Melih Berk Yılmaz

21803702

EEE 486 STATISTICAL FOUNDATIONS OF NATURAL LANGUAGE PROCESSING ASSIGNMENT 3 REPORT

INTRODUCTION

This assignment focuses on corpus preprocessing and collocation analysis. Corpus preprocessing involves transforming raw text data into a structured format, including tasks like tokenization, part-of-speech tagging, and lemmatization. The goal is to identify significant collocations within the corpus, which provide insights into linguistic patterns and thematic associations. Several filtering steps are applied, considering specific part-of-speech combinations and excluding stop words and punctuation. Only bigrams with sufficient frequency are retained. Hypothesis testing methods, such as t-test, chi-square test, and likelihood ratio test, are used to calculate scores and assess the statistical significance of collocations. These scores help rank and identify the most prominent collocations within the corpus.

RESULTS

Part 1:

From steps Part1a to Part1d, the preprocessed text from Fyodor Dostoyevsky is read, tokenized, tagged and lemmatized using the default functions of nltk library. In this part, there were no specific procedure, all the steps were followed according to the directions in the assignment description. In Part1e, the bigrams for window size 1 and 3 are created. Some of the specific counts of bigrams are in the tables pdf. Then, since there are millions of bigrams, a filtering operation is performed. Bigrams with POS tags are not “NOUN-NOUN” or “ADJ-NOUN” are eliminated, bigrams with stop words and punctuation marks are eliminated, and bigrams with frequency is lower than 10 are eliminated. This process is done for both bigrams with window size 1 and 3.

Part 2:

In this part, filtered collocation candidates are analyzed through t-test, chi-square test and likelihood ratio test to compare their scores and likelihood of being a true collocation. There are some minor remarks in this part. For the likelihood ratio test, some of the probabilities of binomial distribution were too small to calculate. Thus, a very small number $\varepsilon = 5 * 10^{-324}$ is added, which is the smallest number in Python. Another remark is that there are 3 times more bigrams for windows size 3 compared to window size 1. Therefore, frequency of word1, frequency of word2 and the total number of token is multiplied by the window size, in this case 3, to normalize and accurately calculate the probabilities. Then, the scores of each test are calculated for filtered collocation candidates and ordered by the score. Their tables can be seen in tables pdf.

Part 3a:

$$c_{w1} = window_{size} * frequency\_of\_word1$$

$$c_{w2} = window\_size * frequency\ \_of\_word2$$

$$c_{w1w2} = frequency\_of\_word1word2$$

$$N = window_{size} * total\_token\_size$$

Melih Berk Yılmaz

21803702

t-test

Assume that we have a collocation [“word1”, “word2”]. To analyze it statistically in terms of t-test, the frequencies of word1, word2 and word1word2 (in consecutive order) are needed.

Then, t-score is evaluated by the formula:

$$\bar{X} = {c_{w1w2}}/{N}, \quad \mu = {c_{w1} * c_{w2}}/{N^2}$$

$$s^2 = \bar{X}(1 - \bar{X})$$

$$t - score = \frac{\bar{X} - \mu}{\sqrt{{s^2}/{N}}}$$

Where t-score has a degrees of freedom N-1, which can be taken as infinite since N is around millions.

χ^2 Test

In this part, we have a 2x2 table to compute the values.

$O_{11} = c_{w1w2}$	$O_{12} = c_{w2} - O_{11}$
$O_{21} = c_{w1} - O_{11}$	$O_{22} = N - O_{12} - O_{21}$

$$\chi^2 - score = \frac{N(O_{11}O_{22} - O_{12}O_{21})^2}{(O_{11} + O_{12})(O_{11} + O_{21})(O_{12} + O_{22})(O_{21} + O_{22})}$$

$\chi^2 - score$ has a degrees of freedom with $(I - 1) * (J - 1) = (2 - 1) * (2 - 1) = 1$, where I and J are the number of columns and rows respectively.

Likelihood Ratio Test

$$p = c_{w2}/N, \quad p_1 = c_{w1w2}/c_{w1}, \quad p_2 = \frac{(c\_w2 - c\_w1w2)}{(N - c\_w1)}$$

$$L_{H1} = Binom(c_{w1w2}, c_{w1}, p) * Binom(c_{w2} - c_{w1w2}, N - c_{w1}, p)$$

$$L_{H2} = Binom(c_{w1w2}, c_{w1}, p_1) * Binom(c_{w2} - c_{w1w2}, N - c_{w1}, p_2)$$

$$likelihood - score = -2 * \log\left(\frac{L_{H1}}{L_{H2}}\right)$$

$likelihood - score$ has asymptotically χ^2 distribution.

Using these formulas, t-test, χ^2 test and likelihood ratio test scores are calculated for collocations [“head”, “clerk”] and [“great”, “man”]. All relevant values are calculated in the code and can be checked from the variables for each bigram.

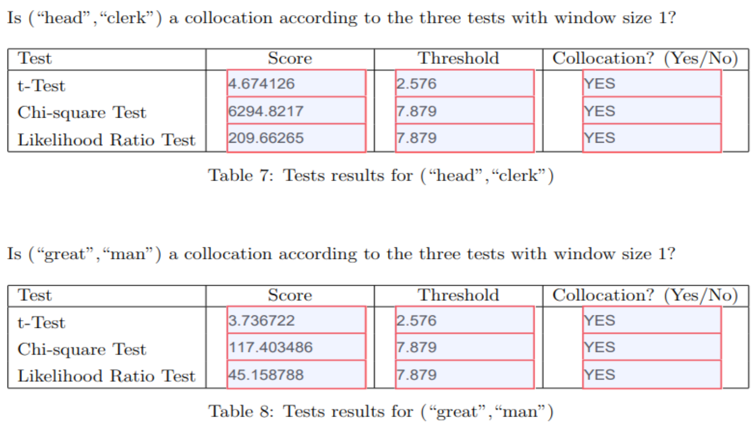
Part 3b:

The scores are calculated for each collocation and test method, which can be seen in the tables pdf. In order to decide whether a bigram is collocation or not, there should be a threshold value. In this statistical test, there are two important value determining the threshold value, significance level and degrees of freedom.

For t-test, the degrees of freedom is equivalent to N-1, which is around millions. Thus, I assumed that the df for t-test is infinity and significance level is given as 0.005. Hence, the threshold value for t-test is 2.576.

For χ^2 test, the degrees of freedom is $(I-1)*(J-1)=(2-1)*(2-1)=1$ as explained. The significance level is the same as in the t-test. Hence, the threshold value is 7.879.

For likelihood ratio test, when the final score is multiplied by -2, this distribution tends towards to χ^2 distribution asymptotically. Thus, the df and significance level are the same as in the χ^2 test. Hence, the threshold value for likelihood ratio test is 7.879. These values are obtained from t and χ^2 tables in the assignment pdf.



Is (“head”,“clerk”) a collocation according to the three tests with window size 1?

Test	Score	Threshold	Collocation? (Yes/No)
t-Test	4.674126	2.576	YES
Chi-square Test	6294.8217	7.879	YES
Likelihood Ratio Test	209.66265	7.879	YES

Table 7: Tests results for (“head”,“clerk”)

Is (“great”,“man”) a collocation according to the three tests with window size 1?

Test	Score	Threshold	Collocation? (Yes/No)
t-Test	3.736722	2.576	YES
Chi-square Test	117.403486	7.879	YES
Likelihood Ratio Test	45.158788	7.879	YES

Table 8: Tests results for (“great”,“man”)

Figure 1: Test Results for Sample Bigrams

As seen in above figure, since the t-test, χ^2 test and likelihood ratio test scores of both bigrams are higher than the threshold values, these are accepted as collocations according to the statistical tests.

CONCLUSION

In conclusion, this assignment successfully focused on corpus preprocessing and collocation analysis. The raw text data was transformed into a structured format through tokenization, part-of-speech tagging, and lemmatization, enabling the identification of significant collocations within the corpus. By applying various filtering steps, including specific part-of-speech combinations and the exclusion of stop words and punctuation, only relevant and meaningful bigrams were retained. The implementation of hypothesis testing methods, such as the t-test, chi-square test, and likelihood ratio test, allowed for the calculation of scores and the assessment of statistical significance for the identified collocations. Through this process, the most prominent collocations were ranked and identified, providing valuable insights into linguistic patterns and thematic associations. Overall, each part of the assignment was accurately completed, leading to the accomplishment of the expected results and further enhancing our understanding of the corpus.

REFERENCES

[1] A. Koç, “Lecture_6_Collocations.pdf,” 2022-2023 Spring Moodle service, (accessed May 16, 2023).

Melih Berk Yılmaz
21803702

APPENDIX

```
# -*- coding: utf-8 -*-
"""Assignment3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1pkUyrwXpUAZV6xx3WRwyNETm1toGJb7C

# **IMPORTS**
"""

!pip -q install --upgrade nltk

import nltk
import time
import numpy as np
import pandas as pd

from tqdm import tqdm
from scipy.stats import binom

from nltk.tag import pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet, stopwords

nltk.download("punkt")
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('universal_tagset')
nltk.download('averaged_perceptron_tagger')

pd.set_option('display.float_format', lambda x: f'{x:.6f}')

"""# **HELPER FUNCTIONS**"""

############# HELPER #############
class CustomLemmatizer:
    tag_dict = {"ADJ": wordnet.ADJ,
                "NOUN": wordnet.NOUN}
    lemmatizer = WordNetLemmatizer()

    def lemmatize(self, word_pos_tuple):
        word = word_pos_tuple[0]
        pos_tag = word_pos_tuple[1]
        if pos_tag in self.tag_dict:
            return self.lemmatizer.lemmatize(word,
self.tag_dict[pos_tag]).lower()
        else:
            return word.lower()

def bigram_counter(text, window_size):
    temp_list = []
    for i in tqdm(range(len(text) - 1)):
        for j in range(window_size):
            try:
                collacation = [text[i],text[i+j+1]]
                temp_list.append(collacation)
            except:
                pass
    bigram_list, bigram_counts = np.unique(temp_list, axis=0,
return_counts=True)
    bigram_dict = {' '.join(key): value for key, value in
zip(bigram_list.tolist(), bigram_counts.tolist())}
    return bigram_list.tolist(), bigram_counts.tolist(), bigram_dict

def filter_collacations(lemmatized_tokens, pos_tags, window_size,
min_frequency=10):
    stop_words = set(stopwords.words('english'))
    collocation_candidates = []
    asd = []
    for i in tqdm(range(len(lemmatized_tokens)-1)):
        for j in range(window_size):
            try:
                bigram = (lemmatized_tokens[i], lemmatized_tokens[i+j+1])
                asd.append(bigram)
                pos_tag_coll = [pos_tags[i][1], pos_tags[i+j+1][1]]
                if pos_tag_coll == ['ADJ', 'NOUN'] or pos_tag_coll == ['NOUN',
'NOUN']:
                    # Eliminate stopwords
                    if bigram[0].lower() not in stop_words and
bigram[1].lower() not in stop_words:
                        # Eliminate punctuations
                        if all(char.isalpha() for char in bigram[0]) and
all(char.isalpha() for char in bigram[1]):
                            # Eliminate bigrams with frequency less than
min_frequency
                            collocation_candidates.append(bigram)
            except:
                pass

    _,_, bigram_dict = bigram_counter(lemmatized_tokens, window_size)
    temp_col = np.unique(collocation_candidates, axis=0).tolist()
    filtered_col = []
    filtered_counts = []

    for j, col in enumerate(temp_col):
        if bigram_dict[" ".join(col)] >= min_frequency:
            filtered_col.append(col)
            filtered_counts.append(bigram_dict[" ".join(col)])

    return filtered_col, filtered_counts

def statistical_test_collacation(bigram_list_count, token_list, method,
window_size=1):
    c_w1 = []
    c_w2 = []
    c_w1w2 = []
    collacation = []
    score_list = []
    total_token = len(token_list)

    eps = 5e-324 # math.ulp(0.0)
    N = window_size * total_token

    words, counts = np.unique(token_list, return_counts=True)
    word_freq_dict = dict(zip(words.tolist(), counts.tolist()))

    for bigram, count in bigram_list_count:
        w1w2 = count
        w1 = window_size * word_freq_dict[bigram[0]]
        w2 = window_size * word_freq_dict[bigram[1]]

        if method == "t-test":
            x_bar = w1w2 / N
            mu = w1*w2 / N ** 2
            s_2 = x_bar # actual value is x_bar * (1 - x_bar), since x_bar is
very
            # close to 0, assumed that (1 - x_bar) = 1

            test_score = (x_bar - mu) / np.sqrt(s_2 / N)

        elif method == "chi-square test":
            O11 = w1w2
            O12 = w2 - O11
            O21 = w1 - O11
            O22 = N - O12 - O21

            num = N * (O11*O22 - O12*O21) ** 2
            denom = (O11 + O12) * (O11 + O21) * (O12 + O22) * (O21 + O22)

            test_score = num / denom
```

```python
        elif method == "likelihood ratio test":
            p = w2 / N
            p1 = w1w2 / w1
            p2 = (w2 - w1w2) / (N - w1)

            L_H1 = binom.pmf(w1w2, w1, p) * binom.pmf(w2-w1w2, N-w1, p)
            L_H2 = binom.pmf(w1w2, w1, p1) * binom.pmf(w2-w1w2, N-w1, p2)

            if L_H1 == 0:
                L_H1 = eps
            if L_H2 == 0:
                L_H2 = eps

            test_score = -2 * np.log(L_H1 / L_H2)

        else:
            raise ValueError(f"There is no {method}")

        c_w1.append(int(w1/window_size))
        c_w2.append(int(w2/window_size))
        c_w1w2.append(w1w2)
        collacation.append(" ".join(bigram))
        score_list.append(test_score)

    data = {
        "Bi-gram": collacation,
        f"{method[:-4]}score": score_list,
        "c(w1w2)": c_w1w2,
        "c(w1)": c_w1,
        "c(w2)": c_w2
    }

    df = pd.DataFrame(data)
    df = df.sort_values(by=f"{method[:-4]}score", ascending=False)
    df.index = pd.RangeIndex(start=1, stop=len(df) + 1, name="rank")
    return df

"""# **PART 1**"""

############# Part1 a #############
t0 = time.time()
file_path = "Fyodor Dostoyevski Processed.txt"
with open(file_path, "r") as file:
    text = file.read()
elapsed_time = time.time() - t0
print(f"Part 1a elapsed time: {elapsed_time:.4f} seconds")
```

```
############# Part1 b #############
t0 = time.time()
tokenized_text = word_tokenize(text)
elapsed_time = time.time() - t0
print(f"Part 1b elapsed time: {elapsed_time:.4f} seconds")

############# Part1 c #############
t0 = time.time()
pos_tags = pos_tag(tokenized_text, tagset="universal")
elapsed_time = time.time() - t0
print(f"Part 1c elapsed time: {elapsed_time:.4f} seconds")

############# Part1 d #############
t0 = time.time()
lemmatizer = CustomLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token_pos) for token_pos in
tqdm(pos_tags)]
elapsed_time = time.time() - t0
print(f"Part 1d elapsed time: {elapsed_time:.4f} seconds")

############# Part1 e #############
t0 = time.time()
bigrams_size_1, bigrams_size_1_counts, _ = bigram_counter(lemmatized_tokens,
window_size=1)
bigrams_size_3, bigrams_size_3_counts, _  = bigram_counter(lemmatized_tokens,
window_size=3)
elapsed_time = time.time() - t0
print(f"Part 1e elapsed time: {elapsed_time:.4f} seconds")

############# Part1 f #############
t0 = time.time()
bigrams_filtered_size_1, bigrams_filtered_size_1_counts =
filter_collacations(lemmatized_tokens, pos_tags, window_size=1,
min_frequency=10)
bigrams_filtered_size_3, bigrams_filtered_size_3_counts =
filter_collacations(lemmatized_tokens, pos_tags, window_size=3,
min_frequency=10)
elapsed_time = time.time() - t0
print(f"Part 1f elapsed time: {elapsed_time:.4f} seconds")

# Print the number of bi-grams
print(f"\nThe number of bi-grams with window_size=1 is\n\tbefore filtering:
{len(bigrams_size_1)} and after filtering: {len(bigrams_filtered_size_1)}")
print(f"\nThe number of bi-grams with window_size=3 is\n\tbefore filtering:
{len(bigrams_size_3)} and after filtering: {len(bigrams_filtered_size_3)}")

"""**Part 1 Answers**"""
```

```
words, counts = np.unique(lemmatized_tokens, return_counts=True)
word_count_dict = dict(zip(words.tolist(), counts.tolist()))

idx1 = bigrams_size_1.index(["magnificent", "capital"])
idx2 = bigrams_size_3.index(["bright", "fire"])

print("Part1b")
print(f"What is the number of tokens in the corpus? N =
{len(tokenized_text)}")
print("Part1d")
print(f"Counts of word [that]   = {word_count_dict['that']}")
print(f"Counts of word [the]    = {word_count_dict['the']}")
print(f"Counts of word [abject] = {word_count_dict['abject']}")
print(f"Counts of word [london] = {word_count_dict['london']}")
print(f"Counts of word [.]      = {word_count_dict['.']}")
print("Part1e")
print(f"'magnificent capital' occur in windows of size 1:
{bigrams_size_1_counts[idx1]}")
print(f"'bright fire' occur in windows of size 1:
{bigrams_size_3_counts[idx2]}")
print("Part1f")
try:
    idx3 = bigrams_filtered_size_1.index(["mr.", "skimpole"])
    print(f"'Mr. Skimpole' occur in windows of size 1:
{bigrams_size_3_counts[idx3]}")
except:
    print("'Mr. Skimpole' doesnt exists after filtering")
try:
    idx4 = bigrams_filtered_size_3.index(["spontaneous", "combustion"])
    print(f"'spontaneous combustion' occur in windows of size 1:
{bigrams_size_3_counts[idx4]}")
except:
    print("'spontaneous combustion' doesnt exists after filtering")

"""# **PART 2**"""

# t-test, Chi-Sqaure test and Likelihood Ratio Test for Collacation Candidates
with window_size=1
df_size_1_t_test   = statistical_test_collacation(zip(bigrams_filtered_size_1,
bigrams_filtered_size_1_counts), lemmatized_tokens, method = "t-test",
window_size = 1)
df_size_1_chi_test = statistical_test_collacation(zip(bigrams_filtered_size_1,
bigrams_filtered_size_1_counts), lemmatized_tokens, method = "chi-square
test", window_size = 1)
df_size_1_mle_test = statistical_test_collacation(zip(bigrams_filtered_size_1,
bigrams_filtered_size_1_counts), lemmatized_tokens, method = "likelihood ratio
test", window_size = 1)
```

```
# t-test, Chi-Sqaure test and Likelihood Ratio Test for Collacation Candidates
with window_size=3
df_size_3_t_test   = statistical_test_collacation(zip(bigrams_filtered_size_3,
bigrams_filtered_size_3_counts), lemmatized_tokens, method = "t-test",
window_size = 3)
df_size_3_chi_test = statistical_test_collacation(zip(bigrams_filtered_size_3,
bigrams_filtered_size_3_counts), lemmatized_tokens, method = "chi-square
test", window_size = 3)
df_size_3_mle_test = statistical_test_collacation(zip(bigrams_filtered_size_3,
bigrams_filtered_size_3_counts), lemmatized_tokens, method = "likelihood ratio
test", window_size = 3)

print("Statistical Test Analysis for Bigrams with window_size=1")
print(df_size_1_t_test.head(20))
print("\n")
print(df_size_1_chi_test.head(20))
print("\n")
print(df_size_1_mle_test.head(20))
print("\n")
print("\n")
print("Statistical Test Analysis for Bigrams with window_size=3")
print(df_size_3_t_test.head(20))
print("\n")
print(df_size_3_chi_test.head(20))
print("\n")
print(df_size_3_mle_test.head(20))

"""# **PART 3**"""

idx1 = bigrams_filtered_size_1.index(["head", "clerk"])
idx2 = bigrams_filtered_size_1.index(["great", "man"])

count1 = bigrams_filtered_size_1_counts[idx1]
count2 = bigrams_filtered_size_1_counts[idx2]

bigrams_list = [["head", "clerk"], ["great", "man"]]
bigrams_count = [count1, count2]

# t-test, Chi-Sqaure test and Likelihood Ratio Test for 2 Specific Collacation
with window_size=1
df_size_1_t_test_sub   = statistical_test_collacation(zip(bigrams_list,
bigrams_count), lemmatized_tokens, method = "t-test", window_size = 1)
df_size_1_chi_test_sub = statistical_test_collacation(zip(bigrams_list,
bigrams_count), lemmatized_tokens, method = "chi-square test", window_size =
1)
df_size_1_mle_test_sub = statistical_test_collacation(zip(bigrams_list,
bigrams_count), lemmatized_tokens, method = "likelihood ratio test",
window_size = 1)
```

```
print("Statistical Test Analysis for Subset of Bigrams with window_size=1")
print("\n")
print(df_size_1_t_test_sub)
print("\n")
print(df_size_1_chi_test_sub)
print("\n")
print(df_size_1_mle_test_sub)
```