МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
“КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра математических методов кибернетической безопасности

КУРСОВАЯ РАБОТА

Дисциплина: «Интеллектуальные методы обработки информации»

Направление подготовки: 8.04030101 «Прикладная математика»

Тема: «Интеллектуальные методы обработки информации»

Выполнил студент группы ФИ-51м

Кригин Валерий Михайлович

Проверил:

Ландэ Дмитрий Владимирович

_____
(подпись)

Оценка:

_____

Киев 2015

ОГЛАВЛЕНИЕ

РЕФЕРАТ

При интеллектуальном анализе текстовых данных важно понимать взаимосвязь между словами внутри текста. Также необходимо учесть, что на важность таких связей влияют ещё и связанные документы: публикации одного автора, сборники литературы одной эпохи, цикл учебников по определённой дисциплине и пр.

В данной работе были проверены эмпирические законы Ципфа и Хипса, которые определяют зависимости частот слов от размера текста. Далее была использована одна из мер значимости слов и словосочетаний в наборе текстов — $TF - IDF$.

TF-IDF, ЗАКОН ЦИПФА, ЗАКОН ХИПСА, ГРАФ, ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ

ABSTRACT

Connections between words within text are needed to know, when analysing text data. Also it's needed to consider, that value of these connections are affected by related documents: books of the same author, literature of one epoch, guides for similar study etc.

Zipf's law and Heaps' law were covered by this work — they show correlation between words' frequencies and text size. $TF - IDF$ was analyzed, as a measure of words value in set of texts.

TF-IDF, ZIPF'S LAW, HEAPS' LAW, GRAPH, DATA MINING, TEXT ANALYSIS

1 ЗАКОН ЦИПФА

1.1 Закон Ципфа

Отношение ранга слова R, то есть его номер в списке слов, отсортированных по частоте в порядке убывания, к частоте слова f, является постоянным [1]

$$Z = R \cdot f,$$

где f — частота слова в тексте, а Z — коэффициент Ципфа. Значит,

$$f = \frac{Z}{R}.$$

1.2 Задание

Под понятием “отфильтровать текст” тут и далее будут подразумеваться следующие действия:

1) очистить текст от всех символов кроме букв и пробелов;
2) буквы привести в нижний регистр, между словами оставить по одному пробелу.
3)

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) составить частотный словарь слов — каждому слову текста сопоставить количество его повторений в тексте;
4) отсортировать частоты в порядке убывания;

5) изобразить полученные значения на графике, выбрав логарифмический масштаб для оси ординат и абсцисс;
6) построить степенную линию тренда и убедиться, что график похож на прямую линию, за исключением, возможно, “хвостов” с обеих концов.

1.3 Фильтр

На Perl написан фильтр, который

1) делает заглавные буквы строчными;
2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 1.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

1.4 Частотный словарь

На Python написан скрипт, который составляет частотный словарь и выводит его в формате csv. Полученный результат можно открыть в программе для работы с электронными таблицами для построения графиков.

Вход считывается из stdin, выход происходит в stdout.

Листинг 1.2 — counter.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from sys import stdin
5  from os import linesep
6
7  words = ' '.join([l.strip() for l in stdin]).split(' ')
8
9  counts = {}
10 for key in set(words):
11     counts[key] = 0
12 for w in words:
13     counts[w] += 1
14
15 result = sorted(counts.iteritems(),key=lambda x: x[1],
16                  reverse=True)
17
18 print linesep.join('%s,%d'%r for r in result)
```

1.5 График

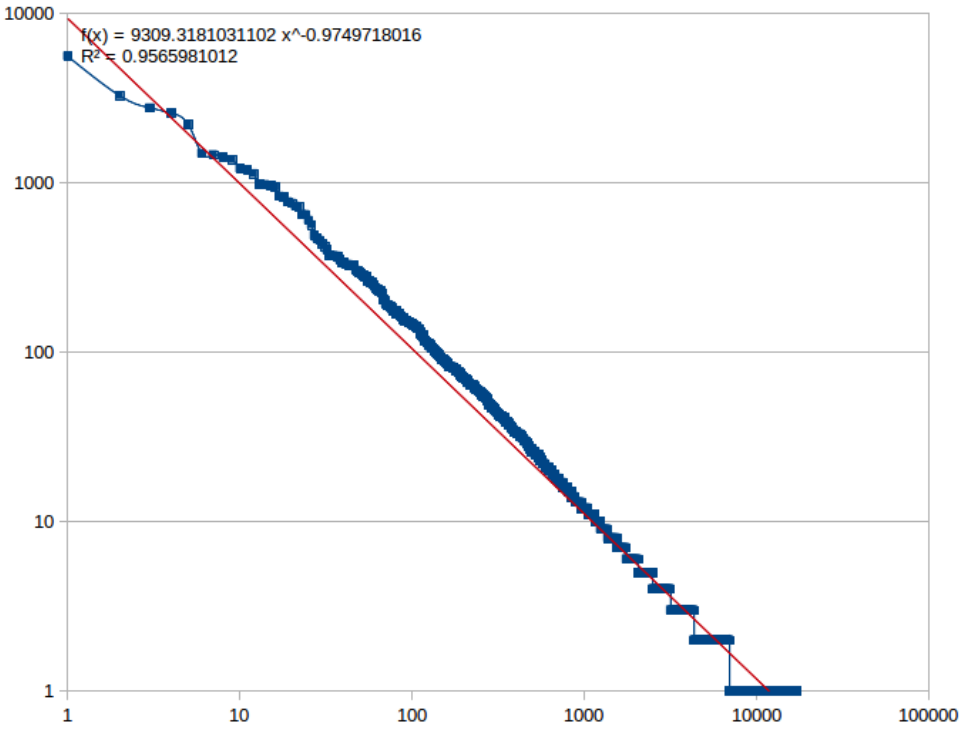

Рисунок 1.1 — Результат

2 ЗАКОН ХИПСА

2.1 Закон Хипса

Объём словаря уникальных слов $\nu(n)$ для текста длиной n связан с длиной текста следующим соотношением [2]

$$\nu(n) = \alpha \cdot n^{\beta},$$

где α и β — эмпирические константы, которые разнятся от языка к языку, и для европейских языков колеблятся в пределах от 10 до 100 и от 0.4 до 0.6 соответственно.

2.2 Задание

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) построить зависимость количества уникальных слов в тексте от его размера; для этого достаточно использовать один и тот же текст, изымать из него всё больше и больше слов с каждой итерацией, и подсчитывать число уникальных слов на каждом шаге;
4) изобразить полученные значения на графике;
5) построить степенную линию тренда и убедиться, что полученные параметры α и β близки к теоретическим значениям.

2.3 Фильтр

На Perl написан фильтр, который

1) делает заглавные буквы строчными;
2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 2.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

2.4 Частотный словарь

На Python написан скрипт, который считает зависимость между объёмом текста и объёмом словаря уникальных слов и выводит его в формате csv. Полученный результат можно открыть в программе для работы с электронными таблицами для построения графиков.

Листинг 2.2 — counter.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from sys import stdin
5
6  words = ' '.join([l.strip() for l in stdin]).split(' ')
7
8  found = []
9
10 for i, w in enumerate(words):
11     if w not in found:
12         found.append(w)
13     print '%d,%d'%(i+1, len(found))
```

2.5 График

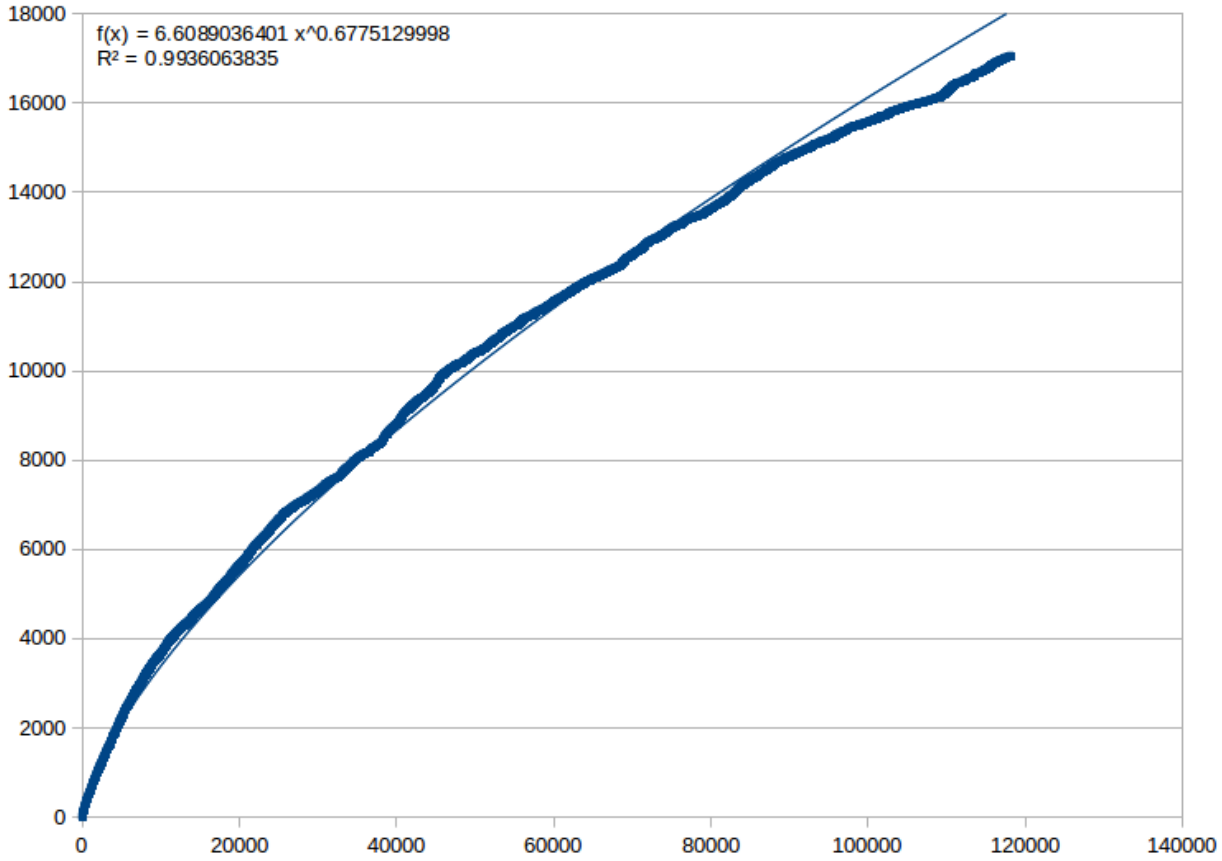

Рисунок 2.1 — Результат

3 $TF - IDF$

3.1 $TF - IDF$

Для i слова (n-граммы) индексы TF и IDF считаются по следующим формулам, где D — множество документов, n_k — количество повторений k слова (n-граммы) в текущем документе [3]

$$TF_i = \frac{n_i}{\sum_k n_k},$$
$$IDF_i = \log \frac{|D|}{|\{d \mid t_i \in d \in D\}|}.$$

Сам индекс $TF - IDF$ является произведением индексов TF и IDF

$$TF - IDF_i = TF_i \cdot IDF_i$$

3.2 Задание

3.2.1 Основное задание

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) подсчитать $TF - IDF$ для каждого слова;
4) изобразить полученные результаты в виде таблицы, отсортировав по значению $TF - IDF$ в порядке убывания.

То же самое нужно проделать с биграммами и триадами слов. Например,

в тексте “мама мыла раму” биграммы следующие: “мама мыла” и “мыла раму”.

3.2.2 Стоп-слова (шумовые слова)

Стоп-слова — те слова, которые не несут смысловую нагрузку. К ним относятся предлоги, частицы и прочее, если анализируемый документ не является учебником русского языка.

Список стоп-слов можно найти в интернете. Например, в разделе 12.9.4 Full-Text Stopwords документации к MySQL 5.5 [4] находится список англоязычных шумовых слов.

Для увеличения скорости и уменьшения объёма обрабатываемых данных

1) при подсчёте $TF - IDF$ для слов можно выбросить из рассмотрения те, которые находятся в списке стоп-слов; например, слово “не” имеет мало смысла в сказке о царе Салтане, чего не скажешь о слове “лебедь”;
2) при подсчёте $TF - IDF$ для биграмм следует исключать те биграммы, которые содержат в себе шумовые слова; например, биграмма “я пришёл” имеет мало смысловой нагрузки, но биграмма “пришёл домой” скажет больше;
3) при подсчёте $TF - IDF$ для триад следует исключать те элементы, которые оканчиваются или начинаются на шумовые слова; скажем, “и она решила” мало о чём говорит, триада “она решила пойти” скажет больше, но “решила пойти домой” несёт определённый смысл.

3.3 Фильтр

На Perl написан фильтр, который

1) делает заглавные буквы строчными;

2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 3.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

3.4 Счётчик $TF - IDF$

На Python написан скрипт, который считает $TF-IDF$ для слов и выводит их в формате csv.

Полученный результат можно открыть в программе для работы с электронными таблицами для сортировки и фильтрации.

Листинг 3.2 — counter.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from sys import stdin, argv
```

```
5  from os import linesep
6  from math import log
7  from stoplist import stop_list
8
9
10 def get_count(words):
11     tfs = {}
12     for key in set(words):
13         tfs[key] = 0
14     for w in words:
15         tfs[w] += 1
16     return tfs
17
18 def group_n_grams(words, n):
19     if n < 2:
20         return [w for w in words if w not in stop_list]
21     return ['␣'.join(w for w in words[i:i+n])
22             for i in range(len(words)-n)
23             if words[i+n-1] not in stop_list
24                and words[i] not in stop_list]
25
26 if __name__ == '__main__':
27     n_grams_length = 1
28
29     if len(argv) > 1:
30         n_grams_length = int(argv[1])
31
```

```
32      texts = ([l.strip().split('␣') for l in stdin])
33      names = map(lambda text: text[0], texts)
34      texts = map(lambda text: group_n_grams(text[1:],
35                                             n_grams_length), texts)
36
37
38      tfs = map(get_count, texts)
39
40
41      idf = {}
42      for word in set(sum(texts, [])):
43          idf[word] = 0
44
45
46      for tf in tfs:
47          for word in tf:
48              idf[word] += 1
49
50
51      logN = log(len(texts))
52      for word in idf:
53          idf[word] = logN - log(idf[word])
54
55
56      tf_idfs = []
57      for i, tf in enumerate(tfs):
58          tf_idfs.append({})
```

```
59          for word in tf:
60              tf_idfs[i][word] = tf[word] * idf[word] / len(tf)
61
62      result = [(names[i], word, tf_idf[word])
63                for i, tf_idf in enumerate(tf_idfs)
64                for word in tf_idf]
65      result = sorted(result, key=lambda x: x[2], reverse=True)
66      print linesep.join('%s,%s,%f'%(r) for r in result)
```

3.5 Результат

На 3.1 изображены первые 18 строк таблицы со значениями $TF - IDF$ для слов из 144 документов автора Льва Николаевича Толстого, 27 документов Фёдора Михайловича Достоевского и 31 документа Александра Сергеевича Пушкина, отсортированных по значению $TF - IDF$ в порядке убывания.

Объём документов Толстого 18MB, Достоевского 7.6MB, Пушкина — 2.8MB. Фильтрация происходит соответственно 10.3, 3.2 и 2 секунды. Далее каждый документ имеет только один перенос строки, который говорит об окончании документа, и их можно объединить в один файл. Подсчёт $TF - IDF$ происходит за 6.5 секунд, на выходе получается *.csv* файл объёмом 39MB.

На 3.2 изображены первые 18 строк таблицы с биграммами, а на 3.3 изображены первые 18 строк таблицы с триадами.

№	Книга	Слово	$TF-IDF$
1	TolstoiVorobei	воробей	0.910196
2	TolstoiEchizayac	ёж	0.723855
3	TolstoiVorobei	лён	0.717333
4	TolstoiTelenoknaldu	телёнок	0.649992
5	TolstoiLetuchayamysh	летучая	0.645765
6	PushkinKamennyigost	гуан	0.625634
7	TolstoiVolgaiVazuza	волга	0.616940
8	TolstoiFilipok	филипок	0.603935
9	TolstoiShatiDon	шат	0.591983
10	TolstoiShakalyislon	слон	0.570469
11	TolstoiVolgaiVazuza	вазуза	0.570408
12	TolstoiPesnyaprosrachenienarekeChernoi	bis	0.523350
13	TolstoiZaicyilyagushki	зайцы	0.517134
14	TolstoiLetuchayamysh	мышь	0.507136
15	PushkinKamennyigost	дон	0.471297
16	TolstoiMyshi	кота	0.467739
17	TolstoiShatiDon	дон	0.454054
18	TolstoiSobakaieeten	собака	0.441651

Таблица 3.1 — Результат для слов

№	Книга	Биграмма	$TF - IDF$
1	TolstoiLetuchayamysh	летучая мышь	2.051165
2	PushkinKamennyigost	дон гуан	0.803516
3	TolstoiMyshi	кота спастись	0.558765
4	PushkinKamennyigost	дона анна	0.461864
5	TolstoiSobakaieeten	бросила своё	0.408328
6	TolstoiSobakaieeten	своё мясо	0.408328
7	TolstoiSobakaieeten	тень собака	0.408328
8	TolstoiSobakaieeten	своё волною	0.408328
9	TolstoiSobakaieeten	мясо несёт	0.408328
10	TolstoiShatiDon	шат иваныч	0.407217
11	TolstoiShatiDon	дон иваныч	0.407217
12	TolstoiVolk	ай ай	0.372557
13	TolstoiVorobei	лён воробей	0.366087
14	TolstoiSobakaieeten	зубах несла	0.355009
15	TolstoiSobakaieeten	волною унесло	0.355009
16	TolstoiSobakaieeten	собака шла	0.355009
17	TolstoiSobakaieeten	собака мясо	0.355009
18	TolstoiSobakaieeten	кинулась отнимать	0.355009

Таблица 3.2 — Результат для биграмм

№	Книга	Триада	$TF-IDF$
1	TolstoiOtecisynovya	отец и сыновья	0.865335
2	TolstoiMyshi	коту на шею	0.663533
3	TolstoiZaicyilyagushki	зайцы и лягушки	0.629335
4	TolstoiTelenoknaldu	телёнок на льду	0.497650
5	TolstoiSobakaieeten	своё волною унесло	0.408328
6	TolstoiSobakaieeten	бросила своё мясо	0.408328
7	TolstoiSobakaieeten	несёт она бросила	0.408328
8	TolstoiSobakaieeten	тень собака шла	0.408328
9	TolstoiSobakaieeten	собака мясо несёт	0.408328
10	TolstoiVorobei	птицы не послушались	0.393205
11	TolstoiShakalyislon	шакалы и слон	0.384593
12	TolstoiSobakaieeten	унесло и осталась	0.355009
13	TolstoiSobakaieeten	собаки того мяса	0.355009
14	TolstoiSobakaieeten	воде и подумала	0.355009
15	TolstoiSobakaieeten	зубах несла мясо	0.355009
16	TolstoiSobakaieeten	мясо и кинулась	0.355009
17	TolstoiSobakaieeten	дощечке через речку	0.355009
18	TolstoiSobakaieeten	несла мясо увидала	0.355009

Таблица 3.3 — Результат для триад

4 ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ СЕТИ СЛОВ

4.1 Задание

Изобразить граф, отображающий взаимосвязи между словами, биграммами и триадами. Привести его матрицу весов.

4.2 Прорисовка графа

Связи строились следующим образом: от слов шли дуги к биграммам, в которые они входят, а от биграмм — к триадам, в которые входят эти биграммы. Было взято 10 самых значимых триад из произведения “Каменный гость” Александра Сергеевича Пушкина, биграммы, которые в них входят, и слова, которые входят в эти биграммы и не попадают в чёрный список. Вес дужки — индекс $TF - IDF$ “родителя”. То есть, вес дужки между словом и биграммой — $TF - IDF$ биграммы, вес дужки между биграммой и триадой — $TF - IDF$ триады.

Поскольку между словами и триадами дужек нет, для уменьшения объёма таблицы с матрицей весов было решено построить две таблицы: слова и биграммы (табл. 4.1), биграммы и триады (табл. 4.2).

Для визуализации графа была использована библиотека $graph-tool$. Прорисовка выполнялась с помощью иерархического разделения блоков [5]. Результат можно увидеть на рисунке 4.1. Чем жирнее вершина, тем больше у неё $TF - IDF$.

Ниже приведён код программы визуализации.

Листинг 4.1 — draw.py

```
1 from graph_tool.all import *
```

```
2 from words import data as words
3 from bigrams import data as bigrams
4 from trigrams import data as trigrams
5 from math import log10, log
6 from sys import argv
7 from os import linesep
8
9
10 def clear_entries(entries, containers, treshold=0):
11     non_needed_entries = set(entries.keys())
12     non_needed_containers = set(containers.keys())
13     for entry in entries:
14         if entries[entry] <= treshold:
15             continue
16         exists = False
17         for container in containers:
18             if entry in container:
19                 if container in non_needed_containers:
20                     non_needed_containers.remove(container)
21                 non_needed_entries.remove(entry)
22                 break
23     for container in non_needed_containers:
24         del containers[container]
25     for entry in non_needed_entries:
26         del entries[entry]
27
28
```

```
29 def get_vertex(g, name, word, vertices):
30     # If we already have this vertex, just use it from cache
31     if name in vertices:
32         return vertices[name]
33     # Otherwise we have to create new one
34     a = g.add_vertex()
35     word[a] = name
36     # Add the vertex to cache
37     vertices[name] = a
38     return a
39
40
41 def build_graph(g, entries, containers,
42                 weight, word, color, vertices,
43                 treshold=0, last_word=-1, width_scale = 3.0,
44                 entry_color='red', container_color='red'):
45     min_tfidf = min(containers.values())
46     def add_to_container(entry, container):
47         a = get_vertex(g, entry, word, vertices)
48         color[a] = entry_color
49         b = get_vertex(g, container, word, vertices)
50         color[b] = container_color
51         e = g.add_edge(a, b)
52         # Set weight for new edge (tf-idf)
53         # Just empirical formula
54         raw_weight = containers[container]/min_tfidf
55         weight[e] = log(log(raw_weight)+1) * width_scale + 1
```

```
56      for entry in entries:
57          if entries[entry] < treshold:
58              continue
59          for container in containers:
60              if entry in container:
61                  add_to_container(entry, container)
62      return
63
64
65  def get_matrix(entries, containers):
66      result = {}
67      for container in containers:
68          result[container] = dict((entry, entries[entry] if entry
69                                    for entry in entries)
70      return result
71
72
73  def draw_matrix(matrix, entries):
74      keys = entries.keys()
75      result = ''
76      result += ','.join(['container'] + keys) + linesep
77      for container, line in matrix.items():
78          result += ','.join([container] + [str(line[key]) for key
79      return result
80
81
82  def init_graph():
```

```
83      # Create directed graph
84      g = Graph(directed=True)
85      # Create 'weight' property for edge:
86      # will contain tf-idf
87      weight = g.new_edge_property('float')
88      # Create 'word' property for vertex:
89      # will contain string with current word
90      word = g.new_vertex_property('string')
91      color = g.new_vertex_property('string')
92      return g, weight, word, color, {}
93
94
95  if __name__ == '__main__':
96      img_name='output.png'
97      if len(argv) > 2:
98          if argv[1] in ['-w', '--word']:
99              words = dict(words.items()[:int(argv[2])])
100             clear_entries(words, bigrams)
101         elif argv[1] in ['-b', '--bigram']:
102             bigrams = dict(bigrams.items()[:int(argv[2])])
103             clear_entries(bigrams, trigrams)
104         elif argv[1] in ['-t', '--triad']:
105             trigrams = dict(trigrams.items()[:int(argv[2])])
106             clear_entries(bigrams, trigrams)
107     if len(argv) > 3:
108         img_name = argv[3]
109     g, weight, word, color, vertices = init_graph()
```

```
110     # Dictionary with existent vertices (cache)
111     clear_entries(words, bigrams)
112     clear_entries(bigrams, trigrams)
113     print draw_matrix(get_matrix(words, bigrams), words)
114     print draw_matrix(get_matrix(bigrams, trigrams), bigrams)
115     build_graph(g, entries=words, containers=bigrams,
116                 color=color, weight=weight, word=word,
117                 vertices=vertices,
118                 entry_color='red', container_color='blue')
119     build_graph(g, entries=bigrams, containers=trigrams,
120                 color=color, weight=weight, word=word,
121                 vertices=vertices,
122                 entry_color='blue', container_color='purple')
123     # Draw the graph;
124     # Weight is responsible for edges widths
125     # Word contains labels for vertices
126     # graph_draw(g, vertex_font_size=10, edge_pen_width=weight,
127     #             vertex_text=word, vertex_fill_color=color,
128     #             vertex_text_position=0, output=img_name,
129     #             output_size=(300, 500))
130     # Alternative
131     # graph_draw(g, edge_pen_width=weight, vertex_text=word,
132     #             node_first=True, vertex_text_position=0,
133     #             vertex_size=20, vertex_shape='double_square')
134     # Or even
135     state=minimize_nested_blockmodel_dl(g)
136     draw_hierarchy(state, vertex_text=word,
```

```
137                 vertex_text_position=1, edge_pen_width=weight,
138                 vertex_fill_color=color,
139                 output=img_name, output_size=(800, 600))
```

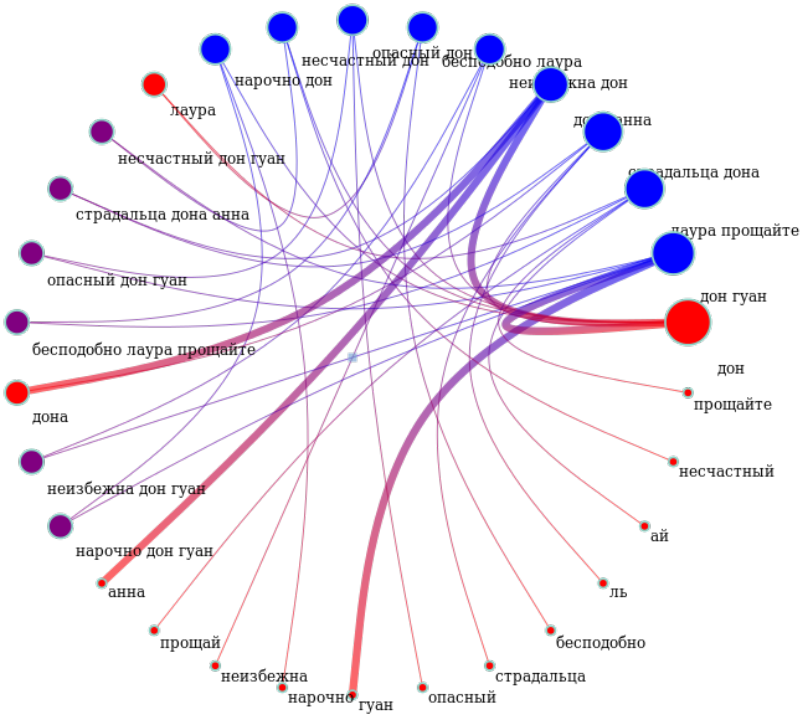

Рисунок 4.1 — triads

	анна	прощай	неизбежна	лаура	гуан	дон	несчастный	дона	бесподобно	страдальца	опасный	прощайте	нарочно
несчастный дон	0	0	0	0	0	0.471297	0.002683	0	0	0	0	0	0
опасный дон	0	0	0	0	0	0.471297	0	0	0	0	0.006097	0	0
дон гуан	0	0	0	0	0.625634	0.471297	0	0	0	0	0	0	0
бесподобно лаура	0	0	0	0.192471	0	0	0	0	0.007633	0	0	0	0
дона анна	0.143801	0	0	0	0	0.471297	0	0.222531	0	0	0	0	0
неизбежна дон	0	0	0.004184	0	0	0.471297	0	0	0	0	0	0	0
страдальца дона	0	0	0	0	0	0.471297	0	0.222531	0	0.003048	0	0	0
нарочно дон	0	0	0	0	0	0.471297	0	0	0	0	0	0	0.001116
лаура прощайте	0	0.003127	0	0.192471	0	0	0	0	0	0	0	0.003458	0

Таблица 4.1 — Матрица весов графа слов и биграмм

	неизбежна дон	лаура прощайте	опасный дон	дона анна	несчастный дон	нарочно дон	страдальца дона	дон гуан	бесподобно лаура
страдальца дона анна	0	0	0	0.461864	0	0	0.006327	0	0
несчастный дон гуан	0	0	0	0	0.006327	0	0	0.803516	0
бесподобно лаура прощайте	0	0.006327	0	0	0	0	0	0	0.006327
нарочно дон гуан	0	0	0	0	0	0.006327	0	0.803516	0
неизбежна дон гуан	0.006327	0	0	0	0	0	0	0.803516	0
опасный дон гуан	0	0	0.006327	0	0	0	0	0.803516	0

Таблица 4.2 — Матрица весов графа биграмм и триад

ВЫВОДЫ

В работе была проанализирована русскоязычная классика: Пушкин, Толстой и Достоевский. Эмпирические законы Ципфа и Хипса подтвердились. $TF - IDF$, хоть и не является хорошей мерой, дал хорошие результаты.

Полученная в итоге сеть слов была изображена в виде графа, построенного с помощью метода иерархического разделения блоков (Hierarchical Block Partition). Метод $TF - IDF$ проявил себя как простой и рабочий, а визуализация графов посредством библиотеки $graph - tool$ для $Python$ — удобная и простая в использовании утилита, позволяющая получить наглядную картину.

СПИСОК ЛИТЕРАТУРЫ

1. Pearson, Karl. Jean-Baptiste Estoup and the origins of Zipf's law / Karl Pearson // *Bolet'in de Estad'istica e Investigaci'on Operativa.* — 2014. — Vol. 30, no. 1. — Pp. 66–67.

2. Ландэ, Д.В. Интернетика: навигация в сложных сетях : модели и алгоритмы / Д.В. Ландэ, А.А. Снарский, И.В. Безсуднов. — УРСС, 2009. `https://books.google.com.ua/books?id=P_l_kgAACAAJ`.

3. Jones, Karen Sparck. A statistical interpretation of term specificity and its application in retrieval / Karen Sparck Jones // *Journal of Documentation.*

4. Oracle Corporation. Full-Text Stopwords. — `https://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html`. — 2015. — Online; accessed 11 December 2015.

5. Holten, D. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data / D. Holten // *Visualization and Computer Graphics, IEEE Transactions.* — 2006. — Vol. 12, no. 5. — Pp. 741–748.