Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 01.03.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А.А. Замышляева
«____»_____20___ г.

Разработка и исследование алгоритмов информационного поиска по тематической модели с применением эволюционного подхода

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2019.55.ПЗ ВКР

Руководитель работы, доцент
_____/Т.Ю. Оленчикова
«____»_____2019 г.

Автор работы
Студент группы ЕТ-412
_____/ Н.Л. Квитченко
«____»_____2019 г.

Нормоконтролер, ассистент
_____/Н.С. Мидоночева
«____»_____2019 г.

Челябинск
2019

АННОТАЦИЯ

Квитченко Н.Л. Разработка и исследование алгоритмов информационного поиска по тематической модели с применением эволюционного подхода. – Челябинск: ЮУрГУ, ЕТ-412, 54 с., 7 ил., библиогр. список – 20 наим., 1 прил.

В работе исследуются методы построения тематических моделей и алгоритмы реализации разведочного информационного поиска. Решается задача повышения релевантности поисковой выдачи за счет учета предпочтений пользователя. В качестве решения задачи предлагается применение эволюционного подхода. Целью работы является разработка алгоритмов информационного поиска с применением эволюционного подхода.

Написана программа для построения тематической модели на основе подхода аддитивной регуляризации, разработан программный модуль, реализующий построенный алгоритм поиска и учета предпочтений пользователя. На примере конкретных запросов проведено экспериментальное исследование описанного метода.

Разработанный алгоритм позволяет расширить область поиска и вывести пользователю релевантные документы, даже если его запрос находится рядом с интересуемой областью. Программный модуль, реализующий разработанный алгоритм, может быть подключен к другим задачам и проектам.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С приходом компьютеров и сети Интернет мы можем найти любую нужную нам информацию, просто зайдя в поисковую систему. Но количество информации растет, классифицировать ее становится все труднее, и требуется найти новый способ обработки информации. Одним из возможных выходов является тематическое моделирование.

Суть тематического моделирования заключается в следующем: на входе коллекция документов, мы знаем, какие слова или термины встречаются в каждом документе, а на выходе мы хотим получить информацию о том, как документы раскладываются по темам и как каждая тема представлена в словаре терминов.

Данная задача похожа на задачу кластеризации документов, где каждый документ целиком соотносится с одним конкретным кластером. Но в тематической модели осуществляется мягкая кластеризация, позволяющая разделять документ между несколькими кластерами. Таким образом, обходится проблема синонимии и полисемии слов [1].

Концепция тематического моделирования применяется в следующих приложениях: многоязычный информационный поиск, поиск тематических сообществ (в социальных сетях), обнаружение текстового спама, анализ изображений и видеопотоков.

Особенно важно применение тематического моделирования к задаче реализации системы «разведочного поиска». Дело в том, что известные нам поисковые системы подходят, прежде всего, для поиска точных и емких ответов на короткие запросы. Но такой формат поиска не подходит для пользователей, которым нужно разобраться в новой для них предметной области, иногда даже в ограниченные сроки. Такой пользователь не знает ключевых терминов области, почти не имеет представления о структуре предметной области, не может указать точных формулировок запроса и не всегда подра-

зумевает единственный правильный ответ. Для этих случаев необходим поиск по смыслу, а не набору ключевых слов. Запросом в парадигме разведочного поиска может являться, например, отрывок текста или документ целиком, а результатом являются актуальные данные, близкие по тематике к запросу.

Такой поиск может быть полезен как ученым и студентам, так и рядовым работникам, – задача поиска полезной информации и релевантных документов в сжатые сроки становится все более актуальной.

При реализации подобной системы может возникнуть такая проблема, как локализация поиска результатов запроса. Иногда пользователю может понадобиться документ, относящийся к другой, смежной области, и вероятность того, что этот документ будет выдан пользователю по его запросу, очень мала. Нужно так попробовать улучшить систему поиска, чтобы радиус поиска был шире и корректировался в зависимости от предпочтений пользователя.

Тематическое моделирование развивается примерно с 1999 года, когда Томас Хоффман придумал модель вероятностного латентного семантического анализа. За последние 15 лет тематических моделей придумано несколько сотен, а может, даже тысяч. В общем случае задача тематического моделирования некорректно поставлена и поэтому имеет бесконечное множество решений.

Работа посвящена разработке и исследованию алгоритмов информационного поиска по тематической модели с применением эволюционного подхода.

Первый раздел посвящен тематическому моделированию и разведочному поиску. Описаны основные модели разведочного поиска информации в Интернете, сделан обзор существующих алгоритмов, реализующих методы построения тематических моделей, выдвинута гипотеза об использовании эволюционного подхода для информационного поиска по тематической мо-

дели, рассмотрены основные этапы проведения генетического алгоритма. Также была сформулирована постановка задачи для данной работы.

Во втором разделе рассмотрена математическая модель поиска документов, приведены основные математические основы генетического алгоритма, произведено описание косинусной меры близости документов.

В третьем разделе приводятся разработанные алгоритмы предобработки исходных данных, поиска документов по тематической модели и метод учета предпочтений пользователя – генетический алгоритм.

В четвертом разделе описывается методика экспериментального исследования эффективности разработанного алгоритма, и приводятся его результаты.

1 МОДЕЛИ СИСТЕМ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ РЕШЕНИЯ ЗАДАЧИ РАЗВЕДОЧНОГО ПОИСКА

1.1 Модели разведочного поиска информации в Интернете

1.1.1 Тематическое моделирование документов

1.1.1.1 Основные термины и предположения

В тематическом моделировании существует несколько основных параметров: – множество (коллекция) текстовых документов, – множество (словарь) всех термов, употребляющихся в множестве . Термом может служить слово, нормальная форма слова, определенный термин или словосочетание. Каждый документ из коллекции документов представляется вектором термов из множества термов .

С тематическим моделированием также связан термин «тема». Под ним подразумевается набор терминов, совместно часто встречающихся в документах. Каждый документ может относиться к одной или к нескольким темам.

При построении тематической модели выдвигается ряд предположений или гипотез.

1. Гипотеза о существовании тем. Пусть существует – конечное множество тем, и каждое вхождение в документ терма можно связать с определенной темой . Коллекцию документов тогда можно представить как последовательность троек .

2. Гипотеза «мешка слов» («мешка документов»). Гипотеза «мешка слов» заключается в предположении, что порядок термов в документе не важен при отнесении документов к определенной теме, то есть, тему можно определить даже после такой перестановки термов в документе, что для человека смысл текста будет утерян. Гипотеза «мешка документов» заключает-

ся в предположении, что порядок документов в коллекции также не имеет значения.

3. Гипотеза о вероятностном порождении данных. Множество представляет собой конечное вероятностное пространство, с неизвестной функцией вероятности .

4. Гипотеза разреженности. Каждый терм или слово и каждый документ должны относиться к небольшому количеству тем. При этом большинство вероятностей и либо станут равными нулю, либо будут очень близки к нулю. Алгоритмы, которые не требуют хранения нулевых значений, имеют меньшее время работы и используют меньшее количество памяти, поэтому разреженность обязательна для больших коллекций.

5. Гипотеза условной независимости определяется вероятностным распределением . Это означает, что появление терма из темы в документе не зависит от самого документа, а только от темы данного терма.

Задачей тематического моделирования является нахождение параметров – вероятности терминов в каждой теме и – вероятности тем в каждом документе .

В случае если документ может принадлежать большому количеству различных тем, то этот документ следует разделить на несколько фрагментов, и в каждом фрагменте можно выделить меньшее число тем.

В случае если терм принадлежит большому количеству тем, то этот терм можно считать общеупотребимым, и в таком случае данный терм стоит исключить из выборки документов.

Целью тематического моделирования является определение тематики документов и связанных с ними объектов. В общем случае число тем значительно меньше и .

1.1.1.2 Принцип максимума правдоподобия

Согласно принципу максимума правдоподобия, из наблюдаемой выборки выбираются значения параметров, обеспечивающие наибольшую правдоподобность данной выборки.

Для нахождения параметров модели выполняется максимизация функции правдоподобия; она определяется как зависимость вероятности выборки от параметров модели:

(1.1)

Можно перейти от произведения к сумме, прологарифмировав данную функцию. Получим задачу максимизации log-правдоподобия при ограничениях неотрицательности и нормировки:

(1.2)

(1.3)

1.1.1.3 Частотные оценки условных вероятностей

В пространстве вероятности, связанные с переменными и , выражаются как частоты соответствующих событий (частотная оценка):

—, —, —, —, (1.4)

где – число вхождений терма в документ ;

– длина документа ;

– длина коллекции;

– число вхождений терма во все документы коллекции.

Аналогично можно оценивать вероятности, выражающиеся через скрытую переменную :

—, —, — —, (1.5)

где – число троек, в которых терм документа принадлежит теме ;

– число троек, в которых терм связан с темой ;

– число троек, связанных с темой ;

– число троек, в которых терм документа принадлежит теме .

В отличие от (1.4), эти оценки не могут быть вычислены непосредственно по исходным данным, т. к. темы не известны.

1.1.1.4 Методы, используемые для построения тематических моделей

Рассмотрим основные методы, которые применяются в тематическом моделировании.

1. Вероятностный латентно-семантический анализ PLSA. В методе PLSA (Probabilistic Latent Semantic Analysis) [2] переменные термов и документов являются наблюдаемыми. Здесь каждый документ можно определить как числовой вектор, относящийся к одной или нескольким разным темам с определенной вероятностью для каждой из них [3]. Переменные тем являются скрытыми, или латентными. Их поиск и выявление является задачей данного метода.

После того, как заданы несколько тем, требуется оценить ряд следующих величин:

- – вероятность вхождения слова в документ ;
- – вероятность, что документ тесно связан с темой ;
- – вероятность, что слово тесно связано с темой .

Сначала с вероятностью выбирается случайный документ Затем также случайно выбирается тема с вероятностью и после этого терм с вероятностью . Результатом будет служить пара ( ).

Вероятностная тематическая модель коллекции документов задается следующим образом:

(1.6)

У данного метода есть свои недостатки. Например, при работе с большой коллекций текстов ввиду линейного возрастания числа параметров и имеется склонность к переобучению. Также при добавлении нового документа в исходную коллекцию необходимо каждый раз перестраивать всю модель.

2. Байесовская регуляризация. В байесовском подходе используется принцип максимума апостериорной вероятности. Предполагается, что параметры модели являются случайными, но подчиняются некоторому априорному распределению где – неслучайный гиперпараметр. В результате получаем модификацию задачи (1.2), с логарифмом априорного распределения в роли регуляризатора:

(1.7)

В байесовском выводе попытки комбинирования моделей и добавления дополнительных критериев влекут за собой необходимость повторного проведения всех математических выкладок и в каждом случае требуют новой программной реализации, что критически сказывается на сроках и стоимости таких разработок. Поэтому в практических приложениях выбор метода построения тематической модели, как правило, происходит в пользу более простых моделей.

3. Латентное размещение Дирихле. Исходные данные состоят из набора документов, каждый из которых содержит слов. Слова являются наблюдаемыми переменными в модели, все остальные переменные являются скрытыми. Переменная задает распределение тем для каждого та [4].

выступает фиксированным входным параметром модели и равняется количеству тем. Оптимальное обычно подбирают эмпирическим способом [5]. Переменная задает распределение слов в теме .

Данный метод основан на утверждении, что столбцы матриц и являются случайными векторами, порожденными распределением Дирихле с параметрами и соответственно:

(1.8)

(1.9)

,

где – гамма-функция.

Модель порождения данных в данной модели схожа с генеративным процессом из PLSA. Она является двухуровневой: сначала из распределения Дирихле генерируются вектора , задающие темы. После чего из распределений генерируются слова, формирующие тематические части документов . В результате получаем описание кластерных структур на коллекции текстов, где задают кластерные центры, а распределения описывают точки этих кластеров.

Главными отличием LDA от PLSA являются оценки условных вероятностей (1.4).

В методе PLSA используются несмещенные оценки максимума правдоподобия [6]:

(1.10)

В методе LDA – сглаженные байесовские оценки:

(1.11)

К недостаткам распределения Дирихле можно отнести отсутствие убедительных лингвистических обоснований, а также громоздкие математические выкладки при попытке построения многоцелевых комбинированных моделей.

4. EM-алгоритм. EM-алгоритм является применением метода простых итераций для решения оптимизационной задачи (1.2). Каждая итерация алгоритма состоит их двух шагов – E (expectation) и M (maximization) [7].

Перед первой итерацией необходимо задать начальное приближение параметров и . Ими могут быть нормированные случайные вектора из равномерного распределения.

На E-шаге с помощью текущих и с использованием формулы Байеса происходит вычисление условных вероятностей всех тем в каждом документе для каждого терма :

(1.12)

На M-шаге по значениям условных вероятностей тем вычисляется новое приближение параметров , при помощи величины .

Суммирование по документам и по термам дает оценки , , , посредством которых, в соответствии с (1.5), можно получить частотные оценки условных вероятностей , :

—, , (1.13)

—, , (1.14)

1.1.1.5 Аддитивная регуляризация тематических моделей

ARTM (AdditiveRegularizationofTopicModels) – это подход к построению вероятностных моделей путем комбинирования различных методов тематического моделирования [8].

Задачу называют корректно поставленной по Адамару в случае, если ее решение существует, оно единственно и устойчиво.

Однако задача стохастического матричного разложения таковой не является, т. к. ее решение не единственно: разложение определено с точностью до невырожденного преобразования (где матрицы и тоже стохастические), следовательно, оно имеет бесконечное множество возможных решений.

Данную проблему можно решить с помощью применения регуляризации, представляющей собой добавление к основному критерию правдоподобия (1.2) дополнительного критерия – регуляризатора [9]:

(1.15)

. (1.16)

Переменные называются коэффициентами регуляризации и выражают важность регуляризатора и его влияние на конечную модель.

Также следует заметить, что при решении задачи максимизации правдоподобия будет осуществляться и максимизация нового критерия, что позволяет улучшить релевантность моделей.

На практике нередко используются сразу несколько регуляризаторов, каждый из которых накладывает свое ограничение на решение.

Подход ARTM обладает относительной легкостью понимания, построения и вывода моделей, а также сравнительно простым математическим аппаратом – для добавления регуляризатора достаточно лишь добавить его производные в формулы М-шага (1.13)–(1.14).

Также следует заметить, что в этом подходе регуляризаторам необязательно иметь вероятностную интерпретацию или быть априорными распределениями.

Ниже приведены виды регуляризаторов, наиболее часто используемые в данном подходе.

1. Регуляризатор сглаживания. С помощью данного регуляризатора формализуется требование о том, чтобы часто используемые слова, стоп-слова, а также специфические и редко встречающиеся слова были отнесены в отдельные темы и не засоряли предметные темы. Это оказывает положительный эффект на их интерпретируемость.

Данный регуляризатор осуществляет минимизацию дивергенции Кульбака-Лейблера между распределениями и и дискретными распределениями слов во всей коллекции и тем по всей коллекции соответственно:

(1.17)

где – коэффициенты регуляризации.

Формулы М-шага тогда приобретают вид:

(1.18)

. (1.19)

2. Регуляризатор разреживания. Можно ввести предположение, являющееся вполне естественным, что каждый документ описывается небольшим числом тем *t*. Следовательно, в распределении значения вероятностей таких тем можно увеличить, а вероятности тем, не характеризующих документ, приравнять к нулю.

Аналогично введем предположение, что каждая тема определяется небольшим количеством слов, а вероятности остальных слов в распределении должны равняться нулю.

В случае с LDA распределение Дирихле не допускает нулевых значений в порождаемых им векторах, поэтому осуществить зануление в этом методе невозможно.

В ARTM такой результат достигается добавлением разреживающего регуляризатора. Таким образом, осуществляется максимизация дивергенции между равномерным распределением и искомыми распределениями:

(1.20)

где – коэффициенты регуляризации.

Формулы М-шага:

(1.21)

(1.22)

3. Регуляризатор сокращения незначимых тем. Этот регуляризатор помогает исключить из модели темы, ядро которых состоит из крайне малого количества редких слов. Регуляризатор позволяет привести распределение

тем во всей коллекции к разреженному виду, а также устремить дивергенцию Кульбака-Лейблера к ее максимальному значению между и равномерным распределением:

(1.23)

Формула М-шага:

— , (1.24)

где – коэффициент регуляризации.

Исходя из этой формулы, вероятности темы понижаются для всех документов, если – число отнесенных к ней слов – мало. Таким образом, данный регуляризатор оптимизирует количество тем, если изначально их заданное число было избыточным.

4. Регуляризатор декоррелирования. Способствует тому, чтобы темы как можно сильнее отличались друг от друга. Этот регуляризатор осуществляет минимизацию суммы ковариаций между распределениями и для всех пар тем *:*

(1.25)

Формулы М-шага:

, (1.26)

где – коэффициент регуляризации.

Декоррелирование позволяет разреживать темы, и выделять в них ядра, состоящие из слов с большей величиной вероятности . Кроме этого, регуляризатор способен группировать стоп-слова в отдельные фоновые темы [10].

1.1.1.6 Пакеты ПО для тематического моделирования

1. Gensim [11] – это пакет инструментов для обработки коллекций текстов. Он реализован в виде подключаемого модуля для языка программирования Python. Разработка проекта началась с 2008 года, тогда он представлял собой небольшой набор скриптов для поиска похожих статей, который был разработан для Чешской Библиотеки Цифровой математики.

Gensim хорошо подходит для обработки больших объемов текстовых данных с помощью потоковых данных. В задачах семантического моделирования текстов данный проект выделяется среди пакетов программного обеспечения своей эффективностью и простотой в использовании.

Gensim включает в себя, помимо прочего, реализации алгоритмов латентного семантического анализа, латентного распределения Дирихле и иерархического процесса Дирихле (Hierarchical Dirichlet Process, HDP).

Пакет распространяется по лицензии GNU LGPLv2.1, допускающей свободное академическое и коммерческое использование, исходные коды находятся в открытом доступе.

2. Vowpal Wabbit (VW). Широко используется для работы с алгоритмами машинного обучения. В прошлом библиотека разрабатывалась группой Yahoo! Research, а на текущий момент над проектом работает Microsoft Research [12].

Из достоинств библиотеки можно выделить гибкость в выборе формата входных данных, хорошую масштабируемость, высокую скорость работы, поддержку большого количества различных режимов обучения и множество поддерживаемых алгоритмов.

Главной особенностью библиотеки считается возможность онлайн-обучения, что особенно важно для задач, требующих обработки больших и

высокоразмерных данных. Проект реализован на языке C++, исходные коды библиотеки также находятся в открытом доступе.

3. BigARTM [13] – это кроссплатформенная библиотека с открытым исходным кодом, позволяющая решать задачи тематического моделирования больших коллекций текстовых документов. Разрабатывается А. Фреем (МФТИ), К. Воронцовым (ВЦ РАН) и М. Апишевым (МГУ). Ядро библиотеки написано на C++, имеются интерфейсы для работы из командной строки, с библиотекой на C++, Python.

Библиотека представляет собой реализованный подход аддитивной регуляризации ARTM, благодаря чему ее можно использовать к построенной практически любым способом из возможных тематических моделей на основе мешка слов, включая модели PLSA и LDA.

Отличается наиболее эффективной на сегодняшний день реализацией параллельных распределенных алгоритмов вероятностного тематического моделирования.

Помимо классической оффлайн версии, в BigARTM реализован параллельный онлайновый EM-алгоритм, что позволяет обрабатывать большие коллекции документов за одну итерацию, при этом минимизируя используемый объем оперативной памяти.

Кроме того, EM-алгоритм в BigARTM обобщен для мультимодальных регуляризованных моделей.

Эксперименты на коллекции из 3,7 миллионов статей английской Википедии и словаре из 100 тысяч уникальных слов показали, что BigARTM существенно выигрывает в скорости и точности работы, по сравнению с другими известными программными пакетами.

На текущий момент BigARTM по умолчанию включены следующие метрики качества:

• перплексия;

• разреженность;

• средняя чистота тем и средняя контрастность тем;

• средний размер лексического ядра тем;

• доля фоновых слов во всей коллекции.

Библиотека также адаптирована для работы с мультимодальными тематическими моделями, учитывающими метаданные документов. Метаданные могут быть представлены в виде информации об авторах, метках времени создания документа, изображений в документе и т. д.

1.1.2 Индексирование документов

1.1.2.1 Статистическая мера TF-IDF

TF-IDF – статистическая мера, с помощью которой оценивается важность слова, используемого в отдельном документе, для коллекции документов. Данная мера использует понятия о том, что вес отдельно взятого слова имеет прямую зависимость от частоты употребления этого слова в одном документе и обратную зависимость от частоты употребления слова во всех документах коллекции.

TF (от англ. *term frequency* – частота слова) – с помощью данного параметра оценивается важность слова в пределах отдельного документа. Он показывает то, как соотносится количество вхождений отдельного слова и общее количество слов в документе:

——, (1.27)

где – число вхождений слова в документ , в знаменателе – общее число слов в данном документе.

IDF (от англ. *inverse document frequency* – обратная частота документа) – обратная величина частоты появления отдельного слова в документах коллекции. Учет этой величины понижает веса широко употребляемых слов. Для слов, которые являются уникальными в контексте всей коллекции доку-

ментов, существует только одно значение обратной частоты. Формула для данной величины записывается следующим образом:

_____, (1.28)

где – число документов в коллекции;

– число документов из коллекции , в которых встречается слово *w*.

Непосредственно мера TF-IDF – это произведение частоты слова и обратной частоты документа [14]:

(1.29)

1.1.2.2 Инвертированный индекс

Гипотетически, существует коллекция документов. Задача – найти все документы, удовлетворяющие определенному запросу. Понятно, что слово «документ» также имеет более широкое значение, включая абзац, изображение, музыкальный файл, веб-сайт и т. д.

Каждый документ состоит из термов – индексированных единиц, обычно слов.

При поиске конкретного терма соответствующая строка показывает, в каких документах появляется терм (строку можно рассматривать как вектор). Аналогично для столбца есть вектор, который показывает, какие термы присутствуют в документе.

Инвертированный индекс представляет собой структуру данных, в которой хранится сопоставление термов с документами, в отличие от обычного подхода от документов к термам. Инвертированный индекс обычно делится на две основные части:

• словарь термов;

• список документов, в которых появляется терм (и дополнительные данные, например, позиции терма).

При поиске данного терма сначала система поиска ищет этот терм в словаре, чтобы получить указатель на соответствующий список публикаций. Затем он просматривает список публикаций и получает список всех документов. В зависимости от типа запроса документы могут быть отсортированы по частоте термов или могут быть получены только лучшие k документов.

Частота термов – это количество вхождений термов в одном документе.

Частота документов – это количество документов, в которых встречается терм.

1. Словарь термов. Словарь реализуется путем хеширования или деревьев поиска. Термы являются ключами, а указатели к списку публикаций являются значениями. Тип принятой реализации зависит от того, сколько ключей у нас есть, как часто мы вставляем (или удаляем) новые ключи и как часто мы получаем доступ к определенным термам.

Хеширование обеспечивает поиск в постоянное время, но в нем отсутствует поддержка префиксных запросов. Мы также должны разрешить коллизии хэш-функций.

С другой стороны, деревья поиска поддерживают поиск термов с одинаковым префиксом при стоимости поиска , где – количество термов в словаре, при условии, что дерево сбалансировано. Однако самым большим недостатком этой функции является вставка или удаление термина в словаре.

2. Список документов – это список вхождений соответствующего терма в собрание документов. Выполнение логического запроса «И» достигается просто путем извлечения обоих списков сообщений и их пересечения. Для эффективного пересечения списки документов обычно сортируются в порядке возрастания по идентификатору документа. Каждый документ в списке документов обычно содержит больше информации, чем просто идентификатор документа. Необходимо добавить частоту терма для поддержки ранжи-

рованных запросов. Позиция терма в документе необходима при выполнении контекстных запросов [15].

1.1.3 Нейронные сети

Нейронные сети – это класс методов, принципы работы которых схожи с принципами функционирования мозга. Данные методы позволяют прогнозировать значения переменных для новых данных по имеющимся данным других наблюдений после прохождения этапа обучения.

Важным этапом при построении нейронной сети является выбор ее архитектуры, а именно числа «слоев» и числа «нейронов» в каждом слое. На рисунке 1.1 представлен пример архитектуры сети. При выборе архитектуры сети нужно учитывать, что размер и структура сети должны соответствовать существу исследуемого явления. Из-за того, что на начальном этапе анализа природа явления не бывает хорошо известна, архитектуру сети не всегда удается выбрать правильно, и этот процесс часто связан с длительным процессом «проб и ошибок».

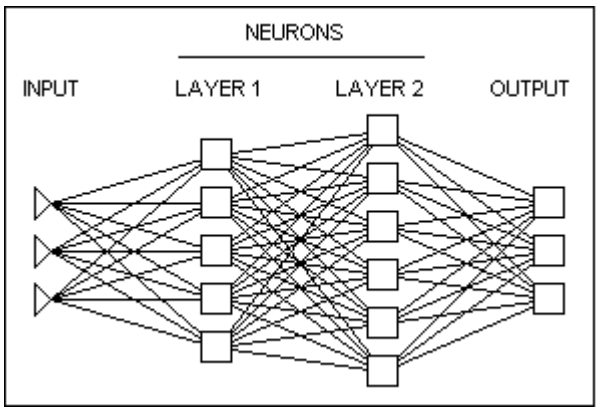

Рисунок 1.1 – Пример
архитектуры нейронной сети

Теперь построенная сеть может приступить к процессу так называемого «обучения». На этом этапе нейроны обрабатывают входные данные и корректируют свои веса таким образом, чтобы сеть смогла наилучшим образом прогнозировать «обучаемые» данные. После обучения на имеющихся данных сеть готова к работе и может использоваться для построения прогнозов.

Иногда нейронные сети выдают прогноз очень высокого качества; однако, они представляют собой типичный пример нетеоретического подхода к исследованию. Это значит, что внимание уделяется исключительно практическому результату – в данном случае, точности прогнозов, – а не сути механизмов, лежащих в основе явления.

Из преимуществ использования нейронных сетей можно выделить то, что они, по крайней мере, теоретически, могут аппроксимировать любую непрерывную функцию, и поэтому исследователь может заранее не принимать какие-либо гипотезы относительно модели.

Однако существенным недостатком нейронных сетей является зависимость окончательного решения от начальных установок сети и его практически невозможно «интерпретировать» в традиционных аналитических терминах, которые обычно применяются при построении теории явления [16].

1.2 Эволюционные алгоритмы

Описанные модели разведочного поиска обладают следующим недостатком – алгоритмы, используемые в этих моделях, локальные. Чтобы иметь возможность получить подходящие решения, не попадающие в область поиска для локальных алгоритмов, нужно использовать алгоритмы для поиска глобальных экстремумов. Эту задачу решают эволюционные алгоритмы.

1.2.1 Определение и этапы алгоритма

Генетические алгоритмы (ГА) применяются для решения многих задач, среди них можно выделить аппроксимацию функций, задачи о кратчайшем пути, задачи размещения, настройку искусственной нейронной сети, игровые стратегии. Главным образом, генетические алгоритмы широко используются для решения задачи поиска глобального экстремума многопараметрической функции.

Пусть существует многопараметрическая функция, которая по заданному набору численных параметров возвращает некоторое значение. Возьмем множество строк, каждая из которых кодирует вектор чисел (длина вектора равна количеству параметров функции). По заданному вектору высчитаем соответствующее значение функции. Будем считать более приспособленными те строки, для которых значение функции велико, аналогично будем считать менее приспособленными строки, для которых значение функции мало. Если мы запустим эволюцию на строках по подобию природной, то на каждом поколении будем получать строки со все большими значениями функции. Таким образом, такого рода эволюция решает задачу максимизации многопараметрической функции.

В классическом ГА начальная популяция формируется случайным образом. Количество особей N фиксируется и не изменяется в течение работы всего алгоритма. Каждая особь представляется случайной L-битной строкой и является одним из решений поставленной задачи. Более приспособленные особи – это более подходящие ответы.

Шаг алгоритма состоит из следующих трех стадий: генерация промежуточной популяции, – она формируется путем отбора текущего поколения – скрещивание особей промежуточной популяции с использованием кроссовера и мутация нового поколения.

Оператор кроссинговера (crossover operator), также называемый кроссовером, является основным генетическим оператором, за счет которого особи обмениваются генетическим материалом друг с другом. Суть кроссинговера состоит в следующем: случайным образом определяется точка внутри особи, в которой обе особи делятся на две части и обмениваются ими.

Схема работы генетического алгоритма изображена на рисунке 1.2.

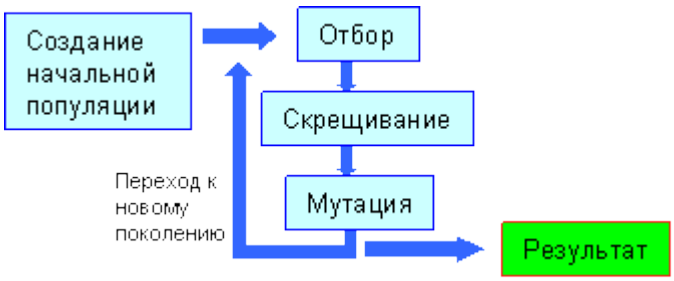

Рисунок 1.2 – Схема работы генетического алгоритма

Промежуточная популяция – это особи, получившие право размножаться. Приспособленные особи могут быть внесены в нее несколько раз, в то время как неприспособленные могут туда вообще не попасть. В общем случае вероятность каждой особи попасть в промежуточную популяцию пропорциональна ее приспособленности. Это явление называется пропорциональным отбором.

Далее особи промежуточной популяции случайным образом разбиваются на пары. Каждая из них с вероятностью скрещивается (применяется оператор кроссовера), в результате чего получаются два потомка, которые записываются в новое поколение. Если же пара не скрещивалась, в новое поколение записываются особи этой пары.

Затем к новому поколению применяется оператор мутации. Определенный бит особи популяции с вероятностью инвертируется. Эта вероятность обычно очень мала, менее 1%. Оператор мутации необходим для «выбивания» популяции из локального экстремума и способствует защите от преждевременной сходимости.

На этом шаг алгоритма завершается, новое поколение объявляется текущим и все действия повторяются.

Такой процесс эволюции может продолжаться до бесконечности. Критерием останова может служить заданное количество поколений или схождение популяции.

Схождение – это такое состояние популяции, когда все строки популяции практически одинаковы, а их значения находятся в области некоторого

экстремума. Схождение популяции чаще всего означает, что найдено лучшее или близкое к лучшему решение.

Ответом может служить набор параметров, кодируемый наилучшей особью последнего поколения [17].

1.2.2 Факторы, создающие сложность для ГА

Для генетических алгоритмов есть некоторые типы функций, с которыми им работать сложнее, чем с другими. Именно на таких функциях тестируют генетические алгоритмы. Ниже приведены свойства функций приспособленности, создающие сложность для ГА.

Многоэкстремальность: сложность поиска глобального экстремума неизбежно возникает в функциях, имеющих много локальных оптимумов, таких, что значения функции в соседних экстремальных точках отличаются на достаточно малую величину.

Обманчивость – это характеристика функции, построенной так, что шаблоны малого порядка уводят популяцию к локальному экстремуму.

Простыми словами, обманчивость целевой функции – это когда её оптимум «скрывается» среди «плохих решений».

В этом случае, ГА вероятнее всего не найдёт глобальный экстремум, т. к. неоптимальные решения, среди которых и «прячется» искомый оптимум, будут отброшены алгоритмом.

Изолированность («поиск иголки в стоге сена») – проблема для любого метода оптимизации, при котором в области существования экстремума значения функции не приближены к значению экстремума, и никак не сигнализируют о том, что минимум или максимум находится близко, т. е. функция не предоставляет никакой информации, подсказывающей, в какой области искать экстремум. Лишь случайное попадание особи в глобальный экстремум может решить задачу [18].

1.3 Постановка задачи

Целью выпускной квалификационной работы является разработка и исследование алгоритмов информационного поиска по тематической модели с применением эволюционного подхода.

Для достижения поставленной цели требуется:

1) разработать и реализовать алгоритм обработки исходных данных;

2) освоить методы работы с библиотекой BigARTM и построить тематическую модель на коллекции документов;

3) разработать и реализовать алгоритм поиска документов по тематической модели;

4) разработать и реализовать эволюционный алгоритм для глобального поиска с учетом предпочтений пользователя;

5) разработать методику экспериментального исследования алгоритма поиска;

6) сделать выводы на основе полученных результатов исследований.

1.4 Выводы по разделу

В первом разделе были описаны общие принципы решения задачи разведочного поиска. Особое внимание уделено тематическому моделированию и его математическим обоснованиям. Были рассмотрены различные методы построения тематических моделей, указаны их недостатки и достоинства. На основе проведенного обзора в качестве используемого в данной работе подхода решено использовать подход аддитивной регуляризации тематических моделей ARTM, так как благодаря его гибкому аппарату регуляризаторов становится возможной тонкая подстройка модели под конкретные нужды решаемой задачи.

Также в данном разделе приводится обзор существующих программных решений, реализующих рассмотренные алгоритмы. Все найденные паке-

ты ПО представляют собой библиотеки функций или их компоненты, в то время как самостоятельных приложений, которые бы осуществляли работу с тематическими моделями, обнаружить не удалось.

В заключение было проведено краткое описание эволюционных алгоритмов, и выдвинута гипотеза о том, что применение эволюционных алгоритмов в построении тематических моделей позволит лучше настроить модель под конкретные нужды пользователя.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ПОИСКА ДОКУМЕНТОВ

2.1 Косинусная мера близости документов

Косинусное сходство − это мера сходства между двумя векторами пространства, используемая для измерения косинуса угла между ними.

Если даны два вектора признаков, то косинусное сходство может быть представлено с использованием скалярного произведения и нормы. Формула косинусной меры между двумя векторами признаков A и B размерности n имеет следующий вид:

(2.1)

В случае информационного поиска, косинусное сходство двух документов изменяется в диапазоне от 0 до 1, поскольку частота терма не может быть меньше нуля. Чем косинусное сходство ближе к 1, тем тематически ближе документы находятся между собой. Угол между двумя векторами частоты терма не может быть больше, чем 90°.

Популярность косинусного сходства состоит в том, что оно эффективно в качестве оценочной меры, особенно для разреженных векторов, так как необходимо учитывать только ненулевые измерения [19].

2.2 Ранжирование документов по запросу, модификация запроса

Пусть – текст запроса произвольной длины .

Для документа-запроса формируется вектор-распределение

(2.2)

по темам, предопределенным в матрице для всех документов модели:

. (2.3)

Данный вектор получается в результате выполнения EM-алгоритма только для запроса. Однако в отличие от используемого при построении тематической модели подхода, матрица распределения тем по словам берется из самой модели и остается фиксированной на всем протяжении работы алгоритма. Обновлению на каждой итерации подвергается только интересующий нас вектор и именно после его схождения работа алгоритма завершается.

Формула косинусной меры для документа из коллекции и документа-запроса имеет следующий вид:

(2.4)

здесь – это вес темы в документе коллекции и в запросе соответственно, – количество тем в модели.

После выполнения запроса и вывода первых n документов с наибольшим значением косинусной близости может случиться так, что пользователь захочет уточнить поиск, задав дополнительный критерий.

В таком случае у него должна быть возможность скорректировать поисковую выдачу так, чтобы максимизировать в ней количество релевантных новому критерию документов и исключить из нее документы, по-прежнему тематически близкие исходному запросу, но не соответствующие заданному критерию:

Этого можно добиться, если позволить пользователю помечать просмотренные им результаты поиска как релевантные и нерелевантные его пожеланиям. От системы требуется самостоятельно выявить пожелания на основе предоставленного пользователем разбиения документов и скорректировать выдачу соответствующим образом.

2.3 Математические основы генетического алгоритма

Генетический алгоритм начинает свою работу с формирования начальной популяции – конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью вероятностных жадных алгоритмов. Начальной популяцией для генетического алгоритма, построенного в этой работе, являются первые s документов, которые близки по косинусной мере близости с запросом.

На каждом шаге эволюции с помощью вероятностного оператора селекции выбираются два решения , . Операторы селекции строятся таким образом, чтобы с ненулевой вероятностью любой элемент популяции мог бы быть выбран в качестве одного из родителей. Одним из операторов селекции является метод рулетки, в котором особи отбираются с помощью N запусков рулетки, где – размер популяции. Для каждого члена популяции отведен свой сектор на колесе. Размер сектора пропорционален вероятности попадания в новую популяцию:

———, (2.5)

где – пригодность -той особи.

Ожидаемое число копий -той хромосомы после оператора рулетки определяется по формуле:

. (2.6)

Пусть оператором селекции выбраны два документа из популяции: и . Как только два решения выбраны, к ним применяется оператор кроссовера. Существует много различных версий этого оператора, среди которых простейшим является простой кроссовер. Простой кроссовер – разделение двух родительских хромосом в случайно выбранной позиции гена и «склеивание» первого фрагмента хромосомы со вторым фрагментом другой хромосомы с получением двух новых дочерних

особей. То есть, создаются новые документы со следующими распределениями вероятностей (потомки):

, (2.7)

(2.8)

где

(2.9)

(2.10)

где – точка разделения.

Оператор мутации с заданной вероятностью меняет значение каждой координаты. Мутацию в генетических алгоритмах вещественного кодирования рекомендуется проводить в 20% генах текущей популяции. Таким образом, количество мутаций, приходящихся на одно поколение популяции, вычисляется по соотношению:

, (2.10)

где вероятность возникновения мутации;

– количество особей в популяции;

– количество генов особи.

Функция приспособленности генетического алгоритма, используемого в данной работе, основана на косинусной мере близости документов.

Документы, которые должна выдавать поисковая система, должны быть как можно ближе к документам, отмеченным пользователем как подходящие, и как можно дальше от документов, отмеченным пользователем как неподходящие.

Обозначим за множество понравившихся пользователю документов, за множество документов, которые пользователю не понравились. Тогда функция приспособленности выразится следующей формулой:

, (2.12)

где – документ-запрос;

– косинусная мера близости между документами a и b;

– количество понравившихся и непонравившихся документов соответственно.

2.4 Выводы по разделу

В данном разделе была рассмотрена мера, с помощью которой в данной работе оценивается близость двух документов – косинусная мера близости. Также был описан алгоритм предобработки исходных документов и приведения их в пригодный для работы с ними тематической модели формат.

Были рассмотрены общие принципы и алгоритмы выполнения поисковых запросов по тематической модели, алгоритм ранжирования документов по запросу.

Также были описаны математические основы генетического алгоритма для более глобального поиска релевантных документов. Были рассмотрены основные этапы проведения генетического алгоритма: формирование начальной популяции, выбор особей-родителей, кроссинговер, мутация, функция приспособленности для определения потомков.

3 РАЗРАБОТКА АЛГОРИТМОВ ПОИСКА

3.1 Алгоритм предварительной обработки документов

Прежде чем использовать документы для построения модели, документы необходимо особым образом подготовить. Схема алгоритма предобработки документов выглядит следующим образом:

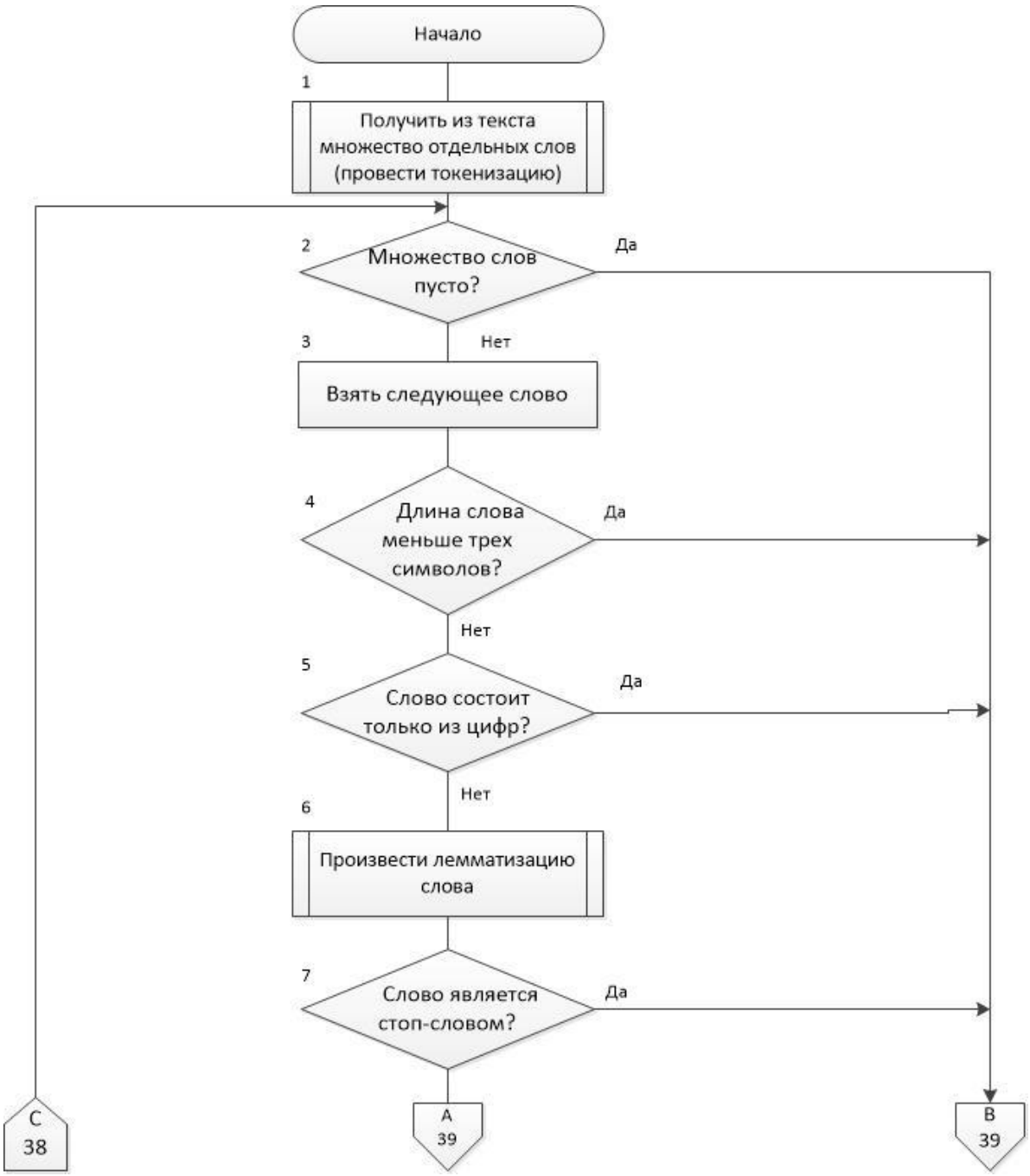

Рисунок 3.1 – Алгоритм предобработки документов (начало)

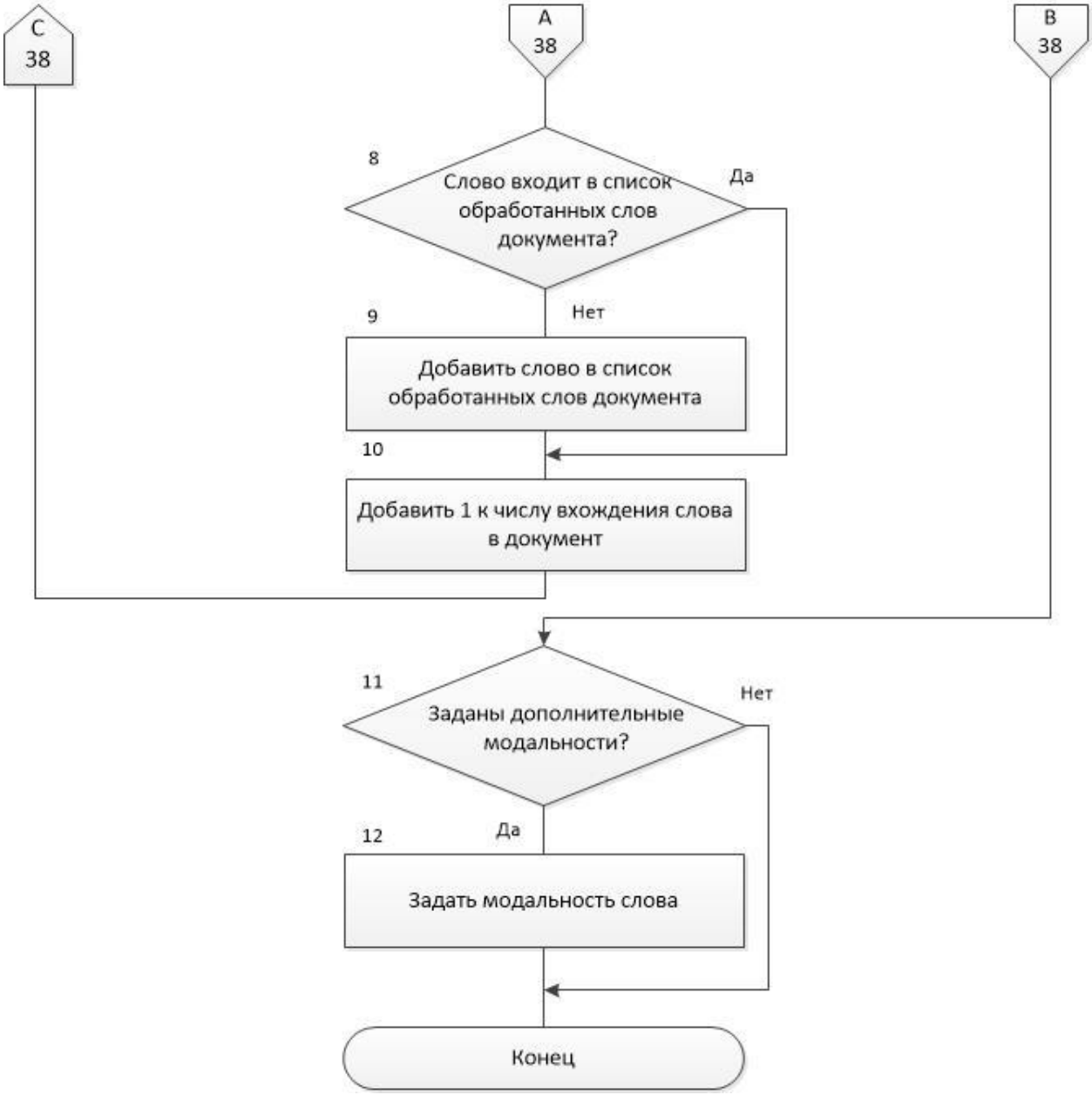

Рисунок 3.1 – Продолжение

Для того чтобы можно было выполнять операции не над текстом, а над отдельными словами, нужно для начала провести токенизацию, т. е. разбить текст на более мелкие части, или токены. Токенами могут служить слова, числа и знаки пунктуации. Данное разбиение позволяет выделить слова в самостоятельные элементы, над каждым из которых в дальнейшем можно проводить индивидуальные операции.

Затем все слова необходимо привести к единой форме. Это можно сделать двумя способами – либо находя основу слова и удаляя все остальные его

части, иными словами, проведя стемминг, либо посредством проведения лемматизации, то есть приведения слов к их нормальной (словарной) форме. По сравнению с операцией стемминга, в задаче лемматизации требуется гораздо больше как вычислительных, так и временных затрат, но при этом она дает хорошее качество обработки, в то время как при стемминге информация может быть утеряна. Для русского языка, а также для ряда других языков, более подходящим вариантом предобработки данных является лемматизация, в силу некоторых грамматических особенностей.

Следующим этапом является удаление стоп-слов из исходных текстов. Стоп-слова – это слова, которые не несут в себе конкретной тематической окраски, и поэтому понижают качество тем конечной модели. К стоп-словам можно отнести, например, предлоги, частицы, союзы. Также стоп-словами считаются те слова, число вхождений которых в исходный набор документов чрезмерно велико. Подвергаются удалению и слова с низкой частотой вхождений в исходный корпус, даже при условии, что они хорошо интерпретируются по темам, потому что такие редкие слова не способствуют формированию качественных тем.

Кроме редко встречающихся и часто встречающихся слов в некоторых случаях убирают числовые значения, т. е. слова, состоящие только из цифр. Правда это не всегда целесообразно, в этом случае все зависит от целей, которым должна отвечать модель, и от особенностей исходной коллекции документов [20].

Далее полученную в результате выполнения предыдущих шагов коллекцию необходимо привести к формату Vowpal Wabbit. Данный формат представляет собой текстовый файл, в котором документы располагаются друг за другом по одному на строку в виде мешка слов. Каждая строка начинается с идентификатора документа, затем, опционально, указывается название какой-либо модальности слов, вслед за которым перечисляются все слова, отнесенные к данной модальности и число их вхождений в данный доку-

мент. По умолчанию задается одна модальность – модальность основного текста. Дополнительные модальности – это метаданные документа, то есть информация, дополнительная по отношению к основному содержанию документа. Такими данными могут быть автор, время создания документа, категории, цитируемые документы и т. д. Предполагается, что метаданные могут помогать выявлять тематику документа, или, наоборот, могут быть предугаданы на основе тематики документа.

3.2 Алгоритм ранжирования документов по запросу

Схемы алгоритмов представлены на рисунке 3.2 и рисунке 3.3.

Рисунок 3.2 – Общий алгоритм поиска документов по запросу

Для документа-запроса и для корпуса текстов выполняется предварительная обработка. Затем для документа-запроса формируется вектор-распределение и матрица распределения тем по документам модели .

По полученному вектору с помощью косинусной меры вычисляется близость запроса ко всем документам исходной коллекции.

Из результатов расчетов составляется рейтинг релевантности каждого документа коллекции к запросу – чем выше значение косинусной меры, тем большей тематической близостью обладает документ.

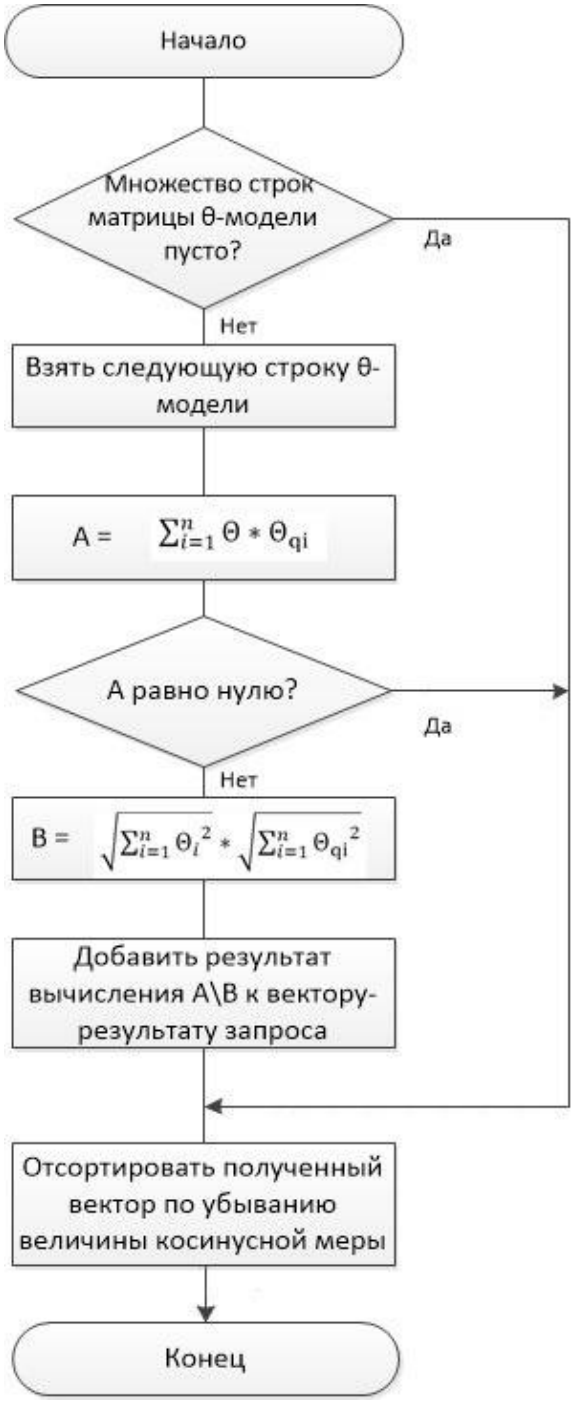

Рисунок 3.3 – Алгоритм расчета
косинусной меры
документа-запроса

3.3 Алгоритм поиска релевантных документов

Алгоритм поиска заключается в определении тематической близости между документом-запросом и документами, входящих в модель, и последующей коррекции результатов поиска с учетом предпочтений пользователя на основе разметки документов на релевантные или нерелевантные его запросу. Коррекция достигается путем применения эволюционного подхода.

Алгоритм начинает свою работу с формирования «начальной популяции», т. е. вывода наиболее близких по косинусной мере близости документов. Далее выведенные документы размечаются на релевантные и нерелевантные. Если документу не поставлена оценка, он автоматически становится нерелевантным.

После того, как произойдет разметка выведенных документов, происходит «эволюция». Между парами соседних релевантных документов происходит операция кроссинговера – обмена частями векторов – и мутации – замены одного вещественного числа в определенном релевантном документе.

Затем из коллекции убирают те документы, которые уже были ранее выданы пользователю и происходит вычисление функции приспособленности. Для каждого документа из оставшейся коллекции вычисляется сумма косинусных мер между документом коллекции и релевантными документами (а точнее векторами распределений, полученных после операции кроссинговера), а затем из нее вычитается сумма косинусных мер между документом коллекции и нерелевантными документами, умноженная на некоторый коэффициент от 0 до 1. Полученные значения функции приспособленности сортируются и документы с наибольшими значениями выдаются пользователю для повторной разметки, если это необходимо.

Схема алгоритма представлена на рисунке 3.4.

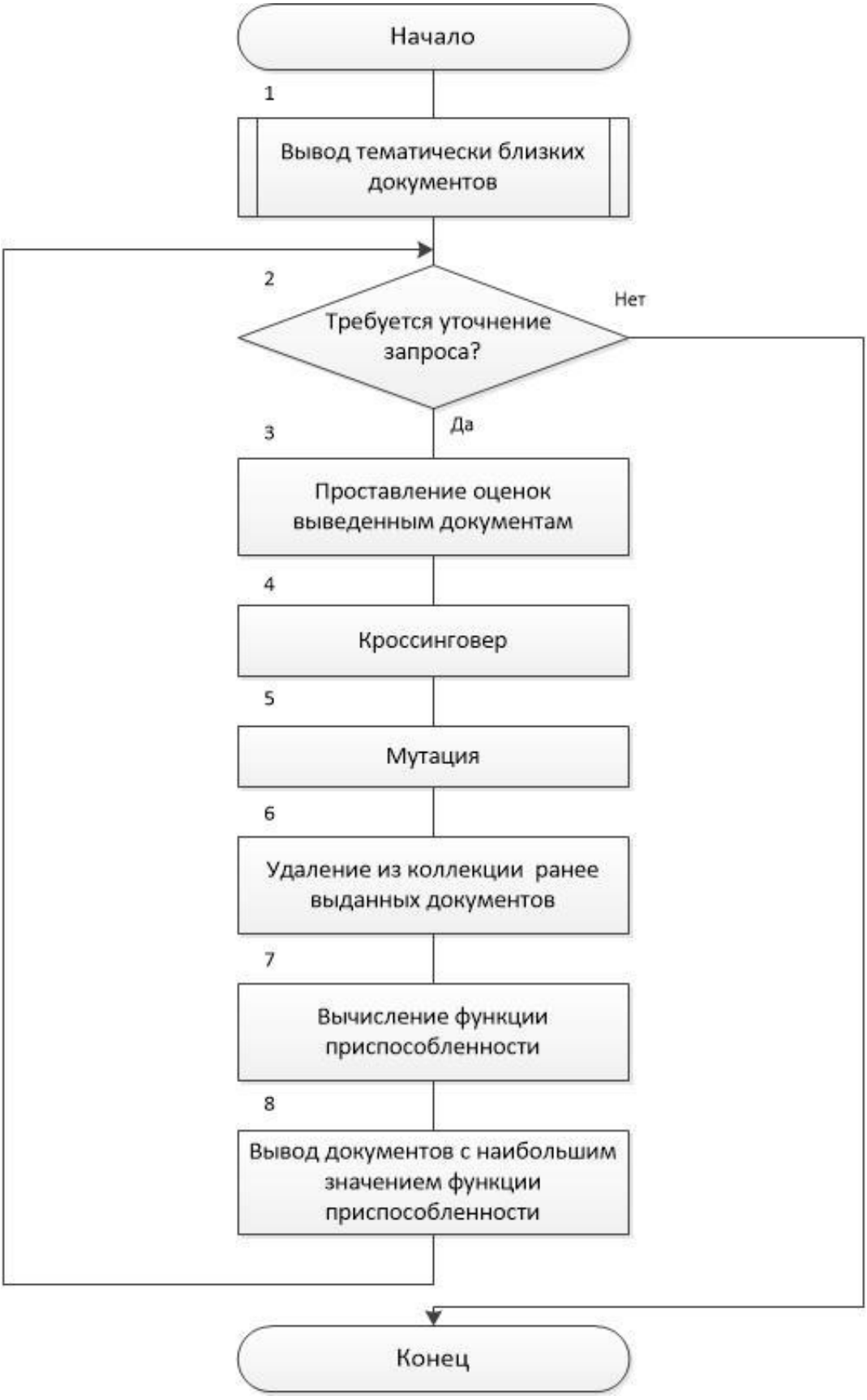

Рисунок 3.4 – Алгоритм поиска документов

3.4 Выводы по разделу

В данном разделе была рассмотрена предварительная обработка текста, описаны процедуры стемминга и лемматизации, представлена схема алгоритма предобработки текста, проведено описание формата Vowpal Wabbit.

Были рассмотрен алгоритм ранжирования документов по запросу.

Также были описаны математические основы генетического алгоритма для более глобального поиска релевантных документов. Были рассмотрены основные этапы проведения генетического алгоритма: формирование начальной популяции, выбор особей-родителей, кроссинговер, мутация, функция приспособленности для определения потомков.

4 ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМА ИНФОРМАЦИОННОГО ПОИСКА

4.1 Построение тематической модели

Для того чтобы можно было проводить экспериментальное исследование над реализованным алгоритмом поиска, требуется построить тематическую модель по некоторой коллекции документов. Для обработки и построения модели был использован архив статей с интернет ресурсов Habrahabr и Geektimes за 2011–2015 годы. Количество документов в коллекции было решено взять равным 67 000. Статьи в коллекции представлены по разным тематикам, но, как правило, они связаны со сферой IT-технологий, программированием, инженерией.

Тематическая модель была построена, исходя из количества тем, равного 100, и 20 из них были отнесены к темам, в которых должны содержаться общеупотребительные слова, а также стоп-слова, которые плохо влияют на интерпретируемость темы (фоновые темы), а остальные 80 – под основные темы.

Каждый документ в используемой коллекции содержит мета информацию, представленную в следующем виде:

– дата публикации;

– теги;

– хабы (более обобщенные теги);

– автор (его никнейм);

– количество комментариев;

– рейтинг статьи.

Данную мета информацию можно использовать и при построении тематической модели.

Также при построении модели использовались регуляризаторы сглаживания / разреживания для матриц и , регуляризатор улучшения когерент-

ности тем и регуляризатор декоррелирования матрицы (раздел 1.1.1.5). Следует отметить, что сглаживание производилось только для фоновых тем, а разреживание и декорелирование – только для основных тем.

Алгоритм, по которому строилась модель, имеет следующий вид:

1) активировать регуляризатор сглаживания на всех 100 темах, коэффициент сглаживания на фоновых темах сделать больше, чем на основных;

2) провести 15 проходов по коллекции;

3) отключить сглаживание основных тем, активировать для них регуляризатор декоррелирования;

4) провести 20 проходов по коллекции;

5) активировать регуляризатор разреживания для основных тем и регуляризатор улучшения когерентности тем;

6) провести 35 проходов по коллекции.

Первоначальное сглаживание всех тем проводится для предотвращения зануления слов и тем с низкой частотой оценкой, но потенциально важной ролью для интерпретируемости тем и качества модели в целом.

Для оценки состояния модели использовались метрики перплексии, разреженности матриц и , контрастность и чистота тем.

4.2 Методика эксперимента

Целью экспериментов ставилось определение эффективности предлагаемых методов улучшения релевантности поисковой выдачи.

Для этого были предприняты следующие шаги. Была взята тема для запроса, находящаяся между двумя предметными областями. Затем для определения релевантных документов на эту тему был приглашен независимый эксперт, который выбрал из коллекции 105, по его мнению, релевантных документов. Заинтересовавшие его документы располагаются близко друг к другу, но можно выделить две зоны, документы в которых относятся к определенной предметной области.

Был выполнен поиск этих же документов с помощью использования эволюционного подхода.

Для этого сначала были выбраны три документа-запроса, удовлетворяющие следующим условиям:

1) запрос находится в центре области отобранных документов, относящихся к первой предметной области;

2) запрос находится в центре области отобранных документов, относящихся ко второй предметной области;

3) запрос находится между предметными областями.

Затем для каждого запроса был выполнен поиск тематически близких запросу документов на основе (косинусной меры) косинусного сходства между векторами-распределениями тем документов коллекции и документа-запроса. Для первой итерации были взяты первые 30 наиболее тематически близких документов к запросу. Среди них было подсчитано количество документов, найденных экспертом. Для генетического алгоритма они были оценены как релевантные, остальные оценены как нерелевантные. Также были зафиксированы итерации, на которой были найдены документы, отмеченные экспертом.

Далее на основе полученной разметки на релевантные и нерелевантные произошел этап «эволюции» и наступила вторая итерация. На второй итерации получились новые 30 документов, которые также подвергнутся автоматической разметке и т. д.

Для проведения эксперимента был зафиксирован результат, полученный после прохождения 10 итераций.

Общий алгоритм эксперимента для каждого запроса выглядит следующим образом:

1) выполнить первоначальный поисковый запрос и получить значение его тематической близости ко всем документам коллекции;

2) отобрать первые 30 документов с наибольшим значением тематической близости по косинусной мере;

3) подсчитать количество релевантных документов эксперта;

4) запомнить итерацию, на которой релевантные документы были найдены;

5) считая релевантными документы, выбранные экспертом, а нерелевантными – остальные документы, прогнать генетический алгоритм;

6) для новых 30 документов выполнить шаги 3–5, пока не найдутся все документы, найденные экспертом, либо не пройдет 10 итераций.

4.3 Результаты эксперимента

Эксперименты проводились для разных запросов с различным расположением относительно области документов, отмеченной экспертом.

Результаты эксперимента оценивались по нескольким критериям:

– число шагов: насколько быстро исследуемый метод для каждого из запросов справился с нахождением релевантных документов;

– количество релевантных документов: сколько релевантных документов было найдено для каждого запроса;

– процент найденных документов: отношение найденных релевантных документов ко всем релевантным документам;

– число просмотренных документов: число документов, которые бы пришлось просмотреть, чтобы отобрать найденные релевантные документы.

Количество найденных релевантных документов составило 81, 62 и 95 для первого, второго и третьего запроса соответственно. С этой задачей для данных запросов наш метод справился за 9 шагов для первого запроса, за 7 шагов для второго и за 10 шагов для третьего. Исходя из выбранных 105 документов, процент найденных документов составил 77, 59 и 90%. Чтобы получить такой же результат, вручную нам пришлось бы просмотреть 270, 210, 300 документов для первого, второго и третьего запросов соответственно, в

то время как эксперту для нахождения 105 релевантных документов пришлось просмотреть примерно 500 документов.

В таблице 1 приведены сводные результаты для трех запросов:

Таблица 1 – Сводная таблица результатов эксперимента

Номер запроса	Число шагов	Количество релевантных документов	Процент найденных документов, %	Число просмотренных документов
1	9	81	77,14	270
2	7	62	59,04	210
3	10	95	90,47	300

Ниже на рисунке 4.1 приведен график скорости нахождения релевантных документов из отобранных экспертом для каждого из трех запросов.

Для перового запроса число релевантных документов составило 81 (синяя линия на графике), для второго – 62 (оранжевая линия), для третьего – 95 (зеленая линия).

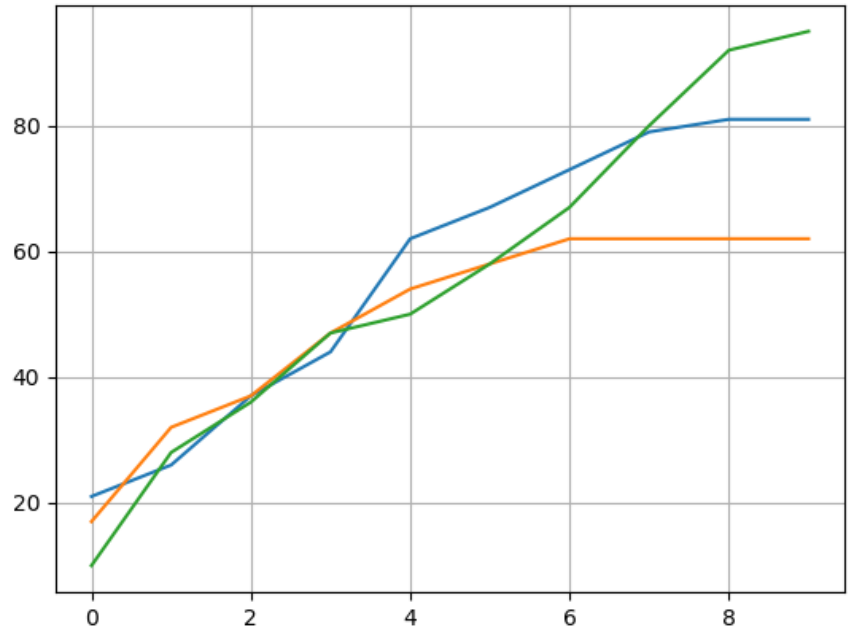

Рисунок 4.1 – График скорости нахождения документов

Пользователь заинтересован в скорейшем получении желаемых результатов – следовательно, чем более стремительно график метода приближается к верхнему значению, тем он более эффективен.

Для измерения качества работы метода была введена следующая метрика:

— (4.1)

где – общее число релевантных документов для данного запроса,

– процент документов, просмотренных для выбора релевантных документов.

Таким образом, максимальное значение, которое может принимать данная величина равна 100, что соответствует нахождению всех релевантных документов на первом шаге. Для поиска документов без использования метода данная метрика составляет 21%, в то время как рассматриваемый метод позволяет получить значение метрики, равное 39%.

Как показали эксперименты, применение метода эволюционного подхода находит в среднем 80% существующих релевантных документов; при этом система поиска находит документы, относящиеся к предметной области, смежной к запросу, но имеющей релевантные документы. Это означает, что пользователю не обязательно составлять свой запрос так, чтобы он идеально попадал в центр интересуемой его области. Также исследуемый метод позволяет в среднем на 18% улучшить скорость поиска документов.

4.4 Выводы по разделу

В разделе предложена методика экспериментов, проверяющих эффективность описанного метода улучшения релевантности поисковой выдачи. Также приводится описание построенной для проведения экспериментов тематической модели.

С помощью проведенных экспериментов было выявлено, что исследуемый метод позволяет сократить число просматриваемых документов, а также позволяет находить релевантные документы, даже если запрос находится рядом с интересуемой областью.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были изучены методики и принципы построения тематических моделей. Также разработан алгоритм подготовки исходных данных для тематических моделей.

С помощью библиотеки BigARTM была построена тематическая модель, с использованием которой разрабатывался алгоритм поиска документов. Был разработан алгоритм улучшения результатов поиска, использующий эволюционный подход.

Разработанные алгоритмы были реализованы в виде подключаемого модуля для языка Python 3. Данный модуль можно использовать для других проектов, где требуется функционал поиска по тематическим моделям или предобработки текстовых данных для работы с ними.

Также была разработана методика для экспериментального исследования описанного алгоритма поиска. Проведённые эксперименты продемонстрировали эффективность разработанного метода – с помощью применения эволюционного подхода эффективность поиска удалось улучшить на 29%.

Таким образом, поставленные на выпускную квалификационную работу задачи были выполнены, цель достигнута, но тематическое моделирование, как и информационный поиск с его использованием – это относительно молодая и чрезвычайно обширная сфера для дальнейших исследований и совершенствований, в которой на сегодняшний день остается множество нерешенных задач и объектов для изучения и улучшения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Воронцов, К.В. Вероятностное тематическое моделирование: обзор моделей и регуляризационный подход / К.В. Воронцов. – Дата обновления: 16.08.2017. URL: http://machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf (Дата обращения: 11.01.2018).

2 Hofmann, T. Probabilistic latent semantic indexing / T. Hofmann // Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. – ACM, 1999. – С. 50–57.

3 Blei, D.M. Introduction to Probabilistic Topic Models / Blei, M. David . – Comm. ACM 55 (4). – С. 77–84.

4 Blei, D.M. Latent dirichlet allocation / D.M. Blei, A.Y. Ng , M.I. Jordan // Journal of machine Learning research. – 2003. Т. 3. №. Jan. – С. 993–1022.

5 Коршунов, А. Тематическое моделирование текстов на естественном языке / А. Коршунов, А. Гомзин. – URL: http://www.ispras.ru/proceedings/docs/2012/23/isp_23_2012_215.pdf (Дата обращения: 15.01.2018).

6 Masada T. Comparing LDA with pLSI as a dimensionality reduction method in document clustering / T. Masada, S. Kiyasu, S. Miyahara // Proceedings of the 3rd International Conference on Large-scale knowledge resources: construction and applications. – LKT'08. Springer-Verlag, 2008. – С. 13–25.

7 Воронцов, К.В. Модификации ЕМ-алгоритмов для вероятностного тематического моделирования / К.В. Воронцов, А.А. Потапенко // Машинное обучение и анализ данных. – 2013. Т.1, №6. – С.657–686.

8 Воронцов, К.В. Аддитивная регуляризация тематических моделей коллекции текстовых документов / К.В. Воронцов // Доклады РАН. – 2014. Т. 456, №3. – С. 268-271

9 Тихонов, А.Н. Методы решения некорректных задач / А.Н. Тихонов, В.Я. Арсенин // М.: Наука, 1986 . – 285с.

10 Vorontsov, K.V. Tutorial on probabilistic topic modeling: Additive regularization for stochastic matrix factorization / K.V. Vorontsov, A.A. Potapenko // AIST'2014, Analysis of Images, Social networks and Texts. – 2014. Vol. 436. – С. 29–46.

11 Gensim official page. URL: http://radimrehurek.com/gensim/about.html. – Дата обновления: 10.12.2017 (Дата обращения: 15.01.2018).

12 VowpalWabbit official page. URL: https://github.com/JohnLangford/vowpal_wabbit/wiki. – Дата обновления: 27.04.2017 (Дата обращения: 15.01.2018).

13 BigARTM documentation. URL: http://docs.bigartm.org/en/stable/index.html. – Дата обновления: 27.07.2017 (Дата обращения: 15.01.2018).

14 Salton, G. Introduction to modern information retrieval / M. J. McGill, G. Salton. – McGraw-Hill, 1983. – 400 p.

15 Dyer, C. Data-intensive text processing with MapReduce / C. Dyer, J. Lin. – Morgan and Claypool Publishers, 2010. – 177 p.

16 Методы добычи данных. URL: http://www.iki.rssi.ru/magbase/REFMAN/STATTEXT/modules/stdatmin.html#mining. – Дата обновления: 20.04.2017 (Дата обращения: 15.01.2019).

17 Дударов, С.П. Математические основы генетических алгоритмов / С.П. Дударов // М.: РХТУ им. Д.И. Менделеева, 2012 . – 56 с.

18 Whitley, D. An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls / Darrel Whitley // Journal of Information and Software Technology. – 2001. Т.1, №43. – P. 817–831.

19 Manning, C. An introduction to information retrieval draft / C.D. Manning, P. Raghavan, H. Schütze. – Cambridge University Press. – 2009. – 544 p.

20 Жердеева, М.В. Стемминг и лемматизация в lucene.net / М.В. Жердеева, В.М. Артюшенко // Лесной вестник. – 2016. Т.20, №3. – С. 131–133.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. ТЕКСТ ПРОГРАММЫ

Модуль doc2VowWab.py:

```
import nltk
import urllib.parse as urlparse
import collections
import time
import sys
import pymorphy2
import os
import json
from bs4 import BeautifulSoup
from subprocess import Popen
from nltk.corpus import stopwords
import multiprocessing as mp
from functools import partial
import matplotlib.pyplot as plt
import numpy as np

SUBLIME_PATH = r'C:\Program Files\Sublime Text
3\sublime_text.exe'
DATASET_PATH = r'D:\Habrahabr\habr_posts'
FILE_BOW = r'text_test135.txt'

# nltk.download('stopwords')
nltk.download('punkt')
morph = pymorphy2.MorphAnalyzer()
stop_words = set(stopwords.words('russian'))


def run_editor(file=None, path=SUBLIME_PATH, filename=None):
    with open('file_to_read.txt', 'w', encoding='utf-8') as fout:
        if type(file) == type(dict()):
            for item in file:
                print(item, ':', file[item], file=fout)
        else:
            print(file, file=fout)
    handle = Popen([path, 'file_to_read.txt'])


def get_post(post_id):
    with open(os.path.join(DATASET_PATH, str(post_id))) as fin:
        post = json.load(fin)
    return post


def post_to_corpus_line(post):
    post_id = post['_id']
    author = post['author']
    tags = post['tags']
```

```
            date = post['published'][:10]
            print(post['published'][:4])

            words = collections.Counter()
            soup = BeautifulSoup(post['content_html'], 'lxml')
            [x.extract() for x in soup.findAll('code')]
            content_text = soup.getText()

            space_chars = u'«»“”’„*…/_.\\<>"' + u"'"
            for c in space_chars:
                content_text = content_text.replace(c, ' ')
            tokens = nltk.word_tokenize(content_text)
            for token in tokens:
                if len(token) > 2:
                    try:
                        isnumb = float(token)
                        continue
                    except:
                        token = token.lower().replace(u'ë', u'e')
                        word = morph.parse(token)[0].normal_form
                        if len(word) > 0 and word not in stop_words:
                            words[word] += 1

            def parse_hub_id(hub_pair, get_name=True):
                if get_name:
                    hub_id = hub_pair[0].replace(' ', '_').lower()
                    return hub_id
                else:
                    url_parts = list(filter(lambda s: len(s) > 0,

urlparse.urlsplit(hub_pair[1]).path.split('/')))
                    if len(url_parts) >= 2:
                        hub_id = '_'.join(url_parts[-2:])
                        return hub_id

            hubs = []
            for hub_pair in post['hubs']:
                hub_id = parse_hub_id(hub_pair)
                if hub_id:
                    hubs.append(hub_id)

            def construct_bow(words):
                return [
                    (
                            word.replace(' ', '_').replace(':',
'_').replace('|', '_').replace('\t', '_') +
                            ('' if cnt == 1 else ':{}'.format(cnt))
                    )
                    for word, cnt in words.items()
                ]

            parts = (
                    ['{}'.format(post_id)] +
                    ['|date ' + date] +
                    ['|author'] + construct_bow({author: 1} if author is
not None else {}) +
```

```
                ['|tags'] + construct_bow({tag: 1 for tag in tags}) +
                ['|hubs'] + construct_bow({hub_id: 1 for hub_id in
hubs}) +
                ['|words'] + construct_bow(words)
        )
        return ' '.join(parts)


    def print_time(cur_time, coll_size, ready_cnt):
        print(str(ready_cnt) + '/' + str(coll_size), '-
{0:.0f}%'.format(100 * ready_cnt / coll_size))
        print('{0:.2f} seconds'.format(cur_time),
              '({2:.0f}h{0:.0f}m{1:.0f}s) '.format(cur_time // 3600,
                                                   (cur_time) // 60 %
60,
                                                   cur_time - 60 *
(cur_time // 60)), end='')
        print('since begining...')


    def listener(q, notif_rate, coll_size):
        ready_cnt = 0
        start = time.time()
        with open(FILE_BOW, 'w', encoding='utf-8') as fout:
            while 1:
                message = q.get()
                if message == 'kill':
                    break
                print(message, file=fout)
                fout.flush()
                ready_cnt += 1
                if ready_cnt % (notif_rate) == 0:
                    print_time(time.time() - start, coll_size,
ready_cnt)
                    sys.stdout.flush()


    def worker(q, post_id):
        line = post_to_corpus_line(get_post(post_id))
        q.put(line)
        return line


    def do_parallel(post_ids, notif_rate, coll_size):
        manager = mp.Manager()
        q = manager.Queue()
        func = partial(worker, q)
        pool = mp.Pool(mp.cpu_count() + 1)

        watcher = pool.apply_async(listener, (q, notif_rate,
coll_size))
        pool.map(func, post_ids)

        q.put('kill')
        pool.close()
```

```
    def do_usual(post_ids, notif_rate, coll_size):
        start = time.time()
        with open(FILE_BOW, 'w', encoding='utf-8') as fout:
            ready_cnt = 0
            really_cnt = 0
            for post_id in post_ids:
                if get_post(post_id)['published'][:4] >= '2011':
                    line = post_to_corpus_line(get_post(post_id))
                    print(line, file=fout)
                    really_cnt += 1
                ready_cnt += 1
                if ready_cnt % notif_rate == 0:
                    print_time(time.time() - start, coll_size,
ready_cnt)
            print(really_cnt)


    def main(start=None, finish=None, parallel=True, notifica-
tions=20):
        post_ids = [int(filename) for filename in
os.listdir(DATASET_PATH)
                    if not filename.startswith('.')][start:finish]
        coll_size = len(post_ids)
        start_time = time.time()
        if parallel:
            do_parallel(post_ids, coll_size // notifications,
coll_size)
        else:
            do_usual(post_ids, coll_size // notifications, coll_size)
        cur_time = time.time() - start_time

        print('\nTotal time ({0:.0f} items) from '.format(coll_size),
end='')
        print('{0:.0f} - {1:.2f} seconds '.format(start, cur_time),
end='')
        print('({0:.0f}h{1:.0f}m{2:.0f}s)'.format(cur_time // 3600,
                                                  (cur_time // 60) %
60,
                                                  cur_time - 60 *
(cur_time // 60)))
```

Модуль BuildTM.py:

```
from sys import path
if r'C:\BigARTM\python' not in path:
    path.append(r'C:\BigARTM\python')

import os
import glob
import artm
import time
import matplotlib.pyplot as plt

DATA_PATH = r'text_test135.txt'
QUERY = r'query.txt'
BATCHES = 'habr_batches'
QUERY_BATCHES = 'habr_query_batches'
```

```
    def save_phi_theta(model, save_phi=False, save_theta=True):
        if save_phi:
            model.get_phi().transpose().to_hdf('phi_store.h5',
'table', mode='w')
        if save_theta:
            model.get_theta().transpose().to_hdf('theta_store.h5',
'table', mode='w')

    def get_query_data(model, predict_class=None, query_file=QUERY,
save_theta_q=True,
                       save_classes=True, filename='thetaQ_store'):
        query_bv = artm.BatchVectorizer(data_path=query_file,
                                        data_format='vowpal_wabbit',
                                        batch_size=10,
                                        target_folder=QUERY_BATCHES,
                                        gather_dictionary=True)
        print(query_bv)
        thetaQ = model.transform(batch_vectorizer=query_bv)
        if predict_class:
            classes = model.transform(batch_vectorizer=query_bv, pre-
dict_class_id=predict_class)
        if save_theta_q:
            thetaQ.transpose().to_hdf(filename + '.h5', 'table')
        if predict_class is not None and save_classes:
            classes.sort_values(classes.columns[0]).to_csv(filename +
'_classes.csv', sep='\t', encoding='utf-8')

    def get_tokens(model, flname='res.txt'):
        tokens = model.score_tracker['top_tok'].last_tokens
        with open(flname, 'w', encoding='utf-8') as fl:
            for topic_name in model.topic_names:
                print(topic_name, file=fl)
                for token in tokens[topic_name]:
                    print(token, end=' ', file=fl)
                print(file=fl)

    def draw_data(model, ConPur=False):
        perp_values = model.score_tracker['PerplAll'].value[1:]
        perp_values_w = model.score_tracker['PerplWords'].value[1:]
        sparsity_phi = model.score_tracker['SparPhiAll'].value
        sparsity_phi_main = model.score_tracker['SparPhiMain'].value
        sparsity_phi_bckgr = mod-
el.score_tracker['SparPhiBckgr'].value
        sparsity_theta = model.score_tracker['SparThetAll'].value
        sparsity_theta_main = mod-
el.score_tracker['SparThetMain'].value
        sparsity_theta_bckgr = mod-
el.score_tracker['SparThetBckgr'].value

        #plt.ion()
        # Perp
        fig, axs = plt.subplots(3, 2)
        fig.suptitle('Перплексия')
        axs[0, 0].set_title('All', fontsize=10)
        axs[0, 1].set_title('Words', fontsize=10)
```

```
        for i in range(3):
            axs[i, 0].plot(range(len(perp_values[i * 5:])),
perp_values[i * 5:])
            axs[i, 0].grid(True)
            axs[i, 1].plot(range(len(perp_values_w[i * 5:])),
perp_values_w[i * 5:])
            axs[i, 1].grid(True)
        plt.show()
        # SparPhi

        fig, axs = plt.subplots(3)
        fig.suptitle('SparPhi')
        axs[0].plot(range(len(sparsity_phi)), sparsity_phi)
        axs[1].plot(range(len(sparsity_phi_main)), sparsity_phi_main)
        axs[2].plot(range(len(sparsity_phi_bckgr)),
sparsity_phi_bckgr)
        for i in range(3):
            axs[i].grid(True)
        plt.show()
        # SparTheta
        fig, axs = plt.subplots(3)
        fig.suptitle('SparTheta')
        axs[0].plot(range(len(sparsity_theta)), sparsity_theta)
        axs[1].plot(range(len(sparsity_theta_main)),
sparsity_theta_main)
        axs[2].plot(range(len(sparsity_theta_bckgr)),
sparsity_theta_bckgr)
        for i in range(3):
            axs[i].grid(True)
        plt.show()
        if ConPur:
            contrast = mod-
el.score_tracker['TopicKernelScore'].average_contrast
            purity = mod-
el.score_tracker['TopicKernelScore'].average_purity
            plt.figure(num=4)
            plt.plot(range(len(contrast)), contrast)
            plt.title('Contrast')
            plt.grid(True)
            plt.figure(num=5)
            plt.plot(range(len(purity)), purity)
            plt.title('Purity')
            plt.grid(True)
            plt.show()

    try:
        if len(glob.glob(os.path.join(BATCHES, '*.batch'))) == 0:
            start_bv = time.time()
            bv = artm.BatchVectorizer(data_path=DATA_PATH,
                                      data_format='vowpal_wabbit',
                                      batch_size=1000,
                                      target_folder=BATCHES,
                                      gather_dictionary=True)
            end_bv = time.time() - start_bv
            print('Batches in {0:.0f}m{1:.0f}s'.format((end_bv // 60)
% 60, end_bv % 60, ))
```

```
            dictionary = bv.dictionary
        else:
            bv = artm.BatchVectorizer(data_path=BATCHES,
                                      data_format='batches',
                                      gather_dictionary=True)
            dictionary = artm.Dictionary()
            dictionary.load(dictionary_path=os.path.join(BATCHES,
'dictionary.dict'))

        if not os.path.isfile(os.path.join(BATCHES,
'dictionary.dict')):
            dictionary.gather(data_path=bv.data_path)
            dictionary.save(dictionary_path=os.path.join(BATCHES,
'dictionary.dict'))

        topics = ['topic_' + str(i) for i in range(100)]
        background_topics = topics[:20]
        main_topics = topics[20:]
        doc_passes = 5

        model = artm.ARTM(topic_names=topics,
                          cache_theta=True,
                          dictionary=dictionary,
                          show_progress_bars=True,
                          num_document_passes=doc_passes,
                          class_ids={'words': 1.0, 'hubs': 10.0},
                          theta_columns_naming='title',
                          theta_name='Habr_theta')

        ## Regularaizers
        # SSPhi

'''model.regularizers.add(artm.SmoothSparsePhiRegularizer(name='SSPhiM
ain', topic_names=main_topics, tau=0.01))
        model.regularizers.add(
            artm.SmoothSparsePhiRegularizer(name='SSPhiBckgr', top-
ic_names=background_topics, tau=0.015))
        # SSTheta
        mod-
el.regularizers.add(artm.SmoothSparseThetaRegularizer(name='SSThetaMai
n', topic_names=main_topics, tau=0.01))
        model.regularizers.add(
            artm.SmoothSparseThetaRegularizer(name='SSThetaBckgr',
topic_names=background_topics, tau=0.015))
        # DecorrelatorPhi
        model.regularizers.add(

artm.DecorrelatorPhiRegularizer(name='DecorrelatorPhiMain', top-
ic_names=main_topics, tau=1e+3))
        model.regularizers.add(

artm.DecorrelatorPhiRegularizer(name='DecorrelatorPhiBckgr', top-
ic_names=background_topics, tau=1e+2))

        ## Scores
        # Perplexity
```

```
        model.scores.add(artm.PerplexityScore(name='PerplAll', dic-
tionary=bv.dictionary))
        model.scores.add(artm.PerplexityScore(name='PerplWords',
class_ids=['words'], dictionary=bv.dictionary))
        # SparPhi
        model.scores.add(artm.SparsityPhiScore(name='SparPhiAll',
class_id='words'))
        model.scores.add(artm.SparsityPhiScore(name='SparPhiMain',
topic_names=main_topics, class_id='words'))
        model.scores.add(artm.SparsityPhiScore(name='SparPhiBckgr',
topic_names=background_topics, class_id='words'))
        # SparThet
        model.scores.add(artm.SparsityThetaScore(name='SparThetAll'))
        model.scores.add(artm.SparsityThetaScore(name='SparThetMain',
topic_names=main_topics))
        mod-
el.scores.add(artm.SparsityThetaScore(name='SparThetBckgr', top-
ic_names=background_topics))
        # Other
        model.scores.add(artm.TopTokensScore(name='top_tok',
num_tokens=10, class_id='words'))
        mod-
el.scores.add(artm.TopicKernelScore(name='TopicKernelScore',
                                                class_id='words',
                                                top-
ic_names=main_topics,
                                                probabil-
ity_mass_threshold=0.5))'''

        ## Fitting
        model.fit_offline(batch_vectorizer=bv,
num_collection_passes=15)
        model.dump_artm_model('250_topics_15itr')
        #model1=artm.load_artm_model('250_topics_15itr')
        get_tokens(model, 'res1.txt')
        get_query_data(model, predict_class=None, query_file=QUERY,
save_theta_q=True,
                       save_classes=True, filename='thetaQ_store')

        '''model.regularizers['SSThetaMain'].tau = 0
        model.regularizers['SSPhiMain'].tau = 0
        model.regularizers['DecorrelatorPhiMain'].tau = 1e+4

        model.fit_offline(batch_vectorizer=bv,
num_collection_passes=20)
        model.dump_artm_model('250_topics_35itr')
        get_tokens(model, 'res2.txt')

        alpha_iter = [(10 - x) / 10 for x in range(doc_passes)]
        alpha_iter.reverse()
        model.regularizers['SSThetaMain'].tau = -0.01
        model.regularizers['SSThetaMain'].alpha_iter = alpha_iter
        model.regularizers['SSPhiMain'].tau = -0.01
        mod-
el.regularizers.add(artm.ImproveCoherencePhiRegularizer(name='ImpCohPh
i',
```

```
tau=1.5,

class_ids='words',

dictionary=dictionary))

          model.fit_offline(batch_vectorizer=bv,
num_collection_passes=35)
          model.dump_artm_model('250_topics_70itr')

          save_phi_theta(model, save_phi=False)
          get_tokens(model)
          draw_data(model, ConPur=True)'''


      except:
          print("Unexpected error")
          raise
```

Модуль new_genetic_algo.py:

```
      import pandas as pd
      from math import sqrt
      import numpy as np
      import copy
      from collections import defaultdict

      QUERY_PATH = r'thetaQ_store.h5'
      THETA_PATH = r'theta_store.h5'


      def cos_distance(x, y):
          return np.sum(x * y) / (np.sqrt(np.sum(x ** 2)) *
np.sqrt(np.sum(y ** 2)))


      class GeneticEvolution:  # генетический алгоритм

          def __init__(self, select_size, mutation_eps):
              self.select_size = select_size  # сколько выводить доку-
ментов
              self.mutation_eps = mutation_eps  # вероятность мутации

          def initialize(self, ideal, values, population_eps,
                         good_power=1.0, bad_power=1.0, cast_value=1):
              self.index = ideal > population_eps  # взять значения,
большие порогового значения
              self.cast_value = cast_value  # на сколько умножить зна-
чения популяции
              self.values = values[:, self.index] * cast_value
              self.good_power = good_power  # усиление хороших докумен-
тов
              self.bad_power = bad_power  # усиление плохих документов
              self.good_values = np.array([ideal[self.index] *
cast_value])
              self.bad_values = np.empty((0, self.values.shape[1]),
float)
```

```
            self.check_list = set()
            self.IDEAL = ideal[self.index].copy() * cast_value

        def evolute(self, good_values, bad_values, checked):  #
эволюция
            self.good_values = np.vstack((self.good_values,
good_values))
            self.bad_values = np.vstack((self.bad_values,
bad_values))
            self.check_list |= checked

            def func(x):  # функция приспособленности
                def cos_dist(matrix):  # функция косинусной меры
                    if not matrix.size:
                        return 0
                    return np.sum(np.sum(matrix * x, axis=1) /
                                  (np.sqrt(np.sum(matrix ** 2, ax-
is=1)) * np.sqrt(np.sum(x ** 2))))

                def inv_cos_dist(matrix):
                    if not matrix.size:
                        return 0
                    return np.sum((np.sqrt(np.sum(matrix ** 2, ax-
is=1)) * np.sqrt(np.sum(x ** 2))) /
                                  np.sum(matrix * x, axis=1))

                # return cos_dist(self.good_values) +
inv_cos_dist(self.bad_values)
                return self.good_power * cos_dist(self.good_values) -
\
                       self.bad_power * cos_dist(self.bad_values)

            self.crossing_over()  # сделать кроссинговер и мутацию
            self.mutate()
            selected = [(func(x), i) for i, x in enumer-
ate(self.values) if
                        i not in self.check_list]  # посчитать для
всех функцию приспособленности
            selected.sort(reverse=True)  # отсортировать
            _, indices = zip(*selected)
            #print([cos_distance(self.IDEAL, self.values[i]) for i in
indices[:self.select_size]])
            #print(indices[:self.select_size])
            return indices[:self.select_size]  # вывести столько,
сколько нужно

            # return selected[:self.select_size]

        def mutate(self):  # мутация
            if np.random.random() < self.mutation_eps:  # если
случайное число меньше вероятности мутации
                x = np.random.randint(self.good_values.shape[0])  #
случайно найти x и y
                y = np.random.randint(self.good_values.shape[1])
                self.good_values[x][y] *= np.random.random()  # поме-
нять значение (x,y) в популяции
```

```
        def crossing_over(self):  # кроссинговер
            point = np.random.randint(self.good_values.shape[1])  #
найти точку разделения
            for x, y in zip(self.good_values[:-1:2],
                            self.good_values[1::2]):  # поменять в
этой точке части пары рядом стоящих документов
                dy, dx = y[:point].copy(), x[:point].copy()
                x[:point], y[:point] = dy, dx


    class GeneticManager(GeneticEvolution):  # разделение на плохие и
хорошие документы
        def initialize(self, ideal, values, real_indices,
                       population_eps, good_power=1.0, bad_power=1.0,
cast_value=1):
            super().initialize(ideal, values, population_eps,
                               good_power, bad_power, cast_value)  #
взять параметры из прошлого класса
            real_indices_inverse = {c: i for i, c in enumer-
ate(real_indices)}  # документ:счетчик
            self.get_real_index = real_indices.__getitem__  # вернуть
номера документов
            self.get_inverse_index = real_indices_inverse.__getitem__
# вернуть счетчики
            self.good_document = set()
            self.bad_document = set()

        def evolute(self, good_document, bad_document):
            good_document = [self.get_inverse_index(x) for x in
good_document]
            bad_document = [self.get_inverse_index(x) for x in
bad_document]
            good_document = list(filter(lambda x: x not in
self.good_document, good_document))
            self.good_document.update(good_document)
            bad_document = list(filter(lambda x: x not in
self.bad_document
                                       and x not in
self.good_document, bad_document))
            self.bad_document.update(bad_document)
            good_values = np.array(self.values[good_document])  #
взять значения хороших и плохих документов
            bad_values = np.array(self.values[bad_document])
            indices = super().evolute(good_values, bad_values,
                                      set(good_document +
bad_document))  # прогнать генетический алгоритм
            return [self.get_real_index(x) for x in indices]  #
превратить в номера документов


    if __name__ == "__main__":
        popul = pd.read_hdf(THETA_PATH, 'table')
        index = popul.index
        # ideal = pd.read_hdf(QUERY_PATH, 'table').values[0][20:]
        values = popul.values[:, 20:]
```

```
        ideal = values[3049].copy()
        g = GeneticManager(select_size=100, mutation_eps=0.2)
        g.initialize(ideal, values, index,
                     population_eps=0, good_power=1, bad_power=1,
cast_value=100)
        good_document = []
        bad_document = []
        while True:  # пока не нажали выход, происходит эволюция
            doc = g.evolute(good_document, bad_document)
            print(doc)
            s = input('Введите номера подходящих статьей через про-
бел:')
            good_document = s.split()
            bad_document = doc
            s = input('Введите q, чтобы выйти:')
            if s == 'q':
                break
```

Модуль auto_test.py:

```
    import pandas as pd
    from new_genetic_algo import QUERY_PATH, THETA_PATH,
GeneticManager
    import matplotlib.pyplot as plt
    import numpy as np
    if __name__ == "__main__":
        popul = pd.read_hdf(THETA_PATH, 'table')
        index = popul.index
        values = popul.values[:, 20:]
        ideal1 = values[3049].copy()
        ideal2 = values[47941].copy()
        ideal3 = values[3398].copy()
        g = GeneticManager(select_size=30, mutation_eps=0.2)
        g.initialize(ideal1, values, index,
                     population_eps=0, good_power=1, bad_power=0.5,
cast_value=100)
        remaining_documents = {'110247','224175', '154851', '231287',
'145176', '225187', '134006', '185234', '123966', '142976', '238327',
                               '251409', '172491', '249305',
'230857', '253266', '202866', '116945', '116830', '187594', '212341',
                               '226987', '231525', '245273',
'235409', '200354', '179723', '247509', '165787', '243471', '194514',
                               '200878', '243953', '219801',
'176767',
                               '191898', '222295', '137339',
'237173', '167315', '195830', '120931', '226297', '118482', '213097',
                               '200874', '173109', '251165',
'117513', '112272', '188092', '254761', '176643', '185004', '141476',
                               '218365', '215141', '189676',
'189678', '124322', '216629', '176919', '203884', '235917', '114089',
                               '215693', '190102', '203846',
'166469', '165731', '190230',
                               '136705', '128511', '169189',
'244423', '120772', '143517', '144617', '254527', '132126', '255133',
                               '111513', '133220', '165105',
'183846', '190428', '233265', '154193', '131889', '151360', '158453',
                               '196246', '164035', '115122',
'230595', '199258', '150143', '236567', '237101', '213193', '209910',
```

```
                                '226897', '182780', '220689', '191586'
                               }
        taken_documents = set()
        N = 10
        good_document = []
        bad_document = []
        cum_sum = [0]
        for i in range(N):
            print(i)
            doc = g.evolute(good_document, bad_document)
            good_document = []
            k = 0
            for x in doc:
                if x in remaining_documents:
                    k += 1
                    remaining_documents.remove(x)
                    taken_documents.add((x, i))
                    good_document.append(x)
            cum_sum.append(cum_sum[-1] + k)
            bad_document = doc
            if not remaining_documents:
                break
        print('cum_sum:', cum_sum[1:])
        print('Remaining count:', len(remaining_documents))
        print('Remaining documents:', list(remaining_documents))
        print('Taken count:', len(taken_documents))
        print('Taken documents:', list(taken_documents))
        print('P = {:.2f}'.format(len(taken_documents)/
(len(remaining_documents) + len(taken_documents))))
```