# 50.007 Machine Learning

## Design Project

### Tan Jun Qi (1000382), Ren Yang (1000959), Hazel Lau (1001150)

## **Part 2**

### Approach

For each of the four datasets, we estimated emission parameters using the counts of each possible emission from each state in the training data, and constructed an emission matrix containing the estimated probabilities.

Before processing the data, we initialize a global array of symbols, which is a list inclusive of all seven possible symbols (e.g. ‘O’, ‘B-positive’ etc)

We then process the data using a number of functions:

#### get\_symbol\_word\_counts(training\_file)

This function takes in a training data file formatted with lines like “*word symbol*”, which is how the train files for each data set looks like. The function returns a tuple of two dictionaries, the first of which is a nested dictionary of word counts where the value of d[symbol][word] is the count of “*word symbol*”. The second dictionary is a dictionary of symbol counts, where d[symbol] is the total count of the symbol.

#### estimate\_emission\_params(symbol\_word\_counts, symbol\_counts)

This function takes in the two dictionaries from function 1 containing the word counts and symbol counts, and returns a nested dictionary of emission probabilities where the value of d[symbol][word] is the emission probability for that particular word and symbol.

For each word, the emission parameters are as follows:

#### get\_emission\_probabilities(training\_file)

This function takes in the training file (the “train” file for each data set) and makes use of functions 1 and 2 to return a dictionary of emission probabilities for each word and symbol.

#### emission\_probability(symbol, word, emission\_probabilities, symbol\_counts)

This function takes in a symbol, a word, and the dictionary of emission probabilities from function 3, and the dictionary of symbol counts from function 1, and returns the emission probability for the particular symbol and word. If the word has not been encountered in the training data, we assign it a fixed probability based on the symbol count, as specified in the project requirements.

If word is an unseen word (that did not appear in the training set):

Otherwise, the emission probability for a given word and symbol will be taken from the emission\_probabilities dictionary.

#### find\_symbol\_estimate(dev\_file, emission\_probabilities, symbol\_counts)

This function takes in the unlabeled file (“dev.in”), the dictionary of emission probabilities from function 3, and the dictionary of symbol counts from function 1. When iterating through each word, this function will use function 4 to estimate the emission probability of that word with each symbol, and choose the symbol for which the emission probability is largest for that word. This is a representation of the following equation:

This function then returns a list of all predicted symbols for the dev\_file.

Using the above functions, we then found the most likely tags for each observation, and calculated the Precision, Recall, and F-score for each data set.

### Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **EN** | **ES** | **SG** | **CN** |
| Entity in Gold Data | 662 | 1326 | 4779 | 935 |
| Entity in Prediction | 3845 | 8490 | 24547 | 8165 |
| **Correct Entity** | **180** | **357** | **1357** | **180** |
| Entity Precision | 0.0468 | 0.0420 | 0.0553 | 0.0220 |
| Entity Recall | 0.2719 | 0.2692 | 0.2840 | 0.1925 |
| Entity F-Score | 0.0799 | 0.0727 | 0.0925 | 0.0396 |
| **Correct Sentiment** | **24** | **73** | **303** | **37** |
| Sentiment Precision | 0.0062 | 0.0086 | 0.0123 | 0.0045 |
| Sentiment Recall | 0.0363 | 0.0551 | 0.0634 | 0.0396 |
| Sentiment F-score | 0.0107 | 0.0149 | 0.0207 | 0.0081 |

## **Part 3**

### Approach

Using maximum likelihood estimation, we then estimated the transition parameters for the four datasets and constructed the transition matrix, inclusive of the START and STOP states.

#### get\_symbol\_symbol\_counts(training\_data)

This function takes a training data file and returns a tuple of two dictionaries, the first of which is a nested dictionary of symbol-to-symbol transition counts, where the value of d[symbol1][symbol2] is the counts of “*word symbol1*” and “*word symbol2*”. The second is a dictionary of symbol counts where the value of d[symbol] is the total count of the symbol.

#### estimate\_transition\_params(symbol\_symbol\_counts, symbol\_counts)

This function takes in the outputs from function 1, and returns a nested dictionary of transition probabilities, where the value of d[symbol1][symbol2] is the probability of transitioning from symbol1 to symbol2.

The transition probabilities for a symbol to another is as follows:

If the count of a symbol is 0, the transition probability is then 0 as well.

Otherwise, the transition probability is:

#### get\_transition\_probabilities(training\_file)

This function makes use of the previous two functions to get the transition probabilities of the training data file.

#### get\_observation\_sequences(dev\_file)

This function serves to split the dev.in file according to their respective tweets. This function returns a nested list, each element containing a list of words in a tweet.

#### viterbi(transition\_probabilities, emission probabilities, symbol\_counts, observation\_sequences)

This function takes in the transition probabilities from function 3, emission probabilities from part 2, symbol counts from function 1, and observation sequences from function 4, runs the Viterbi algorithm, and outputs the predicted symbols for each word in the data set.

Using the above functions, we run the Viterbi algorithm on the development set using the models for each dataset.

### Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **EN** | **ES** | **SG** | **CN** |
| Entity in Gold Data | 662 | 1326 | 4779 | 935 |
| Entity in Prediction | 894 | 1339 | 3566 | 1307 |
| **Correct Entity** | **144** | **384** | **889** | **378** |
| Entity Precision | 0.1611 | 0.2868 | 0.2493 | 0.2892 |
| Entity Recall | 0.2175 | 0.2896 | 0.1860 | 0.4043 |
| Entity F-Score | 0.1851 | 0.2882 | 0.2131 | 0.3372 |
| **Correct Sentiment** | **90** | **176** | **436** | **237** |
| Sentiment Precision | 0.1007 | 0.1314 | 0.1223 | 0.1813 |
| Sentiment Recall | 0.1360 | 0.1327 | 0.0912 | 0.2535 |
| Sentiment F-score | 0.1157 | 0.1321 | 0.1045 | 0.2114 |

## **Part 4**

### Approach

Part 4 makes use of functions from the previous 2 parts to get the transition and emission probabilities, and runs a top m Viterbi algorithm to get the top m paths for each observation sequence.

#### top\_m\_viterbi(m, transition probabilities, emission\_probabilities, symbol\_counts, observation\_sequences)

This function is an edited version of the Viterbi algorithm detailed in part 3, and it returns a nested list, each element being the top m paths for each sequence, and within that a tuple of the score and the predicted symbols.

### Results

|  |  |  |
| --- | --- | --- |
|  | **EN** | **ES** |
| Entity in Gold Data | 662 | 1326 |
| Entity in Prediction | 1407 | 1885 |
| **Correct Entity** | **263** | **471** |
| Entity Precision | 0.1869 | 0.2499 |
| Entity Recall | 0.3973 | 0.3552 |
| Entity F-Score | 0.2542 | 0.2934 |
| **Correct Sentiment** | **137** | **215** |
| Sentiment Precision | 0.0974 | 0.1141 |
| Sentiment Recall | 0.2069 | 0.1621 |
| Sentiment F-score | 0.1324 | 0.1339 |

## **Part 5**

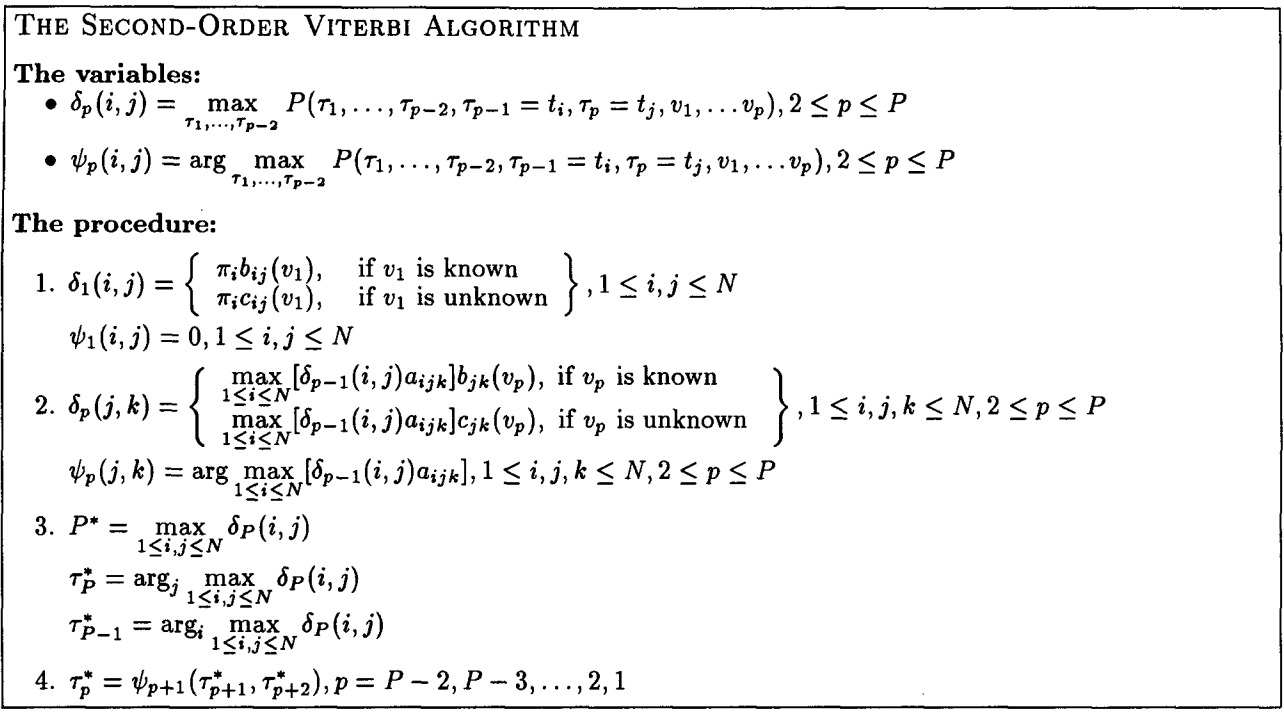
### Approach

We chose to use a second order Markov chain (HMM2), or a trigram model, instead of the usual 1st order Markov chain or bigram model. So instead of the bigram transition probability aij = P(rp= tjIrp-1= ti) with a trigram probability aijk = P(rp= tkIrp-1= tj, rp-2 = ti). Our transition matrix is therefore a 3-Tensor (or a doubly nested dictionary).  
  
Intuitively, we chose to use a second order Markov chain if we take into account both the current state and the previous state in generating the next state, we are allowing for more context-dependence on the word tags (except for the boundary cases). Therefore, we hypothesise that this method would give a better prediction of the underlying states.  
  
Learning

We would still use the counting method to estimate the transition probabilities, the only difference is that this time, our formula will take into account transitions.   
for the boundary cases we will use 2 initial states instead of one.

subsequently, we can get the transition using the formula:

#### Decoding



### Results

|  |  |  |
| --- | --- | --- |
|  | **EN** | **ES** |
| Entity in Gold Data | 662 | 1326 |
| Entity in Prediction |  |  |
| **Correct Entity** |  |  |
| Entity Precision |  |  |
| Entity Recall |  |  |
| Entity F-Score |  |  |
| **Correct Sentiment** |  |  |
| Sentiment Precision |  |  |
| Sentiment Recall |  |  |
| Sentiment F-score |  |  |