CSE 538 - Natural Language Processing: HW 2

Mihir Chakradeo – 111462188

[I] Description of Viterbi Implementation

Step 1:

I first created two matrices:

a. Transition Matrix: This matrix is the LxN dp matrix, where each cell T[i,j] represents the score of the best tag sequence for w1, …, wj-1
b. Backpointer matrix: This is a LxN matrix, who's each cell holds the index of the best tag in the previous column (word). This matrix is useful in the back pass of the algorithm, when we trace back the path from last word to the first word

Step 2:

I populated the first column of the transition matrix by adding the emission_scores for each tag and the start_scores for the corresponding tag

Step 3:

I then populated the rest of the matrix using the formula given in the assignment pdf:

$$T(i, y) = \psi_x(y, i, \mathbf{x}) + \max_{y'} \psi_t(y', y) + T(i-1, y')$$

The basic idea is as follows:

The current cell T[i,j] is calculated as emission_score of current word added to maximum of k: 1->L T[k,j-1] + trans_score[k,j-1]

For this, I populated a temporary list called prev_list which holds the max(prev to current transition part for each label) and the took the max and argmax

from the temp list. I inserted the argmax in the corresponding cell of the backpointer matrix, and the max in the transition matrix.

Step 4:

Added the end_scores to the last word across each tag

Step 5

Calculated the max from the last column of T matrix and assigned it to score variable. Backtracked the path from last column of T matrix to first path, following the pointers stored in the backpointer matrix to obtain the y list

Returned score and y at the end

Issues faced:

I had to take care of the indexing of emission_scores matrix, as it is NxL and not LxN as the trans_scores. Initially I got indexing error, then realized after printing shape of the matrix that it is transposed.

An improvement to my implementation can be vectorizing the transition matrix populating step.

[II] Description of added features and Comparison against basic features

Before adding features, the base model accuracy was as follows:

CRF = 84.30, LR = 84.39

1. IS_SINGULAR: I used the inflect library to check if a word is singular or plural. Using base model + IS_SINGULAR the new accuracies are as follows: CRF = 84.53, LR = 84.24

2. IS_PUNCTUTATION: I used the string.punctuation() function to check if the word is a punctuation or not. Using base model + IS_PUNCTUATION the new accuracies are as follows: CRF = 84.48, LR = 84.57

3. IS_X: On analyzing the train data, I noticed that there are words which have twitter specific keywords like '#', '@', 'rt'. So, this feature IS_X checks for the twitter specific keywords. Using base model + IS_X the new accuracies are as follows: CRF = 84.24, LR = 84.34

4. IS_ADVERB: I checked if the last 2 characters of a word are 'ly'. Usually adverbs end with 'ly', hence I wanted to check if the word is a possible adverb or not. The motivation for this feature was to improve the ADV accuracy. Using base model + IS_ADVERB the new accuracies are as follows: CRF = 84.53, LR = 84.95

5. IS_1_UPPER: Checked if the first letter of the word is capitalized or not. The motivation behind this feature was to identify proper nouns. Using base model + IS_1_UPPER the new accuracies are as follows: CRF = 84.53, LR = 84.67

6. HAS_HYPHEN: This feature checks for '-' in the word. The motivation for this feature was to capture compound adjectives and ranges. Using base model + HAS_HYPHEN the new accuracies are as follows: CRF = 84.10, LR = 84.62

7. IS_ADJECTIVE: Checked if the word ends with string like 'able', 'ible', 'ent', etc. Usually these words are possible adjectives. The motivation was to increase the accuracy of ADJ tags. Using base model + IS_ADJECTIVE the new accuracies are as follows: CRF = 84.43, LR = 84.72

8. IS_VERB: Checked if the word ends with string like ‘ing’, or ‘ed‘. Usually these words are possible verbs. The motivation was to increase the accuracy of VERB tags. Using base model + IS_VERB the new accuracies are as follows: CRF = 85.19, LR = 84.62

9. IS_DT: Created a hashset with usually used determiners like ‘the’, ‘a’, ‘an’, ‘this’, etc. Motivation of this feature was improving accuracy of DET tags. Using base model + IS_DT the new accuracies are as follows: CRF = 84.34, LR = 84.34

To summarize:

Sr. No	Feature	CRF accuracy (%)	LR accuracy (%)
1	No feature (Base model)	84.30	84.39
2	IS_SINGULAR	84.53	84.24
3	IS_PUNCTUATION	84.48	84.57
4	IS_X	84.24	84.34
5	IS_ADVERB	84.53	84.95
6	IS_1_UPPER	84.53	84.67
7	HAS_HYPHEN	84.10	84.62
8	IS_VERB	85.19	84.62
9	IS_DT	84.34	84.34

10. Brown Clustering:

Brown clustering makes use of word embeddings and clustering them together. The result of the brown clustering is a <word, bitstring, word frequency>. Basically, the bitstring words which have the same prefix are in the same cluster. For example: the word ‘dog‘ has a bitstring ‘000’, and the word ‘cat’ has a bitstring ‘0010’

The way I used brown clustering is modified an already existing implementation in python (can be found here: https://github.com/mheilman/tan-clustering). I parsed the twitter_train.pos

file to get all the tweets on separate lines, without their corresponding tags. The output of the brown clustering is a file called parsed_sorted.txt, which contains the words and bitstrings.

I loaded the parsed_sorted.txt file in the preprocess_corpus() function inside feat_gen.py and created a hashmap of {word:cluster_number} from this file. So, the features are basically 1 for the cluster to which the word belongs to, 0 for the rest of them. Accuracy after using all the aforementioned features: CRF = 86.37, LR = 86.42

[III] Comparison of MEMM and CRF

Note: all the configurations involved also have the base features included

Sr. No	Configuration	MEMM	CRF	Comments
1	IS_SINGULAR, IS_PUNCTUATION, IS_X, IS_ADVERB, IS_1_UPPER, HAS_HYPHEN, IS_VERB, IS_DT	85.67	85.67	Accuracy of each separate token also improved
2	IS_SINGULAR, IS_PUNCTUATION, IS_X, IS_ADVERB, IS_1_UPPER, HAS_HYPHEN, IS_VERB, IS_DT and stemmed words	85.15	85.38	Stemming gives a good base accuracy, however, as I added more features along with stemming, there were no changes. The reason being that stemming removes information about singular/plural words
3	All features + Brown clustering	86.42	86.37	Brown clustering made use of embeddings, hence was able to get spatial information. This is the best model with brown and overall

I tried various hyperparameters with brown clustering like:

Changing the batch size, bitstring size, minimum word count. The trend which I noticed was that smaller bitstring size meant lesser clusters, which gave a bad accuracy. Very large bitstring size also gave a bad score. The reason being, there is a tradeoff between the cluster size and accuracy.

As the cluster size decreases, the correlation between the features decreases, but the spatial relation between the words is lost, and the accuracy decreases.

Conversely, if the cluster size is large, we introduce many zero entries in the feature vector, which leads to high correlation in the features. So again, accuracy decreases.

My experimentation with brown cluster helped me figure out the best parameters: min word count = 2, bitstring = 7, batchsize = 50000

Here are the results for the best model used:

Min word count = 2, bitstring = 7, batchsize = 50000

CRF

Dev evaluation

Token-wise accuracy 86.37653736991486

Token-wise F1 (macro) 85.71545780134761

Token-wise F1 (micro) 86.37653736991486

Sentence-wise accuracy 13.392857142857142

	precision	recall	f1-score	support
.	0.95	0.98	0.97	254
ADJ	0.64	0.52	0.57	99
ADP	0.89	0.89	0.89	151
ADV	0.87	0.70	0.77	129

CONJ	0.97	0.93	0.95	42
DET	0.97	0.90	0.94	130
NOUN	0.78	0.88	0.83	479
NUM	0.84	0.76	0.80	34
PRON	0.99	0.94	0.97	194
PRT	0.91	0.93	0.92	57
VERB	0.86	0.86	0.86	362
X	0.84	0.81	0.83	183
micro avg	0.86	0.86	0.86	2114
macro avg	0.88	0.84	0.86	2114
weighted avg	0.86	0.86	0.86	2114

MEMM

Dev evaluation

Token-wise accuracy 86.42384105960265

Token-wise F1 (macro) 85.36599100681404

Token-wise F1 (micro) 86.42384105960265

Sentence-wise accuracy 14.285714285714285

	precision	recall	f1-score	support
.	0.96	0.99	0.97	254
ADJ	0.72	0.39	0.51	99
ADP	0.91	0.90	0.91	151
ADV	0.90	0.68	0.78	129
CONJ	1.00	0.93	0.96	42
DET	0.99	0.90	0.94	130

NOUN	0.73	0.92	0.82	479
NUM	0.89	0.71	0.79	34
PRON	0.99	0.94	0.97	194
PRT	0.88	0.91	0.90	57
VERB	0.88	0.84	0.86	362
X	0.87	0.81	0.84	183
micro avg	0.86	0.86	0.86	2114
macro avg	0.89	0.83	0.85	2114
weighted avg	0.87	0.86	0.86	2114

[IV] Specifications

Python 2.7

Libraries used: nltk, inflect, string