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,

_____
«____»_____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А. Замышляева
«____»_____2019 г.

Разработка и исследование локальных алгоритмов разведочного поиска по тематическим моделям

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2019.173.ПЗ ВКР

Руководитель работы, к.т.н.,
доцент кафедры ПМиП
_____/Т.Ю. Оленчикова
«____»_____2019 г.

Автор работы
Студент группы ЕТ-412
_____/С.Е. Кащеев
«____»_____2019 г.

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

Челябинск
2019

АННОТАЦИЯ

Кащеев С.Е. Разработка и исследование локальных алгоритмов разведочного поиска по тематическим моделям. – Челябинск: ЮУрГУ, ЕТ-412, 65 с., 2 табл., 12 ил., библиогр. список – 21 наим., 1 прил.

В работе исследуются методы построения тематических моделей и реализация разведочного информационного поиска с их использованием. Решается задача повышения релевантности поисковой выдачи за счет учета предпочтений пользователя. В качестве решения задачи разведочного поиска предлагаются два метода – наивная байесовская классификация и предсказание релевантности с помощью логистической регрессионной модели.

Написана программа для построения тематической модели на основе подхода аддитивной регуляризации, разработан программный модуль, реализующий построенные алгоритмы поиска и учета предпочтений пользователя. На примере конкретных запросов проведено экспериментальное исследование описанных методов.

Полученные алгоритмы повышают релевантность поиска, а реализующий их программный модуль может быть подключен к другим задачам и проектам.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В условиях все большего увеличения объемов информации и ускорения темпов ее прироста, все более актуальной становится задача поиска новых способов обработки, анализа и хранений данных. Одним из возможных решений выступает тематическое моделирование – способ построения модели коллекции текстовых документов, которая определяет, к каким темам относится каждый из документов. Цель построения вероятностной тематической модели заключается в автоматическом извлечении тем из коллекции текстовых документов. При этом происходит поиск информации о том, какими терминами описывается каждая тема, и каким темам принадлежит каждый из документов коллекции [1].

Данная задача имеет некоторое сходство с задачей кластеризации документов. Однако при кластеризации каждый документ целиком соотносится с одним конкретным кластером, в то время как в тематической модели осуществляется мягкая кластеризация (альтернативно, нечеткая кластеризация), допускающая принадлежность документа нескольким кластерам-темам. Такой подход позволяет обойти проблему многозначности и синонимии слов.

К числу конкретных приложений тематического моделирования можно отнести: анализ коллекций научных статей, анализ новостных потоков, рубрикация коллекций изображений, видео, музыки, коллаборативную фильтрацию, аннотацию генома и другие задачи биоинформатики [2].

Отдельно выделяется применение к задаче реализации системы «разведочного поиска». В стандартных поисковых системах поиск осуществляется по короткому запросу из набора ключевых слов, что предполагает наличие у пользователя достаточного конкретного представления о том, что он желает найти. Разведочный поиск же в качестве запроса может принимать отрывок текста, документ целиком или коллекцию документов, а в качестве результа-

та должен выдавать наиболее близкие по тематике и актуальные на сегодняшний день данные.

Предполагается, что такой поиск мог бы быть особенно полезен для научной, исследовательской и инженерной сфер, где часто возникает необходимость получить наиболее релевантную информацию по какой-либо заданной теме.

Одной из проблем, с которой приходится сталкиваться при реализации подобной системы поиска является персонализация поисковой выдачи под нужды пользователя. В общем случае единственным рычагом контроля над результатом поиска выступает только сам документ-запрос, однако далеко не во всех случаях такой документ идеально соответствует тематикам, которые пользователь хотел бы получить в результате поиска. Как следствие, важную роль обретает наличие у пользователя способов коррекции выдачи и ее подстройка под некие дополнительные критерии.

Задача тематического моделирования в общем случае является некорректно поставленной и обладает бесконечным множеством решений. Наиболее популярные алгоритмы построения тематических моделей PLSA (Probabilistic latent semantic analysis) [3] и LDA (Latent Dirichlet Allocation) [4] на выходе выдают одно из этих решений. Работа посвящена разработке программного модуля, реализующего локальные алгоритмы разведочного поиска по тематическим моделям и улучшения релевантности поисковой выдачи посредством учета предпочтений пользователя.

Первый раздел посвящен тематическому моделированию. Описаны общие принципы и методы построения тематических моделей, сделан обзор существующих пакетов ПО, реализующих эти методы. На основе проведенного обзора принято решение использовать подход ARTM и реализующую его библиотеку BigARTM. Также была сформулирована постановка задачи для данной работы.

1 МЕТОДЫ ТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ

1.1 Математическая модель тематического моделирования

1.1.1 Терминология и базовые предположения

Исходными данными для тематического моделирования является множество (коллекция) текстовых документов и множество (словарь) всех употребляемых в них термов [5]. Термами могут выступать слова, нормальные формы слов, или определенные термины и словосочетания (граммы) – это зависит от того, как и какими методами осуществилась предварительная обработка исходных текстов. Каждый документ из множества всех документов представляется вектором слов (термов) ( ) из словаря , где – это длина документа. Каждый термин может повторяться в документе несколько раз.

Под термином «тема» подразумевается множество признаков, принадлежащих одной и той же предметной области. При этом каждый документ может принадлежать более чем одной теме, а каждый признак, в свою очередь, может принадлежать произвольному числу тем.

Вероятностные тематические модели основаны на следующих предположениях [6]:

1) порядок документов в коллекции не имеет значения;

2) порядок слов в документе не имеет значения, документ – мешок слов;

3) слова, встречающиеся часто в большинстве документов, не важны для определения тематики;

4) коллекцию документов можно представить как выборку пар документ-слово ;

5) каждая тема описывается неизвестным распределением на множестве слов ;

6) Каждый документ описывается неизвестным ем на множестве тем ;

7) Гипотеза условной независимости .

1.1.2 Принцип максимума правдоподобия

В математической статистике для оценки неизвестных параметров вероятностной модели по наблюдаемым данным используется принцип максимума правдоподобия. Согласно этому принципу, выбираются такие значения параметров, которые обеспечивают наибольшую правдоподобность наблюдаемой выборки.

Для нахождения параметров модели , выполняется максимизация функции правдоподобия, определяемой как зависимость вероятности выборки от параметров модели:

(1.1)

Переход от произведения к сумме осуществляется с помощью логарифмирования, после отбрасываются слагаемые, не зависящие от параметров модели. В результате получается задача максимизации log-правдоподобия:

, (1.2)

, ,

, .

При этом устанавливаются ограничения неотрицательности

и нормировки

1.1.3 Частотные оценки условных вероятностей

В пространстве (× ×) вероятности, связанные с переменными и , можно оценивать по выборке как частоты (будем обозначать частотные оценки через :

—, —, —, —, (1.3)

– длина документа ;

– число вхождений терма в документ ;

– число вхождений терма во все документы коллекции;

– длина коллекции.

Вероятности, выражающиеся через скрытую переменную *t*, аналогично можно оценивать как частоты, если рассматривать коллекцию документов как выборку троек (*d*,*w*,*t*):

—, —, —, —, (1.4)

где – число троек, связанных с темой *t*;

– число троек, в которых терм *w* документа *d* относится к теме *t*;

– число троек, в которых терм *w* связан с темой *t*;

– число троек, в которых терм *w* документа *d* относится к теме *t*.

При частотные оценки (1.3) – (1.4) стремятся к соответствующим вероятностям *p* в соответствии с законом больших чисел.

1.1.4 Перплексия в тематических моделях

Для оценки качества работы алгоритмов построения тематических моделей в общем случае наиболее часто используется перплексия:

– – . (1.5)

Для ее расчета обычно используется некоторая выборка контрольных данных , не входящих в обучающую выборку. Каждый документ произвольно делится на две части и . Оценка параметров осуществ-

ляется по обучающей выборке , параметры оценивается по . Перплексия же подсчитывается по из контрольной выборки [7].

В тематическом моделировании численное значение перплексии интерпретируется следующим образом. Если каждый документ генерируется из множества всех термов , то перплексия сходится к его мощности .

Таким образом, перплексия является зависимой от размера словаря и распределения частоты слов в коллекции , из чего следует ряд ее недостатков [7]:

1) невозможность проведения сравнения между моделями с различной n-граммностью;

2) невозможность проведения сравнения различных методов разреживания словаря;

3) невозможность оценки качества удаления стоп-слов и слов, не участвующих в формировании тем.

Важной характеристикой тематической модели является способность к обобщению. Говорят, что модель переобучается, если перплексия контрольной выборки существенно превышает перплексию обучающей выборки. На данный момент не существует адекватных теоретических оценок переобучения вероятностных тематических моделей, в силу чего их качество принято оценивать и сравнивать эмпирически.

1.1.5 Методы построения тематических моделей

1.1.5.1 Латентно-семантический анализ LSA

Метод обработки информации на естественном языке, анализирующий взаимосвязь между библиотекой документов и терминами, в них встречающимися, и выявляющий характерные факторы (тематики), присущие всем документам и терминам.

В основе метода латентно-семантического анализа лежат принципы факторного анализа, в частности, выявление латентных связей изучаемых явлений или объектов. При классификации кластеризации документов этот ме-

тод используется для извлечения контекстно-зависимых значений лексических единиц при помощи статистической обработки больших корпусов текстов.

В области информационного поиска данный подход называют латентно-семантическим индексированием (ЛСИ) [3].

ЛСА можно сравнить с простым видом нейросети, состоящей из трех слоев: первый слой содержит множество слов (термов), второй – некое множество документов, соответствующих определенным ситуациям, а третий, средний, скрытый слой представляет собой множество узлов с различными весовыми коэффициентами, связывающих первый и второй слои.

В качестве исходной информации ЛСА использует матрицу термы-на-документы, описывающую набор данных, используемый для обучения системы. Элементы этой матрицы содержат, как правило, веса, учитывающие частоты использования каждого терма в каждом документе и участие терма во всех документах (TF-IDF). Наиболее распространенный вариант ЛСА основан на использовании разложения матрицы по сингулярным значениям (SVD – Singular Value Decomposition). С помощью SVD-разложения любая матрица раскладывается во множество ортогональных матриц, линейная комбинация которых является достаточно точным приближением к исходной матрице.

Согласно теореме о сингулярном разложении [8], любая вещественная прямоугольная матрица может быть разложена на произведение трех матриц:

, (1.6)

где и – ортогональные матрицы, – диагональная матрица, значения на диагонали которой называются сингулярными значениями матрицы .

Такое разложение обладает особенностью: если в матрице оставить только наибольших сингулярных значений, а в матрицах и – только соответствующие этим значениям столбцы, то произведение получившихся матриц будет наилучшим приближением исходной матрицы к матрице ранга :

. (1.7)

Основная идея латентно-семантического анализа состоит в том, что если в качестве матрицы использовалась матрица термы-на-документы, то матрица , содержащая только первых линейно независимых компонент , отражает основную структуру различных зависимостей, присутствующих в исходной матрице. Структура зависимостей определяется весовыми функциями термов.

Таким образом, каждый терм и документ представляются при помощи векторов в общем пространстве размерности (так называемом пространстве гипотез). Близость между любой комбинацией термов и/или документов легко вычисляется при помощи скалярного произведения векторов.

Как правило, выбор зависит от поставленной задачи и подбирается эмпирически. Если выбранное значение слишком велико, то метод теряет свою мощность и приближается по характеристикам к стандартным векторным методам. Слишком маленькое значение не позволяет улавливать различия между похожими термами или документами.

Существуют три основных разновидности решения задачи методом латентно-семантического анализа:

1) сравнение двух термов между собой;

2) сравнение двух документов между собой;

3) сравнение терма и документа.

Достоинства метода:

1) метод является наилучшим для выявления латентных зависимостей внутри множества документов;

2) метод может быть применен как с обучением, так и без обучения (например, для кластеризации);

3) используются значения матрицы близости, основанной на частотных характеристиках документов и лексических единиц;

4) частично снимается полисемия и омонимия.

Недостатки:

1) значительное снижение скорости вычисления при увеличении объема входных данных (например, при SVD-преобразовании) [9]. Скорость вычисления соответствует порядку , где – сумма количества документов и термов, – размерность пространства факторов;

2) вероятностная модель метода не соответствует реальности. Предполагается, что слова и документы имеют нормальное распределение, хотя ближе к реальности распределение Пуассона. В связи с этим для практических применений лучше подходит вероятностный латентно-семантический анализ, основанный на полиномиальном распределении.

1.1.5.2 Вероятностные тематические модели

Вероятностное тематическое моделирование представляет собой набор алгоритмов, позволяющих анализировать слова в больших наборах документов и извлекать из них темы, связи между темами [10].

Документы исходной выборки представляется в виде мешка слов, порядок которых не имеет значения. Каждый документ определяется распределением составляющих его слов по множеству тем , то есть для каждой темы имеется величина, характеризующая вероятность присутствия данной темы в данном документе.

Темы представляются в виде распределения на множестве словаря, то есть для каждого слова по результатам обучения определяется вероятность его принадлежности данной теме.

Вероятностные модели являются генеративными, то есть могут быть использованы для генерации документов. Но главная цель тематического моделирования заключается в извлечении тем из уже имеющегося набора документов, а не генерация. Таким образом, получаем задачу, обратную генерации: выявить наиболее вероятные скрытые структуры (темы и их распределение), с помощью которых мог быть сгенерирован исходный набор данных.

1.1.5.3 Вероятностный латентно-семантический анализ PLSA

PLSA (Probabilistic Latent Semantic Analysis) является одним из распространённых инструментов вероятностного тематического моделирования. Впервые метод был предложен Томасом Хоффманом в 1999 году в работе [3]. За основу метод берет аспектную модель (aspect model), связывающую между собой латентные (скрытые) переменные тем и наблюдаемые переменные термов и документов .

Задача определяется как поиск и выявление этих латентный переменных. Каждый документ при этом представлен в виде числового вектора и может относиться к множеству разных тем с некоторой вероятностью для каждой из них. В этом состоит основная особенность PLSA, отличающая ее от методов, основанных не на вероятностном моделировании.

После задания определенного количества тем в качестве желаемого параметра модели метод PLSA позволяет оценить ряд величин:

1) – вероятность того, что слово будет находиться в случайно выбранном из коллекции документе ;

2) – вероятность того, что документ это наиболее тесно связанный с темой документ;

3) – вероятность того, что слово наиболее тесно связанно с темой .

Вероятностную модель порождения документов возможно описать следующим образом:

1) случайным образом выбирается документ с вероятностью ;

2) случайным образом выбирается тема с вероятностью ;

3) случайным образом выбирается терм с вероятностью .

В результате получаем пару ( ). Вероятностная модель тогда задается следующим образом:

(1.8)

Отдельно стоит отметить, что модель также можно записать двумя другими способами:

(1.9)

где и являются распределением тем и слов по всей коллекции документов соответственно.

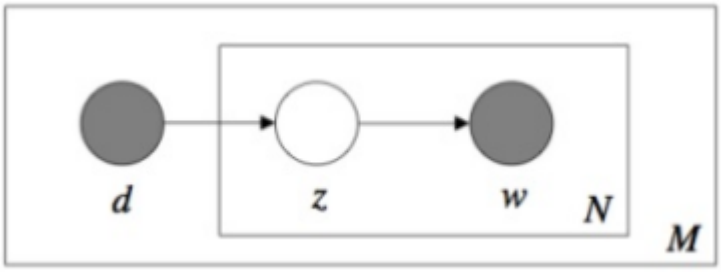

Рисунок 1.1 – Модель PLSA

Вершины графа на рисунке соответствуют переменным, а соединяющие их ребра – вероятностным взаимоотношениям между ними. Закрашенные вершины обозначают наблюдаемые переменные, не закрашенная вершина – латентная переменная тем. является множеством всех документов, а – множеством слов в каждом отдельном документе. Направленность ребер означает отношение условной зависимости вершин, на которые указывают ребра, от вершин, из которых ребра исходят.

PLSA обладает более гораздо более лучшим статистическим обоснованием, нежели LSA, основанный на методах линейной алгебры.

К недостаткам PLSA относят:

1) склонность к переобучению при работе с большой коллекций текстов в виду линейного возрастания числа параметров и ;

2) необходимость перестройки всей модели при добавлении в исходную коллекцию нового документа;

3) значительное снижение скорости сходимость модели на больших коллекциях документов, так как обновляются после каждого прохода коллекции;

4) алгоритм не предусматривает специальной обработки общеупотребительных слов. Как результат, снижается интерпретируемость выделенных моделью тем;

5) PLSA не позволяет управлять ни разреженностью ни разреженностью ;

6) необходимость хранить трехмерную матрицу .

1.1.5.4 Байесовская регуляризация

В байесовском подходе используется принцип максимума апостериорной вероятности (maximuma posteriori probability; MAP), в рамках которого предполагается, что параметры модели являются случайными, но подчиняются некоторому априорному распределению где – неслучайный гиперпараметр. Формула максимизация совместного правдоподобия в этом случае обретает следующий вид:

(1.9)

После логарифмирования получается модификация задачи (1.2), с логарифмом априорного распределения в роли регуляризатора:

(1.10)

Для снижения размерности задачи и риска переобучения в байесовском подходе может применяться принцип неполного правдоподобия, в котором гиперпараметры оптимизируются с помощью интегрирования параметров . Это возможно в силу того, что размерность вектора не зависит от объема исходной коллекции и, чаще всего, много меньше матриц

В байесовском подходе оцениваются не сами параметры а их апостериорное распределение . Но в практических приложения в этом нет необходимости, так как полученное распределение нужно только

для того, чтобы затем вернуться к точечным оценкам математического ожидания.

Несмотря на доминирование байесовского подхода в литературе по тематическому моделированию, он представляет скорее академический интерес, нежели практический метод.

Объясняется это тем, что 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.

Байесовские модели также чрезвычайно сложны для понимания, вывода и сравнения, отсутствует возможность их комбинирования и взаимной замены. При этом модели, опирающиеся на байесовский вывод, часто не обладают достаточными лингвистическими обоснованиями и подвержены проблемам неединственности и неустойчивости.

В силу этих причин в практических приложениях выбор метода построения тематической модели, как правило, происходит в пользу других, более простых моделей.

1.1.5.5 Латентное размещение Дирихле

LDA (Latent Dirichlet Allocation) – генеративная вероятностная модель, предложенная Дэвидом Блеем, Эндрю Ыном и Майклом Джорданом в 2003 году для решения проблемы переобучения PLSA, которая предсказывала вероятности распределения слов на новых заметно хуже, чем на обучающей выборке. На сегодняшний день LDA считается доминирующим методом в области вероятностного тематического моделирования и имеет огромное множество модифицированных вариантов, специализированных на решение более конкретных задач.

Процесс порождения документов в данной модели происходит независимо для каждого документа и схож с генеративным процессом из PLSA (рисунок 1.2):

1) случайным образом для документа выбирается его распределение по темам ;

2) для каждого слова в документе:

а) случайным образом выбирается тема из распределения , полученного на 1-м шаге;

б) случайным образом выбирается слово из распределения слов в выбранной теме .

Исходные данные состоят из набора документов, каждый из которых содержит слов. Слова являются единственными наблюдаемыми переменными в модели, все остальные переменные являются скрытыми. Переменная характеризует значение темы с шага 2а по отношению слову , а задает распределение тем для каждого документа .

выступает фиксированным входным параметром модели и равняется количеству тем. Оптимальное обычно подбирают эмпирическим способом, но существуют и подходы к автоматическому определению оптимального количества тем [11]. Переменная задает распределение слов в теме *k*.

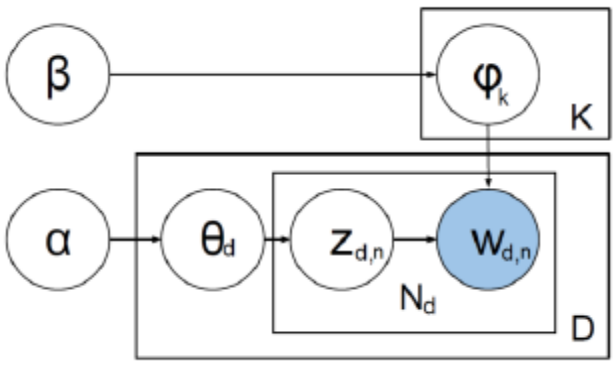

Рисунок 1.2 – Модель LDA

LDA тоже основана на разложении (1.1), но с дополнительным предположением, что столбы матриц и являются случайными векторами, по-

рожденными распределением Дирихле с параметрами и соответственно:

$$\_\_\_\_\_$$

$$\_\_\_\_\_$$

где – гамма-функция, а векторы и называются гиперпараметрами.

Распределение Дирихле способно порождать как плотные, так и разреженные векторы распределений. С уменьшением растет разреженность компонента в порождаемых векторах .

В процессе порождения данных из распределения Дирихле сперва 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 PLSA – несмещенные оценки максимума правдоподобия:

$$\_\_ \quad \_\_;$$

- в LDA – сглаженные байесовские оценки:

$$\_\_\_\_\_ \quad \_\_\_\_\_.$$

Недостатки распределения Дирихле:

1) отсутствие убедительных лингвистических обоснований;

2) байесовский вывод требует интегрирования по пространству параметров модели, которое элементарно только в простейшем варианте LDA;

3) попытки построения многоцелевых комбинированных моделей приводят к громоздким математическим выкладкам;

4) практические исследования показывают, что на больших коллекциях правдоподобие LDA и PLSA отличается незначительно [12].

Практическая распространенность метода в области задач тематического моделирования объясняется скорее математическим удобством и популярностью байесовского обучения. Байесовский вывод в данном случае значительно упрощен благодаря сопряженности распределения Дирихле к мультиноминальному распределению.

1.1.5.6 EM-алгоритм

Для решения оптимизационной задачи (1.2) в PLSA применяют итерационный алгоритм, в котором каждая итерация состоит их двух шагов – E (expectation) и M (maximization) [13].

Перед первой итерацией осуществляется выбор начального приближения параметров и . В простейшем случае их можно задать нормированными случайными векторами из равномерного распределения.

На E-шаге с помощью текущих и и формулы Байеса вычисляются условные вероятности всех тем для каждого терма в каждом документе :

———— ————. (1.11)

На M-шаге по условным вероятностям тем вычисляется новое приближение параметров , при помощи величины

. (1.12)

Суммирование по документам и по термам дает оценки , , , посредством которых, в соответствии с (1.4), можно получить частотные оценки условных вероятностей , :

—, , , (1.13)

—, , . (1.14)

1.1.6 Аддитивная регуляризация тематических моделей

ARTM (AdditiveRegularizationofTopicModels) – это подход к построению вероятностных тематических моделей, охватывающий PLSA, LDA и многие байесовские модели. ARTM позиционируют как альтернату байесовскому обучению тематических моделей [14].

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

Однако стохастическое матричное разложение не единственно и определенно с точностью до невырожденного преобразования (где матрицы и тоже стохастические), что означает наличие у него, в общем случае, бесконечного множества возможных решений. Таким образом, задача построения тематической модели является некорректно поставленной.

Для решения этой проблемы применяют регуляризацию, представляющую собой общий подход для решения обратных задач с некорректной постановкой [15]. В рамках данного подхода к основному критерию правдоподобия (1.2) добавляется дополнительный критерий, называемый регуляризатором:

(1.15)

(1.16)

Переменная здесь выполняет роль коэффициента регуляризации и напрямую задает параметр, определяющий, насколько важен данный регулятор, и насколько сильным должно быть его влияние на конечную модель.

Как результат, при решении задачи максимизации правдоподобия будет осуществляться максимизация нового критерия, что позволяет учитывать специфику решаемой задачи и получать более релевантные модели.

На практике в задачах обработки текстов зачастую используются сразу несколько регуляризаторов, каждый из которых накладывает свое ограничение на решение.

Подход ARTM обладает множеством преимуществ над типичным байесовским обучением и графическими моделями, в рамках которых, главным образом, до сих пор развивалось вероятностное тематическое моделирование. В их числе можно выделить:

1) относительная легкость понимания и вывода моделей, снижающая порог входа в область тематического моделирования для исследователей их смежных областей;

2) очень простой математический аппарат – для добавления регуляризатора достаточно лишь добавить его производные в формулы М-шага (1.6)–(1.7);

3) регуляризаторам в ARTM необязательно иметь вероятностную интерпретацию или быть априорными распределениями;

4) регулязатор Дирихле утрачивает свою особую роль, исчезает необходимость использовать его в каждой модели для всех тем;

5) легкость построения многоцелевых комбинированиях моделей посредством суммирования регуляторов, взятых из разных моделей.

1.1.6.1 Регуляризатор сглаживания

Данный регуляризор формализует требование о том, чтобы слова общей лексики, стоп-слова и специфические редко встречающиеся слова собрались в специальные фоновые темы и не засоряли предметные темы, что положительно сказывается на их интерпретируемости.

С его помощью осуществляется минимизация 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:

.

1.1.6.2 Регуляризатор разреживания

Естественным является предположение, каждый документ описывается небольшим числом тем из общего словаря коллекции . Значение вероятностей таких тем в распределении в таком случае следует увеличивать, в то время вероятности тем, не характеризующих документ, следует занулять.

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

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

В ARTM таким результатов можно добиться добавлением сглаживающего регуляризатора. Таким образом, осуществляется максимизация дивергенции между равномерным распределением и искомыми распределениями:

, (1.18)

где – коэффициенты регуляризации.

Формулы М-шага:

,

.

1.1.6.3 Регуляризатор сокращения незначимых тем

Формализует требование исключения из модели тем, ядро которых состоит из крайне малого количества редких слов. Регуляризатор вводит требование разреженности распределения тем во всей коллекции и максимизирует дивергенцию Кульбака-Лейблера между и равномерным распределением:

(1.19)

Формула М-шага принимает вид:

—

где – коэффициент регуляризации. В соответствии с этой формулой, вероятности темы понижаются для всех документов, если число отнесенных к ней слов мало. Регулязатор способен оптимизировать количество тем, если их изначально заданное число было заведомо избыточным.

1.1.6.4 Регуляризатор декоррелирования

Формализует требование к предметным темам, согласно которому темы должны как можно сильнее отличаться друг от друга. С помощью этого регуляризотора осуществляется минимизация суммы ковариаций между распределениями и для всех пар тем , :

(1.20)

Формулы М-шага:

,

,

где – коэффициент регуляризации. Декоррелирование способствует разрежеванию тем и выделению в них ядер, состоящих слов с доминирующей величиной вероятности . Регулязатор также обладает способностью группировать стоп-слова в отдельные фоновые темы [16].

1.2 Методы классификации и кластеризации

1.2.1 Наивный байесовский классификатор

Простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости.

В зависимости от точной природы вероятностной модели, наивные байесовские классификаторы могут обучаться очень эффективно. Во многих

практических приложениях для оценки параметров для наивных байесовых моделей используют метод максимального правдоподобия.

———————, (1.21)

———————. (1.22)

Достоинством наивного байесовского классификатора является малое количество данных необходимых для обучения, оценки параметров и классификации [17].

В основе NBC (Naïve Bayes Classifier) лежит теорема Байеса:

———————— (1.23)

где – вероятность, что документ принадлежит классу ;

– вероятность встретить документ среди всех документов класса ;

– безусловная вероятность встретить документ класса в корпусе документов;

– безусловная вероятность встречи документа в корпусе документов.

Цель классификации состоит в том, чтобы понять к какому классу принадлежит документ, поэтому нам нужна не сама вероятность, а наиболее вероятный класс. Байесовский классификатор использует оценку апостериорного максимума (maximum a posteriori estimation) для определения наиболее вероятного класса.

Для реализации байесовского классификатора необходимы:

1) относительные частоты классов в корпусе документов. То есть, как часто встречаются документы того или иного класса;

2) суммарное количество слов в документах каждого класса;

3) относительные частоты слов в пределах каждого класса;

4) размер словаря выборки;

5) количество уникальных слов в выборке.

Совокупность этой информации мы будем называть моделью классификатора. Затем на этапе классификации необходимо для каждого класса рассчитать значение следующего выражения и выбрать класс с максимальным значением:

(1.24)

где – количество документов в обучающей выборке принадлежащих классу ;

– общее количество документов в обучающей выборке;

– количество уникальных слов во всех документах обучающей выборки;

– суммарное количество слов в документах класса в обучающей выборке;

– сколько раз i-ое слово встречалось в документах класса в обучающей выборке;

– множество слов классифицируемого документа (включая повторы).

1.2.2 Логистическая регрессия

Применяется для прогнозирования вероятности возникновения некоторого события по значениям множества признаков. Для этого вводится так называемая зависимая переменная принимающая лишь одно из двух значений – как правило, это числа 0 (событие не произошло) и 1 (событие произошло), и множество независимых переменных – вещественных чисел, на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной.

1.2.3 Линейная регрессия

Используемая в статистике регрессионная модель зависимости одной (объясняемой, зависимой) переменной от другой или нескольких других пе-

ременных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

Пусть дана выборка объёмом n наблюдений переменных y и x. Обозначим – номер наблюдения в выборке. Тогда – значение переменной в -м наблюдении, – значение -го фактора в -м наблюдении. Соответственно, – вектор регрессоров в -м наблюдении. Тогда линейная регрессионная зависимость имеет место в каждом наблюдении:

(1.25)

Модель линейной регрессии можно представить в матричной форме:

,

где Y– вектор наблюдений зависимой переменой y,

X – матрица факторов,

ε – вектор случайных ошибок.

1.2.4 Полиномиальная линейная регрессия

Метод классификации, который обобщает логистическую регрессию для задач мультикласса, т. е. дискретных результатов. Эта модель, которая может быть использована для прогнозирования вероятности различных возможных результатов отдельных переменных (которые могут быть действительными, двоичными, категориальными и т. д.).

Полиномиальная логистическая регрессия используется, когда рассматриваемая зависимая переменная является номинальной (эквивалентно категориальной, что означает, что она попадает в любую из набора категорий, которые не могут быть упорядочены каким-либо значимым образом) и для которых существует более двух категорий.

Полиномиальная логистическая регрессия – это конкретное решение задач классификации, в которых для оценки вероятности каждого конкретно-

го значения зависимой переменной используется линейная комбинация наблюдаемых признаков и некоторых специфических для задачи параметров. Наилучшие значения параметров для данной проблемы обычно определяются из некоторых данных обучения (например, некоторых людей, для которых известны результаты диагностического теста и группы крови, или из некоторых примеров произносимых известных слов).

Полиномиальная логистическая модель предполагает, что данные зависят от конкретного случая, то есть каждая независимая переменная имеет одно значение для каждого случая. Полиномиальная логистическая модель также предполагает, что зависимая переменная не может быть точно предсказана из независимых переменных для любого случая. Как и в случае других типов регрессии, нет необходимости, чтобы независимые переменные были статистически независимы друг от друга (в отличие, например, от наивного байесовского классификатора); однако предполагается, что коллинеарность является относительно низкой, поскольку становится трудно различить влияние нескольких переменных, если это не так.

Если полиномиальный логит используется для моделирования выбора, он опирается на допущение независимости нерелевантных альтернатив, что не всегда желательно. Это предположение утверждает, что шансы на предпочтение одного класса другому не зависят от наличия или отсутствия других «нерелевантных» альтернатив.

Классификация на множество классов:

– , (1.26)

. (1.27)

1.3 Пакеты ПО для тематического моделирования

1.3.1 Gensim

Gensim представляет собой пакет инструментов с открытым исходным кодом для обработки коллекций текстов, реализованный в виде подключаемого модуля для языка программирования Python [18]. Проект берет свое начало с 2008 года, тогда он являлся небольшим набором скриптов для поиска похожих статей, разработанным для нужд Чешской Библиотеки Цифровой математики.

Gensim специально заточен под обработку больших объемов текстовых данных с помощью потоковых данных и эффективных инкрементальных алгоритмов. Проект позиционирует себя как один из самых робастных, эффективных и простых в использовании пакетов программного обеспечения для работы с задачами семантического моделирования текстов.

Gensim включает в себя, помимо прочего, реализации алгоритмов латентного семантического анализа, латентного распределения Дирихле и иерархического процесса Дирихле (Hierarchical Dirichlet Process, HDP). Для каждого из алгоритмов также реализованы его распределенные параллельные версии.

Пакет распространяется по лицензии GNU LGPLv2.1, допускающей свободное академическое и коммерческое использование.

1.3.2 Vowpal Wabbit (VW)

Является одной из наиболее широко используемых библиотек для работы с алгоритмами машинного обучения в индустрии. На текущий момент проект разрабатывается усилиями Microsoft Research, в прошлом был разработкой группы Yahoo! Research [19].

Отличается, прежде всего, гибкостью в выборе формата входных данных, высокой скоростью работы, хорошей масштабируемостью, поддержкой

большого количества различных режимов обучения и огромным множеством поддерживаемых алгоритмов, включая метод LDA.

Главной особенностью библиотеки считается возможность онлайн-обучения, что представляет особый интерес для задач, требующих обработки больших и высокоразмерных данных.

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

1.3.3 BigARTM

BigARTM [20] – это кроссплатформенная библиотека с открытым исходными кодом, специализированная на решении задач 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 с библиотекой на C++, Python и из командной строки.

Библиотека является программной реализацией подхода аддитивной регуляризации ARTM, благодаря чему способна к построенной практически любой 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. В библиотеке, помимо его классической оффлайн версии, реализован параллельный онлайновый EM-алгоритм, что позволяет обрабатывать большие коллекции документов за одну итерацию, минимизируя используемый объём оперативной памяти. Кроме того, EM-алгоритм в BigARTM обобщён 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 дру-

гими известными программными пакетами. Результаты эксперимента приведены в таблице 1.

Таблица 1 – Сравнение BigARTM с другими программными пакетами

Библиотека	Количество параллельных потоков	Затраченное время (мин.)	Перплексия
BigARTM	1	35	4000
Gensim.LdaModel	1	369	4161
VowpalWabbit.LDA	1	73	4108
BigARTM	4	9	4061
Gensim.LDAMulticore	4	60	4111
BigARTM	8	4,5	4304
Gensim.LDAMulticore	8	57	4455

На текущий момент BigARTM по умолчанию включены следующие метрики качества:

1) перплексия;
2) разреженность;
3) средняя чистота тем;
4) средняя контрастность тем;
5) средний размер лексического ядра тем;
6) 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, цитируемых документах, цитируемых авторов и т. д.

1.4 Постановка задачи

Целью выпускной квалификационной работы является разработка алгоритма для локального поиска с учетом предпочтений пользователя и его исследование.

Для достижения поставленной цели требуется:

1) разработать и реализовать алгоритм предобработки исходных данных: коллекции документов и документа-запроса;

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 алгоритм учета предпочтений пользователя и корректировки результатов запроса в соответствии с ними;

5) реализовать модуль для поиска документов по тематической модели с учетом предпочтений пользователя;

6) разработать методику экспериментального исследования алгоритмов для локального поиска;

7) сделать выводы на основе полученных результатов исследований.

1.5 Выводы по разделу

В первом разделе описаны общие принципы тематического моделирования и его математические обоснования, а также рассмотрены различные подходы к тематическому моделированию. Оказалось, что тематические модели, основанные на латентно-семантическом анализе (LSA), не подходят к решению данной задаче, так как вероятностная модель данного подхода не соответствует реальности. В основе LSA предполагается, что слова и документы имеют нормальное распределение, хотя ближе к реальности распределение Пуассона. В связи с этим для практических применений лучше подходит вероятностный латентно-семантический анализ (PLSA), основанный на полиномиальном распределении. Помимо этих двух подходов существует также байесовская регуляризация. Байесовские модели чрезвычайно сложны для понимания, вывода и сравнения, отсутствует возможность их комбинирования и взаимной замены, часто не обладают достаточными лингвистическими обоснованиями и подвержены проблемам неединственности и неус-

тойчивости. На помощь байесовскому методу приходит латентное размещение Дирихле. Практическая распространенность метода Дирихле в области задач тематического моделирования объясняется скорее математическим удобством и популярностью байесовского обучения. Байесовский вывод в данном методе значительно упрощен благодаря сопряженности распределения Дирихле к полиноминальному распределению, но не удобен для решения поставленной задачи.

В данной главе рассмотрены локальные алгоритмы разведочного поиска в тематических моделях, такие как наивный байесовский классификатор, полиномиальная, линейная, и логистическая регрессии. Наивный байесовский классификатор основан на строгих (наивных) предположениях о независимости и требует точных описаний природы вероятностной модели.

В качестве используемых в данной работе локальных алгоритмов разведочного поиска по тематической модели принято решение использовать логистическую регрессию, так как значением ее функции является вероятность принадлежности исходного значения к определенному классу, а не значение числовой переменной, как в случае с обычной линейной регрессией, и наивный байесовский классификатор, так как для его использования необходимо малое количество данных для обучения, оценки параметров и классификации.

Для подготовки исходных данных к обработке решено использовать формат Vowpal Wabbit, с помощью которого каждый документ из коллекции можно представить в виде «мешка слов», на котором достаточно легко исследовать и проанализировать работу локальных алгоритмов. «Мешок слов» отличается от обычной матрицы тем, что признаки можно разбить на категории, чтобы впоследствии при работе часть из них можно было «отключать».

2 РАЗРАБОТКА МАТЕМАТИЧЕСКОЙ МОДЕЛИ ЛОКАЛЬНЫХ АЛГОРИТМОВ КЛАССИФИКАЦИИ

2.1 Математическая модель поиска и классификации документов по запросу

Документ-запрос сперва необходимо подвергнуть предобработке, аналогично документам исходного корпуса текстов.

Затем для документа-запроса формируется вектор-распределение по темам, предопределенным в матрице распределения тем по документам модели . Данный вектор получается в результате выполнения EM-алгоритма только для запроса.

На E-шаге с помощью текущих и и формулы Байеса вычисляются условные вероятности всех тем для каждого терма в каждом документе :

(2.1)

На M-шаге по условным вероятностям тем вычисляется новое приближение параметров , при помощи величины:

(2.2)

Однако в отличие от используемого при построение 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.

По полученному вектору с помощью косинусной меры вычисляется близость запроса ко всем документам исходной коллекции. Формула косинусной меры имеет следующий вид:

(2.3)

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

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

2.2 Математическая модель модификации запроса с учетом предпочтений пользователя

В основе метода предугадывания лежит предположение о том, что размеченные пользователем документы можно использовать в качестве обучающего множества данных для построения регрессионной модели. Соответственно, все еще не просмотренные пользователем документы тогда можно рассматривать как тестовую выборку. Для каждого документа из тестового набора данных модель способна сделать предсказание о том, к какому классу и с какой вероятностью он относится. Исходными данными здесь выступают вектора распределений тем по документам.

В рамках данной работы рассматриваются два метода для достижения поставленной цели – нахождение релевантных документов на основе логистической регрессии и с помощью наивного байесовского классификатора.

2.2.1 Логистическая регрессия

Для решения данной задачи было принято решение использовать логистическую регрессию, так как значением ее функции является вероятность принадлежности исходного значения к определенному классу, а не значение числовой переменной, как в случае с обычной линейной регрессией. Ее формула выглядит следующим образом [21]:

– (2.4)

где признак -го объекта,

класс -го объекта,

вектор весов,

обратный коэффициент силы регуляризации,

вес -го объекта.

Количество признаков для каждого документа равно количеству его тем и может равняться нескольким сотням. Количество объектов же в обучающей выборке ограничивается размеченным пользователем множеством документов, и, в общем случае, намного меньше количества признаков. Такой расклад приводит к переобучению регрессионной модели и, как следствие, некорректным предсказаниям. Чтобы этого избежать в функцию регрессии дополнительно вводится L-2 регуляризация, представленная первым слагаемым в формуле. Данный регуляризатор позволяет снизить эффект переобучения и улучшить обобщающие способности получившейся модели.

2.2.2 Математическая модель уточнения запроса с помощью метода Байеса

В задаче классификации текстов метод Байеса применяется отдельно для каждого запроса и принимается решение, принадлежит документ категории или нет [17].

Апостериорная вероятность принадлежности документа запросу вычисляется по формуле Байеса, связывающей априорную вероятность с апостериорной:

(2.5)

Знаменатель (вероятность документа) является константой и никак не может повлиять на ранжирование классов, поэтому в нашей задаче мы можем его игнорировать.

Байесовский классификатор использует оценку апостериорного максимума для определения наиболее вероятного класса:

(2.6)

В натуральном языке вероятность появления слова сильно зависит от контекста. Байесовский классификатор представляет документ как набор слов вероятности, которые условно не зависят друг от друга. Этот подход иногда еще называется bag of words model. Исходя из этого предположения, условная вероятность документа аппроксимируется произведением условных вероятностей всех слов входящих в документ.

(2.7)

Подставив (2.7) в (2.6) получаем, что:

(2.8)

При достаточно большой длине документа придется перемножать большое количество очень маленьких чисел. Для того чтобы при этом избежать арифметического переполнения снизу зачастую пользуются свойством логарифма произведения .Так как логарифм функция монотонная, ее применение к обоим частям выражения изменит только его численное значение, но не параметры при которых достигается максимум. При этом, логарифм от числа близкого к нулю будет числом отрицательным, но в абсолютном значении существенно большим чем исходное число, что делает логарифмические значения вероятностей более удобными для анализа:

(2.9)

Оценка вероятностей и осуществляется при выборе пользователем релевантных и нерелевантных документов. Вероятность класса оцениваем следующим образом:

— (2.10)

где количество документов, принадлежащих классу *c,* общее количество документов, отмеченных пользователем.

———, (2.11)

где количество раз, сколько *i*-ое слово встречается в документах класса *c,* словарь корпуса документов (список всех уникальных слов).

Если на этапе классификации вам встретится слово, которого не было на этапе обучения, то значения , а следственно и будут равны нулю. Это приведет к тому, что документ с этим словом нельзя будет классифицировать, так как он будет иметь нулевую вероятность по всем классам. Избавиться от этой проблемы путем анализа б**о**льшего количества документов не получится. Решением проблемы неизвестных слов является аддитивное сглаживание (сглаживание Лапласа). Идея заключается в том, что необходимо прибавить единицу к частоте каждого слова:

——————— ——————— (2.12)

Подставив выбранные оценки в (2.9) выводим формулу, по которой происходит байесовская классификация:

— ——————— (2.13)

На этапе классификации необходимо для каждого класса рассчитать значение следующего выражения и выбрать класс с максимальным значением:

— ——— (2.14)

где – количество документов в обучающей выборке принадлежащих классу ;

– общее количество документов в обучающей выборке;

– количество уникальных слов во всех документах обучающей выборки;

– суммарное количество слов в документах класса в обучающей выборке;

– сколько раз -ое слово встречалось в документах класса в обучающей выборке;

– множество слов классифицируемого документа (включая повторы).

2.3 Выводы по разделу

В данном разделе была описана математическая модель выполнения поисковых запросов по тематической модели, и математические модели двух локальных алгоритмов модификации поисковой выдачи на основе данных о предпочтениях пользователя. В основе математической модели выполнения поискового запроса находится мера косинусной близости, по которой вычисляется тематическая близость документов коллекции к запросу. На данном шаге пользователь может задать дополнительный критерий, тем самым классифицировав на релевантные и нерелевантные документы поисковую выдачу по косинусной мере. После чего с помощью локальных алгоритмов на основе составленной классификации формируется новая поисковая выдача, состоящая из подходящих запросу и дополнительному критерию документов.

3 РАЗРАБОТКА АЛГОРИТМОВ ПОИСКА

3.1 Алгоритм предварительной обработки документов

Перед использованием документов для построения модели или выполнения поисковых запросов их необходимо особым образом подготовить. Схема алгоритма предобработки документов имеет следующий вид:

Прежде всего выполняется токенизация – разбиение целого текста на более мелкие части, токены. К токенам в общем случае относятся слова, числа и знаки пунктуации. Это необходимо для выделения слов в самостоятельные элементы, над каждым из которых можно проводить индивидуальные операции.

Все слова необходимо привести к единой форме либо путем стемминга (нахождение основы слова и удаление всех остальных его частей), либо посредством проведения лемматизации, то есть приведения слов к их нормальной (словарной) форме. Задача лемматизации требует намного больших вычислительных и временных затрат по сравнению со стеммингом, но взамен обеспечивает лучшее качество обработки, поскольку при стемминге информация может быть утеряна. Для ряда языков, в число которых входит и русский язык, в силу их грамматических особенностей лемматизация является намного более предпочтительным вариантом предобработки.

Следующим шагом из исходных текстов следует удалить все стоп-слова, то есть слова, не имеющие конкретной тематической окраски, а потому понижающие качество тем конечной модели. К таким словам можно отнести предлоги, частицы, союзы, слова, число вхождений которых в исходный корпус чрезмерно велико, или, наоборот, мало и т. д. В общем случае слова с очень низкой частотой вхождений не способствуют формированию качественных и хорошо интерпретируемых тем, даже если их можно явно отнести к какой-либо из них, поэтому они также подвергаются удалению.

Вместе со стоп-словами из текстов также имеет смысл убрать числовые значения, то есть слов, состоящих только из цифр, хотя выполнение данного

шага может зависеть от особенностей исходной коллекции документов, и целей, которым должна отвечать модель

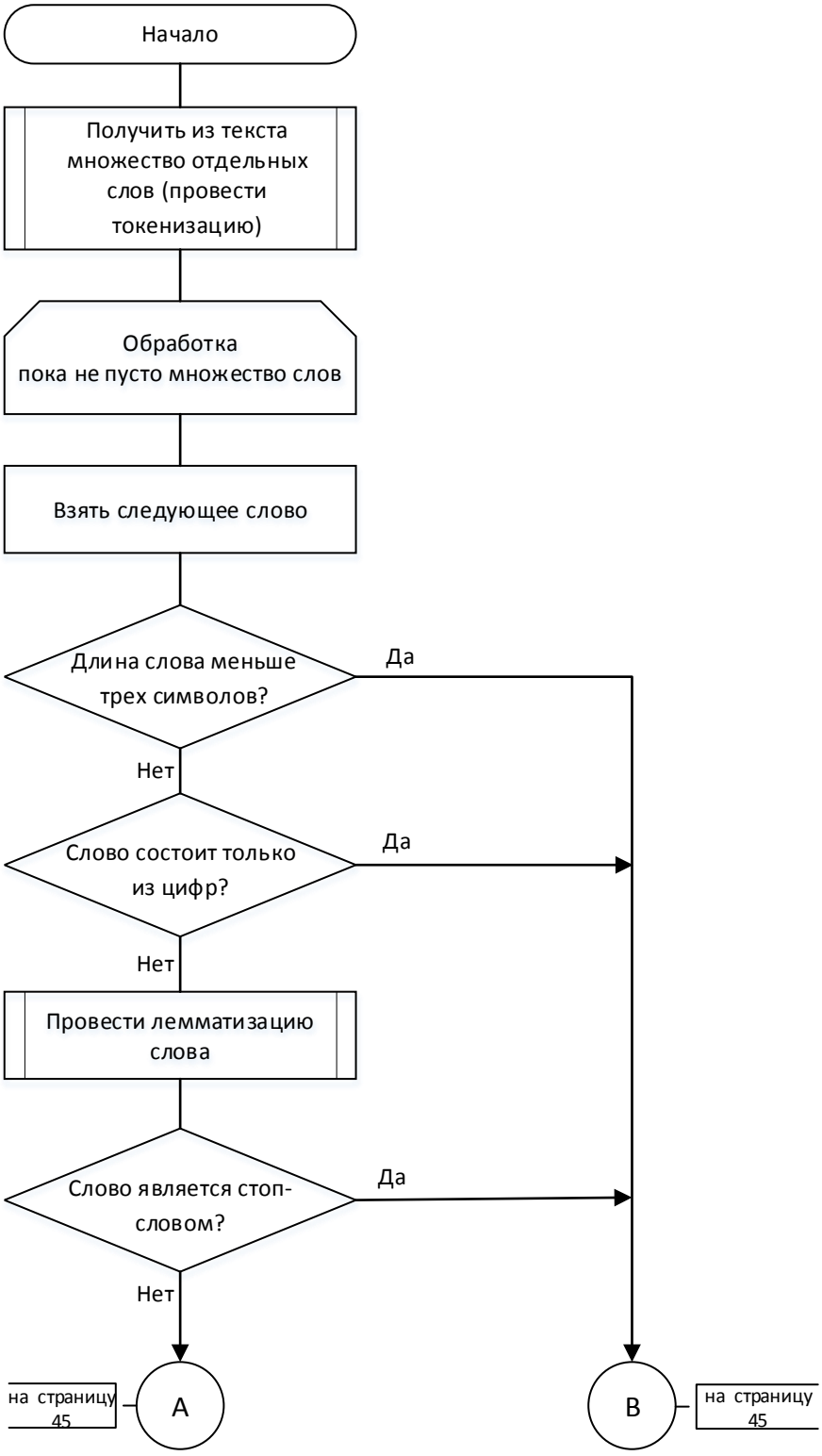

Рисунок 3.1 – Алгоритм предобработки документов

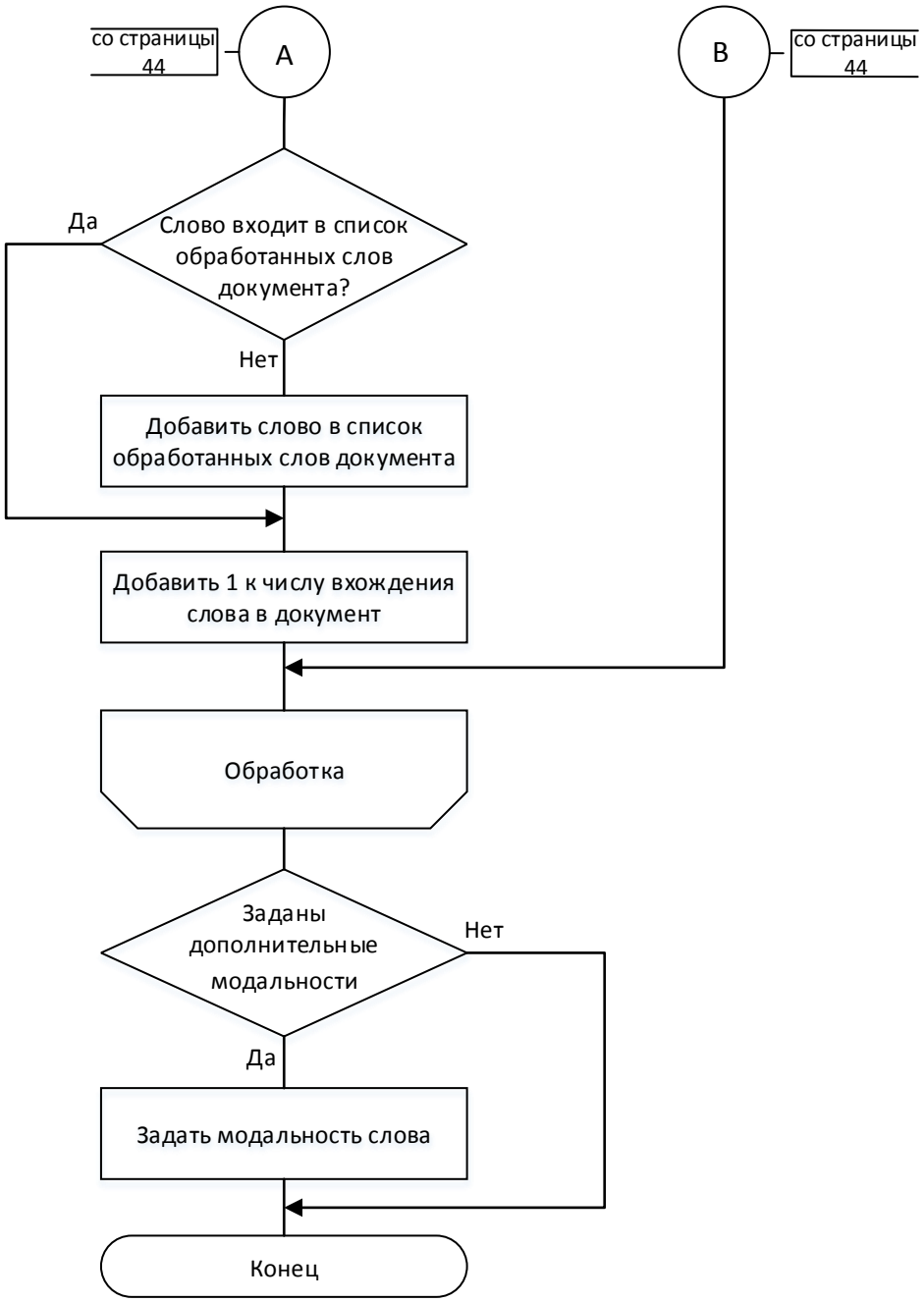

Рисунок 3.1 – продолжение

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 документ. 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 поиска и ранжирования документов по запросу

Документ-запрос сперва необходимо подвергнуть предобработке, аналогично документам исходного корпуса текстов.

Затем для документа-запроса формируется вектор-распределение по темам, предопределенным в матрице распределения тем по документам модели . Данный вектор получается в результате выполнения EM-алгоритма только для запроса. Однако в отличие от используемого при построение 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.

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.

Схемы описанных выше алгоритмов приведены на рисунке 3.2 и рисунке 3.3.

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

Рисунок 3.2 – Общий алгоритм поиска документов по запросу

Такой критерий может выражаться в более конкретной тематике, которой должны обладать результаты поиска.

Этого можно добиться, если позволить пользователю помечать просмотренные им результаты поиска как релевантные и нерелевантные его пожеланиям. Сами пожелания при этом выступают скрытым критерием и явным образом не обозначаются. От системы требуется самостоятельно выявить их на основе предоставленного пользователем разбиения документов на релевантные/нерелевантные и скорректировать выдачу соответствующим образом.

В рамках данной работы рассматриваются два метода для достижения поставленной цели – предугадывание релевантности документов на основе логистической регрессии и с помощью наивного байесовского классификатора.

3.4 Логистическая регрессия

Исходными данными для регрессионного метода выступает отсортированный по убыванию косинусной меры вектор документов, полученный на предыдущем шаге.

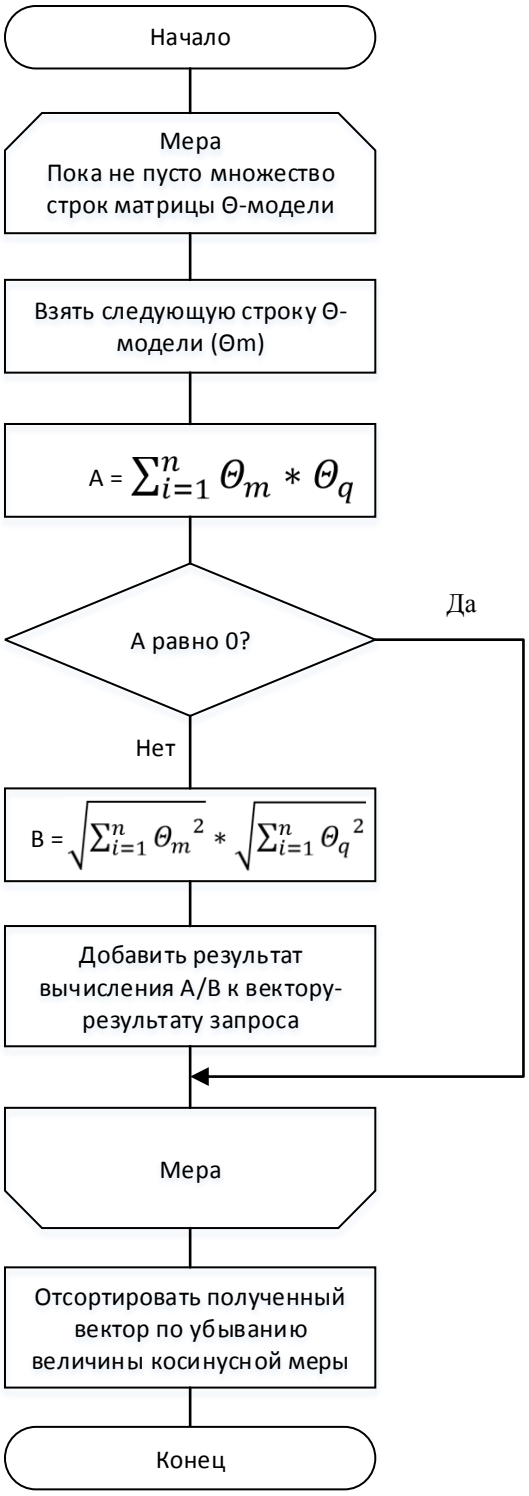

Рисунок 3.3 – Алгоритм расчета
косинусной меры документа-запроса

Пользователю необходимо разметить документы на релевантные и нерелевантные, чтобы максимизировать количество релевантных документов в новой поисковой выдаче.

С помощью логистической регрессии вычисляется рейтинг релевантности документа дополнительному критерию. После чего формируется вектор документов, отсортированный по убыванию рейтинга релевантности.

Схема описанного выше алгоритма приведена на рисунке 3.4

Рисунок 3.4 – Алгоритм вычисления рейтинга релевантности документа к дополнительному критерию с помощью логистической регрессии

3.5 Байесовский классификатор

Исходными данными для байесовского классификатора, также, как и для логистической регрессии выступает отсортированный по убыванию косинусной меры вектор документов, полученный на предыдущем шаге. Пользователю необходимо разметить документы на релевантные и нерелевантные, чтобы максимизировать количество релевантных документов в новой

поисковой выдаче. Потом наивный байесовский классификатор классифицирует еще не просмотренные документы в порядке рейтинга релевантности дополнительному критерию. После чего формируется вектор документов, отсортированный по убыванию рейтинга релевантности.

Рисунок 3.5 – Алгоритм вычисления рейтинга релевантности документа к дополнительному критерию с помощью байесовского классификатора

3.4 Разработка модуля поиска

3.4.1 Диаграмма компонентов

Поисковому модулю searchTM для работы требуется матрица распределения тем по документам, то есть матрица . Данную матрицу же, в свою очередь, можно получить из тематической модели, построить которую можно с помощью библиотеки BigARTM.

Готовую матрицу можно сохранить на диск, в таком случае модуль просто загрузит ее в память. Альтернативно, можно не сохранять саму мат-

рицу, а передать в поисковой модуль объект самой модели, тогда матрица будет извлечена напрямую из нее.

Зависимость от библиотеки BigARTM также выражается в использовании ее метода, позволяющего получить распределение произвольного документа по темам построенной тематической модели.

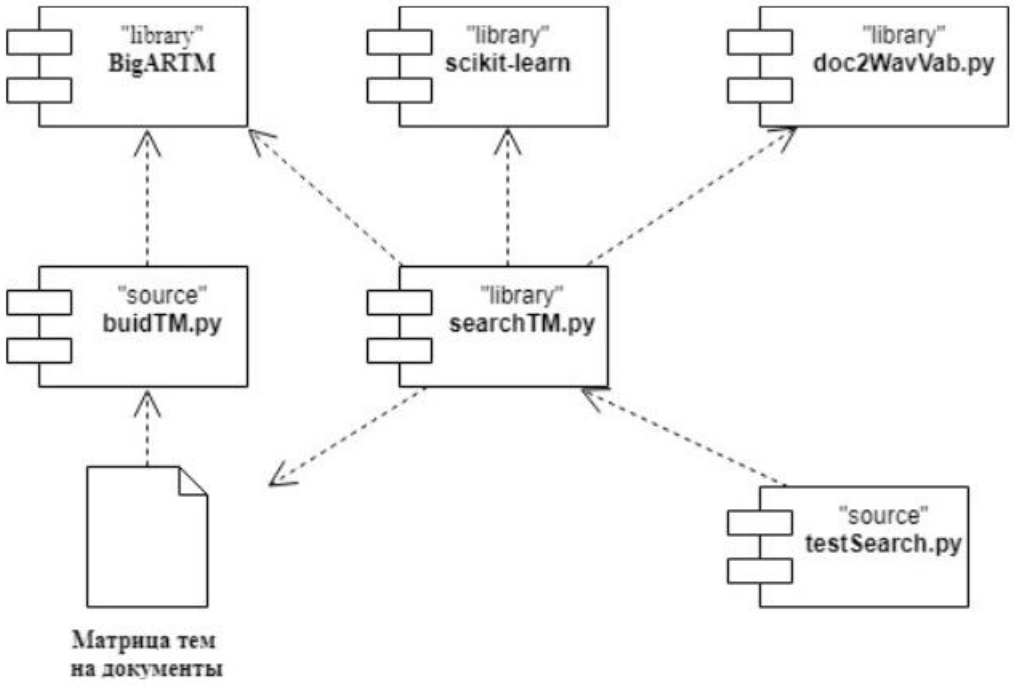

Рисунок 3.1 – Диаграмма компонентов

Также для проведения классификации с помощью наивного байесовского классификатора и логистической регрессии используется библиотека scikit-learn.

Модуль doc2VowWab реализует алгоритмы предобработки исходного блока документов модели и документов запроса. Перед использованием с поисковым модулем любой текстовый документ необходимо привести к определенному формату, называемому Vowpal Wabbit.

Модуль testSearch содержит в себе функции, с помощью которых проводилось тестирование реализованных методов поиска документов и учета предпочтений пользователя.

Модуль buildTM реализует построение используемой для проведения экспериментальных исследований тематической модели.

3.4.2 Диаграмма классов

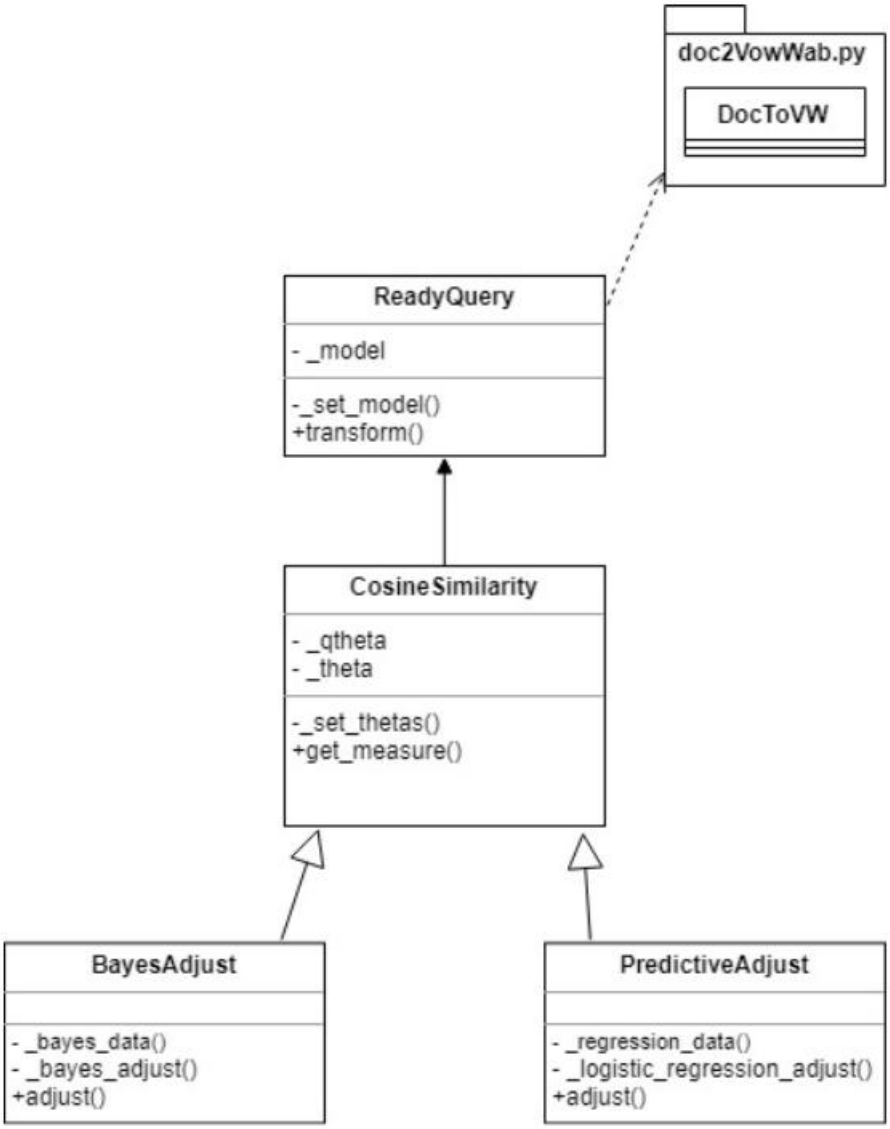

Рисунок 3.2 – Диаграмма классов модуля SearchTM

Класс CosineSimiliraty воплощает базовую поисковую логику модуля. Именно в его полях содержатся матрицы распределения тем по документам тематической модели и запроса, и именно в нем реализован расчет косинусной меры их сходства.

Его логику расширяют наследуемые от него классы BayesAdjust и PredictiveAdjust, реализующие наивную байесовскую классификацию и предсказание релевантности документов с помощью регрессии соответственно.

Класс ReadyQuery обеспечивает взаимодействие с объектом тематической модели, если таковой был предоставлен, и отвечает за получение вектора тематического распределения запроса.

Класс DocToVW реализует предварительную обработку документов и приведение их к формату, пригодному для взаимодействия с тематическими моделями.

3.4.3 Описание библиотеки модуля

Разработанный пакет SearchTM для языка Python состоит из двух модулей. Первый модуль searchTM.py содержит в себе всю логику выполнения поискового запроса по тематическим моделям. Второй модуль является скорее вспомогательным и отвечает за предобработку текстовых данных в исходных документах и приведению их к подходящему для работы основного модуля формату.

Модуль SearchTM предоставляет функции:

1) поиска тематически близких запросу документов на основе расчета косинусной меры близости их векторов распределения тем;

2) модификации поисковой выдачи с учетом предпочтений пользователя методом предсказания релевантности документов посредством наивного байесовского классификатора;

3) модификации поисковой выдачи с учетом предпочтений пользователя методом предсказания релевантности документов посредством регрессионного анализа (логистической регрессии).

Взаимодействие с библиотекой может осуществляться по различным сценариям. Предполагается, что в распоряжении пользователя данной библиотеки есть построенная с помощью средств библиотеки BigARTM тематическая модель (в виде объекта в оперативной памяти или сохраненная на диск), и, желательно, извлеченная из нее матрица распределения тем на документы .

Если передать в библиотеку только объект модели, матица Θ будет извлечена из нее. Если передать путь до модели, сохраненной на диске, она будет считана в оперативную память. Последний вариант является наименее предпочтительным, поскольку размер модели может достигать гигабайт (ее размер зависит от размера исходной коллекции и, строго говоря, может быть сколь угодно большим), и на ее загрузку может потребоваться значительное количество времени.

Поисковым запросом может быть вектор тематического распределения некоторого документа (соответствующий тематикам имеющейся тематической модели), либо сам этот документ в необработанном виде. Во втором случае автоматически будет произведена предобработка документа и получение его вектора тем.

Наиболее общий сценарий взаимодействия с модулем может быть следующим:

1) создается объект CosineSimiliraty, в его конструктор передаются матрица тематической модели и вектора документа-запроса, либо сам этот документ и тематическая модель;

2) выполняется метод get_measure(), возвращающий значение косинусной меры близости между документом запроса и каждым документом коллекции; в метод можно передать новый документ запроса или вектор , запрос тогда будет выполнен именно с ним;

3) создается объект класса BayesAdjust или PredictiveAdjust (возможно их одновременное использование), в метод adjust() которого в виде двух списков передаются имена (совпадающие с идентификаторами документов матрицы) документов, оцененных пользователем как релевантные или нерелевантные к его запросу;

4) метод adjust() возвращает скорректированную под предпочтения пользователя поисковую выдачу.

Также возможно создание экземпляра класса DocToVW и вызов его методов для ручной обработки исходных документов и приведению их к

формату Vowpal Wabbit, с которым работает модуль, но обычно в этом нет необходимости.

3.5 Выводы по разделу

В разделе описан алгоритм предобработки исходных документов и приведения их в пригодный для работы с ними тематической модели формат.

Также представлены алгоритмы косинусной меры близости запроса к коллекции документов, объяснены локальные алгоритмы разведочного поиска по тематическим моделям. Описан алгоритм и библиотека модуля поиска, диаграмма классов, используемых для нахождения релевантных документов, а также общий сценарий взаимодействия с модулем поиска.

4 ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ЛОКАЛЬНЫХ АЛГОРИТМОВ

4.1 Методика эксперимента

Для проведения исследований в рамках данной работы была построена тематическая модель, по которой в дальнейшем осуществляется поиск. Детальное описание процесса построения модели и используемой коллекции документов приведено в разделе 4.1.1.

Сам алгоритм поиска заключается в определении тематической близости между документом-запросом и документами, входящими в модель, и последующей коррекции результатов поиска с учетом предпочтений пользователя на основе разметки документов на релевантные или нерелевантные его запросу. Коррекция результатов достигается двумя способами – с помощью логистической регрессии и с применением наивного байесовского классификатора.

Целью экспериментов ставилось определение эффективности предлагаемых алгоритмов улучшения релевантности поисковой выдачи, сравнение их друг с другом и с простой выдачей без их использования.

Для этого были предприняты следующие шаги. В качестве запроса был выбран определенный документ. Затем был выполнен поиск тематически близких ему документов на основе косинусной меры между векторами-распределениями тем документов коллекции и документа-запроса.

Из результатов поиска были взяты первые 100 документов с наибольшим показателем сходства по косинусной мере. Все дальнейшие действия по ходу эксперимента осуществлялись только с ними.

Далее был задан некоторый дополнительный критерий релевантности. В качестве дополнительного критерия релевантности для, например, запроса про Linux, может выступать наличие в статье информации про дистрибутивы Linux, такие как Ubuntu, Mint и т. д. А в запросе про нейронные сети, в качестве дополнительного критерия релевантности может выступать наличие ин-

формации про связь нейронных сетей с искусственным интеллектом. В полученной на предыдущем шаге выборке путем ручного просмотра содержания документов были найдены все отвечающие этому критерию (релевантные ему) документы.

После чего на каждом шаге алгоритма улучшения релевантности выводилось по 20 документов (из отобранных 100), которые размечались на релевантные и нет. На основе полученной классификации производилась корректировка следующих к выдаче 20 документов. Сравнение методов проводилось по количеству найденных ими на каждом шаге релевантных документов и по тому, насколько быстро ими были найдены все релевантные документы в отобранной сотне.

Общий алгоритм эксперимента для обоих методов коррекции можно описать следующим образом:

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

2) отобрать 100 документов с наибольшим значением тематической близости по косинусной мере, последующие шаги осуществлять только с их использованием;

3) задать дополнительный критерий, делающий запрос более строгим;

4) в отобранных ста документах найти все, отвечающие заданному критерию (релевантные) и запомнить их;

5) вывести первые 20 документов с наибольшим косинусным сходством и разметить их на релевантные и нерелевантные;

6) выполнить новый, скорректированный на основе полученного на предыдущем шаге распределения, запрос;

7) из еще не размеченных документов вывести 20 с наибольшим после корректировки показателем релевантности;

8) повторять шаги 6–7 до тех пор, пока не будут найдены все релевантные документы с шага 4.

Собранная в ходе эксперимента информация об эффективности каждого из локальных алгоритмов приводится в разделе 4.2.

4.1.1 Построение тематической модели

Для реализации алгоритма поиска и проведения экспериментов необходима построенная по определенной коллекции документов тематическая модель. В качестве исходного корпуса документов была взята коллекция статей с интернет ресурсов Habr и Geektimes за 2007–2011 годы. Размер коллекции составляет 40 000 документов. Тематика статей достаточно разнообразна, но большинство из них имеет связь с программированием, информационными технологиями, бизнесом и интернетом.

Количество тем тематической модели было решено установить равным 250, из которых 50 тем отведены под фоновые темы (где должна собираться общая лексика и слова, слабо влияющие на интерпретируемость тем), а остальные 200 – под темы, которые должны стать основными.

Используемая в данной работе коллекция документов предоставляет различную мета информацию о каждом документе, такую как: дата публикации, теги, хабы (аналог тегов, но с большей степенью обобщения), никнейм автора. Данную информацию также можно использовать для построения тематической модели. В используемой модели была задана дополнительная модальность хабов, где находится информация о том, к каким хабам относится каждая статья.

В качестве регуляризаторов модели были использованы регуляризаторы сглаживания/разреживания матриц и регуляризатор декоррелирования матрицы и регуляризатор улучшения когерентности тем. Регуляризация проводилась для фоновых и основных тем отдельно – на фоновых осуществлялось сглаживание, в то время как основные подвергались разреживанию и декорелированию.

Алгоритм, по которому строилась модель, имеет следующий вид:

1) активировать регуляризатор сглаживания на всех 250 темах, с большим коэффициентом сглаживания на фоновых темах, нежели 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 регуляризатором 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.

4.2 Результаты экспериментов

Эксперименты проводились для разных запросов с различными тематиками и варьирующимся количеством релевантных документов и их распределением. Ниже для обоих описанных методов улучшения разведочного поиска на примере двух разных запросов приведены графики скорости нахождения всех релевантных дополнительному критерию документов из отобранных на 4 шаге алгоритма. Для перового запроса (рисунок 4.1) число релевантных документов составило 12, для второго (рисунок 4.2) – 16.

Графики отражают то, насколько быстро (за сколько итераций шага 8) каждый из методов справился с нахождением релевантных документов, и сколько их было найдено на каждой итерации. Пользователь заинтересован в скорейшем получении желаемых результатов – следовательно, чем более стремительно график метода приближается к верхнему значению (12 и 16 для запроса №1 и №2 соответственно), тем он более эффективен.

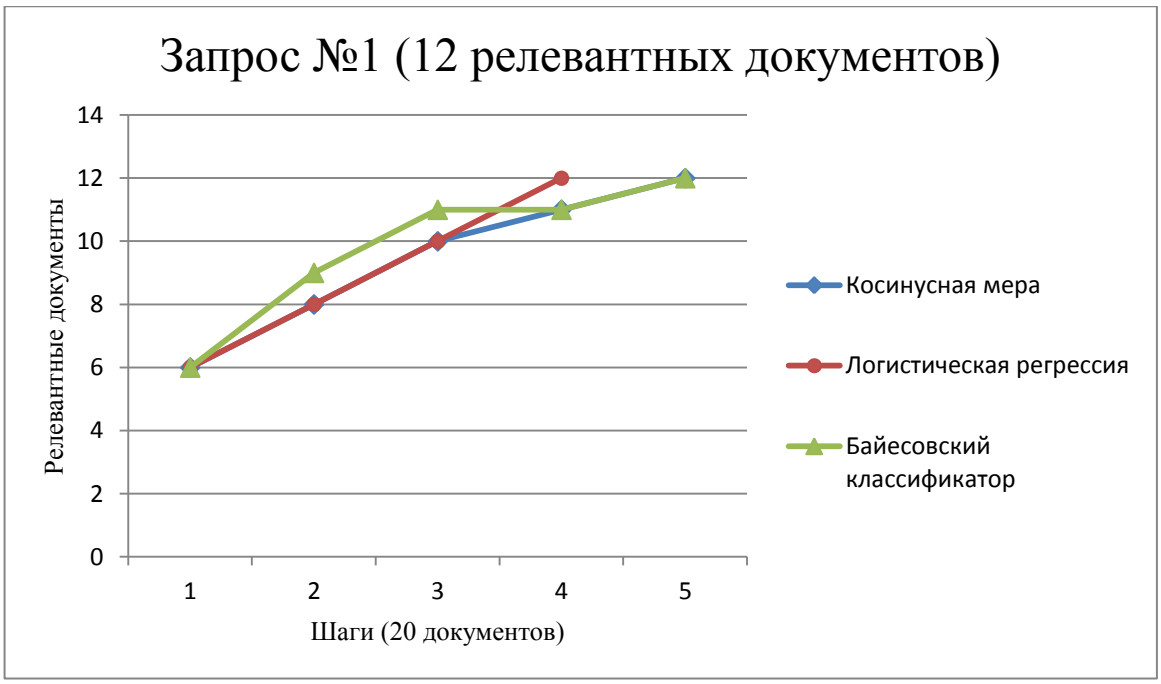

Рисунок 4.1 – Запрос №1 (12 релевантных документов)

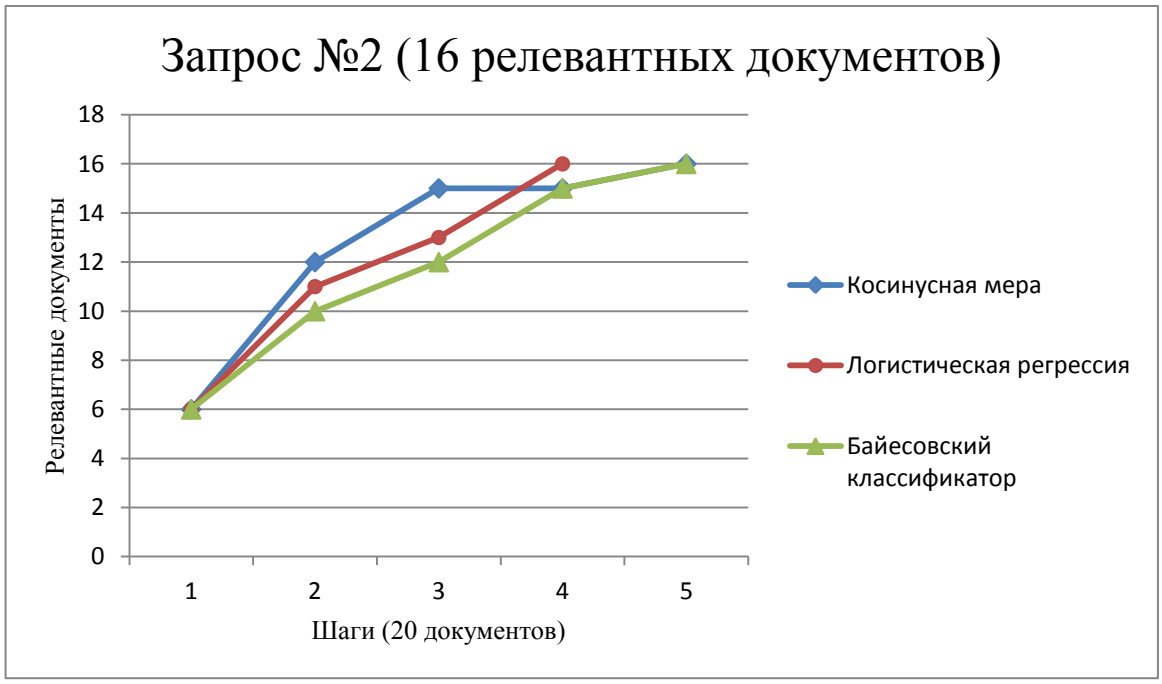

Рисунок 4.2 – Запрос №2 (16 релевантных документов)

Графики косинусной меры здесь, по существу, отражают исходное распределение релевантных дополнительному критерию документов в отобранной (шаг 2 алгоритма) сотне. Таким образом, например, для первого запроса в первых 20 документах с наибольшей косинусной мерой было 6 релевантных документов, в следующих двадцати по косинусной мере их было 2, затем 2 и т. д. Графики имеют одинаковую начальную точку в рамках одного

запроса, поскольку первый шаг поиска (шаг 5 алгоритма) задает первые релевантные документы и совпадает для всех методов.

На графике (рисунок 4.1) видно, что после разметки первых 20 документов регрессионному методу понадобилось меньше, чем байесовскому классификатору итераций, чтобы найти все релевантные документы. На каждом шаге байесовский классификатор находил больше документов, чем логистическая регрессия.

Во втором случае (рисунок 4.2) логистическая регрессия также справилась за меньшее количество итераций, чем байесовский классификатор. Оба метода в данном запросе находили релевантных документов меньше, чем их было распределено, но смогли найти все документы. Регрессионный метод справился за 4-е шага, байесовский классификатор за 5.

Для измерения качества работы каждого метода введена следующая метрика:

$$\frac{}{}$$

где – количество шагов, потребовавшихся методу для нахождения всех релевантных документов,

– общее число релевантных документов для данного запроса,

– отношение релевантных документов на данном шаге от общего числа релевантных.

Максимально возможное значение данной метрики равно 1. Это значит, что все релевантные документы найдены на первом шаге. Минимально возможное значение 0. В таком случае метод не смог найти релевантные документы.

Опираясь на результаты эксперимента, нашли средние значения метрики для косинусной меры, логистической регрессии и байесовского классификатора. Они равны 0,76, 0,75 и 0,75 соответственно.

Согласно метрике, была составлена таблица (таблица 4.1).

Таблица 4.1 – Результаты эксперимента

№ запроса	Локальный алгоритм	Значение метрики	Кол-во итераций
1	Косинусная мера	0.68	5
	Логистическая регрессия	0.68	4
	Байесовский классификатор	0.70	5
2	Косинусная мера	0.85	5
	Логистическая регрессия	0.83	4
	Байесовский классификатор	0.80	5

Регрессионный метод и байесовский классификатор позволяют пользователю одинаково эффективно улучшить поисковую выдачу релевантных документов, но регрессионный метод справляется с задачей за меньшее количество итераций, что делает его более приемлемым для использования пользователем.

4.3 Выводы по разделу

В разделе предложена методика оценки эффективности описанных локальных алгоритмов разведочного поиска. Также приводится описание построенной для проведения экспериментов тематической модели.

Оба рассматриваемых метода показали улучшение результатов поиска, однако регрессионный метод оказался лучше в плане затраченного времени. Ему требуется меньше итераций для нахождения релевантных документов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы автором изучены методики и принципы построения тематических моделей. В частности, такие их вариации как латентно-семантический анализ LSA, его вероятностная версия PLSA, латентное размещение Дирихле LDA и подход аддитивной регуляризации ARTM. Для дальнейшей работы выбран подход ARTM, для которого также рассмотрены некоторые регуляризаторы. Разработан алгоритм подготовки исходных данных для тематических моделей.

С помощью библиотеки BigARTM построена тематическая модель, с использованием которой разрабатывался алгоритм поиска документов и локальные алгоритмы улучшения релевантности поисковой выдачи посредством учета предпочтений пользователя. Реализованы два локальных алгоритма улучшения результатов поиска – метод регрессионного предсказания релевантности документов и метод наивной байесовской классификации.

Разработанные алгоритмы 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 разработана методика для экспериментального исследования описанных алгоритмов поиска. Проведённые эксперименты продемонстрировали эффективность разработанных методов.

Таким образом, поставленные на выпускную квалификационную работу задачи были выполнены, однако тематическое моделирование и разедочный поиск с его использованием молодая и чрезвычайно обширная сфера деятельности, в которой остается еще много нерешенных задач.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Апишев М. Вероятностное тематическое моделирование / М. Апишев Дата обновления: URL: http://www.machinelearning.ru/wiki/images/3/34/MelLain_TM_lecture_MIPT.pdf (дата обращения: 10.02.2019).

2 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: 10.02.2019).

3 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.

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 Воронцов, К.В. Регуляризация, робастность и разреженность вероятностных тематических моделей / К.В. Воронцов, А.А. Потапенко // Компьютерные исследования и моделирование 2012. № 4. – С. 693–706

6 Смелик, Н.Д. Мультимодальная тематическая модель текстов и изображением на основе использования их векторного представления / Н.Д. Смелик, А.А. Фильченков // Машинное обучение и анализ данных 2016. № 4. – С. 421–441.

7 Нижибицкий, Е.А. Относительная перплексия как мера качества тематических моделей / Е.А. Нижибицкий. – Дата обновления: 07.04.2014. URL: http://www.machinelearning.ru/wiki/images/0/0c/NizhibitskyLomonosovSlides14.pdf (дата обращения: 07.03.2019).

8 Сингулярное разложение // Википедия, свободная энциклопедия. – Дата обновления: 24.02.2016. URL: https://ru.wikipedia.org/wiki/Сингулярное_разложение (дата обращения: 10.03.2019).

9 Thomas, L. Introduction to Latent Semantic Analysis / LandauerThomas, Peter W. Foltz, Darrell Laham. – Дата обновления: 07.04.2014. URL: http://lsa.coloradoedu/papers/dp1.LSAintro.pdf (дата обращения: 10.03.19).

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

11 Коршунов, А. Тематическое моделирование текстов на естественном языке / А. Коршунов, А. Гомзин. – Дата обновления: URL: http://www.ispras.ru/proceedings/docs/2012/23/isp_23_2012_215.pdf (дата обращения:15.03.2019).

12 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.

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

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

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

16 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.

17 Боярский, К.К. Введение в компьютерную лингвистику / К.К. Боярский // Учебное пособие. – СПб: НИУ ИТМО, 2013. – С. 51–57.

18 Gensim official page URL: http://radimrehurek.com/gensim/about.html. – Дата обновления: 10.12.2017 (дата обращения: 15.03.2019).

19 VowpalWabbit official page. URL: https://github.com/JohnLangford/vowpal_wabbit/wiki. – Дата обновления: 27.04.2017 (дата обращения: 15.03.2019).

20 BigARTM documentation URL: http://docs.bigartm.org/en/stable/index.html. – Дата обновления: 27.07.2017 (дата обращения: 15.03.2019).

21 Тепляков С.Ю. Повышение релевантности поисковой выдачи и учет предпочтений пользователя в разведочном поиске / С.Ю. Тепляков, Т.Ю. Оленчикова // Точная наука. – 2018. №26. – С. 107–112.

ПРИЛОЖЕНИЕ 1

Исходный текст программы

Модуль searchTM.py

```
import os
import shutil
from time import time
import pandas as pd
from operator import itemgetter
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
import numpy as np
import matplotlib.pyplot as plt

def _timeit(method):
    def timed(*args, **kw):
        ts = time()
        result = method(*args, **kw)
        te = time()
        print('{} completed in {:.2f}s'.format(method.__name__, (te - ts)))
        return result
    return timed


class ReadyQuery:
    def __init__(self, model=None):
        """
        :param model: topic model to get query Theta distribution from, if
None can't use transform method
        :type model: str or reference to BigARTM artm model object
        """
        if model is not None:
            self._model = self._set_model(model)
        else:
            self._model = None

    @classmethod
    def _set_model(cls, arg_model):
        try:
            import artm
        except ModuleNotFoundError as e:
            raise ModuleNotFoundError('Make sure python knows path to BigARTM
library.') from e
        if isinstance(arg_model, str) and os.path.isdir(arg_model):
            model = artm.load_artm_model(arg_model)
        if not isinstance(model, (artm.ARTM, artm.LDA)):
            raise TypeError('Use BigARTM library model')
        return model

    def transform(self, filename):
        """
            :Description: Returns file's Theta distribution which corresponds
to model's Theta matrix

            :param str filename: path to the file to get Theta distribution
of
            :return:
                * pandas.DataFrame: (data, columns, row), where:
                * columns --- the names of topics in topic model;
                * row --- id of the query document;
```

```
                * data --- content of Theta matrix.
            :Note:
                * Needs model to be set
        """
        if self._model is None:
            raise AttributeError('{} object has no model set. Set the model
in the constructor'
                                 .format(self.__class__.__name__))
        import artm
        bv_folder = os.path.join(os.getcwd(), 'query_batches')
        query_bv = artm.BatchVectorizer(data_path=filename,
                                        data_format='vowpal_wabbit',
                                        batch_size=10,
                                        target_folder=bv_folder,
                                        gather_dictionary=True)
        query_theta = self._model.transform(batch_vectorizer=query_bv)
        shutil.rmtree(bv_folder)
        return query_theta


class CosineSimilarity(ReadyQuery):
    def __init__(self, theta, qtheta=None, file=None, model=None):
        """
        :param theta: Theta matrix of the topic model
        :type theta: csv or hdf (.h5) table
        :param qtheta: Theta distribution of the query document
        :type qtheta: csv or hdf (.h5) table
        :param str file: path to an unprepared query document
        :param model: topic model to get query Theta distribution from, if
None can't get\
                      distribution for a new document
        :type model: str or reference to BigARTM artm model object
        """
        super(CosineSimilarity, self).__init__(model)
        self._theta = self._set_thetas(theta)
        if qtheta is not None:
            self._qtheta = self._set_thetas(qtheta)
        elif file is not None and model is not None:
            super(CosineSimilarity, self).__init__(model=model)
            self._qtheta = self.transform(file)
        else:
            raise ValueError('Either qtheta or file and model must be set')

    @classmethod
    def _set_thetas(cls, theta):
        if isinstance(theta, (pd.DataFrame, pd.Series)):
            if isinstance(theta, pd.Series):
                theta = pd.DataFrame(theta)
            return theta
        elif isinstance(theta, str):
            _, file_extension = os.path.splitext(theta)
            if file_extension == '.csv':
                return pd.read_csv(theta, index_col=0)
            elif file_extension == '.h5':
                return pd.read_hdf(theta, 'table')
            else:
                raise TypeError("Theta matrix must have either hdf (.h5) or
csv format. " +
                                "Instead got '{}'".format(file_extension))
        else:
            raise TypeError('theta must be either a DataFrame/Series object
or a path to it')
```

```
    @property
    def qtheta(self):
        return self._qtheta

    @property
    def theta(self):
        return self._theta

    @qtheta.setter
    def qtheta(self, qtheta):
        self._qtheta = self._set_thetas(qtheta)

    @theta.setter
    def theta(self, theta):
        self._theta = self._set_thetas(theta)

    def get_measure(self, qtheta=None):
        """
            :Description: Gets cosine measure for the specified query docu-
ment\
                          and documents in the topic model

            :param qtheta: Theta distribution for a new document, None means
use one that the object has
            :type qtheta: csv or hdf (.h5) table

            :return: list with tuples of documents ids and their correspond-
ing cosine measure in descending order
        """
        if qtheta is not None:
            self.qtheta = self._set_thetas(qtheta)
        mera = dict()
        mera_down1 = pow(pow(self.qtheta.values[0], 2).sum(), 1 / 2)
        for row in range(len(self.theta.index)):
            mera_up = self.qtheta.values[0].dot(self.theta.values[row])
            if mera_up == 0:
                continue
            mera_down2 = pow(pow(self.theta.values[row], 2).sum(), 1 / 2)
            mera[self._theta.index[row]] = mera_up / (mera_down1 *
mera_down2)
        mera = sorted(mera.items(), key=itemgetter(1))
        mera.reverse()
        return mera


class Bayes(CosineSimilarity):
    def __init__(self, theta, qtheta=None, file=None, model=None):
        """
           :param theta: Theta matrix of the topic model
           :type theta: csv or hdf (.h5) table
           :param qtheta: Theta distribution of the query document
           :type qtheta: csv or hdf (.h5) table
           :param str file: path to an unprepared query document
           :param model: topic model to get query Theta distribution from, if
None can't get\
                         distribution for a new document
           :type model: str or reference to BigARTM artm model object
        """
        super(Bayes, self).__init__(theta, qtheta, file, model)

    def _bayes_data(self, rev, irrev, mera, confidence):
        if confidence is None:
            confidence = 5e2
```

```
        items = len(mera)
        x = np.ones((items, self.theta.shape[1]))
        y = np.zeros(items)
        weights = np.ones(items)
        n = (len(rev) + len(irrev))
        koef_rev = len(irrev) / n
        koef_irrev = len(rev) / n

        for i, item in enumerate(mera):
            x[i] = self.theta.loc[item[0]].values
            if item[0] in rev:
                y[i] = 1
                weights[i] = confidence * koef_rev
            elif item[0] in irrev:
                y[i] = 0
                weights[i] = confidence * koef_irrev
        return x, y, weights

    def _bayes_adjust(self, x, y, weights):
        fitting_index = weights > 1
        bayes = MultinomialNB()
        bayes.fit(x[fitting_index], y[fitting_index], weights[fitting_index])
        return bayes.predict_proba(x)

    def adjust(self, rev, irrev, mera=None, qtheta=None, confidence=None):
        """
           :Description: Gets naive bayes model's prediction regarding rele-
vance of the\
                         document in topic model to the query document

           :param list rev: ids of the relevant documents
           :param list rev: ids of the irrelevant documents
           :param mera: cosine measure of the document to predict relevance
of, None means\
                         predict for all document in the topic model
           :type mera: list of tuples with documents ids and their corre-
sponding cosine measure
           :param qtheta: Theta distribution for a new document, None means
use one that the object has
           :type qtheta: csv or hdf (.h5) table

           :return:
            *list with triple tuples with:
            *document ids;
            *corresponding cosine measure;
            *corresponding relevance prediction (from 0 to 1);
            sorted by relevance in descending order
        """
        if mera is None:
            mera = self.get_measure(qtheta)
        x, y, weights = self._bayes_data(rev, irrev, mera, confidence)
        predicted = self._bayes_adjust(x, y, weights)
        predicted = [(*x, y[1]) for (x, y) in sorted(zip(mera, predicted),
                                                     key=lambda pair:
pair[1][1], reverse=True)]
        return predicted

class PredictiveAdjust(CosineSimilarity):
    def __init__(self, theta, qtheta=None, file=None, model=None):
        """
             :param theta: Theta matrix of the topic model
             :type theta: csv or hdf (.h5) table
             :param qtheta: Theta distribution of the query document
```

```
            :type qtheta: csv or hdf (.h5) table
            :param str file: path to an unprepared query document
            :param model: topic model to get query Theta distribution from,
if None can't get\
                          distribution for a new document
            :type model: str or reference to BigARTM artm model object
        """
        super(PredictiveAdjust, self).__init__(theta, qtheta, file, model)

    def _regression_data(self, rev, irrev, mera, confidence):
        if confidence is None:
            confidence = 5e2
        items = len(mera)
        x = np.ones((items, self.theta.shape[1]))
        y = np.zeros(items)
        weights = np.ones(items)
        n = (len(rev) + len(irrev))
        koef_rev = len(irrev) / n
        koef_irrev = len(rev) / n

        for i, item in enumerate(mera):
            x[i] = self.theta.loc[item[0]].values
            if item[0] in rev:
                y[i] = 1
                weights[i] = confidence * koef_rev
            elif item[0] in irrev:
                y[i] = 0
                weights[i] = confidence * koef_irrev
        return x, y, weights

    def _logistic_regression_adjust(self, x, y, weights):
        fitting_index = weights > 1
        logistic = LogisticRegression()
        logistic.fit(x[fitting_index], y[fitting_index],
weights[fitting_index])
        return logistic.predict_proba(x)

    def adjust(self, rev, irrev, mera=None, qtheta=None, confidence=None):
        """
           :Description: Gets logistic regression model's prediction regard-
ing relevance of the\
                         document in topic model to the query document
           :param list rev: ids of the relevant documents
           :param list rev: ids of the irrelevant documents
           :param mera: cosine measure of the document to predict relevance
of, None means\
                         predict for all document in the topic model
           :type mera: list of tuples with documents ids and their corre-
sponding cosine measure
           :param qtheta: Theta distribution for a new document, None means
use one that the object has
           :type qtheta: csv or hdf (.h5) table
           :return:
            *list with triple tuples with:
            *document ids;
            *corresponding cosine measure;
            *corresponding relevance prediction (from 0 to 1);
            sorted by relevance in descending order
        """
        if mera is None:
            mera = self.get_measure(qtheta)
        x, y, weights = self._regression_data(rev, irrev, mera, confidence)
        predicted = self._logistic_regression_adjust(x, y, weights)
```

```
        predicted = [(*x, y[1]) for (x, y) in sorted(zip(mera, predicted),
                                                     key=lambda pair:
pair[1][1], reverse=True)]
        return predicted
```

Модуль doc2VawWab.py

```
import nltk
import urllib.parse as urlparse
import collections
import time
import sys
import pymorphy2
import os
import json
import pandas as pd
import h5py
import shutil
import numpy as np

# Парсер для синтаксического разбора файлов HTML/XML
from bs4 import BeautifulSoup

from subprocess import Popen
from nltk.corpus import stopwords
import multiprocessing as mp
from functools import partial

#nltk.download('stopwords')

#nltk.download('punkt')
from pandas import DataFrame

SUBLIME_PATH = r'C:\Program Files\Sublime Text 3\sublime_text.exe'
EMED_PATH = r'C:\Program Files\EmEditor\EmEditor.exe'
DATASET_PATH = r'Habrahabr\Habrahabr\habr_posts_20150413\habr_posts'
FILE_BOW = r'habr_posts\10k.txt'
HABR_TEXT_COLLECTION = r'habr_collection'


# Морфологический анализ слов
morph = pymorphy2.MorphAnalyzer()
# Установка в качестве стоп-слов словарь русских слов
stop_words = set(stopwords.words('russian'))

# Функция запуска EmEditor
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)
    # Popen выполняет программу в новом процессе
    handle = Popen([path, 'file_to_read.txt'])


# Получить пост
def get_post(post_id):
    #load_text_post(post_id)
    with open(os.path.join(DATASET_PATH, str(post_id))) as fin:
        # десериализует s (экземпляр str, содержащий документ JSON) в объект
```

```
Python.
        post = json.load(fin)
    return post

def load_text_post(post_id):
    shutil.copy(HABR_TEXT_COLLECTION + r'\\post_' +str(post_id), TRAIN_DIR +
'\\' + str(post_id))

# Преобразовать пост
def post_to_corpus_line(post):
    post_id = post['_id']
    author = post['author']
    tags = post['tags']
    date = post['published'][:10]

    # Позволяет нам считать количество неизменяемых объектов
    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"'"
    # Избавляемся от знаков пунктуации и прочего
    for c in space_chars:
        content_text = content_text.replace(c, ' ')
    # Разбили текст на более мелкие части
    tokens = nltk.word_tokenize(content_text)
    for token in tokens:
        if len(token) > 2:
            try:
                isnumb = float(token)
                continue
            except:
                # Заменяем ё на е
                token = token.lower().replace(u'ё', u'е')
                # Парсим токен в нормальную форму и получаем слово
                word = morph.parse(token)[0].normal_form
                # Подсче встречаемости слова в тексте
                if len(word) > 0 and word not in stop_words:
                    words[word] += 1

    # Разбор
    def parse_hub_id(hub_pair, get_name=True):
        if get_name:
            # Замена пробелов на _
            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:
            # Добавление в конец списка
            hubs.append(hub_id)

    # Построение мешка слов
```

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

    # Соединяем части: номер поста, дата публикации, автор, теги, разде-
лы/глоб темы, текст
    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)


# Печать времени прошедшего сначала чего-то
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...')


# Листинг/просмотр
def listener(q, notif_rate, coll_size):
    ready_cnt = 0
    # Отсчет времени
    start = time.time()
    with open(FILE_BOW, 'w', encoding='utf-8') as fout:
        while 1:
            # Получаем задание из очереди
            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)
    # запуск нового процесса в кучу
    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()
    part = {}
    with open(FILE_BOW, 'w', encoding='utf-8') as fout:
        ready_cnt = 0
        for post_id in post_ids:
            line = post_to_corpus_line(get_post(post_id))
            print(line, file=fout)
            ready_cnt += 1
            if ready_cnt % notif_rate == 0:
                print_time(time.time() - start, coll_size, ready_cnt)

def make_Dictionary(train_dir):
    documents = [os.path.join(train_dir, f) for f in os.listdir(train_dir)]
    all_words = []
    for document in documents:
        with open(document, encoding='utf-8') as m:
            for i, line in enumerate(m):
                words = [morph.parse(word)[0].normal_form for word in
line.split()]
                all_words += words
    dictionary = collections.Counter(all_words)
    list_to_remove = [key for key in dictionary.keys()]
    for item in list_to_remove:
        if item.isalpha() == False:
            del dictionary[item]
        elif len(item) < 4:
            del dictionary[item]
        elif item in stop_words:
            del dictionary[item]
    return dictionary

def extract_features(train_dir):
    dictionary = make_Dictionary(TRAIN_DIR)
    files = [os.path.join(train_dir,fi) for fi in os.listdir(train_dir)]
    features_matrix = np.zeros((len(files),len(set(dictionary))))
    docID = 0;
    for fil in files:
      with open(fil, encoding='utf-8') as fi:
        for i,line in enumerate(fi):
            words = [morph.parse(word)[0].normal_form for word in
line.split()]
            for word in words:
              wordID = 0
              for i,d in enumerate(dictionary):
                if d[0] == word:
                  wordID = i
                  features_matrix[docID,wordID] = words.count(word)
        docID = docID + 1
    print('Запись в DataFrame')
    df = pd.DataFrame(features_matrix, in-
```

```
dex=[int(fil[fil.rindex('\\')+1:])for fil in files])
    df.to_hdf('pwd.h5', 'table')

def main(start=None, finish=None, parallel=True, notifications=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_test.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)

    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 = model.score_tracker['SparPhiBckgr'].value
    sparsity_theta = model.score_tracker['SparThetAll'].value
    sparsity_theta_main = model.score_tracker['SparThetMain'].value
    sparsity_theta_bckgr = model.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 = model.score_tracker['TopicKernelScore'].average_contrast
        purity = model.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(250)]
    background_topics = topics[:50]
    main_topics = topics[50:]
    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')
    get_query_data(model, predict_class=None, query_file=QUERY,
save_theta_q=True,
                   save_classes=True, filename='thetaQ_store')

    ## Regularaizers
    # SSPhi
    model.regularizers.add(artm.SmoothSparsePhiRegularizer(name='SSPhiMain',
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='SSThetaMain',
topic_names=main_topics, tau=0.01))
    model.regularizers.add(
        artm.SmoothSparseThetaRegularizer(name='SSThetaBckgr', top-
```

```
ic_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', diction-
ary=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', top-
ic_names=main_topics, class_id='words'))
    model.scores.add(artm.SparsityPhiScore(name='SparPhiBckgr', top-
ic_names=background_topics, class_id='words'))
    # SparThet
    model.scores.add(artm.SparsityThetaScore(name='SparThetAll'))
    model.scores.add(artm.SparsityThetaScore(name='SparThetMain', top-
ic_names=main_topics))
    model.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'))
    model.scores.add(artm.TopicKernelScore(name='TopicKernelScore',
                                           class_id='words',
                                           topic_names=main_topics,
                                           probability_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')

    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='ImpCohPhi',
                                                        tau=1.5,

class_ids='words',
                                                        diction-
ary=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 testSearch.py

```
import pandas as pd
import numpy as np
import operator
import time
import sys
import multiprocessing as mp
from math import log10
from sklearn.linear_model import Ridge
from io import StringIO
from csv import writer
from functools import partial
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

QUERY_PATH1 = r'thetaQ1_store.h5'
QUERY_PATH2 = r'thetaQ_store.h5'
THETA_PATH = r'theta_store.h5'
KOEFS_LOG = [1.00E+01, 1.50E+01, 2.50E+01, 5.00E+01,
         1.00E+02, 2.00E+02, 5.00E+02, 1.00E+03, 1.00E+05, 5.00E+05]
KOEFS_ADJ = [None, 0.1, 0.25, 0.5, 0.75, 0.1, 1, 1.25, 1.5, 1.75]
'''GLOB_REV2 = ['39129', '39097', '39116', '39169', '39177', '3920', '39206',
'39254', '39463',
            '39541', '38723', '38727', '30983', '3968', '39716', '39803',
'39883', '97722',
            '96920', '9663', '96398', '89617', '60265', '58121']'''
'''GLOB_REV1 = ['156579', '228969', '225845', '135529', '142740', '133843',
'209062', '170711',

'94012','81322','135570','96452','148964','123283','180571','174609','222535'
,'5598',
     '149038','218379','222599','198456','118532','156027','155535','180167',
     '115334', '194068']'''

GLOB_REV = ['40231', '89617', '60265', '67283', '99164','48641', '44536',
'88000', '67229',
            '70649', '58121', '88451', '54229', '51419', '76699', '67270']
#GLOB_REV = ['39341', '99030', '40137', '49181', '86688', '46960', '46868',
'51372', '69388', '80730', '58859', '44233']

def timeit(method):
    def timed(*args, **kw):
        ts = time.time()
        result = method(*args, **kw)
        te = time.time()

        print('{} completed in {:.2f}s'.format(method.__name__, (te - ts)))
        return result
    return timed
```

```
#
def swaptheta(name, theta):
    thetaQ = pd.read_hdf(r'queries\thetaQ_'+name+'.h5','table')
    thetaQ = thetaQ[[col for col in thetaQ.columns[50:]]]
    mera = cosin(theta, thetaQ)
    return thetaQ, mera

def cosin(theta, thetaQ_in):
    thetaQ = pd.DataFrame(thetaQ_in)
    mera = {}
    start = time.time()
    # Вычисление нижней меры1
    meraDOWN1= pow(pow(thetaQ.values[0], 2).sum(), 1/2)
    for row in range(len(theta.index)):
        # Вычисление верхней меры
        meraUP = thetaQ.values[0].dot(theta.values[row])
        if meraUP == 0:
            continue
        # Вычисление нижней меры2
        meraDOWN2 = pow(pow(theta.values[row], 2).sum(), 1/2)
        mera[theta.index[row]] = meraUP/(meraDOWN1*meraDOWN2)
    # Сортировка по возрастанию
    mera = sorted(mera.items(), key=operator.itemgetter(1))
    # Разворот
    mera.reverse()
    end = time.time() - start
    print('Calculate in {0:.2f}s...'.format(end))
    return mera

######### logistic ############# start
def data_logistic(theta, mera, rev, irrev, confidence=1e15):
    items = len(mera)
    _x = np.ones((items, 200))
    _y = np.ones(items)
    weights = np.ones(items)
    n = (len(rev) + len(irrev))
    koef_rev = len(irrev) / n
    koef_irrev = len(rev) / n
    for i, item in enumerate(mera):
        _x[i] = theta.loc[item[0]].values
        if item[0] in rev:
            _y[i] = 1
            # Назначаем вес
            weights[i] = confidence * koef_rev
        elif item[0] in irrev:
            _y[i] = 0
            weights[i] = confidence * koef_irrev
    return _x, _y, weights

def logistic(theta, mera, rev, irrev, confidence):
    _x, _y, weights = data_logistic(theta, mera, rev, irrev, confidence)
    fitting_index = weights > 1
    # Логистическая регрессия тут
    logistic = LogisticRegression()
    # Установить модель в соответствии с данными тренировки.
    logistic.fit( x[fitting index],  y[fitting index],
weights[fitting_index])
    # Возвращает вероятностные оценки.
    return logistic.predict_proba(_x)

def print_logistic(predicted, mera, rev, irrev, GLOB_REV, entries=20,
forme=True):
    new_mera = [x for (x, y) in sorted(zip(mera, predicted),
```

```
                                        key=lambda pair: pair[1][1], re-
verse=True)]
    to_print = []
    for item in new_mera:
        if item[0] in rev or item[0] in irrev:
            continue
        if len(to_print) == entries:
            break
        to_print.append(item)

    new_rev = []
    for item in to_print:
        if item[0] in GLOB_REV:
            # Новый релевантный список
            new_rev.append(item)
    if forme:
        print(*to_print, sep='\n')
        print('\nNew relevant documents here [{}]:'.format(len(new_rev)))
        print(*new_rev, sep='\n')
        print('rev/irrev - [{}/{}]'.format(len(rev), len(irrev)))
    return to_print, new_rev

def first_logistic(mera, GLOB_REV, rev_n, entries, frequency):
    rev = GLOB_REV[:rev_n]
    irrev = get_irrev_logistic(mera[:entries], rev, [], frequency)
    print('1. Найдено {0} релевантных документов:'.format(len(rev)), rev)
    print('1. Найдено {0} нерелевантных документов:'.format(len(irrev)),
irrev)
    return rev, irrev

def iterate_logistic(theta, mera, rev, irrev, frequency, confidence, entries,
forme=False):
    #predicted, x, y, w = logistic(theta, mera, rev, irrev, confidence)
    predicted = logistic(theta, mera, rev, irrev, confidence)
    #check_logistic(c, y, mera, GLOB_REV, detailed)
    printed, new_rev = print_logistic(predicted, mera, rev, irrev, GLOB_REV,
entries, forme)
    rev.extend([item[0] for item in new_rev])
    if forme:
        print(len(new_rev), end=' ')
    irrev = get_irrev_logistic(printed, rev, irrev, frequency)
    return rev, irrev

def get_irrev_logistic(printed, rev, irrev, frequency=1):
    new_irrev = []
    for item in printed:
        if item[0] not in rev:
            # Расстановка нового списка нерелевантных документов
            new_irrev.append(item[0])
    irrev.extend(new_irrev[::frequency])
    return irrev

def check_logistic(c, y, mera, GLOB_REV, detailed=False):
    buf1 = []
    # Создаем три списка и сортируем документы по ним
    buf2 = {'in c, not in glob': [],
            'in glob not in c': [],
            'in c and glob not in rev': []}
    for item in c.nonzero()[0]:
        if item not in y.nonzero()[0]:
            buf1.append(mera[item])
            if mera[item][0] in GLOB_REV:
                buf2['in c and glob not in rev'].append(mera[item])
```

```
        if mera[item][0] not in GLOB_REV:
            buf2['in c, not in glob'].append(mera[item])
    print('Items in c but not in y [{}]:'.format(len(buf1)))
    if detailed: print(*buf1, sep='\n')

    buf3 = []
    for item in y.nonzero()[0]:
        if item not in c.nonzero()[0]:
            buf3.append(mera[item])
    print('\nItems in y but not in c (forgotten) [{}]:'.format(len(buf3)))
    if detailed: print(*buf3, sep='\n')

    for item in GLOB_REV:
        if item not in list(map(lambda x: mera[x][0], c.nonzero()[0])):
            buf2['in glob not in c'].append(item)

    print('\nItems in c but not in glob (incorrectly predicted)
[{}]:'.format(len(buf2['in c, not in glob'])))
    if detailed: print(*buf2['in c, not in glob'], sep='\n')
    print('\nItems in glob but not in c (not predicted)
[{}]:'.format(len(buf2['in glob not in c'])))
    if detailed: print(*buf2['in glob not in c'], sep='\n')
    print(
        '\nItems in c and glob but not in y (correctly predicted)
[{}]:'.format(len(buf2['in c and glob not in rev'])))
    if detailed: print(*buf2['in c and glob not in rev'], sep='\n')
    print('\nItems in c - [{}]'.format(c.nonzero()[0].size))

def log_forme(theta, koefs, mera, GLOB_REV, n_rev, entries, frequency,
no_det=True):
    print('\n###### LOG ADJUSTS RESULTS ######')
    for koef in koefs:
        step = 1
        koef_str = '{:E}'.format(koef)
        koef_str = koef_str.split('E')[0].rstrip('0').rstrip('.') + 'e' +
koef_str.split('E')[1]
        print('For koef [{}] :'.format(koef_str))
        rev, irrev = first_logistic(mera, GLOB_REV, n_rev, entries, frequen-
cy)
        while len(rev) < len(GLOB_REV):
            step = step + 1
            rev, irrev = iterate_logistic(theta, mera, rev, irrev, frequency,
koef, entries, no_det)
            print('{1}. Найдено {0} релевантных документов:'.format(len(rev),
step), rev)
            print('{1}. Найдено {0} нерелевантных
документов:'.format(len(irrev), step), irrev)
        print('\n')
######### logistic ############# end|

######### bayes ############# start
def data_bayes(theta, mera, rev, irrev, confidence=1e15):
    #print(confidence, end=' ')
    items = len(mera)
    _x = np.ones((items, 200))
     y = np.ones(items)
    weights = np.ones(items)
    n = (len(rev) + len(irrev))
    koef_rev = len(irrev) / n
    koef_irrev = len(rev) / n
    #print(n, koef_rev, koef_irrev) #За это можно зацепиться
    for i, item in enumerate(mera):
        _x[i] = theta.loc[item[0]].values
```

```
        if item[0] in rev:
            _y[i] = 1
            # Назначаем вес
            weights[i] = confidence * koef_rev
        elif item[0] in irrev:
            _y[i] = 0
            weights[i] = confidence * koef_irrev
    return _x, _y, weights

def bayes(theta, mera, rev, irrev, confidence):
    _x, _y, weights = data_bayes(theta, mera, rev, irrev, confidence)
    fitting_index = weights > 1
    # Логистическая регрессия тут
    bayes = MultinomialNB()
    # Установить модель в соответствии с данными тренировки.
    bayes.fit(_x[fitting_index], _y[fitting_index], weights[fitting_index])
    # Возвращает вероятностные оценки.
    return bayes.predict_proba(_x)

def print_bayes(predicted, mera, rev, irrev, GLOB_REV, entries=20,
forme=True):
    new_mera = [x for (x, y) in sorted(zip(mera, predicted),
                                       key=lambda pair: pair[1][1], re-
verse=True)]
    to_print = []
    for item in new_mera:
        if item[0] in rev or item[0] in irrev:
            continue
        if len(to_print) == entries:
            break
        to_print.append(item)

    new_rev = []
    for item in to_print:
        if item[0] in GLOB_REV:
            # Новый релевантный список
            new_rev.append(item)
    if forme:
        print(*to_print, sep='\n')
        print('\nNew relevant documents here [{}]:'.format(len(new_rev)))
        print(*new_rev, sep='\n')
        print('rev/irrev - [{}/{}]'.format(len(rev), len(irrev)))
    return to_print, new_rev

def first_bayes(mera, GLOB_REV, rev_n, entries, frequency):
    rev = GLOB_REV[:rev_n]
    #print(rev)
    irrev = get_irrev_bayes(mera[:entries], rev, [], frequency)
    #print(irrev)
    print('1. Найдено {0} релевантных документов:'.format(len(rev)), rev)
    print('1. Найдено {0} нерелевантных документов:'.format(len(irrev)),
irrev)
    return rev, irrev

def iterate_bayes(theta, mera, rev, irrev, frequency, confidence, entries,
forme=False):
    predicted = bayes(theta, mera, rev, irrev, confidence)
    # check_logistic(c, y, mera, GLOB_REV, detailed)
    printed, new_rev = print_bayes(predicted, mera, rev, irrev, GLOB_REV, en-
tries, forme)
    rev.extend([item[0] for item in new_rev])
    if forme:
        print(len(new_rev), end=' ')
```

```
    irrev = get_irrev_logistic(printed, rev, irrev, frequency)
    return rev, irrev

def get_irrev_bayes(printed, rev, irrev, frequency=1):
    new_irrev = []
    for item in printed:
        if item[0] not in rev:
            # Расстановка нового списка нерелевантных документов
            new_irrev.append(item[0])
    irrev.extend(new_irrev[::frequency])
    return irrev

def check_bayes(c, y, mera, GLOB_REV, detailed=False):
    buf1 = []
    # Создаем три списка и сортируем документы по ним
    buf2 = {'in c, not in glob': [],
            'in glob not in c': [],
            'in c and glob not in rev': []}
    for item in c.nonzero()[0]:
        if item not in y.nonzero()[0]:
            buf1.append(mera[item])
            if mera[item][0] in GLOB_REV:
                buf2['in c and glob not in rev'].append(mera[item])
        if mera[item][0] not in GLOB_REV:
            buf2['in c, not in glob'].append(mera[item])
    print('Items in c but not in y [{}]:'.format(len(buf1)))
    if detailed: print(*buf1, sep='\n')

    buf3 = []
    for item in y.nonzero()[0]:
        if item not in c.nonzero()[0]:
            buf3.append(mera[item])
    print('\nItems in y but not in c (forgotten) [{}]:'.format(len(buf3)))
    if detailed: print(*buf3, sep='\n')

    for item in GLOB_REV:
        if item not in list(map(lambda x: mera[x][0], c.nonzero()[0])):
            buf2['in glob not in c'].append(item)

    print('\nItems in c but not in glob (incorrectly predicted)
[{}]:'.format(len(buf2['in c, not in glob'])))
    if detailed: print(*buf2['in c, not in glob'], sep='\n')
    print('\nItems in glob but not in c (not predicted)
[{}]:'.format(len(buf2['in glob not in c'])))
    if detailed: print(*buf2['in glob not in c'], sep='\n')
    print(
        '\nItems in c and glob but not in y (correctly predicted)
[{}]:'.format(len(buf2['in c and glob not in rev'])))
    if detailed: print(*buf2['in c and glob not in rev'], sep='\n')
    print('\nItems in c - [{}]'.format(c.nonzero()[0].size))

def bayes_forme(theta, koefs, mera, GLOB_REV, n_rev, entries, frequency,
no_det=True):
    print('\n###### BAYES ADJUSTS RESULTS ######')
    for koef in koefs:
        step = 1
        koef_str = '{:E}'.format(koef)
        koef_str = koef_str.split('E')[0].rstrip('0').rstrip('.') + 'e' +
koef_str.split('E')[1]
        print('For koef [{}] :'.format(koef_str))
        rev, irrev = first_bayes(mera, GLOB_REV, n_rev, entries, frequency)
        while len(rev) < len(GLOB_REV):
            step = step + 1
```

```
            rev, irrev = iterate_bayes(theta, mera, rev, irrev, frequency,
koef, entries, no_det)
            print('{1}. Найдено {0} релевантных документов:'.format(len(rev),
step), rev)
            print('{1}. Найдено {0} нерелевантных
документов:'.format(len(irrev), step), irrev)
        print('\n')
######### bayes ############ end|


def usual_check(mera, GLOB_REV):
    print('\nCosine measure distribution:')
    for i in range(0,100,20):
        cnt = 0
        for item in mera[i:i+20]:
            if item[0] in GLOB_REV:
                cnt += 1
        print(cnt)

def main():
    start_glob = time.time()
    thetaQ1 = pd.read_hdf(QUERY_PATH1,'table')
    thetaQ2 = pd.read_hdf(QUERY_PATH2,'table')
    theta = pd.read_hdf(THETA_PATH,'table')
    theta = theta[[col for col in theta.columns[50:]]]
    thetaQ1 = thetaQ1[[col for col in thetaQ1.columns[50:]]]
    thetaQ2 = thetaQ2[[col for col in thetaQ2.columns[50:]]]
    end = time.time() - start_glob
    print('Read matrices in {0:.2f}s...'.format(end))
    # Эксперимент 1. Запрос: Линукс

    # Вычисление косинусной меры близости документов
    mera = cosin(theta, thetaQ1)
    usual_check(mera, GLOB_REV)
    #print(mera[:100])
    # Передаем тету, тету запрсса, первые 100 документов по мере, коэф весов,
релевантные посты, частоту нерелевантных, частоту релевантных
    #adjust forme(theta, thetaQ, mera[:100], KOEFS ADJ, GLOB REV, 1, 1)
    #log_forme(theta, KOEFS_LOG, mera[:100], GLOB_REV, n_rev=6, entries=20,
frequency=1)
    bayes_forme(theta, KOEFS_LOG, mera[:100], GLOB_REV, n_rev=6, entries=20,
frequency=1)

    # Эксперимент 2. Запрос: Нейронные сети
    # Вычисление косинусной меры близости документов
    #mera = cosin(theta, thetaQ2)
    #usual check(mera, GLOB REV)
    # Передаем тету, тету запрсса, первые 100 документов по мере, коэф весов,
релевантные посты, частоту нерелевантных, частоту релевантных
    #adjust_forme(theta, thetaQ2, mera[:100], KOEFS_ADJ, GLOB_REV, 1, 1)
    #log forme(theta, KOEFS LOG, mera[:100], GLOB REV, n rev=6, entries=20,
frequency=1)
    #bayes_forme(theta, KOEFS_LOG, mera[:100], GLOB_REV, n_rev=6, entries=20,
frequency=1)

if   name   == '  main  ':
    main()
```