Design Project Report

01.112 Machine Learning

4th December 2017

Aishwarya Prabhat 1001848

Cheng You Xuan Zanette 1001845

Ng Yi Jun 1001766

Contents

1 **General Instructions** 2

2 Part 2 **Simple Sentiment Analysis Approach** 2-3

2.1 **Implementation method** 2

2.2 **Score summary** 3

3 Part 3 **Viterbi Algorithm** 4-6

3.1 **Implementation method** 4

3.2 **Score summary** 6

4 Part 4 **Alternative Max Marginal Decoding Algorithm** 7-8

4.1 **Implementation method** 7

4.2 **Score Summary** 8

5 Part 5 **Improved Viterbi Algorithm** 9-10

5.1 **Implementation method** 9

5.2 **Score summary** 10

1 General Instructions

Softwares Required:
iPython/Jupyter Notebook with Python 3.6

In order to run code, follow these steps:

1. Open iPython notebook
2. Read instructions in first cell
3. Change input file and output file path variables as instructed
4. Run each cell corresponding to each part of the project as per the instructions

---

Part 2 Simple Sentiment Analysis

2.1 Implementation method

The function `emission(file_path)` takes in the training file as input and returns a dictionary in the format `{(word,tag) : emission_probability}` after computing the emission probabilities based on the maximum likelihood estimate given by:

$$e(x|y) = \text{Count}(y \rightarrow x) \,/\, \text{Count}(y)$$

The function `modify_training_set(training_data, k)` takes in training data and the k value as inputs and replaces all words with frequency<k with #UNK#

The function `sentiment_anal(train_file_path, test_file_path)` takes the file paths to the training and testing files and calls all the helper functions to perform necessary computations and data formatting. It finally computes the predicted tags for each word in the test file based on:

$$y* = \arg\max_y e(x|y)$$

2.2 **Score Summary**

EN	**FR**
#Entity in gold data: 226 #Entity in prediction: 737 #Correct Entity : 163 Entity precision: 0.2212 Entity recall: 0.7212 Entity F: 0.3385 #Correct Sentiment : 84 Sentiment precision: 0.1140 Sentiment recall: 0.3717 Sentiment F: 0.1745	#Entity in gold data: 223 #Entity in prediction: 712 #Correct Entity : 181 Entity precision: 0.2542 Entity recall: 0.8117 Entity F: 0.3872 #Correct Sentiment : 105 Sentiment precision: 0.1475 Sentiment recall: 0.4709 Sentiment F: 0.2246
CN	**SG**
#Entity in gold data: 362 #Entity in prediction: 1820 #Correct Entity : 165 Entity precision: 0.0907 Entity recall: 0.4558 Entity F: 0.1512 #Correct Sentiment : 49 Sentiment precision: 0.0269 Sentiment recall: 0.1354 Sentiment F: 0.0449	#Entity in gold data: 1382 #Entity in prediction: 961 #Correct Entity : 370 Entity precision: 0.3850 Entity recall: 0.2677 Entity F: 0.3158 #Correct Sentiment : 248 Sentiment precision: 0.2581 Sentiment recall: 0.1795 Sentiment F: 0.2117

Part 3 **Viterbi Algorithm**

3.1 **Implementation Method**

For part 3, we take the approach of using Viterbi Algorithm to tag the words via a generative model.

Generally, a sentence for Viterbi Algorithm is as

$$\text{START We are good students END}$$

$$\text{Where } y_0 = START \text{ and } y_{n+1} = STOP$$

As the raw file does not have $y_0 = START$ and $y_{n+1} = STOP$, the data was preprocessed to include $y_0 = START$ and $y_{n+1} = STOP$.

`all_pi` is an important dictionary. The structure is:
`{1:{'O':...,'B-positive':...,...},2:{...},...}`
where `{node number : {tag:score of best subpath to node if use tag}}`

In the first for loop, following the notes

$$\pi(0,u) = \begin{cases} 1 & \text{if } u = \texttt{START} \text{ (starting state, no observations)} \\ 0 & \text{otherwise} \end{cases}$$

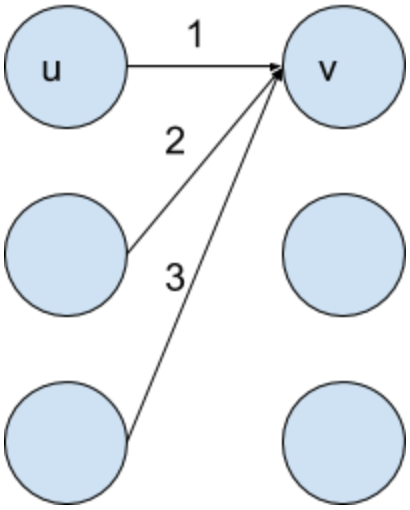
ss checked by the function `check_basecase` which checks the state and returns True or False. $\pi(k-1,u)$ is `pi_prev` in the code.
If true, $\pi(k-1=0,u)=1$ else $\pi(k-1=0,u)=0$

If k-1 is not 0, then do the recursive case

$$\pi(k,v) = \max_{u \in \mathcal{T}} \{\pi(k-1,u) \cdot a_{u,v} \cdot b_v(x_k)\}$$

For when transition or emission probability was not calculated from the training date (ie missing), they will be defaulted to be 0.

In `for v in T` and in `for u in T`:



Finds probability of all sub-paths & get the max via `max(temp_pi.values())`

To calculate the score for STOP, the code was separated as a special case as stated in the notes:

$$\max_{y_1,\ldots,y_n} p(x_1,\ldots,x_n,y_0=\texttt{START},y_1,\ldots,y_n,y_{n+1}=\texttt{STOP}) = \max_{v\in\mathcal{T}}\{\pi(n,v)\cdot a_{v,\texttt{STOP}}\}$$

For back-tracing, stop is separated as shown in the notes. Because the transition states will always be (y_n, y_{n+1}), the states can hardcoded to be $(v, 'STOP')$. Returns y^* for one sentence.

y^*_n for $y^*_{n-1} = \arg\max\{\pi(n-1,u)\cdot a_{u,y^*_n}\}$ is captured from y^* via `y_star[0]` because the tags are being inserted in reverse. Argmax is done by `max(yn_1_star_values, key = yn_1_star_values.get)`

At the end, `y_star` is appended to `massive_y_star`, which contains all tags of all the sentences in the file.

Function `process_results` returns `(word,tag)` , which is the format needed for output for dev.p3.out .

Dynamic programming is done here through getting the previous pi value `pi_prev = all_pi[k - 1][u]` from `all_pi`, saving computation time from regenerating the previous pi value.

3.2 **Score Summary**

EN	**FR**
#Entity in gold data: 226 #Entity in prediction: 162 #Correct Entity : 104 Entity precision: 0.6420 Entity recall: 0.4602 Entity F: 0.5361 #Correct Sentiment : 64 Sentiment precision: 0.3951 Sentiment recall: 0.2832 Sentiment F: 0.3299	#Entity in gold data: 223 #Entity in prediction: 166 #Correct Entity : 112 Entity precision: 0.6747 Entity recall: 0.5022 Entity F: 0.5758 #Correct Sentiment : 72 Sentiment precision: 0.4337 Sentiment recall: 0.3229 Sentiment F: 0.3702
CN	**SG**
#Entity in gold data: 362 #Entity in prediction: 158 #Correct Entity : 64 Entity precision: 0.4051 Entity recall: 0.1768 Entity F: 0.2462 #Correct Sentiment : 47 Sentiment precision: 0.2975 Sentiment recall: 0.1298 Sentiment F: 0.1808	#Entity in gold data: 1382 #Entity in prediction: 723 #Correct Entity : 22 Entity precision: 0.0304 Entity recall: 0.0159 Entity F: 0.0209 #Correct Sentiment : 11 Sentiment precision: 0.0152 Sentiment recall: 0.0080 Sentiment F: 0.0105

Part 4 **Max Marginal Decoding Algorithm**

4.1 **Implementation Method**

Forward & backwards dictionary are in the same format as `all_pi` for viterbi.
`forward[1] = alpha_base #this should be constant` is constant because

$$\alpha_u(1) \quad = \quad a_{\texttt{START},u} \quad \forall u \in 1, \ldots, N-1$$

And u is fixed in T thus will always form the $\alpha_v(j)$ for when $j = 1$ in the recursive case `range(len(sentence)-2,0,-1)`
The sentence being read in is: START We are good students END
Format: `len(sentence-2)` refers to students here, the equivalent of `j = n-1`

Decoding

$$y_i^* = \arg\max_u p(y_i = u | x_1, \ldots, x_n; \theta) = \arg\max_u \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)} = \arg\max_u \alpha_u(i)\beta_u(i)$$

Is done in

```
temp[u] = forward[j][u] * backwards[j][u]
YSTAR.append(max(temp,key=temp.get))
```

Beta is not fixed because the last word X_n constantly changes for each sentence thus needs to be constantly regenerated.

4.2 **Score Summary**

EN	**FR**
#Entity in gold data: 226 #Entity in prediction: 175 #Correct Entity : 107 Entity precision: 0.6114 Entity recall: 0.4735 Entity F: 0.5337 #Correct Sentiment : 69 Sentiment precision: 0.3943 Sentiment recall: 0.3053 Sentiment F: 0.3441	#Entity in gold data: 223 #Entity in prediction: 173 #Correct Entity : 113 Entity precision: 0.6532 Entity recall: 0.5067 Entity F: 0.5707 #Correct Sentiment : 73 Sentiment precision: 0.4220 Sentiment recall: 0.3274 Sentiment F: 0.3687
CN	**SG**
#Entity in gold data: 362 #Entity in prediction: 191 #Correct Entity : 68 Entity precision: 0.3560 Entity recall: 0.1878 Entity F: 0.2459 #Correct Sentiment : 48 Sentiment precision: 0.2513 Sentiment recall: 0.1326 Sentiment F: 0.1736	#Entity in gold data: 1382 #Entity in prediction: 778 #Correct Entity : 23 Entity precision: 0.0296 Entity recall: 0.0166 Entity F: 0.0213 #Correct Sentiment : 11 Sentiment precision: 0.0141 Sentiment recall: 0.0080 Sentiment F: 0.0102

Part 5 **Improved Viterbi Algorithm**

5.1 **Implementation Method**

For part 5, the model is improved by making the following changes based on empirically observed changes in f-scores of all 4 datasets:

a. Improving pre-processing of input data by using a more stringent policy on replacement of low frequency words
b. Using a 'safe' policy for unseen words in test data by assigning O-tag

Improving pre-processing of input data by using a more stringent policy on replacement of low frequency words

As highlighted in part 2, one problem with estimating the emission parameters is that some words that appear in the test set do not appear in the training set. To handle this issue, we replace those words that appear less than k times in the training set with a special token #UNK# before training. For part 2, the k value was set to 3. However, empirical experimentation with the 4 given datasets revealed that as K value was increased from 3 to 7, there was a general consistent improvement in the f-scores of all 4 datasets. Values beyond 7 not only resulted in poorer f-scores but also did not appeal semantically because replacing words appearing more than 7 times seems too heavy a penalty.

Using a 'safe' policy for unseen words in test data by assigning O-tag

It was observed in the data that, generally, words that were replaced with #UNK# tended to produce the most incorrect tags as their corresponding outputs. Given the 4 datasets, it was also prominent that words replaced with #UNK# had a tendency to appear 'outside' major entities in the sentence. Hence, we experimented with a policy whereby whenever an #UNK# appeared in the parsed test data, the tag was asserted to 'O'. Empirically, this resulted in increment in f-scores across all datasets.

5.3 **Score Summary**

EN	**FR**
#Entity in gold data: 226 #Entity in prediction: 169 #Correct Entity : 102 Entity precision: 0.6036 Entity recall: 0.4513 Entity F: 0.5165 #Correct Sentiment : 67 Sentiment precision: 0.3964 Sentiment recall: 0.2965 Sentiment F: 0.3392	#Entity in gold data: 223 #Entity in prediction: 178 #Correct Entity : 127 Entity precision: 0.7135 Entity recall: 0.5695 Entity F: 0.6334 #Correct Sentiment : 80 Sentiment precision: 0.4494 Sentiment recall: 0.3587 Sentiment F: 0.3990