EEE 486/586 Statistical Foundations of Natural Language Processing Assignment 3

Mehmet Rıfat Özkurt, *B.Sc., 22003187, Bilkent University Electrical & Electronics Engineering Department*

I. Corpus Preprocessing

A. About the Corpus

The corpus at hand is a concatenated version of six novels by Fyodor Dostoevsky, which are: *Crime and Punishment, Devils, Notes from the Underground, The Brothers Karamazov, The Gambler and The Idiot.* This corpus was imported as a string into the Python environment to be preprocessed. Tokenization, lemmatization and POS-tagging were applied to the corpus as the initial preprocessing methods.

B. Tokenization

Tokenization in NLP is the decomposition of a text into smaller units of meaning in order to perform further computations. In this case, tokenization refers to extracting each word of the sequence into an ordered list. For this purpose, Python's NLTK (Natural Language Toolkit) library was used. This library offers many methods to tokenize a corpus, from which word_tokenize() was chosen. This function splits the text based on whitespaces and punctuation marks. While it may leave some unwanted half words and orphan characters behind, it is sufficient for our purposes and the unwanted characters were hopefully removed during stop word removal. More complex methods include PunktTokenizer and TreebankWordTokenizer. For other purposes, sentence tokenization and Tweet tokenization also exist.

C. POS Tagging

POS tagging refers to the detection of what part of speech each token is within the context of the sentence. It is a more complex task than tokenization, since one token type may serve as different parts of speech depending on its position in the sequence and the meaning. If the syntactic role of each token is ignored and a rule-based approach is employed, most of the results are likely to be wrong. Python's NLTK library uses pre-trained language models to label each token, taking into consideration the semantic roles. The pos_tag() function was used along with the tagset 'universal', which is a set of labels designated to apply across many languages. It is a more general representation of the parts of speech, compared to more complex set such as Penn Treebank Target. Although the labeling was correct for most of the tokens, there were some very obvious mistakes in the resultant data, as well.

D. Lemmatization

Lemmatization is an even more complex task that aims to eliminate different inflections of tokens and reduce all tokens to their simplest, canonical forms called "lemma". It differs from stemming through its more intricate approach and generation of full words at the output, as opposed to incomplete words generated during stemming. For this assignment, NLTK's WordNetLemmatizer class utilizes WordNet to do this lemmatization. This WordNet includes the synonyms, semantically-related words and lemmas for each word. As a result, a set of lemmas were obtained. The resultant dataset was still full of mistakes and punctuation marks, which were removed later.

E. Creating Bigrams

Bigrams are essentially a sequence of two tokens that occur within a known proximity in the sequence. The maximum value of this proximity is called the window size. Bigrams provide statistical information about the co-occurrence of pairs of tokens in a sequence. Depending on the window size, each token's co-occurrence probability with every other token is represented in this information. In this task, bigrams are used to analyze collocations, which refer to words that frequently occur together, with window sizes of 1 and 3. These collocations are important in the analysis of a language because they reveal certain patterns about the usage of words in a language and goes beyond the meanings of single words.

The generation of bigrams were done manually, using nested for loops. The returned output is a list of tuples, containing the word pairs. These bigrams will be simplified further according to different criteria given in the task, to obtain a smaller and more specific dataset for collocation analysis.

F. Filtering Bigrams

The bigrams obtained in Part E were further filtered in this section. Bigrams with POS tags NOUN-NOUN and NOUN-ADJ were kept, were the rest was discarded. Note that the ordering of the POS tags is also important, since the semantic relationships of the collocations is also reflected by the order of these tags. Then, stop-word removal was performed on the set, which effectively cleared most of the unwanted tokens generated during tokenization, such as single 's' characters, half-words, along with stop words in the sequences, which do not carry a significant information on their own for our purposes. The list of stop words was provided as is, and no further modifications were made. Then, all bigrams containing non-alphabetic characters were eliminated. This prevents the generation of the same bigrams multiple times, due to the existence of punctuation marks. Finally, a frequency list of all bigram types were made and the bigrams that occur less than 10 times were eliminated. Bigram lists before and after this final elimination were saved for further use.

II. Finding Collocations

A. Student's t-test

Student's t-test is a statistical method of hypothesis testing that evaluates the significance of the difference between the means of two different datasets. It is widely used for collocation analysis, where the observed frequency and the expected frequency of collocations are compared statistically. The expected frequency value is obtained by assuming that the two tokens are statistically independent and occur in a bigram randomly. This random occurrence represents the null hypothesis in out hypothesis testing method. The acceptance of null hypothesis would mean that no collocations exist between the two tokens in question, and the bigram is a result of a random incident. The second hypothesis indicates the existence of a strong dependence between the two words. Mathematically, the general formula for t-test is shown below:

$$t\_test = \frac{\bar{X} - \mu}{\sqrt{\frac{S^2}{N}}} \tag{1}$$

Where for our application, the formula transforms into:

$$t_{test} = \frac{MLE - H_0}{\sqrt{\frac{H_0}{N}}} \tag{2}$$

Where:

$$MLE = \frac{C(word1, word2)}{N} = occurrence\ rate\ of\ the\ bigram \tag{3}$$

$$H_0 = \frac{C(word1) \cdot C(word2)}{N^2} = occurrence\ of\ null\ hypothesis \tag{4}$$

$$N = total\ number\ of\ bigrams \tag{5}$$

B. Pearson's Chi-square test

The Chi-Square Test is a statistical method used to evaluate the significance between two variables. It differs from t-test because the variables in chi-square test are categorical. Similar to t-test, its null hypothesis assumes that the two variables are independent and the bigrams occur as random incidents. Its main formula refers to a contingency table that classifies the occurrence frequencies of the tokens in a given bigram. A critical value is used to determine the existence of a significant relationship between the two tokens. The contingence table for a bigram model and the formula are provided below:

TABLE I
CONTINGENCE TABLE FOR THE OCCURRENCE OF THE TOKENS IN THE BIGRAM. THE EXPRESSIONS IN THE PARENTHESES ARE USED IN THE FORMULA BELOW TO CALCULATE THE CHI-SQUARE TEST RESULT. THE EXPRESSIONS ARE IN THE FORM OF TOTAL OCCURRENCES OF THE DESCRIBED SCENARIOS IN ALL BIGRAMS.

W1 –W2 (O11)	not W1 –W2 (O12)
W1- not W2 (O21)	not W1- not W2 (O22)

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

C. Likelihood-Ratio Test

The likelihood-ratio test is another hypothesis test that compares two models in terms of their fitness to the observed data. In our collocation analysis, it determines whether the occurrence of a collocation is higher than its expected value so as to indicate a potential strong relationship between the tokens. Similar to the previous models, a critical value is set to determine the significance of the relationship. The independence hypothesis is the simplest model to reflect the occurrence of tokens, and the test looks for more complex models, indicating dependence between the tokens. The Common formulae are given below:

(6) $$p = \frac{c_2}{N}, p_1 = \frac{c_2-c_{12}}{N-c_1},\ p_2 = \frac{c_2}{N}$$

Where c1, c2, c12 are defined in a contingence table.

(7) $$L(H0) = b(c_{12};\ c_1, p)b(c_2 - c_{12}; N - c_1, p)$$

(8) $$L(H1) = b(c_{12};\ c_1, p1)b(c_2 - c_{12}; N - c_1, p2)$$

In the equations above, b() is the binomial distribution with parameters n, k, and probability of successful trials p. In order to reduce computational load, the likelihood ratio operation is carried out in the log domain, as shown. The '-2' factor is used for a normalization that enables this test to be treated similarly to chi-square test, especially in the selection of critical values:

(9) $$-2\log(\lambda) = \frac{\log(LH(0))}{\log(LH(1))}$$

Another computational issue is the potential generation of divide by zero or infinity problems. In such cases, the numbers are converted to very small or very large positive numbers as approximation.

III. Explaining the Statistical Tests

The first question of Part 3 is explained with equations in the previous sections. Detailed information about the implementation of these algorithms are provided and the path to obtain these scores for any given bigram are shown. More detailed implementation can be seen within the code in the appendix.

In the second part, the threshold value of significance is chosen to be 0.005, and two bigrams are evaluated with respect to the given tables and their corresponding test scores outputted by the algorithms.

The output result of the t-test for the bigrams 'head clerk' and 'great man' are observed to be 4.6782 and 3.8061, respectively. In the table, it is seen that the required minimum score for an alpha variable of 0.005 is 2.576, which is below both of the real scores. Therefore, both samples are collocations given the alpha criterion.

The output of Chi-Squared test for 'head clerk' and 'great man' are 5082.7515 and 1.8302, respectively. The required minimum score from the table is observed to be 7.879, given the alpha value. Therefore, 'head clerk' is a collocation while 'great man' is not, according to this test and the alpha value.

The output of Likelihood Ratio Test test for 'head clerk' and 'great man' are 61.2090 and 56.0788, respectively. The required minimum score from the table is observed to be 7.879, given the alpha value. Therefore, both 'head clerk' and 'great man' are collocations, according to this test and the alpha value.

References

[1] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Stanford, CA, USA: Thomson Higher Education, 2020. [Online]. Available: https://web.stanford.edu/jurafsky/slp3/. [Accessed: May 4, 2023].

APPENDIX

```
[2]  # -*- coding: utf-8 -*-
[3]  """
[4]  Created on Tue May 16 12:29:32 2023
[5]
[6]  @author: rozku
[7]  """
[8]
[9]  #%% Tokenization, POS Tagging, Lemmatization
[10]
[11] import nltk
[12] from scipy.stats import binom
[13] from collections import Counter
[14] from math import log
[15] import numpy as np
[16]
[17] # nltk.download('popular')
[18] # nltk.download('averaged_perceptron_tagger')
[19] # nltk.download('universal_tagset')
[20] # nltk.download('wordnet')
[21]
[22] from nltk.tokenize import word_tokenize
[23] from nltk import pos_tag
[24] #from nltk.corpus import wordnet
[25] import custom_lemmatizer
[26]
[27]
[28]
[29] file = open(r"C:\Users\rozku\Desktop\Student Release\Fyodor Dostoyevski Processed.txt", "r", encoding="utf-8")
[30] corpus = file.read()
[31] file.close()
[32]
[33] tokenized = word_tokenize(corpus)
[34] tagged = pos_tag(tokenized, tagset='universal')
[35]
[36] cm = custom_lemmatizer.custom_lemmatizer()
[37] lemmatized = []
[38] for token in tagged:
[39]     lemmatized.append((cm.lemmatize(token), token[1]))
[40]
[41] #%% Creating Bigrams -- returns bigrams of window size 1 and 3, in the form of a list of tuples
[42]
[43] def create_bigrams(tokens, window_size):
[44]     bigrams = []
[45]     for i in range(0, len(tokens)):
[46]         for j in range(1, window_size + 1):
[47]             if i+j < len(tokens):
[48]                 bigrams.append((tokens[i], tokens[i + j]))
[49]     return bigrams
[50]
[51] bigram_ws1 = create_bigrams(lemmatized, 1)
[52] bigram_ws3 = create_bigrams(lemmatized, 3)
[53]
[54] #%% Calculating the frequency of bigrams in a dictionary
[55]
[56] def create_freq_dict(bigramlist):
[57]     freq_dict = {}
[58]     for bigram in bigramlist:
[59]         if bigram in freq_dict:
[60]             freq_dict[bigram] += 1
[61]         else:
[62]             freq_dict[bigram] = 1
[63]     return freq_dict
[64]
[65] bigram_freq_dict_ws1 = create_freq_dict(bigram_ws1)
[66] bigram_freq_dict_ws3 = create_freq_dict(bigram_ws3)
[67]
[68] #%% Filtering bigrams according to POS tags, stop words, frequency and punctuations
[69]
[70] def create_filtered_dict(freq_dict):
[71]     file = open(r"C:\Users\rozku\Desktop\Student Release\stopwords.txt", "r", encoding = "utf-8")
[72]     stopwords = file.read().split("\n")
[73]     file.close()
[74]
```

```
[75]     filtered_dict = {}
[76]     filtered_dict_before_cutoff = {}
[77]     for key in freq_dict:
[78]         if (key[0][0].isalpha()) and (key[1][0].isalpha()) and (key[1][1] == 'NOUN') and ((key[0][1] == 'NOUN') or (key[0][1] == 'ADJ')) and (key[0][0] not in stopwords) and (key[1][0] not in stopwords):
[79]             if (freq_dict[key] >= 10):
[80]                 if (key[0][0], key[1][0]) not in filtered_dict:
[81]                     filtered_dict[(key[0][0], key[1][0])] = freq_dict[key]
[82]                 elif (key[0][0], key[1][0]) in filtered_dict:
[83]                     filtered_dict[(key[0][0], key[1][0])] += freq_dict[key]
[84]                 if (key[0][0], key[1][0]) not in filtered_dict_before_cutoff:
[85]                     filtered_dict_before_cutoff[(key[0][0], key[1][0])] = freq_dict[key]
[86]                 elif (key[0][0], key[1][0]) in filtered_dict_before_cutoff:
[87]                     filtered_dict_before_cutoff[(key[0][0], key[1][0])] += freq_dict[key]
[88]             else:
[89]                 if (key[0][0], key[1][0]) not in filtered_dict_before_cutoff:
[90]                     filtered_dict_before_cutoff[(key[0][0], key[1][0])] = freq_dict[key]
[91]                 elif (key[0][0], key[1][0]) in filtered_dict_before_cutoff:
[92]                     filtered_dict_before_cutoff[(key[0][0], key[1][0])] += freq_dict[key]
[93]
[94]     return filtered_dict, filtered_dict_before_cutoff
[95]
[96]  bigram_freq_filtered_dict_ws1, bigram_freq_filtered_dict_before_cutoff_ws1 = create_filtered_dict(bigram_freq_dict_ws1)
[97]  bigram_freq_filtered_dict_ws3, bigram_freq_filtered_dict_before_cutoff_ws3 = create_filtered_dict(bigram_freq_dict_ws3)
[98]
[99]
[100] #%% Student's t-test results
[101]
[102] def t_tester(tokens, bigram_freq, filename):
[103]     token_freq_dict = Counter(tokens)
[104]     n = len(tokens)
[105]
[106]     t_test = {}
[107]     for bigram, freq in bigram_freq.items():
[108]
[109]         null = token_freq_dict[bigram[0]] * token_freq_dict[bigram[1]] / n**2
[110]         mle = freq / n
[111]         sample_mean = mle
[112]         t_test[bigram] = (mle - null) / (sample_mean /n)**0.5
[113]
[114]     t_test = sorted(t_test.items(), key=lambda x: x[1], reverse=True)
[115]     #write the first 20 bigrams to a file
[116]     with open(filename, "w", encoding='utf-8') as file:
[117]         for bigram, score in t_test[:20]:
[118]             file.write(f"Bigram: {bigram}, Score: {score}, Bigram Count: {bigram_freq[bigram]}, Word Counts: {token_freq_dict[bigram[0]], token_freq_dict[bigram[1]]}\n")
[119]     return t_test
[120]
[121] t_test_ws1 = t_tester(tokenized, bigram_freq_filtered_dict_ws1, "t_test_ws1.txt")
[122] t_test_ws3 = t_tester(tokenized, bigram_freq_filtered_dict_ws3, "t_test_ws3.txt")
[123]
[124] #%% Pearson's Chi-Square Test results
[125]
[126] def chi_square_tester(tokens, bigrams, bigrams_before_cutoff, filename):
[127]     token_freq_dict = Counter(tokens)
[128]     chi_square_test = {}
[129]
[130]     for key in bigrams:
[131]         n = sum(bigrams_before_cutoff.values())
[132]         o11 = bigrams[key]
[133]         o22 = n - o11
[134]         o21 = 0
[135]         o12 = 0
[136]
[137]         for key2 in bigrams_before_cutoff:
[138]             if (key[0] == key2[0]) and (key[1] != key2[1]):
[139]                 o12 += bigrams_before_cutoff[key2]
[140]             elif (key[0] != key2[0]) and (key[1] == key2[1]):
[141]                 o21 += bigrams_before_cutoff[key2]
[142]         chi2 = n * ((o11*o22 - o12*o21) ** 2) / ((o11 + o12)*(o11 + o21)*(o12 + o22)*(o21 + o22))
[143]
[144]         chi_square_test[key] = (chi2, o11,o12, o21, o22)
[145]
[146]     chi_square_test = sorted(chi_square_test.items(), key=lambda x: x[1], reverse=True)
[147]
```

```
    with open(filename, "w", encoding='utf-8') as file:
        for bigram, score in chi_square_test[:20]:
            file.write(f"Bigram: {bigram}, Score: {score}, Bigram Count: {bigrams[bigram]}, Word Counts: {token_freq_dict[bigram[0]], token_freq_dict[bigram[1]]}\n")
    return chi_square_test

chi_square_test_ws1 = chi_square_tester(tokenized, bigram_freq_filtered_dict_ws1, bigram_freq_filtered_dict_before_cutoff_ws1, "chi2_test_ws1.txt")
chi_square_test_ws3 = chi_square_tester(tokenized, bigram_freq_filtered_dict_ws3, bigram_freq_filtered_dict_before_cutoff_ws3, "chi2_test_ws3.txt")

#%% Likelihood Ratio Test results

def likelihood_ratio_tester(tokens, bigrams, bigrams_before_cutoff, filename):
    likelihood_test = {}
    token_freq_dict = Counter(tokens)
    n = sum(bigrams_before_cutoff.values())

    for key in bigrams:
        c12 = bigrams[key]
        c1 = token_freq_dict[key[0]]
        c2 = token_freq_dict[key[1]]
        p0 = c2 / n
        ph21 = c12 / c1
        ph22 = (c2- c12) / (n-c1)

        likelihood_1 = binom.pmf(c12, c1, p0) * binom.pmf(c2-c12, n-c1, p0)
        likelihood_2 = binom.pmf(c12, c1, ph21) * binom.pmf(c2-c12, n-c1, ph22)

        if likelihood_1 == 0:
            likelihood_ratio = 10**-10
        elif likelihood_2 == 0:
            likelihood_ratio = 10**10
        else:
            likelihood_ratio = likelihood_1 / likelihood_2

        likelihood_ratio_result = -2 * np.log(likelihood_ratio)
        likelihood_test[key] = (likelihood_ratio_result, c12, c1, c2)

        likelihood_test_sorted = sorted(likelihood_test.items(), key=lambda x: x[1], reverse=True)

        with open(filename, "w", encoding='utf-8') as file:
            for bigram, score in likelihood_test_sorted[:20]:
                file.write(f"Bigram: {bigram}, Score: {score}, Bigram Count: {bigrams[bigram]}, Word Counts: {token_freq_dict[bigram[0]], token_freq_dict[bigram[1]]}\n")
    return likelihood_test_sorted

likelihood_test_ws1 = likelihood_ratio_tester(tokenized, bigram_freq_filtered_dict_ws1, bigram_freq_filtered_dict_before_cutoff_ws1, "likelihood_ratio_tester_ws1.txt")
likelihood_test_ws3 = likelihood_ratio_tester(tokenized, bigram_freq_filtered_dict_ws3, bigram_freq_filtered_dict_before_cutoff_ws3, "likelihood_ratio_tester_ws3.txt")
```