BBM 497 NLP LAB. ASSIGNMENT 2

Tolgahan Dikmen
21327929
tolgahandikmen95@gmail.com

April 12, 2019

1 Introduction

Part-of-speech (PoS) tagging is the task of assigning syntactic categories to words in a given sentence according to their syntactic roles in that context, such as a noun, adjective, verb etc.

A PoS tagger is implemented using Hidden Markov Models with Viterbi algorithm. These Hidden Markov Models are created by nested dictionary method in Python programming language. First model is as follows:

$$stateDictionary = \{state = \{followingState : Count\}\} \tag{1}$$

Keys of the 'stateDictionary' hold the previous state. Values of the 'stateDictionary' indicates the after state from the key of the 'stateDictionary'. Also each 'state' has 'totalValueOfTheState' value. That value keeps the $following\ state|state$ count. For example:

stateDictionary = { 'Start' = { 'Noun':9, 'Det':5 , 'Adj':2 , 'totalValueOfTheState':16 },
'Noun' = { 'Post':2 , 'Ques':4 , 'Verb':12 , 'totalValueOfTheState':18 }
'Punc' = { 'Interj':7 , 'Pron':2 , 'Noun':12 , 'totalValueOfTheState':21 }
'Interj' = { 'Ques':3 , 'Post':2 , 'Verb':4 , 'totalValueOfTheState':9 }}

Other Hidden Markov Model is as follows: 0

$$tagWordDictionary = \{state = \{word : Count\}\} \tag{2}$$

Keys of the 'tagWordDictionary' hold the state. Values of the 'tagWordDictionary' indicates the word which belongs the state. Also each 'state' has 'totalValueOfTheState' value. That value keeps the $state|word$ count.

tagWordDictionary = { 'Conj' = { 'ama':9, 'ki':4 , 'ne':3 , 'de':6 , 've':8,'totalValueOfTheState':30 },
'Noun' = { 'yaşadıklarımdan':2 , 'gözleri':3 , 'kor':1 , 'totalValueOfTheState':6 }
'Num' = { 'iki':3 , 'doksandokuz':1 , 'altı':1 , 'totalValueOfTheState':5 }}

The other using technique is the Viterbi algorithm to PoS tagging. There is an matrix structure to apply Viterbi algorithm.

The program executes with an input text. The text includes huge amoung of data. The data is actually Turkish sentences.There is a sentence in every line in the text. Also the data contains tags of the words which in the sentences. The program uses up to %70 percent of the data for training part. Remaining part of data is used for the test part.

2 Part Of The Programs

2.1 First Part: Build a Bigram Hidden Markov Model (HMM)

In this task, given file is parsed to tags and words. After this parsing operation, 'Start / <s>' tag is added to begining of the sentences and 'End / <e>' tag is added to end of the sentences. Then all words converted to lowercase and started to building Hidden Markov Models.

The program needs to build two different dictionaries to $state - following\ state$ relationship information and $state - word$ relationship information. Actually when these two dictionaries are created and filled they up correctly, Hidden Markov Models are build successfully. In this assignmet bigram Hidden Markov Model is used. So, all stored informations about 'state-state' and 'state-word' are build according to bigram model.

After HHMs are created, there is one more job to complete to Task I. The job is that calculate to all possibilities to the HHMs.
Possibilities to the $'Start'$ key of the $stateDictionary$ gives us to $Initial\ probability$.
Possibilities to the remaining keys of the $stateDictionary$ gives us to $Transition\ probability$.
Possibilities to the keys of the $tagWordDictionary$ gives us to $Emission\ probability$.

2.2 Second Part: Viterbi Algorithm

In this part, is expected to find most probabily PoS tags on the test set from the program. To do this estimation, Viterbi algorithm is used with build HMMs. The Viterbi algorithm is performed in two steps:

1. Compute the probability of the most likely tag sequence.
2. Trace the back pointers to find the most likely tag sequence from the end of the sentence till the beginning.

In my solution on that assignment, a matrix should be created. First row represents the sentence from taken the test part of the file. First row of the matrix represents all tags which program has seen in training part without 'Start' tag.

Let us say that the sentence is 'I go to school .' and the program has these tag from the training set: 'Noun, Pron, Adj, Punc, Verb'

(0,0)	I(0,1)	go(0,2)	to(0,3)	school(0,4)	.(0,5)	<e>(0,6)
Noun(1,0)	a(1,1)	b(1,2)	c(1,3)	d(1,4)	e(1,5)	f(1,6)
Pron(2,0)	a(2,1)	b(2,2)	c(2,3)	d(2,4)	e(2,5)	f(2,6)
Adj(3,0)	a(3,1)	b(3,2)	c(3,3)	d(3,4)	e(3,5)	f(3,6)
Punc(4,0)	a(4,1)	b(4,2)	c(4,3)	d(4,4)	e(4,5)	f(4,6)
Verb(5,0)	a(5,1)	b(5,2)	c(5,3)	d(5,4)	e(5,5)	f(5,6)

To fill the second column of the table (matrix), the word in the first column and tags in all rows are used. According to this, equations of 'a' will be similar like this:

$$a(i,1) = [P(matrix(i,0)|'Start') * P(matrix(i,0)|matrix(0,1)), came\ from\ information] \tag{3}$$

The first column probabilities are calculated by multiplying of two different possibilities. First possibility is the initial probability of related rows' tag. Second one is the probability that the word belongs to the tag of the related row which is called emission probability.

To fill the rest of the table (matrix), the word in the related column, tags in all rows and previous column with related column are used. We need to add transition probability to (3) equation.

The program needs to an algorithm for unseen words or tags in training part, otherwise some probabilities may be zero and it will be a problem for possibility logic. Therefore $add\ one\ smoothing$ is used for unknows situations on each calculation of probabilty. Add one smoothing equation is as follows:

$$P(t_s|t_i) = \frac{C(t_i|t_s) + 1}{C(t_i) + V} \tag{4}$$

After filled the table (matrix) up, the program decide the maximum possibility sentence tagging version. The maxiumum value of the last column of the matrix is selected. The next step is $traceback$. The selected element of matrix has two information. First one is an informaiton about probability value and second one is information where did it come from. The program looks to the second information and goes to previous columns' row of the caming information tag until reach to beginning. After that a path is created for the sentence.

2.3 Third Part: Evaluation

In this part, the program finds out the accuracy of the test part of the file. To do this, two different value is kept on the program. $True\ tagging\ number$ and $Total\ tagging\ number$. When path is created on part II, the program compares tags on the path with correct path tags. The accuracy of the program is calculated according to this formula:

$$Accuracy = \frac{True\ Tagging\ Number}{Total\ Tag\ Number} \tag{5}$$

For given text file the accuracy of the program is about %57. Tagging of the punctuations at the beginning of the sentence reduce accuracy. Because no sentence starts with punctuation in training part. So the program can not catch the correct tag in the begginning then other taggings will be wrong high probably.

Also I tried two different stemming algorithm to increase the accuracy. But Turkish Stemmers are not nice enough. They decreased the accuracy when they were tried.

3 References

https://www.w3schools.com/python/python_dictionaries.asp
https://web.cs.hacettepe.edu.tr/~burcucan/BBM495.htm
https://www.tutorialspoint.com/python/python_functions.htm
https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf
https://en.wikipedia.org/wiki/Additive_smoothing https://en.wikipedia.org/wiki/Viterbi_algorithm https://pdfs.semanticscholar.org/715a/6a62f714edd1b3345a62cc6133cdac7d7740.pdf