BBM497: Introduction to Natural Language Processing Lab. **(Due: 20/04/2020)**

Submission Assignment #2

Instructor: Burcu Can *Name:* Metin DEMİR, *Netid:* 21526902

1 Introduction

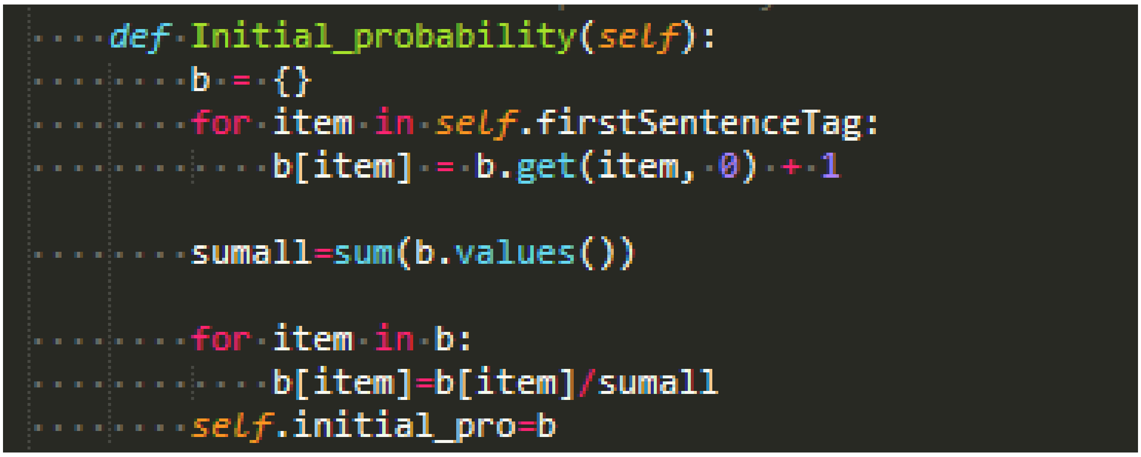
In this assignment, we will implement a NER tagger using Hidden Markov Models (HMMs). Therefore, we will practice HMMs and Viterbi algorithm in this assignment.

2 Approach

First of all things I read my dataset in this dataset I return sentence, all tag of word which in sentences, tag of first word of sentences and existing tags. After that I get all tag of sentences. For calculation of initial probability I apply this formula

$$InitialProbability = \frac{oftaginarray}{ofalltagsinarray}$$

then I record this information in a dictionary where key is tag, probability is value of it. After that I have to calculate transit probability for this I created a dictionary where (tag(n+1), tag(n)) is key and probability is value of this key. For calculation of emission probability I created a dictionary key is word and value is a list in this list I store dictionary which store tag and probability of (word, tag) after calculation these probability.

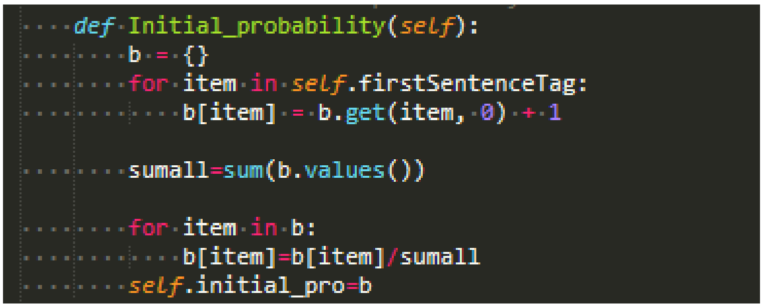


```
    def Initial_probability(self):
        b = {}
        for item in self.firstSentenceTag:
            b[item] = b.get(item, 0) + 1

        sumall=sum(b.values())

        for item in b:
            b[item]=b[item]/sumall
        self.initial_pro=b
```

Figure 1: Class Figure

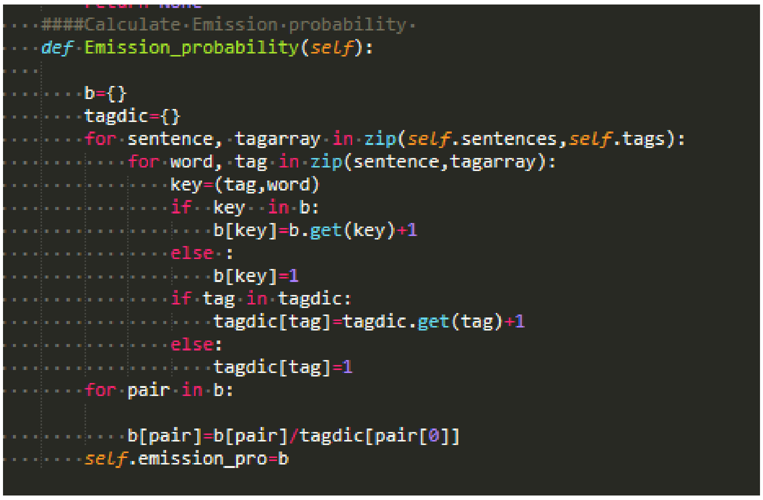


```
    def Initial_probability(self):
        b = {}
        for item in self.firstSentenceTag:
            b[item] = b.get(item, 0) + 1

        sumall=sum(b.values())

        for item in b:
            b[item]=b[item]/sumall
        self.initial_pro=b
```

Figure 2: Class Figure



```
        return None
    ####Calculate Emission probability 
    def Emission_probability(self):
    
        b={}
        tagdic={}
        for sentence, tagarray in zip(self.sentences,self.tags):
            for word, tag in zip(sentence,tagarray):
                key=(tag,word)
                if  key  in b:
                    b[key]=b.get(key)+1
                else :
                    b[key]=1
                if tag in tagdic:
                    tagdic[tag]=tagdic.get(tag)+1
                else:
                    tagdic[tag]=1
        for pair in b:

            b[pair]=b[pair]/tagdic[pair[0]]
        self.emission_pro=b
```

Figure 3: Class Figure

I created my viterbi algorithm. In there I have state as many as tag number and words then for all word I find best state way with backtracking logic. After that I return which tags visited. The visited tag is my outputs. For calculation of accuracy I just get tags of test and compare with my result. Then just print predict result of this formula

$$A(W) = \frac{ofcorrectfoundtags}{oftotalwords}$$

3 Result

Accuracy is 86 that's show us to that our model doesn't so bad but need some improvement. I did not chance any word (all is not lowercase or uppercase). If apply all word lowercase I can get better result. I did not use smoothing too for this I just return 0 value. (Smoothing and return 0 to close each other efficiency is my first concern that's why I did not smoothing value) that's my structure and result. In this assignment I learn what's initial, transit and emission value. And implementing of viterbi algorithm. I saw also where we can dictionary or list in python.