Part 2

EvalScript Results

EN dataset:

#Entity in gold data: 226

#Entity in prediction: 981

#Correct Entity : 149

Entity precision: 0.1519

Entity recall: 0.6593

Entity F: 0.2469

#Correct Sentiment : 58

Sentiment precision: 0.0591

Sentiment recall: 0.2566

Sentiment F: 0.0961

CN dataset:

#Entity in gold data: 362

#Entity in prediction: 2608

#Correct Entity : 146

Entity precision: 0.0560

Entity recall: 0.4033

Entity F: 0.0983

#Correct Sentiment : 56

Sentiment precision: 0.0215

Sentiment recall: 0.1547

Sentiment F: 0.0377

FR dataset:

#Entity in gold data: 223

#Entity in prediction: 964

#Correct Entity : 176

Entity precision: 0.1826

Entity recall: 0.7892

Entity F: 0.2965

#Correct Sentiment : 65

Sentiment precision: 0.0674

Sentiment recall: 0.2915

Sentiment F: 0.1095

SG dataset:

#Entity in gold data: 1382

#Entity in prediction: 5112

#Correct Entity : 644

Entity precision: 0.1260

Entity recall: 0.4660

Entity F: 0.1983

#Correct Sentiment : 283

Sentiment precision: 0.0554

Sentiment recall: 0.2048

Sentiment F: 0.0872

Part 3

EvalScript Results

EN Dataset:

#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

CN Dataset:

#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

FR Dataset:

#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

SG Dataset:

#Entity in gold data: 1382

#Entity in prediction: 723

#Correct Entity : 386

Entity precision: 0.5339

Entity recall: 0.2793

Entity F: 0.3667

#Correct Sentiment : 244

Sentiment precision: 0.3375

Sentiment recall: 0.1766

Sentiment F: 0.2318

Part 4

EvalScript Results

EN Dataset:

#Entity in gold data: 226

#Entity in prediction: 159

#Correct Entity : 83

Entity precision: 0.5220

Entity recall: 0.3673

Entity F: 0.4312

#Correct Sentiment : 53

Sentiment precision: 0.3333

Sentiment recall: 0.2345

Sentiment F: 0.2753

FR Dataset:

#Entity in gold data: 223

#Entity in prediction: 80

#Correct Entity : 41

Entity precision: 0.5125

Entity recall: 0.1839

Entity F: 0.2706

#Correct Sentiment : 25

Sentiment precision: 0.3125

Sentiment recall: 0.1121

Sentiment F: 0.1650

Our Algorithm

The execution of our max-marginal algorithm is done mainly in the maximum\_marginal\_analysis() function. We first read our training file as a list of sentences with its sequence of labels, and ran the count() function over it to count the number of occurrences of each label, word, emissions and transitions. We then use these counts to get our emission and transition parameters using the get\_parameters() function. These functions can be found in our external module.py file.

With our emission and transition parameters, we can estimate the most probable “current transition” by observing the current word and the previous label. When words appear less than a total of 3 times in our training set, we will modify the word into #UNK#, then calculate the emission and transition parameters accordingly.

Our training file is also read in as a list of sentences. Iterating through each sentence in the list, words in the sentence that do not appear in the training set are replaced with an #UNK#. The sentence, the emission parameters and the transition parameters are inputted into our max-marginal algorithm and the resulting word and label sequence of each sentence is appended/concatenated to our final output string. Our max-marginal algorithm is done in the maximum\_marginal\_sentence() function.

0

.

.

.

.

.

.

1

.

.

.

n-1

. . .

To begin the algorithm, the alpha and beta lists are as matrices of zeroes with size n x T, where n refers to the length of the sentence, and T refers to the total number of labels. Alpha refers to our forward probabilities (the sum of scores of all the paths taken from “START” position to the input position with input state), while Beta refers to our backward probabilities (the sum of scores of all paths taken from the input position and input state to the “STOP” position). By multiplying Alpha and Beta, we are basically getting the sum scores for all the paths that pass through the input state and the input position. If we find the label that produces the best sum scores at the input position, we will predict that this label is the optimal label for the input position/word.

In our base case, we set the values of Alpha at position 0 of each label to be the transition probability from “START” to that label, with “START” as its parent node. We also set the values of Beta at position n-1 of each label to be the product of transition probability from that label to “STOP” and the emission probability of that label to the input word. The values of Alpha and Beta for the remaining positions are updated iteratively, summing the scores of all the paths leading to each label at each position.

After getting all our Alpha and Beta values, we multiply the corresponding values to find the label that gives the optimum score at each position. These optimal labels are then returned from our maximum\_marginal\_sentence() function. These optimal labels are then concatenated next to the words in the test data to give us our dev.p4.out.

Part 5

For part 5, we implemented 2 different types of algorithms.

The first is found under ess\_p5.py, which is the “Entity-Sentiment-Separation” Algorithm. This basically does Viterbi twice in succession, however the first run only analyses the Entities, and the second run only analyses the Sentiments. After both are run, we combine the results of both runs of Viterbi to determine whether or not a sentence has those entities and sentiments.

For each sentence, we get a state sequence output just for entities and another state sequence output just for sentiments. Then we iterate over the sentence index and the 2 state different sequence outputs. If at a given index, the entity label is not ‘O’, we check if the sentiment label is also not ‘O’. If both these conditions are satisfied then we merge the results of both sequence outputs to give us our final label for that given index. If the entity label is not ‘O’, but the sentiment label is ‘O’, then we just assume that the label at that index Is ‘O’.