# Machine Learning Report

By Liu Sidian 1000909, Ku Ping Cheng 1001282, Wang Yiran 1000906

In the design of sequence labelling system, we employed Hidden Markov Model to predict the sentiments of words in a sentence. Given the training data sets labeled with sentiments and positions, we compiled the data into readily-readable format, computed model parameters using Maximum Likelihood Estimates and Viterbi algorithm, and eventually identified the best label for each word in a supervised learning process.

Following parts illustrate the detailed approaches we used in constructing the model.

## *Part Two:*

### 1.Algorithm

First of all, we imported the data set and stored them systematically in lists and dictionary. It is convenient for us to reference in the later part of the project.

- Dictionary that stores all 'count(y)' values from the training file: **count_y_trg(trgfile)**
- List that stores each x appears in testing file: **get_x_test(x)**

Secondly, to calculate emission parameters, we scanned through all XY pairs in the training set and increase the count whenever the same XY pair shows up. Subsequently, we calculated the likelihood of XY in three scenarios:

1. X is in testing set but not training set
2. X is in training set but not testing set
3. X is in both testing and training set

In the dictionary 'emission' that stores all the emission parameters $e(x|y)$

- Keys are all x values attained from both training and testing files
- Value for each key $x_i$is a dictionary that stores all possible states $y$that can generate value $x_i$: $e(x_i|y)$, where keys are $y's$and value for each $y$is $e(x_i|y)$

Lastly, we computed y* for each x to be the argument that output the maximum MLE.

### 2. Result

| EN | ES | CN | SG |
|---|---|---|---|
| #Entity in gold data: 662<br>#Entity in prediction: 2119<br><br>#Correct Entity : 332<br>Entity precision: 0.1567<br>Entity recall: 0.5015<br>Entity F: 0.2388<br><br>#Correct Sentiment : 68<br>Sentiment precision: 0.0321<br>Sentiment recall: 0.1027<br>Sentiment F: 0.0489 | #Entity in gold data: 1326<br>#Entity in prediction: 4663<br><br>#Correct Entity : 735<br>Entity precision: 0.1576<br>Entity recall: 0.5543<br>Entity F: 0.2454<br><br>#Correct Sentiment : 185<br>Sentiment precision: 0.0397<br>Sentiment recall: 0.1395<br>Sentiment F: 0.0618 | #Entity in gold data: 935<br>#Entity in prediction: 2750<br><br>#Correct Entity : 323<br>Entity precision: 0.1175<br>Entity recall: 0.3455<br>Entity F: 0.1753<br><br>#Correct Sentiment : 88<br>Sentiment precision: 0.0320<br>Sentiment recall: 0.0941<br>Sentiment F: 0.0478 | #Entity in gold data: 4779<br>#Entity in prediction: 10953<br><br>#Correct Entity : 2086<br>Entity precision: 0.1905<br>Entity recall: 0.4365<br>Entity F: 0.2652<br><br>#Correct Sentiment : 471<br>Sentiment precision: 0.0430<br>Sentiment recall: 0.0986<br>Sentiment F: 0.0599 |

## *Part Three:*

1. Transition Parameters:

- Create a dictionary Y that stores $count(y_i)$ for each state $y_i$, where $y_i \in$ {“start”, "B-negative", "B-neutral", "B-positive", "O", "I-negative", "I-neutral", "I-positive", “stop”}
    - Once we detect a ‘\n’ in the input file, count(“start”) +=1, count(“stop”) +=1
- Calculate $count(y_u \rightarrow y_v)$ from the training data file:

$$a_{y_{previous}y_{current}} = {count(y_{previous} \rightarrow y_{current})}\Big/{count(y_{previous})}$$

- Create a dictionary ‘transition’ that stores all transition parameters transits from $y_u$ to $y_v$: transition[$y_u$][ $y_v$] = $a_{y_u y_v}$; if there is no instance transits from $y_u$ to $y_v$, then store transition[$y_u$][ $y_v$]= 0. The dictionary looks as following:

*all possible states ($y_1$, $y_2$, $y_3$, $y_4$, $y_5$, $y_6$, $y_7$) =*
*("B-negative", "B-neutral", "B-positive", "O", "I-negative", "I-neutral", "I-positive")*

{“start”: {$y_1 : a_{y_1\text{start}}$, $y_2 : a_{y_2\text{start}}$ … $y_7 : a_{y_7\text{start}}$},
$y_1$: {$y_1 : a_{y_1y_1}$, $y_2 : a_{y_2y_1}$ … $y_7 : a_{y_7y_1}$, "stop": $a_{y_{stop}y_1}$ },
……
$y_v$: {$y_1 : a_{y_1y_v}$, $y_2 : a_{y_2y_v}$ … $y_7 : a_{y_7y_v}$, "stop": $a_{y_{stop}y_v}$ },
……
$y_7$: {$y_1 : a_{y_1y_7}$, $y_2 : a_{y_2y_7}$ … $y_7 : a_{y_7y_v}$, "stop": $a_{y_{stop}y_7}$}}

2. Viterbi Algorithm:

- Full states set: ("start", "B-negative", "B-neutral", "B-positive", "O", "I-negative", "I-neutral", "I-positive", "stop")
- Main states: ($y_1$, $y_2$, $y_3$, $y_4$, $y_5$, $y_6$, $y_7$) = ("B-negative", "B-neutral", "B-positive", "O", "I-negative", "I-neutral", "I-positive")
- Layers (keys): 0("start"), 1, ..., n, n+1("stop"), n is the length of each observed sentence
- For each layer k, we store both parameter $\pi(k, y_j)$ and optimal previous state $y_{u^j}{}^*$
  - $\pi(0, "start") = 1$
    $y_{u^j}{}^* = "NA"$
  - $\pi\big(k, y_j\big) = \max_{y_{u^j}}\{\ \pi\big(k-1, y_{u^j}\big) \cdot a_{y_{u^j} y_j} \cdot b_{y_j}(x_k)\}$, $\forall\ k = 1, 2 \ldots n$
    $y_{u^j}{}^* = \operatorname{argmax}_{y_{u^j}}\{\ \pi\big(k-1, y_{u^j}\big) \cdot a_{y_{u^j} y_j} \cdot b_{y_j}(x_k)\}$
  - $\pi(n+1, "stop") = \max_{y_{u^{stop}}}\{\ \pi(n, y_{u^{stop}}) \cdot a_{y_{u^{stop}} "stop"}\}$
- The Viterbi dictionary for each observed sentence as follows:
  {
  0: {"start": {"pi":1.0, "opt previous": "NA"},
  1: {$y_1$: {"pi": $\pi(1, y_1)$, "opt previous": "start"}, $y_2$: {"pi": $\pi(1, y_2)$, "opt previous" :"start"}, …… ,
  $y_7$: {"pi": $\pi(1, y_7)$, "opt previous state": "start" }},
  ……
  k: {$y_1$: {"pi": $\pi(k, y_1)$, "opt previous": $y_{u^1}$ }, $y_2$: {"pi": $\pi(k, y_2)$, "opt previous": $y_{u^2}$}, …… ,
  $y_7$: {"pi": $\pi(k, y_7)$, "opt previous": $y_{u^7}$}},
  ……
  n+1: {"stop": {"pi": $\pi(n+1, "stop")$, "opt previous": $y_{u^{stop}}$}}
  }
- Back Tracking to attain the optimal path $Y^*$ list for each observed sentence
  For each layer k from n+1 to 2, append the $Y^*$ list with "opt previous" state of k["opt previous" state of k+1]

3. Result

| EN | ES | CN | SG |
|---|---|---|---|
| #Entity in gold data: 662<br>#Entity in prediction: 1040<br><br>#Correct Entity : 231<br>Entity precision: 0.2221<br>Entity recall: 0.3489<br>Entity F: 0.2714 | #Entity in gold data: 1326<br>#Entity in prediction: 2042<br><br>#Correct Entity : 510<br>Entity precision: 0.2498<br>Entity recall: 0.3846<br>Entity F: 0.3029 | #Entity in gold data: 935<br>#Entity in prediction: 1446<br><br>#Correct Entity : 400<br>Entity precision: 0.2766<br>Entity recall: 0.4278<br>Entity F: 0.3360 | #Entity in gold data: 4779<br>#Entity in prediction: 4447<br><br>#Correct Entity : 1335<br>Entity precision: 0.3002<br>Entity recall: 0.2793<br>Entity F: 0.2894 |

| | | | |
|---|---|---|---|
| #Correct Sentiment : 108<br>Sentiment precision: 0.1038<br>Sentiment recall: 0.1631<br>Sentiment F: 0.1269 | #Correct Sentiment : 267<br>Sentiment precision: 0.1308<br>Sentiment recall: 0.2014<br>Sentiment F: 0.1586 | #Correct Sentiment : 251<br>Sentiment precision: 0.1736<br>Sentiment recall: 0.2684<br>Sentiment F: 0.2108 | #Correct Sentiment : 492<br>Sentiment precision: 0.1106<br>Sentiment recall: 0.1030<br>Sentiment F: 0.1067 |

## Part Four

*1.* Generate a Top_K_Viterbi dictionary based on previous Viterbi dictionary in Part Three
For each state $y_v^j$ in the $j^{th}$ layer, use a dictionary to store the information on its top k optimal paths:

**Following $y_u^{j-1}$ 's and $num\_k$ pairs can be different in rows since they representing the $i^{th}$ optimal path of previous state which is on the optimal path leads to current state $y_v^j$*

| Keys in dictionary of state $y_v^j$ in $j^{th}$ layer | Keys in the dictionary the $i^{th}$ optimal path of state $y_v^j$ in $j^{th}$ layer | | |
|---|---|---|---|
| $i^{th}$ optimal paths in top k | Probability of $i^{th}$ optimal paths from "start" to $y_v^j$ | Previous state in (j-1) layer | The rank number of the previous state |
| 1st | $\pi_1(i, y_v^j)$ | $y_u^{j-1}$ | $num\_k$ |
| 2nd | $\pi_2(i, y_v^j)$ | $y_u^{j-1}$ | $num\_k$ |
| … | …. | … | $num\_k$ |
| $k^{th}$ | $\pi_k(i, y_v^j)$ | $y_u^{j-1}$ | $num\_k$ |

2. Method to select top 5 scores for each state in layer calculated from all the top 5 paths from all the states in previous layer:

    1) Initialize dictionaries for all rank of "start" state in 0 layer: {"prob":1.0, "previous": NA, "$num\_k$": -1}
    2) Initialize for all rank of every state in 1, 2…. n, n+1 layer with dictionary {"prob": -1.0, "previous": NA, "$num\_k$": -1}
    3) For dictionary of each state layer j:

        For each $i^{th}$ optimal paths of all states in layer j-1:

        i. Calculate score:

        $$\pi(j, y_v) = \pi(j-1, y_u) \cdot a_{y_u y_v} \cdot b_{y_v}(x_j) \text{ j=2,3 ….n}$$

        $$\pi(n+1, "stop") = \pi(n, y_{u^{stop}}) \cdot a_{y_{u^{stop}}"stop"}$$

        ii. Compare the score with current top k probabilities

        iii. Reorder the top k optimal paths from the these (k+1) tuples

3. Back Tracking to attain the $k^{th}$ optimal labeling for each observed sentence:

- Initialize the path list ['stop'], and get the state in layer n that leads to the $k^{th}$ optimal path in the dictionary of 'stop'
- For layers from n to 2, add the previous state that is the key in the dictionary of the $k^{th}$ optimal dictionary of the current state
- Add ['start']
- Reverse the path

Result

| EN | ES | CN | SG |
|---|---|---|---|
| #Entity in gold data: 662<br>#Entity in prediction: 6571<br><br>#Correct Entity : 248<br>Entity precision: 0.0377<br>Entity recall: 0.3746<br>Entity F: 0.0686<br><br>#Correct Sentiment : 109<br>Sentiment precision: 0.0166<br>Sentiment recall: 0.1647<br>Sentiment F: 0.0301 | #Entity in gold data: 1326<br>#Entity in prediction: 14933<br><br>#Correct Entity : 629<br>Entity precision: 0.0421<br>Entity recall: 0.4744<br>Entity F: 0.0774<br><br>#Correct Sentiment : 293<br>Sentiment precision: 0.0196<br>Sentiment recall: 0.2210<br>Sentiment F: 0.0360 | #Entity in gold data: 935<br>#Entity in prediction: 14615<br><br>#Correct Entity : 317<br>Entity precision: 0.0217<br>Entity recall: 0.3390<br>Entity F: 0.0408<br><br>#Correct Sentiment : 146<br>Sentiment precision: 0.0100<br>Sentiment recall: 0.1561<br>Sentiment F: 0.0188 | #Entity in gold data: 4779<br>#Entity in prediction: 47292<br><br>#Correct Entity : 3299<br>Entity precision: 0.0698<br>Entity recall: 0.6903<br>Entity F: 0.1267<br><br>#Correct Sentiment : 1405<br>Sentiment precision: 0.0297<br>Sentiment recall: 0.2940<br>Sentiment F: 0.0540 |

## *Part Five*

We used Bigram Tagger with Perceptron Algorithm to update transition and emission parameters and identify the optimal sentiments for sentences.

### ***1.Initialization***

We considered sentiment labels in a pair of two and label-word pair as transition and emission parameters, and constructed dictionaries to store them. Index for every parameter pair is subsequently initialized as 0. Performed Viterbi Algorithm to select a temporary optimal path.

| Transition parameter | Emission parameter |
|---|---|
| {"start": {$y_1$ : $s_{starty_1}$ , $y_2$ : $s_{starty_2}$ … $y_7$ : $s_{starty_7}$ },<br>$y_1$ : {$y_1$ : $s_{y_1y_1}$ , $y_2$ : $s_{y_1y_2}$ … $y_7$ : $s_{y_1y_7}$ , "*stop*" : $s_{y_1y_{stop}}$ },<br>……<br>$y_v$ : {$y_1$ : $s_{y_vy_1}$ , $y_2$ : $s_{y_vy_2}$ … $y_7$ : $s_{y_vy_7}$ , "*stop*" : $s_{y_vy_{stop}}$ },<br>……<br>$y_7$ : {$y_1$ : $s_{y_7y_1}$ , $y_2$ : $s_{y_7y_2}$ … $y_7$ : $s_{y_7y_7}$, "*stop*" : $s_{y_7y_{stop}}$ }}<br>* keys of the main dictionary are "previous states", while the keys of the sub-dictionary are the "current states" transits from the "previous states | {$x_1$: {$y_1$: $S_{y_1x_1}$, $y_2$: $S_{y_2x_1}$ …. $y_7$: $S_{y_7x_1}$}<br>…….<br>{$x_v$: {$y_1$: $S_{y_1x_n}$, $y_2$: $S_{y_2x_n}$ …. $y_7$: $S_{y_7x_n}$}<br>…….<br>{$x_n$: {$y_1$: $S_{y_1x_n}$, $y_2$: $S_{y_2x_n}$ …. $y_7$: $S_{y_7x_n}$}<br>*keys of the dictionary are words, while the keys of the sub-dictionary are the associated states. |

## 2.Algorithm

***Perceptron algorithm:***

```
n=0
while n <100
        for file in file set:
                for sentence in file:
                        for transition pair in sentence:
                                if gold entity pair != current entity pair
                                        S(current entity pair) -= 1
                                        S( gold entity pair) + = 1
                        viterbi to find optimal path for the next sentence
        for file in file set:
                for sentence in file:
                        for emission pair in sentence:
                                if gold entity pair != current entity pair
                                        S(current entity pair) -= 1
                                        S( gold entity pair) + = 1
                        viterbi to find optimal path for the next sentence
        n+=1
```

***Viterbi Algorithm:***

```
for layer in path-dic
        if layer==0:
                continue
        if layer=n+1:
                p=max(p_previous+S(yistop)
                previous_state= argmax p
        for current state in path_dic:
```

```
        if x not in dic.keys
                add x to dic keys
        if current state in last layer:
                p=p_previous+S(yi->stop)
        else:
                p=p_previous+S(yi1yi2)+S(yixi)
```

Back Tracking to attain the optimal path Y* list for each observed sentence

For each layer k from n+1 to 2, append the Y* list with "opt previous" state of k["opt previous" state of k+1]

***Result***

Compared with the predict result from part 3 and 4, the result in part 5 has actually been improved as shown in the following chart:

| EN - dev.in | ES - dev.in |
|---|---|
| #Entity in gold **data**: 662<br>#Entity in **prediction**: 217<br><br>#Correct Entity : 137<br>Entity precision: 0.6313<br>Entity recall: 0.2069<br>Entity F: 0.3117<br><br>#Correct Sentiment : 105<br>Sentiment precision: 0.4839<br>Sentiment recall: 0.1586<br>Sentiment F: 0.2389 | #Entity in gold data: 1326<br>#Entity in prediction: 902<br><br>#Correct Entity : 410<br>Entity precision: 0.4545<br>Entity recall: 0.3092<br>Entity F: 0.3680<br><br>#Correct Sentiment : 193<br>Sentiment precision: 0.2140<br>Sentiment recall: 0.1456<br>Sentiment F: 0.1732 |