**Цель работы:** проведение семантической предобработки текстов с  
использованием библиотек языка Python и метода TF-IDF.  
**Задачи:**

1. написать программу для предобработки текста
2. Создать биграммы и триграммы слов, встречающихся в тексте.
3. Постройте словарь, корпус и TFIDF модель обработанного текста.
4. Повторить операцию для выбранного текста, с минимальным объемом в 15 страниц.

**1 Написать программу для предобработки текста**

Была написана программа на Python, убирающая из текста все знаки препинания, так же удаляющая все цифры и приводящая весь текст в нижний регистр.

Листинг 1 – код для предобработки текста

|  |
| --- |
| public string RefactorThis()  {  try  {  text = text.ToLower();  text = text.Replace("\r", " ");  text = text.Replace("\n", " ");  text = Regex.Replace(text, @"s{2,}", " ").Trim();  var result = new StringBuilder();  foreach (char ch in text)  {  if (!charsToDelete.Contains(ch))  {  result.Append(ch);  }  }  return result.ToString();  }  catch  {  Exception ex = new Exception("Ошибка при удалении симвлов.");  return ex.Message;  }  } |

**2 Создать биграммы и триграммы из слов, встречающихся в тексте**

Для обнаружения часто встречающихся пар слов (биграмм) мы используем модель Phrases из библиотеки gensim.models.phrases. Параметр min\_count=3 указывает, что биграмма должна встречаться как минимум 3 раза в тексте, чтобы быть включенной в результат.

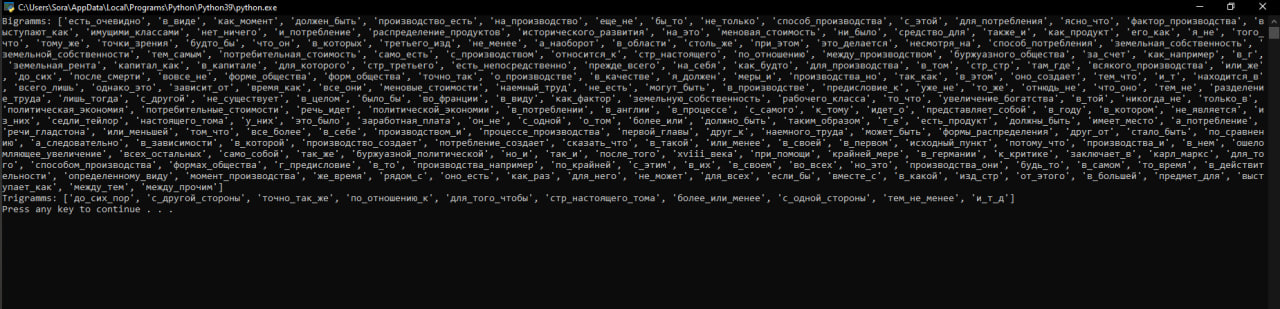
Параметр threshold=10 определяет чувствительность при объединении слов в биграммы: чем выше значение, тем более значительными должны быть взаимосвязи между словами для их объединения.

Аналогично, для создания триграмм мы используем биграммы в качестве основы.

Листинг 2 – код программы для создания биграмм и триграмм.

|  |
| --- |
| import gensim  def read\_from\_file(file\_path):  with open(file\_path, 'r', encoding='utf-8') as file:  texts = [line.strip().split() for line in file.readlines()]  return texts  file\_path = 'result.txt'  texts = read\_from\_file(file\_path)  bigr = gensim.models.phrases.Phrases(texts, min\_count=3, threshold=3)  words\_underscore = []  for word in bigr[texts[0]]:  if '\_' in word:  words\_underscore.append(word)  res = list(set(words\_underscore))  print("Bigramms:", res)  trigr = gensim.models.phrases.Phrases(bigr[texts], threshold=2)  words\_underscore = []  for word in trigr[bigr[texts[0]]]:  if word.count('\_') == 2:  words\_underscore.append(word)  res = list(set(words\_underscore))  print("Trigramms:", res) |

На рисунке 1 видны биграммы и триграммы для текста на 27 страниц, поскольку в тексте из лабораторной работы 1 они не нашлись.

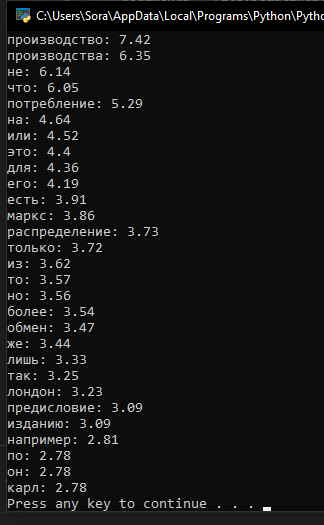


**3 Построить словарь, корпус, TFIDF модель текста**

Листинг 3 – Код для создания словаря, корпуса, модели

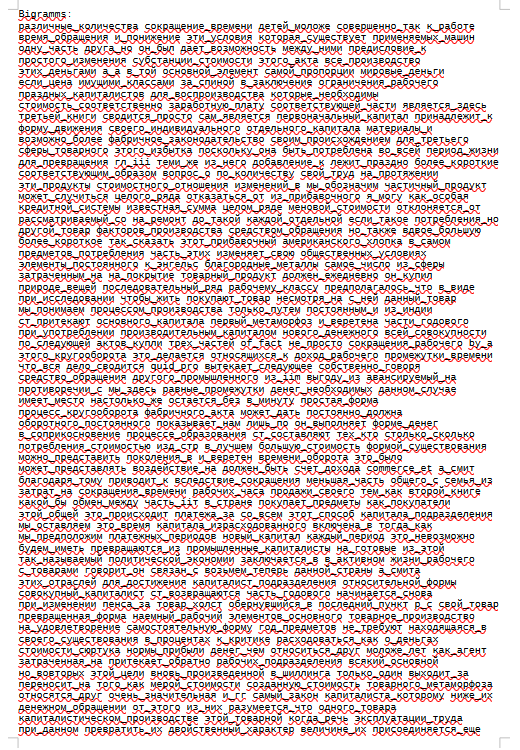
|  |
| --- |
| import gensim  from gensim import models, corpora  from gensim.utils import simple\_preprocess  import numpy as np  with open('result.txt', 'r', encoding='utf-8') as file:  documents = file.readlines()  mydict = corpora.Dictionary([simple\_preprocess(line) for line in documents])  corpus = [mydict.doc2bow(simple\_preprocess(line)) for line in documents]  tfidf = models.TfidfModel(corpus)  tfidf\_corpus = tfidf[corpus]  all\_tfidf\_weights = []  for doc in tfidf\_corpus:  all\_tfidf\_weights.extend([[mydict[id], freq] for id, freq in doc])  word\_weights = {}  for word, weight in all\_tfidf\_weights:  if word in word\_weights:  word\_weights[word] += weight  else:  word\_weights[word] = weight  sorted\_word\_weights = sorted(word\_weights.items(), key=lambda x: x[1], reverse=True)[:30]  for word, weight in sorted\_word\_weights:  print(f"{word}: {np.around(weight, decimals=2)}") |

Результат работы программы представлен на рисунке 2:

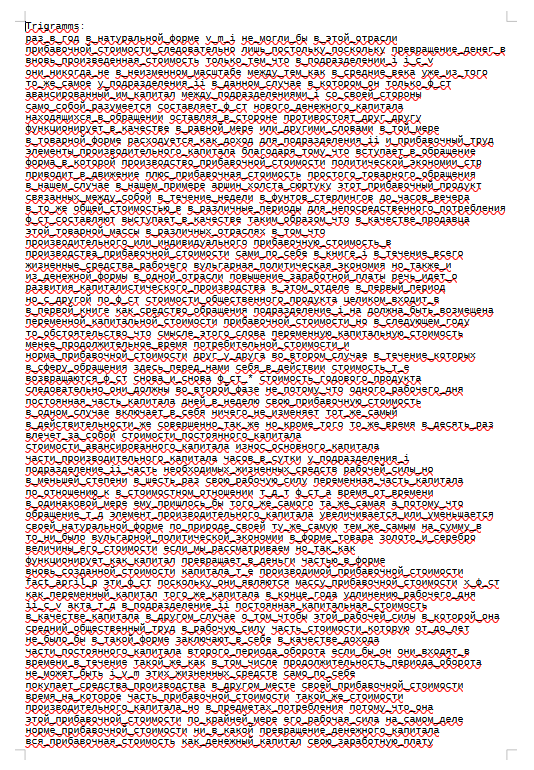


**4 Повторить операцию для выбранного текста с минимальным объемом 15 страниц**Для данного задания был взят текст на 27 страниц , поэтому я повторил тесты для полного текста 1 тома капитала.

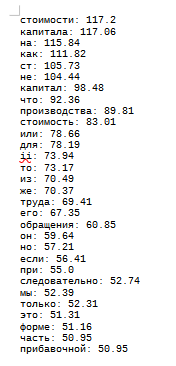
Для Биграмм (1 из 20 листов):



Для Триграмм (1 из 4 листов):

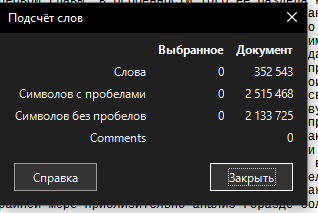


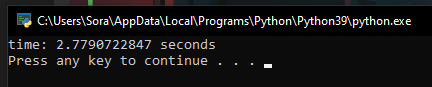
Словарь, корпус и TFIDF модель всего текста:



Для определения времени работы программы использовалась библиотека Time.

|  |
| --- |
| import gensim  from gensim import models, corpora  from gensim.utils import simple\_preprocess  import numpy as np  import time # Импортируем модуль time  # Начинаем отсчет времени  start\_time = time.time()  with open('result.txt', 'r', encoding='utf-8') as file:  documents = file.readlines()  mydict = corpora.Dictionary([simple\_preprocess(line) for line in documents])  corpus = [mydict.doc2bow(simple\_preprocess(line)) for line in documents]  tfidf = models.TfidfModel(corpus)  tfidf\_corpus = tfidf[corpus]  all\_tfidf\_weights = []  for doc in tfidf\_corpus:  all\_tfidf\_weights.extend([[mydict[id], freq] for id, freq in doc])  word\_weights = {}  for word, weight in all\_tfidf\_weights:  if word in word\_weights:  word\_weights[word] += weight  else:  word\_weights[word] = weight  sorted\_word\_weights = sorted(word\_weights.items(), key=lambda x: x[1], reverse=True)[:30]  with open('output.txt', 'w') as f:  for word, weight in sorted\_word\_weights:  f.write(f"{word}: {np.around(weight, decimals=2)}\n")  # Завершаем отсчет времени  end\_time = time.time()  execution\_time = end\_time - start\_time  print(f"time: {execution\_time:.10f} seconds") |





**Вывод:** Были освоены библиотеки Gensim, spacy, time. С их помощью был изучен метод TD-IDF для малых и больших объемов текста.