BBM 497: NATURAL LANGUAGE PROCESSING LABORATORY ASSIGNMENT 2

Dilşad Ergün

April 11, 2019

ABSTRACT

Part of Speech (POS) Tagging in Turkish language is a open problem since the Turkish language is a complex language with different syntax, semantics and word affixes/suffixes. The main goal of this work is the implementation of a Part-Of-Speech Tagging tool using Hidden Markov Models (HMMs) and Viterbi algorithm. The corpus file from Middle East Technical University is used for both training and testing. On evaluation stage 86.8% accuracy is achieved in the project.

1 Introduction

"In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context, its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc."(1)

From a very small age, we have been made accustomed to identifying part of speech tags. For example, reading a sentence and being able to identify what words act as nouns, pronouns, verbs, adverbs, and so on. All these are referred to as the part of speech tags. Identifying part of speech tags is much more complicated than simply mapping words to their part of speech tags. This is because POS tagging is not something that is generic. It is quite possible for a single word to have a different part of speech tag in different sentences based on different contexts. That is why it is impossible to have a generic mapping for POS tags.

POS-tagging algorithms fall into two distinctive groups:

- Rule-Based POS Taggers
- Statistical POS Taggers

Typical rule-based approaches use contextual information to assign tags to unknown or ambiguous words. Disambiguation is done by analyzing the linguistic features of the word, its preceding word, its following word, and other aspects. But for this project we had focused on statistical POS taggers to build scalable and effective models. The term 'statistical tagger' can refer to any number of different approaches to the problem of POS tagging. Any model which somehow incorporates frequency or probability may be properly labelled statistical.

The simplest statistical taggers disambiguate words based solely on the probability that a word occurs with a particular tag. In other words, the tag encountered most frequently in the training set with the word is the one assigned to an ambiguous instance of that word. The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags.

An alternative to the word frequency approach is to calculate the probability of a given sequence of tags occurring. This is sometimes referred to as the n-gram approach, referring to the fact that the best tag for a given word is determined by the probability that it occurs with the n previous tags. This approach makes much more sense than the one defined before, because it considers the tags for individual words based on context.

The next level of complexity that can be introduced into a statistical tagger combines the previous two approaches, using both tag sequence probabilities and word frequency measurements. This is known as the Hidden Markov Model (HMM). The project is implemented with using HMM idea to model the train data. After building the model for evaluation stage the Viterbi algorithm is used to allow dynamic programming.

2 Related Work

Part-of-Speech tagging in itself may not be the solution to any particular NLP problem. It is however something that is done as a pre-requisite to simplify a lot of different problems. Let us consider a few applications of POS tagging in various NLP tasks. Such as text to speech conversion, word sense disambiguation, named entity resolution etc. So we can define Part-Of-Speech Tagging as a root algorithm to define further applications. Most of the NLP tools uses Part-Of-Speech taggers in background.

3 Implementation

3.1 Pre-Process Stage

I have started my implementation with the pre-processing stage with differentiating the test and train data with using number of sentences. To be able to avoid problems because of Turkish characters I used UTF-8 encoding while reading data. The input data is format showed as "word/related tag" so the tags and words should be separated from tags. For train set I have used the tags with words while creating the model but for test data in a different method I collected the tags in order to use in evaluation.

The words needed to be standardized for processing and calculating correct possibilities. Differently from the previous project I have not implemented a complex phase for pre-processing stage. Both train and test words are turned into lower case for standardization. I did not need any other process technique to be able to get accurate results.

Before starting the project I thought that I needed to use stemming for standardization. In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s.

In Turkish language stemming is a complex mechanism. So I started to make surveys about stemming libraries. I founded two different libraries: TurkishStemmer and PyStemmer support for Turkish. After test stages I have seen that they are not successful on their job. So I started to think about writing my own stemmer but I put that in second place. After finishing whole work I saw that stemming is not an important thing for this corpus. The libraries only did 1% improvement. Also when I started to develop new techniques and the probability reached above 83% the stemmer started to decrease the results so I decided not to use it. I saw that stemming is not a powerful tool for English language by looking to performance and results. Stemming takes too much time. Libraries need to be develop to be able to become effective.

3.2 Hidden Markov Model (HMM)

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. Hidden Markov Model holds significant terms which are states, observations, transition possibilities and observation likelihoods (emission probabilities). In our data set states refers to tags which can be 'Noun', 'Verb' etc. Observations refer to the words that have these tags. I separated tags and observations with a basic split action in my implementation.

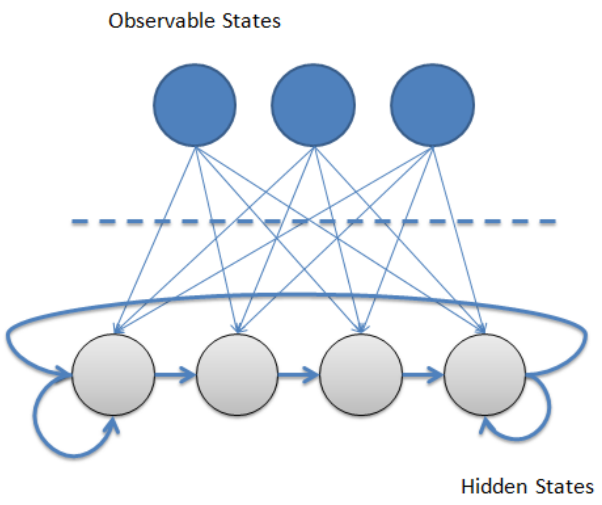


Figure 1: Hidden Markov Model Visualization

As we see in the visualization in Figure 1 after getting states and observations I needed to define the arrows which actually shows the possibilities. I defined transition possibilities as bi-grams. We can define the transition probability as a_{ij} and the equation is described below.

$$a_{ij} = P(state\ q_j\ at\ t+1 \mid state\ q_i\ at\ t) \tag{1}$$

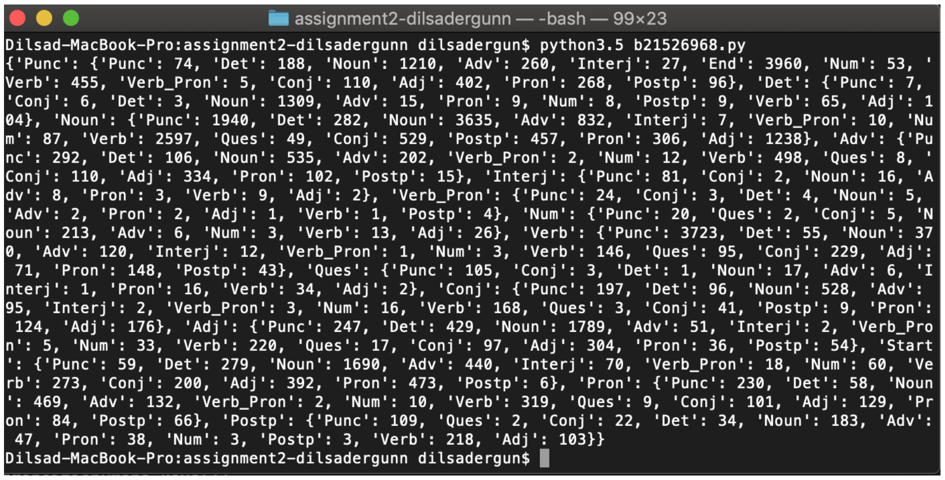
Bigram implementation of states is shown in pseudecode below to make my implementation's definition clear. Also I defined initial probabilities as 'Start' tag. Since algorithm is applied to each sentence I appended 'Start' each sentence's start and appended 'End' each sentence's end.

Algorithm 1 Hidden Markov Model / Bigram Transition Probabilities

```
1:  procedure HMMTRANSITION($TrainDataStates = seperatedbysentences$)
2:      $transitionProbabilities \leftarrow emptyDictionary$
3:      for eachSentence in TrainDataStates  do
4:          $Putstarttagasfirstelementofsentence$
5:          $Putendtagaslastelementofsentence$
6:      for $q\ \epsilon\ TrainDataStates$ do
7:          if $q_t$ not in transitionProbabilities then
8:              Apply smoothing
9:      for $t$ from $startOfSentence$ to $EndOfSentence$ do
10:         if ($q_t\epsilon$ TrainDataStates) not in transitionProbabilities[($q_{t-1}\epsilon$ TrainDataStates)] then
11:             Create empty dictionary, assign to transitionProbabilities[($q_{t-1}$]
12:             Initialize transitionProbabilities[($q_{t-1}$][$q_t$] probability
13:         elseincrease transitionProbabilities[($q_{t-1}$][$q_t$] probability
        return Final nested dictionary structure
```

The general structure of HMM's piece for transition possibilities is hown in the pseudecode above. When we came to its usage to be able to see initial possibilities can be shown asq_0 the nested dictionary structure can be used as TransitionProbabilities['Start']. Others also are reached like that. Another important point is I defined a variable named "Log of zero for viterbi". This variable's job is not actually smoothing. On paper we do not continue calculations from zero cells while applying Viterbi algorithm. If Noun did not came after Verb we do not need to follow that probability. We use 0s on paper to see it. But on implementation if we put 0 or log of zero which is 1 on a table cell and since we choose the maximum one to follow we have a chance to choose that cell. But that cell should be unusable means there is no chance to follow that path. To give no chance to these probabilities I have assigned a too small value to act like an usable cell. Also with calculating logarithms I have put the "log zero for viterbi" variable in action with them which I have mentioned. Like if Adj is not followed by a Det in the corpus never the dictionary[Adj][Det] value becomes log zero for viterbi variable.

The visualization of transition probability model is shown in the Figure 2 with the nested dictionary structure before probability calculations. After this phase the counts turned into logarithm form of the probabilities.



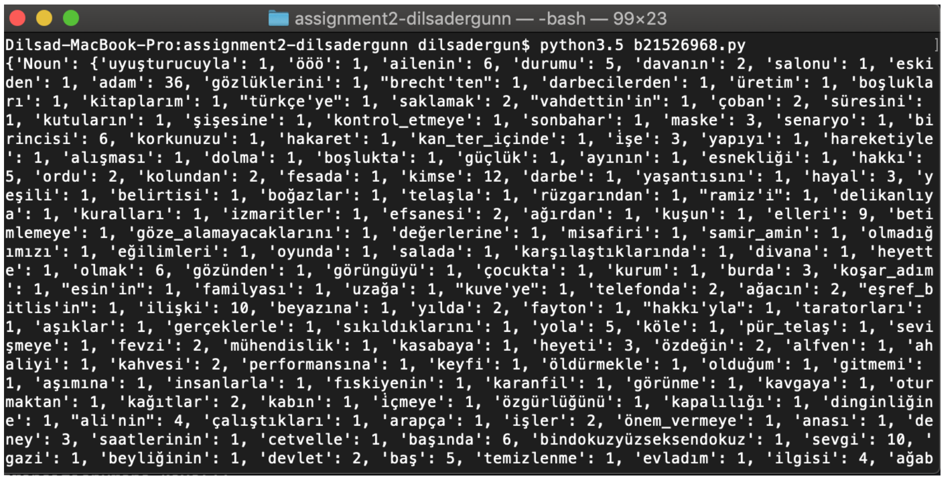
```
Dilsad-MacBook-Pro:assignment2-dilsadergunn dilsadergun$ python3.5 b21526968.py
{'Punc': {'Punc': 74, 'Det': 188, 'Noun': 1210, 'Adv': 260, 'Interj': 27, 'End': 3960, 'Num': 53, '
Verb': 455, 'Verb_Pron': 5, 'Conj': 110, 'Adj': 402, 'Pron': 268, 'Postp': 96}, 'Det': {'Punc': 7,
'Conj': 6, 'Det': 3, 'Noun': 1309, 'Adv': 15, 'Pron': 9, 'Num': 8, 'Postp': 9, 'Verb': 65, 'Adj': 1
04}, 'Noun': {'Punc': 1940, 'Det': 282, 'Noun': 3635, 'Adv': 832, 'Interj': 7, 'Verb_Pron': 10, 'Nu
m': 87, 'Verb': 2597, 'Ques': 49, 'Conj': 529, 'Postp': 457, 'Pron': 306, 'Adj': 1238}, 'Adv': {'Pu
nc': 292, 'Det': 106, 'Noun': 535, 'Adv': 202, 'Verb_Pron': 2, 'Num': 12, 'Verb': 498, 'Ques': 8, '
Conj': 110, 'Adj': 334, 'Pron': 102, 'Postp': 15}, 'Interj': {'Punc': 81, 'Conj': 2, 'Noun': 16, 'A
dv': 8, 'Pron': 3, 'Verb': 9, 'Adj': 2}, 'Verb_Pron': {'Punc': 24, 'Conj': 3, 'Det': 4, 'Noun': 5,
'Adv': 2, 'Pron': 2, 'Adj': 1, 'Verb': 1, 'Postp': 4}, 'Num': {'Punc': 20, 'Ques': 2, 'Conj': 5, 'N
oun': 213, 'Adv': 6, 'Num': 3, 'Verb': 13, 'Adj': 26}, 'Verb': {'Punc': 3723, 'Det': 55, 'Noun': 37
0, 'Adv': 120, 'Interj': 12, 'Verb_Pron': 1, 'Num': 3, 'Verb': 146, 'Ques': 95, 'Conj': 229, 'Adj':
 71, 'Pron': 148, 'Postp': 43}, 'Ques': {'Punc': 105, 'Conj': 3, 'Det': 1, 'Noun': 17, 'Adv': 6, 'I
nterj': 1, 'Pron': 16, 'Verb': 34, 'Adj': 2}, 'Conj': {'Punc': 197, 'Det': 96, 'Noun': 528, 'Adv':
95, 'Interj': 2, 'Verb_Pron': 3, 'Num': 16, 'Verb': 168, 'Ques': 3, 'Conj': 41, 'Postp': 9, 'Pron':
 124, 'Adj': 176}, 'Adj': {'Punc': 247, 'Det': 429, 'Noun': 1789, 'Adv': 51, 'Interj': 2, 'Verb_Pro
n': 5, 'Num': 33, 'Verb': 220, 'Ques': 17, 'Conj': 97, 'Adj': 304, 'Pron': 36, 'Postp': 54}, 'Start
': {'Punc': 59, 'Det': 279, 'Noun': 1690, 'Adv': 440, 'Interj': 70, 'Verb_Pron': 18, 'Num': 60, 'Ve
rb': 273, 'Conj': 200, 'Adj': 392, 'Pron': 473, 'Postp': 6}, 'Pron': {'Punc': 230, 'Det': 58, 'Noun
': 469, 'Adv': 132, 'Verb_Pron': 2, 'Num': 10, 'Verb': 319, 'Ques': 9, 'Conj': 101, 'Adj': 129, 'Pr
on': 84, 'Postp': 66}, 'Postp': {'Punc': 109, 'Ques': 2, 'Conj': 22, 'Det': 34, 'Noun': 183, 'Adv':
 47, 'Pron': 38, 'Num': 3, 'Postp': 3, 'Verb': 218, 'Adj': 103}}
Dilsad-MacBook-Pro:assignment2-dilsadergunn dilsadergun$
```

Figure 2: Transition Counts Visualization

Calculating emission probabilities was much easier. I again defined a nested dictionary structure for observation likelihoods as $(Tag : (Word : Possibility))$. Since it is a unigram-like definition I only assigned the word probabilities of related tags. We can define the emission probability as $b_j(k)$ and the equation is described below.

$$b_j(k) = P(observation\ k\ at\ t \mid state\ q_j\ at\ t) \tag{2}$$

A small visualization of emission counts are shown in Figure 3. After this phase the counts turned into logarithm form of the probabilities. Since I did not use stemming the words are directly in their corpus form in obsevation likelihoods. The structure only holds seen words and their probabilities under the tags. For the unseen observations the smoothing-like operations is done under test data process stage while applying Viterbi algorithm.



```
Dilsad-MacBook-Pro:assignment2-dilsadergunn dilsadergun$ python3.5 b21526968.py
{'Noun': {'uyuşturucuyla': 1, 'ööö': 1, 'ailenin': 6, 'durumu': 5, 'davanın': 2, 'salonu': 1, 'eski
den': 1, 'adam': 36, 'gözlüklerini': 1, "brecht'ten": 1, 'darbecilerden': 1, 'üretim': 1, 'boşlukla
rı': 1, 'kitaplarım': 1, "türkçe'ye": 1, 'saklamak': 2, "vahdettin'in": 1, 'çoban': 2, 'süresini':
1, 'kutuların': 1, 'şişesine': 1, 'kontrol_etmeye': 1, 'sonbahar': 1, 'maske': 3, 'senaryo': 1, 'bi
rincisi': 6, 'korkunuzu': 1, 'hakaret': 1, 'kan_ter_içinde': 1, 'İşe': 3, 'yapıyı': 1, 'hareketiyle
': 1, 'alışması': 1, 'dolma': 1, 'boşlukta': 1, 'güçlük': 1, 'ayının': 1, 'esnekliği': 1, 'hakkı':
5, 'ordu': 2, 'kolundan': 2, 'fesada': 1, 'kimse': 12, 'darbe': 1, 'yaşantısını': 1, 'hayal': 3, 'y
eşili': 1, 'belirtisi': 1, 'boğazlar': 1, 'telaşla': 1, 'rüzgarından': 1, "ramiz'i": 1, 'delikanlıy
a': 1, 'kuralları': 1, 'izmaritler': 1, 'efsanesi': 2, 'ağırdan': 1, 'kuşun': 1, 'elleri': 9, 'beti
mlemeye': 1, 'göze_alamayacaklarını': 1, 'değerlerine': 1, 'misafiri': 1, 'samir_amin': 1, 'olmadığ
ımızı': 1, 'eğilimleri': 1, 'oyunda': 1, 'salada': 1, 'karşılaştıklarında': 1, 'divana': 1, 'heyett
e': 1, 'olmak': 6, 'gözünden': 1, 'görüngüyü': 1, 'çocukta': 1, 'kurum': 1, 'burda': 3, 'koşar_adım
': 1, "esin'in": 1, 'familyası': 1, 'uzağa': 1, "kuve'ye": 1, 'telefonda': 2, 'ağacın': 2, "eşref_b
itlis'in": 1, 'ilişki': 10, 'beyazına': 1, 'yılda': 2, 'fayton': 1, "hakkı'yla": 1, 'taratorları':
1, 'aşıklar': 1, 'gerçeklerle': 1, 'sıkıldıklarını': 1, 'yola': 5, 'köle': 1, 'pür_telaş': 1, 'sevi
şmeye': 1, 'fevzi': 2, 'mühendislik': 1, 'kasabaya': 1, 'heyeti': 3, 'özdeğin': 2, 'alfven': 1, 'ah
aliyi': 1, 'kahvesi': 2, 'performansına': 1, 'keyfi': 1, 'öldürmekle': 1, 'olduğum': 1, 'gitmemi':
1, 'aşımına': 1, 'insanlarla': 1, 'fıskiyenin': 1, 'karanfil': 1, 'görünme': 1, 'kavgaya': 1, 'otur
maktan': 1, 'kağıtlar': 2, 'kabın': 1, 'İçmeye': 1, 'özgürlüğünü': 1, 'kapalılığı': 1, 'dinginliğin
e': 1, "ali'nin": 4, 'çalıştıkları': 1, 'arapça': 1, 'işler': 2, 'önem_vermeye': 1, 'anası': 1, 'de
ney': 3, 'saatlerinin': 1, 'cetvelle': 1, 'başında': 6, 'bindokuzyüzseksendokuz': 1, 'sevgi': 10, '
gazi': 1, 'beyliğinin': 1, 'devlet': 2, 'baş': 5, 'temizlenme': 1, 'evladım': 1, 'ilgisi': 4, 'ağab
```

Figure 3: Emission Counts Visualization

With 2 nested dictionary structures, transition probabilities and observation likelihoods, our Hidden Markov Model became ready with the probabilities in $\log_2$ form. The model is directly used to tag test words with the Viterbi algorithm.

3.3 Viterbi Algorithm

We already know that the probability of a label sequence given a set of observations can be defined in terms of the transition probability and the emission probability. We created our Hidden Markov Model using these probabilities. For test data our aim was to tag sentences by looking at to this model by the help of emission and transition probabilities. To be able to use the model effectively Viterbi algorithm is applied for the task. The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models. (2)

Algorithm 2 Viterbi Algorithm

```
1:  procedure VITERBI($States, Transition, Observations = x, Emissions, Initial = q_0$)
2:      Initialize data structure for viterbi
3:      for $q \epsilon States$ do
4:          P(q,1)=Transition($q_0$,q)Emission(q,$x_1$)
5:          Back(q,1)=$q_0$
6:      for $t$ from 2 to $T$ do
7:          for $q \epsilon States$ do
8:              if $x_t$ not in Emission then
9:                  Apply smoothing
10:             P(q,t)= $max_{q_p \epsilon States}$ P($q_p$,t-1)Transition($q_p$,q)Emission(q,$x_t$)
11:             Back(q,t) $argmax_{q_p \epsilon States}$ P($q_p$,t-1)Transition($q_p$,q)
12:     $Path_{startPoint}$ =$argmax_{q_p \epsilon States}$ P($q_p$,T)
        return Backtrace path following pointers
```

The general structure of my Viterbi algorithm is described above with the peseudecode. States shows the tags which are Noun, Verb, Adj etc. The parameter named Transition is the nested dictionary structure which holds the transition probabilities. Observations shows a single sentence from the test data. The viterbi method is called for each sentence. Emissions shows the nested dictionary which holds the observation likelihoods and lastly Initial is the initial probabilities, which means which tag is most likely to come at the beginning of the sentence.

The data structure that I used to hold Viterbi table, is a list of dictionaries. For each observation a dictionary is appended and the dictionary keys are tags. The values are the probabilities for the observation probabilities for this tag. Also I used the same data structure for holding backtraces too.

Table 1: Visualization of Viterbi Table

	start symbol	x[0]	x[...]	end symbol
Start	0	0 + emission(x[0]) + -1000	...	-3045
Noun	-1000	-1000 +emission(x[0]) + -1	...	-1780
Verb	-1000	-1000 +emission(x[0])+ -3.5	...	-1945
Adv	-1000	-1000 +emission(x[0]) + -3.01	...	-2569
Adj	-1000	-1000 +emission(x[0]) + -3.3	...	-2854
Num	-1000	-1000 +emission(x[0]) + -6	...	-1690
Det	-1000	-1000 +emission(x[0]) + -3.1	...	-2390
Punc	-1000	-1000 +emission(x[0]) + -6.8	...	-1237
Interj	-1000	-1000 +emission(x[0]) + -5.8	...	-2786
Postp	-1000	-1000 +emission(x[0]) + -5	...	-1986
Verb Pron	-1000	-1000 +emission(x[0]) + -7	...	-2138
Conj	-1000	-1000 +emission(x[0]) + -5.2	...	-2567
Ques	-1000	-1000 +emission(x[0]) + -1000	...	-2668
Pron	-1000	-1000 +emission(x[0]) + -7.5	...	-1996
End	-1000	-1000 +emission(x[0]) + -1000	...	**-900**

We can see a small visualization of a Viterbi table that created in my implementation. After completing viterbi table with choosing the maximum value for the tag of related observation I take the maximum probability in last column, which is the last dictionary. For example for the table the maximum value is related to End tag so we start by putting End to the path. Then for determining the path I only follow the back traces and put them in a list. Since I come from end to start while following back traces I append the tags to list from end to start, that is why I reverse the list at the end to get the observed path for the sentence.

After getting the last path the method returns the observed sequence and the sequence is compared with the correct tags which is read in another method. A count value counts the matched word and increases the total count for the calculation at the end. In evaluation stage the total count is divided to total number of words to find the percentage of success.

The Viterbi working mechanism does not change in any situation on the implementation that I have created. I only added some additional mechanisms to make model and probabilities more accurate which results better found tag sequences on Viterbi. The extra mechanisms is described in the section following in details.

3.4 Extra Mechanisms

I have defined some mechanisms to improve my success. We can list these mechanisms as:

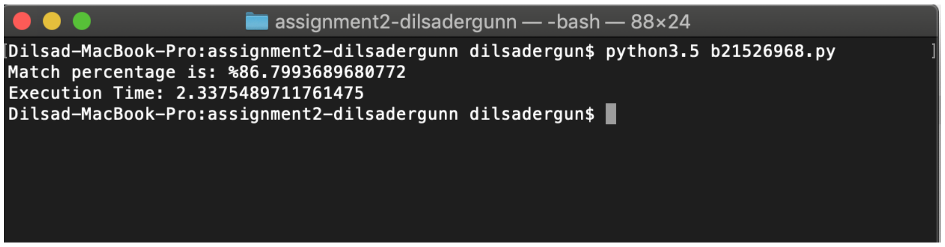
- The start of sentence symbol are so important for this purpose. Actually it is about the initial possibilities. We need to consider the fact that which tag is most likely to come at the beginning of the sentence. The first possibility is directly determined by that so it affects whole chain respectively. Noun tag usually won at the start of the sentence.
- Like start of sentence I also used the end of sentence symbol. The point is that we need to consider after which tag the sentence ends at most of the time. This affects the last column directly so actually it determines our starting point to back trace. With this mechanism I find that all sentences' are finished with Punc tag. In real tags it was just like that so it helped to increase the success a lot.
- While I was reviewing the wrong tagged sentences I saw that some sentences were beginning with a punctuation which are the quotes. Since at training phase it is not seen or seen less the tag was coming Noun instead of Punc. So while determining emission probabilities I have added the characters of string.punctuation as single and double both. It became a small rule based approximation to system.
- Another mechanism is the log zero for viterbi value. It is not for increasing success, it is only for true implementation. On theory as we saw in our lectures we do not continue calculations from zero cells while applying Viterbi algorithm. If Noun did not came after Verb we do not need to follow that probability. We use 0s on paper to see it. But on implementation if we put 0 or log of zero which is 1 on a table cell and since we choose the maximum one to follow we have a chance to choose that cell. But that cell should be unusable means there is no chance to follow that path. To give no chance to these probabilities I have assigned a too small value to act like an usable cell. Also with calculating logarithms I have put the "log zero for viterbi" variable in action with them which I have mentioned. Like if Adj is not followed by a Det in the corpus never the dictionary[Adj][Det] value becomes log zero for viterbi variable.
- The last implemented mechanism is a smoothing-like approach. On my implementation stage I reviewed several articles about smoothing techniques for natural language processing tasks. I did not wanted to use the add-1 directly because it was the lazy one. On my researches I was that the known probabilities should be more clear rather than the others. If I have the word in my possibilities I directly get the possibility but if I do not I assign the small value that I used to stock the path. The path does not stop but has smaller probability. It was actually a lazy approach so did not help to increase probability too much. So I added a small detail that is inspired by K-Nearest-Neigbour and Good Turing smoothing. I reviewed the train data and the tag noun is triple more than most of them. So it is actually the most common tag. But I did not rule this directly to avoid rule based approach. For an unseen word I have given a different probability of this word to be a noun. This probability is gathered from the observation likelihoods. The probability is equal to the minimum emission probability under Noun tag. So we can call the smooth value as 1 /number of words tagged as Noun.(3)

By adding these mechanisms I increased my success by 5%, so I reached to 86% success. Before reaching above 83% stemming was helpful but at this phase stemming decreases my success since it does not work successfully so I preferred not to use it.

4 Results and Evaluation

4.1 Results

The output of the result is shown in the Figure 4 below. I have reached 86.79% success on evaluation after all implementations.



```
Dilsad-MacBook-Pro:assignment2-dilsadergunn dilsadergun$ python3.5 b21526968.py
Match percentage is: %86.7993689680772
Execution Time: 2.3375489711761475
Dilsad-MacBook-Pro:assignment2-dilsadergunn dilsadergun$
```

Figure 4: Evaluation Result

4.2 Comparisons and Discussions

I saw my walk on the way to get the 86% result I can define myself successful comparing to myself. Also I think it is a high success rate for an application. Another point is that I decreased the execution time into half comparing to beginning by editing my wrong implementations and changes on data structure definition. I have reached 2-2.5 seconds interval as execution time.

After the implementation at beginning I have taken results like 80%. At my first implementation the described structures were all used and I only had start of sentence symbol. So the found sequence was like Start Noun ... Punc. But I realised that I need to add the end symbol too. Because the probabilities that after which tag the sentence is ended most likely was a considerable fact. By adding end of sentence symbol my success became nearly 82%. At that time the stemming operation with libraries was giving me an increase of 1%. But with increase the execution time was doubling so I was not sure about it. After when I reached my 86% result with the extra mechanisms defined, stemming started to decrease the probability so I directly did not use it.

5 Conclusion

As a result I have completed the assignment with a success rate 86,8%. With these experiments I firstly had a chance to understand different language modeling structures like Hidden Markov Models. Creating a model from scratch helped me to understand the back stage of the applications which I use commonly in daily life. Designing the model was comparing easy because of the n-gram project that I have implemented. I can say that I implemented the model in one shot, did not take my time. But as we see theoretically in our lecture, in practice implementation of Viterbi algorithm was much harder. I wasted nearly a week by only thinking the structure before started coding. I changed too many data structure decisions, tried implementations on them separately.

Another challenging point about the project was the language of the model, Turkish. I believe that the success rate would be more in English, maybe. Turkish has too complex mechanisms about suffixes, affixes, the order of sentence etc. Also when I planned to write my own stemmer and started to think, I realized that I need to consider too many factors about Turkish which cannot be handled in 2-3 weeks.

At the end, I could understand the Hidden Markov Model and Viterbi algorithm clearly in all purposes. After my experiments, now I am able to develop applications based on part of speech tagging. I believe that part of speech tagging can be useful for many natural language processing applications' background.

References

[1] Wikipedia, "Part of speech tagging." [Online]. Available: http://www.wikizero.biz/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvUGFydC1vZi1zcGVlY2hfdGFnZ2luZw

[2] t. f. e. Wikipedia, "Viterbi algorithm." [Online]. Available: http://www.wikizero.biz/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvVml0ZXJiaV9hbGdvcml0aG0

[3] E. K. Steven Bird and E. Loper, "Probability space," in *Taming Text*. Manning, April 29, 2013, pp. 160–190.