{1} \ldots t^{n}\}$ as the key and a sub-dictionary as the value, where the key of the sub-dictionary is the input word $w^{i} \in \{w^{1} \ldots w^{n}\}$, and its value is the emission parameter about t^{i} and w^{i}.

(e.g. emission_dic = {'tag_1':{'word_1': 0.2, 'word_2':0.1, …,'word_n': 0.2}, …, 'tag_n': {'word_1': 0.2, 'word_2':0.5, …,'word_n': 0.3}})

Then, we initialize all the transition parameters to be 0 and store these in a similar way.

(e.g. transition_dic = {'tag_1':{'tag_1': 0.2, 'tag_2':0.1, ...,'tag_n': 0.2}, ..., 'tag_n': {'tag_1': 0.2, 'tag_2':0.5, ...,'tag_n': 0.3}})

Thirdly, with the initiated parameters, we use viterbi algorithm to predict the tags for each sentence, followed by updating the parameters.
Before running the Viterbi Algorithm, we first initialize Φ for all nodes and store the values in a list which contains no. of words dictionaries. In each dictionary, the key is tag $t^i \in \{t^1 ... t^n\}$ and the value is a list which its second element is also a tag, and the first element is the score with respect to these two tags.
(e.g. pi = [{'tag_1': [0, 'tag_2']}, {'tag_2': [0.2, 'tag_5']}, ..., {'tag_3': [5, "tag_1']}])
While running the Viterbi Algorithm, the value for each dictionary will be replaced by the pair with maximum score, where score = $\Sigma_{i=1}^{n} \alpha_{t_{i-1},t_i} + \Sigma_{i=1}^{n} \alpha_{t_i, w_i}$.
Then predicting the tags using backtracking method based on these scores.

Fourthly, when updating the emission and transition parameters, for any tag $t^i \in \{t^1 ... t^n\}$ in the predicted sequence, if t^i is different from the given t'^i in the training set, the emission parameter of pair (t'^i, w^i) will be increased by 1, and the emission parameter of pair (t^i, w^i) will be decreased by 1. Similarly, the transition parameter of pair (t'^i, t'^{i+1}) and pair (t'^{i-1}, t'^i) will be increased by 1, and the transition parameter of pair (t^i, t^{i+1}) and pair (t^{i-1}, t^i) will be decreased by 1.

Finally, it will re-run the Viterbi Algorithm per sentence with the latest estimated parameters to get the predicted tag sequences, and save the results into the file.

Results

Now we improve the model by using the Perceptron Algorithm. Thus, the accuracy is getting higher.

For T = t (means iterating over the training set t times), the precision, recall and F scores for each dataset are as follows:

Dataset name	AL (t = 8)	EN (t = 8)
Scores info.	#Entity in gold data: 8408 #Entity in prediction: 8288 #Correct Entity : 6853 Entity precision: 0.8269 Entity recall: 0.8151 Entity F: 0.8209 #Correct Sentiment : 6235 Sentiment precision: 0.7523	#Entity in gold data: 13179 #Entity in prediction: 12844 #Correct Entity : 10704 Entity precision: 0.8334 Entity recall: 0.8122 Entity F: 0.8227 #Correct Sentiment : 10317 Sentiment precision: 0.8033

	Sentiment recall: 0.7416 Sentiment F: 0.7469	Sentiment recall: 0.7828 Sentiment F: 0.7929