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

Submission Assignment #2

Instructor: Necva BÖLÜCÜ *Name:* Ahmet Faruk BAYRAK, *Netid:* 21426716

1 Introduction

In this assignment, we are expected to build a **bigram Hidden Markov Model (HMM)** and predict name entity of words in the test sentences using **viterbi algortihm.**

1.1 Name Entity Recognition (NER)

Named-entity recognition (NER) is a subtask of information extraction that seeks to locate and classify named entity mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

1.2 Dataset

As a training and test dataset we used CoNLL 2003 NER dataset. This dataset is composed of sentences with words per line 5 possible name entity labels: "LOC, ORG, MISC, PER, O". The "LOC, ORG, MISC, PER, O" labels are abbreviation of "LOCATION, ORGANIZATION, MISCELLANEOUS, PERSON, OTHER" respectively.

2 Data structures

2.1 Preprocessing

Before using the training data I change every tokens to lowercase.

For reading and creating sentences, I used newline as a sentence boundary.

After that my preprocessing of the training data was done.

2.2 Task 1: Building a Bigram Hidden Markov Model (HMM)

When preprocessing part is done building of bigram HMM model part begins. A hidden Markov model (HMM) allows us to talk about both observed events Hidden Markov model (like words that we see in the input) and hidden events (like name entity tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

States: a set of our trained tags which are unique.

```
{'per', 'org', 'misc', 'o', 'loc'}
```

Observations: each word from our test sentence.

```
[['eu', 'org'], ['rejects', 'o'], ['german', 'misc'], ['call', 'o'], ['to', 'o'], ['boycott', 'o'], ['british', 'misc'], ['lamb', 'o'], ['.', 'o']]
```

Initial probability: The probability of the tag of the first word that a sentence begins with.

```
{'org': 0.16380863414959632, 'per': 0.09001134316407554, 'o': 0.6071929005137786, 'misc': 0.03349569627010075, 'loc': 0.10549142590244878}
```

Transition probability: The probability of moving from tag to another tag.

{'org': {'org': 0.3784771573604061, 'misc': 0.0009137055837563451, 'o': 0.6197969543147208, 'per': 0.0006091370558375635, 'loc': 0.0002030456852791878}, 'per': {'per': 0.41594708800293956, 'o': 0.5836854675730295, 'misc': 9.186110600771634e-05, 'loc': 0.000275583318023149}, 'o': {'org': 0.024268023423250455, 'misc': 0.01830981408594925, 'o': 0.8888419570573741, 'per': 0.0331701385555662 2, 'loc': 0.03541006687785991}, 'misc': {'org': 0.00876010781671159, 'misc': 0.2672955974842767, 'o': 0.7079964061096137, 'per': 0.013701707097933512, 'loc': 0.002246181491464 5105}, 'loc': {'org': 0.0013518495760108148, 'loc': 0.14354184588914834, 'o': 0.8512965466388104, 'per': 0.00012289541600098315, 'misc': 0.003686862480029495}}

Emission probability: The probability of generating the word from the hidden state.

{'org': {'microsoft': 0.0007980049875311721, 'warszawa': 9.975062344139652e-05, 'chemical': 9.975062344139652e-05, 'wigan': 0.000997506234413965, 'moss': 0.0002992518703241895, 'team': 0.0002992518703241895, 'banc': 9.975062344139652e-05, 'lebanese': 9.975062344139652e-05, 'middlesbrough': 0.0004987531172069825, 'm&r': 0.0001995012468827930 3, 'apa': 9.975062344139652e-05, 'securities': 0.0016957605985037406, 'al-watani': 9.975062344139652e-05, 'perrin': 0.0001995012468827930 3, 'vasco': 0.0002992518703241895, 'lidl': 9.975062344139652e-05, 'v': 0.0001995012468827930 3, 'olympiakos': 0.0005985037406483791, 'touchstone':

2.3 Smoothing

When testing the test data there might be a word which never appeared in the train data. We call them **unkown word**. Since these words are not in our model, the probability of these words will be zero. So this zero values will effect upcoming operations. For handling this problem, we needed to use smoothing.

For smoothing, I used simple **Good Turing Algorithm** to estimate the emission probabilities for unknown words. In this approach, the emission probabilities for the known words are estimated using maximum-likelihood estimation and the emission probabilities for the unknown words are estimated separately.

P(u—t): Probability
n0: number of unknown words.
n1: number of words which appeared once with tag t.
N(t): total number of words which appeared with tag t.

$$P(u|t) = n_1(t)/n_0N(t)$$

2.4 Task 2: Viterbi Algorithm

In this part, we tried to predict the **name entity** tags in a given raw text by using the Hidden Markov Model that we build in the first part. For calculating probabilities we implemented the Viterbi algorithm.

For my implementing I use a dictionary in dictionary for keeping probabilities. Structure of dictionary is like: (currentTag: (previousTag1: probability), (previousTag2: probability), ...)

For example If I give "['eu', 'org'], ['rejects', 'o'], ['german', 'misc'], ['call', 'o'], ['to', 'o'], ['boycott', 'o'], ['british', 'misc'], ['lamb', 'o'], ['.', 'o']" as an input to Viterbi function:

Start Probabilities for first word "Eu":

[{'org': ('-', -11.316268808749548), 'o': ('-', -16.977541108488186), 'per': ('-', -17.66800740510828), 'loc': ('-', -18.80037757697121), 'misc': ('-', -19.87510764020469)}]

Probabilities for word "call":

{'loc': ('misc', -65.03250544998677), 'o': ('misc', -53.30849426179028), 'org': ('org', -58.92998159624329), 'misc': ('misc', -57.557339651504705), 'per': ('misc', -61.06237901603821)}

Last probability for ".":

{'o': ('o', -106.93751544876154), 'org': ('org', -116.62516804536983), 'misc': ('misc', -115.8883744954887), 'per': ('o', -121.34201021807205), 'loc': ('o', -118.0718517232215)}

For backtracing algorithm looks for maximum value in the last probability values. So it will start backtracing from "o" tag.

Predicted Tags:

```
['org', 'o', 'misc', 'o', 'o', 'o', 'misc', 'o', 'o']
```

2.5 Error Analysis

For calculating probabilities I used **logarithm** and **addition** instead of multiplication. The main purpose of that I used logarithm is we were dealing with very small numbers. So sometimes interpreter rolls very small number to zero. Also addition is more efficient than multiplication.

2.6 Task 3: Evaluation

In this part, we were expected to calculate accuracy of our model in the test data. The formula of accuracy is:

$$A(W) = \frac{\#of\_correct\_found\_tags}{\#of\_total\_words}$$

I have two functions for finding accuracy of model.

In the preprocessing part I made "B-TAG" and "I-TAG" into "tag". For example I made "B-ORG" and "I-ORG" to "org"

So my first function calculates accuracy according to preprocessed tags ("org", "o", "loc", "per", "misc"):

```
Correct predicted tag count: 41149
Total tag count: 46435
Accuracy: 88.61634542909444
```

But for kaggle I had to change them to original positions.

So my other evaluation function calculates accuracy according to original tags ("B-ORG", "I-ORG", "B-LOC", "I-LOC", "B-PER", "I-PER", "B-MISC", "I-MISC", "O")

```
Correct predicted tag count: 40780
Total tag count: 46435
Accuracy: 87.82168622806073
```

I think the cause of difference between percentages is my translation mistakes on "tag" to "B-TAG" or "I-TAG"