Hacettepe University
Department
Of
Computer Science

BBM-497 Natural Language Processing Laboratory
Assignment 1

Simge ACIMIS / 21327598

March 21st - 2019

Contents

Chapter 1

Authorship Detection

1.1 Introduction

Language Modeling is one of the most important parts of modern Natural Language Processing. There are many sorts of applications for Language Modeling, like: Machine Translation, Spell Correction Speech Recognition, Summarization, Question Answering, Sentiment analysis etc. Each of those tasks require use of language model. Language model is required to represent the text to a form understandable from the machine point of view. Our aim is to practice language models. It defines characteristics of the word in the language.

1.2 Task 1: Building Language Models

I created **unigram, bigram and trigram language models** using some preprocessing steps such as removing punctiations, lowercasing tokens. I removed punctuations with **regular expressions**.

```python
# task 1 unigram model
def build_unigram(author, name):
    dict = {}
    print("building unigram...")
    for arr in author[name]:
        for word in arr:
            if word in dict:
                dict[word] += 1
            else:
                dict[word] = 1
    return dict
```

```python
for line in file:

    line = re.sub(r'[^\w\s]', '', line).lower()
    line = line.rstrip('\n').split(' ')
    lines.append(line)
```

```python
# task 1 trigram model
def build_trigram(author, name):
    print('building trigram...')
    dict = {}
    for arr in author[name]:
        for i in range(len(arr) - 2):
            if arr[i] + ' ' + arr[i+1] + ' ' + arr[i+2] in dict:
                dict[arr[i] + ' ' + arr[i+1] + ' ' + arr[i+2]] += 1
            else:
                dict[arr[i] + ' ' + arr[i+1] + ' ' + arr[i+2]] = 1
    return dict
```

```python
# task 1 bigram model
def build_bigram(author, name):
    # len - 1 for dönecek
    dict = {}
    print("building bigram...")
    for arr in author[name]:
        for i in range(len(arr) - 1):
            if arr[i] + ' ' + arr[i+1] in dict:
                dict[arr[i] + ' ' + arr[i+1]] += 1
            else:
                dict[arr[i] + ' ' + arr[i+1]] = 1
    return dict
```

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

I also used add-one Laplace smoothing because smoothing reduces the variance.

1.3 Task 3: Classification and Evaluation

- ```
  calculating bigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating bigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating bigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating bigram perplexity...
  Unsuccessful prediction for Hamilton Essay
  ----------------------------------------------
  calculating bigram perplexity...
  Successful prediction for Hamilton Essay
  ----------------------------------------------
  calculating bigram perplexity...
  Successful prediction for Hamilton Essay
  ----------------------------------------------
  ```

  - When I am using bigram language model that I created, I observed that model is predict correctly 5 essay out of 6 essay.

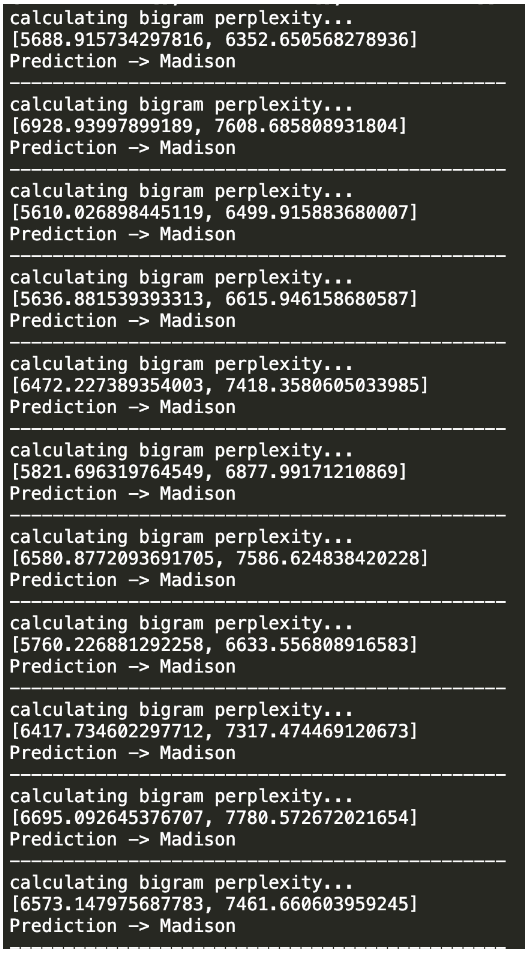
- ```
  calculating trigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating trigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating trigram perplexity...
  Successful prediction for Madison Essay
  ----------------------------------------------
  calculating trigram perplexity...
  Unsuccessful prediction for Hamilton Essay
  ----------------------------------------------
  calculating trigram perplexity...
  Unsuccessful prediction for Hamilton Essay
  ----------------------------------------------
  calculating trigram perplexity...
  Unsuccessful prediction for Hamilton Essay
  ----------------------------------------------
  ```

  - But, If I use trigram language model, this correctness decreased. It predicts 3 out of 6.

```
def calculate_bigram_perplexity(text, d1, d2):
    print('calculating bigram perplexity...')
    total_log_of_probabilities_d1 = 0
    total_log_of_probabilities_d2 = 0
    dict_text = {}
    for word in text:
        dict_text[word] = 0

    # madison total sum of log
    for i in range(len(text) - 1):
        if text[i] + ' ' + text[i+1] in d1:
            total_log_of_probabilities_d1 += math.log2(d1[text[i] + ' ' + text[i+1]])
        else:
            total_log_of_probabilities_d1 += math.log2(1 / (len(d1) + len(text)))

    # hamilton total sum of log
    for i in range(len(text) - 1):
        if text[i] + ' ' + text[i+1] in d2:
            total_log_of_probabilities_d2 += math.log2(d2[text[i] + ' ' + text[i+1]])
        else:
            total_log_of_probabilities_d2 += math.log2(1 / (len(d2) + len(text)))


    return [1 / math.pow(2, (total_log_of_probabilities_d1 / len(text))),
            1 / math.pow(2, (total_log_of_probabilities_d2 / len(text)))]
```

```python
def calculate_trigram_perplexity(text, d1, d2):
    print('calculating trigram perplexity...')
    total_log_of_probabilities_d1 = 0
    total_log_of_probabilities_d2 = 0
    dict_text = {}
    for word in text:
        dict_text[word] = 0
    # madison total sum of log
    for i in range(len(text) - 2):
        if text[i] + ' ' + text[i+1] + ' ' + text[i+2] in d1:
            total_log_of_probabilities_d1 += math.log2(d1[text[i] + ' ' + text[i+1] + ' ' + text[i+2]])
        else:
            total_log_of_probabilities_d1 += math.log2(1 / len(d1))  #laplace smoothing

    # hamilton total sum of log
    for i in range(len(text) - 2):
        if text[i] + ' ' + text[i+1] + ' ' + text[i+2] in d2:
            total_log_of_probabilities_d2 += math.log2(d2[text[i] + ' ' + text[i+1] + ' ' + text[i+2]])
        else:
            total_log_of_probabilities_d2 += math.log2(1 / len(d2)) # laplace smoothing

    return [1 / math.pow(2, total_log_of_probabilities_d1 / len(text)),
            1 / math.pow(2, total_log_of_probabilities_d2 / len(text))]
```

1.4 Testing Selected Model on Test Set



```
calculating bigram perplexity...
[5688.915734297816, 6352.650568278936]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6928.93997899189, 7608.685808931804]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[5610.026898445119, 6499.915883680007]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[5636.881539393313, 6615.946158680587]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6472.227389354003, 7418.3580605033985]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[5821.696319764549, 6877.99171210869]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6580.8772093691705, 7586.624838420228]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[5760.226881292258, 6633.556808916583]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6417.734602297712, 7317.474469120673]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6695.092645376707, 7780.572672021654]
Prediction -> Madison
---------------------------------------------
calculating bigram perplexity...
[6573.147975687783, 7461.660603959245]
Prediction -> Madison
---------------------------------------------
```

As it seen in the last figure, selected model which is bigram language model is the best for my solution. It predicts **9 out of 11**.