Project Report

Assignment 2: Part-Of-Speech Tagging

Name: Nagarajan Shanmuganathan

Abstract:
The implementation of Viterbi decoder has been done using Python. The program accepts two files: One for training and one for test file. Training file is '**berp-POS-training.txt**' and the test file is '**assgn2-test-set.txt'**. The output will be generated in '**output.txt'** file.

(Note: If the test file's name is to be changed, please change them in lines 23 and 26 in the program)

Libraries Used: numpy, Collections

Implementation and Assumptions:

1) Initial training was done by shuffling the training set and splitting it into 80:20 for testing.
2) For the baseline tagger, if the word is not present while tagging, then it will be tagged as a *noun* ["NN"].
3) For the Viterbi algorithm, two matrices are computed and fed as input. These matrices correspond to concepts of Hidden Markov Model.
    a. The first matrix is the tag transition matrix, which has the bigram probability values of the tags in the training set
    b. The second matrix is the emission/observation matrix, which has the probability values that given a tag that it will be associated with a given word
4) The tag transition matrix has been smoothened using **Kneser-Neys smoothing** technique.
    a. The parameter ***d*** for the smoothing has been chosen as 0.75
    b. All the bigram words have been considered from the training set and unique bigram tokens are used in the computation
    c. Also, the positions of the occurrence of tags in the tuples are taken into consideration
5) The emission matrix values are filled for every tag, word pair in the following manner

   a. If the frequency of the word in the corpus is less than or equal to the threshold = 1, then a new word called ***"UNK"*** is created and that gets associated with the particular tag. So, all the words that occur less than or equal to threshold with a particular tag will be associated with ***"UNK"*** and those words won't be in the matrix
   b. If the frequency is greater than the threshold, then it is directly inserted into the matrix divided by the number of occurrences of the tag
6) Once these matrices are created, they are sent to the Viterbi decoder. It accepts the words from the test set and computes two new matrices
   a. The ***viterbi*** matrix, which has the probability values corresponding to (tag, word) pair where the words are from the test data and the cells' values are computed from the previous column values
   b. If the test word is not present in the vocabulary, then the probability value for the "UNK" corresponding to tag value is fetched from the emission matrix
   c. The ***back*** matrix's cells have the indices from which we reached the current position
   d. The ***viterbi*** matrix is then scaled during every iteration of the tag, so that the values in the matrix don't converge to zero. The scaling is done by taking the inverse of the sum of the current column's value multiplied with the all the rows corresponding to the current column
7) After computing both the matrices, the maximum value from the last column of ***viterbi*** is taken and the corresponding (I,j)th cell's value in the ***back*** matrix is used to trace back to figure out the tags corresponding to each word in the test set

Results

Upon splitting the training set into 80:20 for training and test, the overall accuracy on running the algorithm multiple times is **94%** on average. The following results were taken from one run of the algorithm.

Confusion Matrix

Confusion Matrix:

Predicted / Actual	.	:	CC	CD	DT	EX	FW	HYPH	IN	JJ	JJR	JJS	MD	NL	NN	NNP	NNS	...	PRP$	RB	RBR	RBS	RP	TO	UH	VB	VBD	VBG	VBN	VBP	VBZ	WDT	WP	WRB	__all__
.	3150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3150
:	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
CC	0	0	442	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	442
CD	0	0	0	795	2	0	0	0	0	1	0	0	0	0	3	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	801
DT	0	0	0	0	1627	0	1	0	0	4	0	0	0	0	1	0	0	...	0	1	0	0	0	0	2	0	0	0	0	0	0	51	0	0	1690
EX	0	0	0	0	0	89	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89
FW	0	0	0	0	5	0	105	0	0	9	0	0	0	0	57	1	8	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	185
HYPH	0	0	0	0	0	0	0	194	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	194
IN	0	0	0	0	11	0	0	0	2315	1	0	0	0	0	2	0	0	...	0	29	0	0	26	193	0	17	0	0	0	0	0	15	0	0	2609
JJ	0	0	0	0	53	0	2	0	0	1515	0	0	0	0	63	4	0	...	0	5	0	0	0	0	15	0	0	0	0	10	0	0	0	0	1670
JJR	0	0	0	0	4	0	0	0	0	0	354	0	0	0	0	0	0	...	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	360
JJS	0	0	0	0	0	0	0	0	0	0	0	76	0	0	0	0	0	...	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	79
MD	0	0	0	0	0	0	0	0	0	0	0	0	1003	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	1008
NL	0	0	0	0	0	0	0	0	0	0	0	0	0	3139	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3139
NN	0	0	0	6	14	0	28	0	0	80	0	0	0	0	4796	21	15	...	0	9	0	0	0	0	0	37	1	1	1	22	0	0	0	0	5037
NNP	0	0	0	0	0	0	8	0	0	12	0	0	0	0	45	79	5	...	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	157
NNS	0	0	0	0	2	0	4	0	0	8	0	0	0	0	67	0	1124	...	0	0	0	0	0	0	0	0	0	0	0	2	9	0	0	0	1216
PDT	0	0	0	0	8	0	0	0	0	0	0	0	0	0	3	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21
POS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	261
PRP	0	0	0	1	0	0	0	0	0	0	0	0	0	0	11	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	2661
PRP$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55
RB	0	0	0	0	6	42	0	0	56	23	3	0	0	0	22	0	1	...	0	1015	0	0	5	0	1	1	0	0	0	0	0	0	0	0	1175
RBR	0	0	0	0	0	0	1	0	0	0	75	0	0	0	1	0	0	...	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	81
RBS	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
RP	0	0	0	0	0	0	0	0	7	1	0	0	0	0	1	0	0	...	0	0	0	0	179	0	0	0	0	2	0	0	0	0	0	0	190
TO	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1053	0	0	0	0	0	0	0	0	0	0	1054
UH	0	0	0	0	48	0	1	0	0	4	0	0	0	0	10	0	0	...	0	0	0	0	0	0	741	15	0	0	0	0	0	0	0	0	819
VB	0	0	0	0	0	0	1	0	1	5	0	0	0	0	56	0	0	...	0	0	0	0	0	0	0	2768	0	0	0	83	0	0	0	0	2914
VBD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	1	...	0	0	0	0	0	0	0	0	100	0	4	0	0	0	0	0	119
VBG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	...	0	0	0	0	0	0	0	0	0	200	0	0	0	0	0	0	215
VBN	0	0	0	0	0	0	1	0	0	7	0	0	0	0	14	0	0	...	0	0	0	0	0	0	0	0	2	0	67	2	0	0	0	0	93
VBP	0	0	0	0	0	0	0	0	0	4	0	0	0	0	14	1	0	...	0	0	0	0	0	0	0	128	0	0	2	1022	0	0	0	0	1171
VBZ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	13	...	0	0	0	0	0	0	0	0	0	0	0	0	356	0	0	0	470
WDT	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	80	9	0	104
WP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	142	0	142
WRB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	215	217
__all__	3150	2	442	802	1796	131	152	194	2379	1674	432	83	1003	3139	5200	107	1167	...	55	1061	4	3	210	1246	759	2966	103	203	74	1153	373	146	151	215	33597

[37 rows x 37 columns]

Other Statistics

Overall Statistics:

Accuracy: 0.9411554603089561
95% CI: (0.9385857877065393, 0.9436485324199779)

Classes	.	:	CC	CD	DT	EX	...	VBN	VBP	VBZ	WDT	WP	WRB
Population	33597	33597	33597	33597	33597	33597	...	33597	33597	33597	33597	33597	33597
P: Condition positive	3150	2	442	801	1690	89	...	3007	4085	3384	1794	142	1392
N: Condition negative	30447	33595	33155	32796	31907	33508	...	30590	29512	30213	31803	33455	32205
Test outcome positive	3150	2	442	802	1796	131	...	3040	4119	3339	1942	151	1276
Test outcome negative	30447	33595	33155	32795	31801	33466	...	30557	29478	30258	31655	33446	32321
TP: True Positive	3150	2	442	795	1627	89	...	2835	4001	3124	1773	142	1230
TN: True Negative	30447	33595	33155	32789	31738	33466	...	30385	29394	29998	31634	33446	32159
FP: False Positive	0	0	0	7	169	42	...	205	118	215	169	9	46
FN: False Negative	0	0	0	6	63	0	...	172	84	260	21	0	162