EEE 586

Assignment 3

Pouya Ghahramanian (21804034)

May 2023

1. Part 1: Corpus Preprocessing

In the first part of the assignment, we aim to identify, analyze, and save collocations from the text corpus. I used the following steps in this part.

- Text processing: The text is read, tokenized, POS-tagged, and lemmatized using NLTK functions and a custom lemmatizer.
- Collocation finding: Bigram collocations are identified using the find-collocations function. This function uses an object of the BigramCollocationFinder from the NLTK library, and applies frequency and word filters to exclude certain bigrams.
- Dataframe creation: The get_collocations_dataframe function creates pandas dataframes for the collocations. These dataframes contain the individual words in each bigram, their frequencies, and the bigram frequencies.
- Saving results: Word frequencies are saved to a pickle file, and the dataframes are saved to CSV files.

The complete code for this part is given in the appendix as p1.py.

2. Part 2: Finding the Collocations

In this part, we conduct statistical analysis on the collocations. The complete code for this part is given in the appendix as p2.py, and it consists of the following steps:

- Loading data: First, it read two CSV files containing the collocation data and a pickle file containing the word frequencies, all of which were created in the previous part.

- Statistical tests: It defines functions to calculate the t-score, chi-square score, and likelihood ratio for each row (bigram) in the dataframes. These calculations are based on the observed and expected frequencies of the words and bigrams.
- Applying tests: It applies the test functions to each row in the dataframes, creating new columns for the results.
- Saving results: It saves the updated dataframes with the score columns to new CSV files, collocations1_scores.csv and collocations3_scores.csv, one for each window size.
- Printing top scores: Finally, it sorts the dataframes by each score in turn, and prints the top 20 bigrams for each score and window size. The results for each test and window size are available in the answer sheet.

3. Part 3: Explaining the Statistical Tests

Part (a) I obtained the scores for all bigrams in part 2 and saved them to a pandas dataframe (collocations1_scores.csv and collocations3_scores.csv for two window sizes). Then I obtained the scores for the given bigrams by locating them in the pandas dataframe.

- t-score: The t-score is calculated as follows:

  $t = (O11 - E11)/sqrt(O11)$

  where O11 is the observed frequency of the bigram and E11 is the expected frequency, calculated as:

  $E11 = (R1 \times C1)/N$

  R1 is the frequency of the first word in the corpus, C1 is the frequency of the second word, and N is the total number of bigrams.

- Chi-square score: The chi-square score is calculated using the formula:

  $X^2 = N * (O11O22 - O12O21)^2/((O11 + O12) * (O11 + O21) * (O12 + O22) * (O21 + O22))$

  O12 is the frequency of the first word without the second, O21 is the frequency of the second word without the first, and O22 is the frequency of neither the first word nor the second word occurring.

- Log-likelihood score: The log-likelihood score is calculated as:

  $G^2 = 2 \times [(O11 \times log(O11/E11)) + (O12 \times log(O12/E12)) + (O21 \times log(O21/E21)) + (O22 \times log(O22/E22))]$

  Here, E12, E21, and E22 are the expected frequencies for O12, O21, and O22 respectively, calculated in a similar manner to E11.

Part (b)
We use the following procedure for each test to decide whether the bigrams are collocations or not. The obtained scores and threshold values are given in the answer sheet.

- For the t-score, we compare the calculated t-score with the critical t-value for $= 0.005$ in the t-distribution table. If the absolute value of the t-score is greater than the critical value, we reject the null hypothesis that the words occur together by chance, and consider the bigram to be a collocation.

- For the chi-square test, we compare the calculated chi-square value with the critical chi-square value for $= 0.005$ in the chi-square distribution table. If the chi-square value is greater than the critical value, we reject the null hypothesis and consider the bigram to be a collocation.

- For the log-likelihood test, we compare 2 times the calculated log-likelihood score with the critical chi-square value for $= 0.005$ in the chi-square distribution table (because $2G^2$ follows a chi-square distribution). If $2G^2$ is greater than the critical value, we reject the null hypothesis and consider the bigram to be a collocation.

The degrees of freedom (DF) for the chi-square and log-likelihood tests in this case is 1, as we are considering bigrams (two-word combinations). The degrees of freedom for the t-test would be N-1 where N is the number of observations, which in this case would be quite large; therefore, we can use the standard normal distribution as an approximation.

4. Appendix

The complete code for the assignment is given in the appendix.

Listing 1: p1.py

```
import numpy as np
import pandas as pd
import string
import nltk
from nltk import pos_tag
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist
import pickle

# Read Text Corpus
addr = 'Student Release/Fyodor Dostoyevski Processed.txt'
with open(addr, 'r') as file:
    text = file.read()

# Download Sources from NLTK
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('wordnet')
# nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Tokenize the corpus
tokens = nltk.word_tokenize(text)

# POS Tagging with NLTK
pos_tags = pos_tag(tokens, tagset='universal')

# Define Costum Lemmatizer
tag_dict = {'ADJ': wordnet.ADJ, 'NOUN': wordnet.NOUN, 'VERB': wordnet.VERB}
lemmatizer_wn = WordNetLemmatizer()
def lemmatize(token):
    word, tag = token[0], token[1]
    if tag in tag_dict: return lemmatizer_wn.lemmatize(word, tag_dict[tag])
    return word.lower()

# Lemmatize the tokens
tokens_lemmatized = [lemmatize(t) for t in pos_tags]

# Define a function to find collocations from lemmatized tokens (ds) and a window size
```

```
def find_collocations(ds, window_size = 1):
    # Find collocations with given window size
    finder = BigramCollocationFinder.from_words(ds, window_size)
    # Eliminate bigrams that occur less than 10 times
    finder.apply_freq_filter(10)
    # Eliminate bigrams that include stopwords or including any punctuation marks
    finder.apply_word_filter(lambda w: w in stop_words or not w.isalpha())
    # Eliminate all bigrams except those with POS tags NOUN-NOUN or ADJ-NOUN.
    finder.apply_ngram_filter(lambda w1, w2: (pos_tag([w1])[0][1], pos_tag([w2])[0][1])
    result = finder.ngram_fd.items()
    # result_10 = finder.nbest(BigramAssocMeasures.likelihood_ratio, 10)
    return result

# Define filters to find bigrams with
collocations1 = find_collocations(tokens_lemmatized, window_size = 2)
collocations3 = find_collocations(tokens_lemmatized, window_size = 4)

# Calculate word frequencies
word_freqs = FreqDist(tokens_lemmatized)

# Save word frequencies to a file
with open('word_frequencies.pkl', 'wb') as f:
    pickle.dump(word_freqs, f)

def get_collocations_dataframe(collocations, word_freqs):
    # Build DataFrame
    df = pd.DataFrame(columns=["Word1", "Word2", "Word1_Freq", "Word2_Freq", "Bigram_Fre

    # Fill DataFrame
    for bigram, bigram_freq in collocations:
        word1_freq = word_freqs[bigram[0]]
        word2_freq = word_freqs[bigram[1]]
        df = df.append({
            "Word1": bigram[0],
            "Word2": bigram[1],
            "Word1_Freq": word1_freq,
            "Word2_Freq": word2_freq,
            "Bigram_Freq": bigram_freq
        }, ignore_index=True)

    return df

df1 = get_collocations_dataframe(collocations1, word_freqs)
df3 = get_collocations_dataframe(collocations3, word_freqs)

# Save the DataFrames to disk
df1.to_csv('collocations1.csv', index=False)
df3.to_csv('collocations3.csv', index=False)
```

```
print("Collocations for window size 1:")
print(df1.head())
print("\nCollocations for window size 3:")
print(df3.head())
```

Listing 2: p1_answers.py

```
import numpy as np
import pandas as pd
import pickle

# Read the dataframes from the saved CSV files in part 1 without eliminating below 10 oc
df1 = pd.read_csv('collocations1_all.csv')
df3 = pd.read_csv('collocations3_all.csv')

# PART 1.b
# Load word frequencies from disk
with open('word_frequencies.pkl', 'rb') as f: word_freqs = pickle.load(f)
# Get total number of words
N = word_freqs.N()
print('----------------------------------------------------------------')
print('Total number of words in the corpus: {}'.format(N))
print('----------------------------------------------------------------')

# Helper methods to answer part 1 questions
def find_word_freq(freq_dic, word):
    freq = freq_dic[word]
    return freq

def find_bigram_freq(df, word1, word2):
    row = df.loc[(df['Word1'] == word1) & (df['Word2'] == word2)]
    if row.empty: return None
    return row['Bigram_Freq'].values[0]

# PART 1.d
words = ['that', 'the', 'abject', 'london', '.']
for word in words:
    print('Word: {} - Frequency: {}'.format(word, find_word_freq(word_freqs, word)))

word1 = 'magnificent'
word2 = 'capital'
print('Word1: {} - Word2: {} - Frequency: {}'.format(word1, word2, find_bigram_freq(df1,

word1 = 'bright'
word2 = 'fire'
print('Word1: {} - Word2: {} - Frequency: {}'.format(word1, word2, find_bigram_freq(df3,

word1 = 'mr.'
word2 = 'skimpole'
```

```
print('Word1: {} - Word2: {} - Frequency: {}'.format(word1, word2, find_bigram_freq(df1,

word1 = 'spontaneous'
word2 = 'combustion'
print('Word1: {} - Word2: {} - Frequency: {}'.format(word1, word2, find_bigram_freq(df3,
```

Listing 3: p2.py

```
import numpy as np
import pandas as pd
from nltk import FreqDist
from nltk.collocations import BigramCollocationFinder, BigramAssocMeasures
import pickle

# Read the dataframes from the saved CSV files in part 1
df1 = pd.read_csv('collocations1.csv')
df3 = pd.read_csv('collocations3.csv')

# Load word frequencies from disk
with open('word_frequencies.pkl', 'rb') as f: word_freqs = pickle.load(f)
# Get total number of words
N = word_freqs.N()

df1 = df1.sort_values(by='Bigram_Freq', ascending=False)
df3 = df3.sort_values(by='Bigram_Freq', ascending=False)

print('-----------------------------------------------------------------')
print('\tWindow Size 1 Frequencies')
print('-----------------------------------------------------------------')
print(df1.head(20))
print('-----------------------------------------------------------------')
print("\tWindow Size 3 Frequencies")
print('-----------------------------------------------------------------')
print(df3.head(20))

window_size = 1

def t_score(row):
    word1_freq = row['Word1_Freq']
    word2_freq = row['Word2_Freq']
    bigram_freq = row['Bigram_Freq']
    # expected_freq = (word1_freq * word2_freq) / (window_size * N)
    expected_freq = (word1_freq * word2_freq) / total_words
    t_score = (bigram_freq - expected_freq) / np.sqrt(bigram_freq)
    return t_score

def chi_square(row):
    word1_freq = row['Word1_Freq']
    word2_freq = row['Word2_Freq']
```

```
    bigram_freq = row['Bigram_Freq']
    # expected_freq = (word1_freq * word2_freq) / (window_size * N)
    expected_freq = (word1_freq * word2_freq) / total_words
    # return window_size * N * (bigram_freq - expected_freq)**2 / (word1_freq * word2_fr
    return total_words * (bigram_freq - expected_freq)**2 / (word1_freq * word2_freq)

from scipy.stats import binom

def likelihood_ratio(row):
    # Constants
    # SMALL_PROB = 1e-10
    SMALL_PROB = np.finfo(float).eps

    # Calculate probabilities
    p1 = max(row['Bigram_Freq'] / row['Word1_Freq'], SMALL_PROB)
    # p2 = max(row['Word2_Freq'] / row['Bigram_Freq'], SMALL_PROB)
    # p2 = max(row['Word2_Freq'] / (window_size * N), SMALL_PROB)
    p2 = max(row['Word2_Freq'] / total_words, SMALL_PROB)

    # Calculate likelihoods for the null and alternative hypotheses
    L_null = binom.pmf(row['Bigram_Freq'], row['Word1_Freq'], p2)
    L_alt = binom.pmf(row['Bigram_Freq'], row['Word1_Freq'], p1)

    # Avoid division by zero
    if L_null == 0:
        L_null = SMALL_PROB

    # Calculate and return likelihood ratio test statistic
    return -2 * np.log(L_null / L_alt)

# from scipy import stats

# Add t-score, chi-square, and likelihood ratio columns to the DataFrames
window_size = 1
total_words = df1['Word1_Freq'].sum() + df1['Word2_Freq'].sum()
df1['t_score'] = df1.apply(t_score, axis=1)
df1['chi_square'] = df1.apply(chi_square, axis=1)
df1['likelihood_ratio'] = df1.apply(likelihood_ratio, axis=1)

window_size = 3
total_words = df3['Word1_Freq'].sum() + df3['Word2_Freq'].sum()
df3['t_score'] = df3.apply(t_score, axis=1)
df3['chi_square'] = df3.apply(chi_square, axis=1)
df3['likelihood_ratio'] = df3.apply(likelihood_ratio, axis=1)

# Convert scientific numbers format to numbers
df1['t_score'] = df1['t_score'].astype(float)
df1['chi_square'] = df1['chi_square'].astype(float)
df1['likelihood_ratio'] = df1['likelihood_ratio'].astype(float)
```

```
df3['t_score'] = df3['t_score'].astype(float)
df3['chi_square'] = df3['chi_square'].astype(float)
df3['likelihood_ratio'] = df3['likelihood_ratio'].astype(float)

pd.set_option('display.float_format', '{:.5f}'.format)

df1.to_csv('collocations1_scores.csv', index=False)
df3.to_csv('collocations3_scores.csv', index=False)

# Sort by scores and print the top 20 candidates
print('-----------------------------------------------------------')
print('\tWindow Size 1 - top 20 t-scores')
print('-----------------------------------------------------------')
print(df1.sort_values('t_score', ascending=False).head(20))
print('-----------------------------------------------------------')
print('\tWindow Size 1 - top 20 chi-square')
print('-----------------------------------------------------------')
print(df1.sort_values('chi_square', ascending=False).head(20))
print('-----------------------------------------------------------')
print('\tWindow Size 1 - top 20 likelihood-ratio')
print('-----------------------------------------------------------')
print(df1.sort_values('likelihood_ratio', ascending=False).head(20))

print('-----------------------------------------------------------')
print('\tWindow Size 3 - top 20 t-scores')
print('-----------------------------------------------------------')
print(df3.sort_values('t_score', ascending=False).head(20))
print('-----------------------------------------------------------')
print('\tWindow Size 3 - top 20 chi-square')
print('-----------------------------------------------------------')
print(df3.sort_values('chi_square', ascending=False).head(20))
print('-----------------------------------------------------------')
print('\tWindow Size 3 - top 20 likelihood-ratio')
print('-----------------------------------------------------------')
print(df3.sort_values('likelihood_ratio', ascending=False).head(20))
```

Listing 4: p3.py

```
import numpy as np
import pandas as pd
import pickle

df1 = pd.read_csv('collocations1.csv')
df3 = pd.read_csv('collocations3.csv')

df1_scores = pd.read_csv('collocations1_scores.csv')

word1, word2 = 'head', 'clerk'
```

```
# Call the function for a specific bigram
print('-------------------------------------------------------------')
print('\tScores for words: {} and {}'.format(word1, word2))
print('-------------------------------------------------------------')
print(df1_scores.loc[(df1_scores['Word1'] == word1) & (df1_scores['Word2'] == word2)])

word1, word2 = 'great', 'man'
print('-------------------------------------------------------------')
print('\tScores for words: {} and {}'.format(word1, word2))
print('-------------------------------------------------------------')
print(df1_scores.loc[(df1_scores['Word1'] == word1) & (df1_scores['Word2'] == word2)])
```