МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра информатики и программирования

ТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ДЛЯ АНАЛИЗА ТЕКСТОВЫХ ДОКУМЕНТОВ

КУРСОВАЯ РАБОТА

студента 2 курса 241 группы
направления 02.03.03 — Математическое обеспечение и администрирование
информационных систем
факультета КНиИТ
Диева Ильи Николаевича

Научный руководитель
Зав. кафедрой ИиП, к. ф.-м. н, доцент _____ М. В. Огнева

Заведующий кафедрой
ИиП, к. ф.-м. н, доцент _____ М. В. Огнева

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Тематическое моделирование – современный инструмент, позволяющий автоматически выявлять тематическую структуру больших текстовых коллекций, что является актуальной задачей в эпоху больших интернет-данных. В настоящее время процесс накопления информации настолько стремителен, что простого информационного поиска уже недостаточно для оперативного и адекватного получения информации. Построение тематической модели может рассматриваться как задача одновременной кластеризации документов и слов по одному и тому же множеству кластеров – тем. Тема – результат би-кластеризации, то есть одновременной кластеризации и слов, и документов по их семантической близости.

Обычно выполняется нечёткая кластеризация, то есть документ может принадлежать нескольким темам в различной степени. Таким образом, сжатое семантическое описание слова или документа представляет собой вероятностное распределение на множестве тем. Процесс нахождения этих распределений и называется тематическим моделированием.

Цель курсовой работы:

— построить тематическую модель для анализа текстовых документов на естественном языке.

Задачи курсовой работы:

1. рассмотреть различные методы построения тематических моделей;
2. осуществить предварительную обработку текстов для построения тематической модели;
3. построить тематическую модель.

1 Обзор источников

В качестве основного источника теоретической части работы выступила статья [1].

Основоположниками метода латентного размещения Дирихле для построения тематических моделей в машинном обучении выступили Дэвид Блэй, Эндрю Нг и Майкл И. Джордан в 2003 году со статьей [2]. Также принципы LDA изложены в статьях [3], [4], а примеры использования латентно-семантического анализа взяты из статьи [5]

В статье [6] описано, как тематическое моделирование интенсивно развивается с конца 90-х годов. Предложено множество моделей для решения самых разнообразных задач: тематическая сегментация текстов, классификация и категоризация документов, многоязычный информационный поиск, поиск тематической структуры в сообществах, анализ тональности, тематическая визуализация больших текстовых коллекций и др. Тематические модели могут учитывать различные особенности языка и текстовых коллекций. Существуют модели, выявляющие ключевые фразы, учитывающие морфологию слов и синтаксическую структуру предложений, а также различные характеристики документов — авторство, тэги, ссылки и др., отслеживающие изменения тем во времени, строящие иерархические отношения между темами и др.

Принципы кластеризации описаны в книгах [7], [8], [9], [10]. Также описаны различные подходы к тематическому моделированию, доминирующим из которых в настоящее время является байесовское обучение. Большинство моделей разрабатываются на основе модели латентного размещения Дирихле. Также активно развивается и альтернативный, многокритериальный подход, получивший название «аддитивная регуляризация тематических моделей», в котором модель оптимизируется по взвешенной сумме критериев.

Наиболее углубленно подход к кластеризации рассмотрен в статье [11].

2 Теоретическая часть

2.1 Представление текста перед его обработкой

Для представления документов принято пользоваться векторной моделью, в которой каждому слову сопоставляется вес в соответствии с выбранной весовой функцией. С помощью такого представления для всех документов можно находить расстояние между точками пространства и тем самым решать задачу подобия документов – чем ближе расположены точки, тем больше похожи соответствующие документы.

Классическим методом назначения весов является TF-IDF.

TF-IDF – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Мера TF-IDF часто используется в задачах анализа текстов и информационного поиска, например, как один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации [1].

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D) \tag{1}$$

TF *(term frequency)* – нормализованная частота слова в тексте:

$$TF(t, d) = \frac{freq(t, d)}{\max_{w \subset D} freq(w, d)} \tag{2}$$

Здесь $freq(t, D)$ – число вхождений слова t в документе D.

IDF *(inverse document frequency)* – обратная частота документов:

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3}$$

Здесь в числителе – количество документов в наборе, а в знаменателе – количество документов, в которых встречается слово t.

В зависимости от решаемой задачи используются различные варианты метрики TF-IDF. В классическом случае это $TF \times IDF$, однако, используются и более сложные комбинации данных показателей. В любом случае, учёт IDF

уменьшает вес широкоупотребительных слов, и общее значение TF-IDF для терма будет тем больше, чем выше частота встречаемости терма в конкретном документе и ниже в других документах коллекции.

Несмотря на то, что построение простейшей векторной модели для задач сравнения текстов между собой актуально и сейчас, модель неприменима для текстов больших размеров, к тому же она не позволяет разрешить проблему синонимии и полисемии слов. Развитием векторной модели стало представление наборов векторов термов из документа как общей терм-документной матрицы. Первым из методов, реализующих терм-документное представление коллекции документов, стал метод латентно-семантического индексирования [6].

2.2 Кластеризация

Цель алгоритмов кластеризации – выявить скрытую структуру немаркированных данных с использованием признаков для организации экземпляров в существенно различающиеся группы. При работе с текстовыми данными под экземпляром понимается единственный документ или высказывание, а под признаками – лексемы, словарь, структура, метаданные и т. д [8].

Популярный метод для задач обучения без учителя, алгоритм кластеризации k-средних начинается с произвольно выбранного количества кластеров, k, и разбивает векторизованные экземпляры на кластеры согласно их близости к центроидам, которые минимизируют суммы квадратов внутри кластеров [7].

Кластеризация k-средних – один из самых простых и наиболее часто используемых алгоритмов кластеризации. Сначала выбирается число кластеров k. После выбора значения k алгоритм k-средних отбирает точки, которые будут представлять центры кластеров. Затем для каждой точки данных вычисляется его евклидово расстояние до каждого центра кластера. Каждая точка назначается ближайшему центру кластера. Алгоритм вычисляет центроиды – центры тяжести кластеров. Каждый центроид -– это вектор, элементы которого представляют собой средние значения характеристик, вычисленные по всем точкам кластера [9]. Центр кластера смещается в его центроид. Точки заново назначаются ближайшему центру кластера. Этапы изменения центров кластеров и переназначения точек итеративно повторяются до тех пор, пока границы кластеров и расположение центроидов не перестанут изменяться, т.е. на каждой итерации в каждый кластер будут попадать одни и те же точки данных [10].

Другой подход к кластеризации – алгомеративная. Агломеративная кла-

стеризация относится к семейству алгоритмов кластеризации, в основе которых лежат одинаковые принципы: алгоритм начинает свою работу с того, что каждую точку данных заносит в свой собственный кластер и по мере выполнения объединяет два наиболее схожих между собой кластера до тех пор, пока не будет удовлетворен определенный критерий остановки [1].

2.3 Методы построения тематических моделей

Тематическое моделирование – прием машинного обучения без учителя для определения тем коллекций документов. Если цель кластеризации – разделить корпус документов на группы, то цель тематического моделирования – выделить основные темы из набора высказываний; кластеризация дедуктивна, а тематическое моделирование – индуктивно.

2.3.1 Латентный семантический анализ

Латентно-семантический анализ (Latent semantic analysis, LSA) – это метод обработки информации на естественном языке, анализирующий взаимосвязь между библиотекой документов и терминами, в них встречающимися, и выявляющий характерные факторы, присущие всем документам и терминам с целью повышения эффективности работы информационно-поисковых систем [12].

Основная идея метода состоит в оценивании корреляции терминов путем анализа их совместной встречаемости в документах. К примеру, в коллекции всего 100 документов, содержащих термины «доступ» и/или «поиск». При этом только 95 из них содержат оба термина вместе. Логично предположить, что отсутствие термина «поиск» вместе с термином «доступ» ошибочно и возвращать данный документ по запросу, содержащему только термин «доступ». Подобные выводы можно делать не только из простой попарной корреляции терминов.

С другой стороны, анализируя корреляцию терминов в запросе, можно более точно определять интересубщий пользователя смысл основного термина и повышать позиции документов, соответствующих этому смыслу, в результатах поиска.

Таким образом, при латентно-семантическом анализе документов задача состоит в том, чтобы спроецировать часто встречающиеся вместе термины в одно и то же измерение семантического пространства, которое имеет пониженную размерность по сравнению с оригинальной терм-документной матрицей, кото-

рая обычно довольно разрежена. Элементы этой матрицы содержат веса терминов в документах, назначенные с помощью выбранной весовой функции [5].

Наиболее распространенный вариант LSA основан на использовании резложения терм-документной матрицы по сингулярным значениям – сингулярном разложении, хорошо зарекомендовавшем себя в факторном анализе.

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

$$A = TSD^T \tag{4}$$

где матрицы T и D – ортогональные, а S – диагональная матрица, элементы на диагонали которой называются сингулярными значениями матрицы A.

Если в матрице S оставить только k наибольших сингулярных значений, а в матрицах T и D – только соответствующие этим значениям столбцы, то произведение получившихся матриц S, T и D будет наилучшим приближением исходной матрицы A к матрице Å ранга k.

Если в качестве матрицы A взять терм-документную матрицу, то матрица Å , содержащая только k первых линейно независимых компонент A, отражает основную структуру различных зависимостей, присутствующих в исходной матрице. Структура зависимостей определяется весовыми функциями терминов.

Таким образом, каждый термин и документ представляются при помощи векторов в общем семантическом пространстве размерности k. Близость между любой комбинацией терминов и/или документов легко вычисляется при помощи скалярного произведения векторов.

Выбор k зависит от поставленной задачи и подбирается эмпирически. Если выбранное значение k слишком велико, метод теряет свою мощность и приближается по характеристика к стандартным векторным методам. Слишком маленькое значение k не позволяет улавливать различия между похожими терминами или документами.

2.3.2 Латентное размещение Дирихле

Впервые предложенный Дэвидом Блеем, Эндрю Ыном и Майклом Джорданом в 2003 г., метод латентного размещения Дирихле (Latent Dirichlet Allocation, LDA) является методом определения темы. Он принадлежит семейству порож-

дающих вероятностных моделей, в которых темы представлены вероятностями появления каждого слова из заданного набора. Документы, в свою очередь, могут быть представлены как сочетания этих тем [3]. Уникальная особенность моделей LDA состоит в том, что темы не обязательно должны быть различными и слова могут встречаться в нескольких темах; это придает некоторую нечеткость определяемым темам, что может пригодиться для совладания с гибкостью языка [11].

Распределение Дирихле, семейство непрерывных распределений (способ измерения группировки по распределениям), – это удобный способ выявления тем, присутствующих в корпусе (наборе текстов), а также проявляющихся в разных сочетаниях в каждом документе в корпусе. Фактически, метод латентного размещения Дирихле дает нам наблюдаемое слово или лексему, по которому можно попытаться определить вероятную тему, распределение слов в каждой теме и сочетание тем в документе [4].

LDA – это вероятностная модель, которая использует как Дирихле, так и мультиномиальные распределения. Распределения Дирихле и мультиномиальные распределения являются обобщениями бета- и биномиальных распределений. В то время как бета- и биномиальные распределения можно понимать как случайные процессы, связанные с подбрасыванием монет (возвращением дискретного значения), Дирихле и мультиномиальные распределения имеют дело со случайными процессами.

Чтобы лучше понимать, как работают распределения в LDA, представим, что имеется документ, полностью состоящий из слов p и q, при этом документ генерируется путем просчитывания вероятности двух событий – $P(p)$ и $P(q)$, где $P(q) = 1 - P(p)$. Если заранее знать эти вероятности, можно просчитать вероятность получения документа, содержащего n слов, среди которых k слов – p, с помощью формулы Бернулли:

$$P = C_n^k P(p)^k P(q)^{n-k} \tag{5}$$

Однако вероятности $P(p)$ и $P(q)$ заранее неизвестны, но если рассматривать конкретный документ, состоящий, например, из семи слов p и трех слов q, можно было бы предположить, что $P(p) = 0.7$, а $P(q) = 0.3$. Рассмотрение документа, состоящего из большего числа слов, могло бы уточнить эти вероятности, если в документе из 1000 слов встретилось бы 700 слово p и 300

слово q. Именно бета-распределение дает возможность количественно оценить укрепление этих убеждений после получения дополнительных доказательств. Бета-распределение принимает два параметра α и β и создает распределение вероятностей. Параметры α и β представляют, сколько предварительных знаний имеется о вероятностях. Более низкие значения α и β приводят к более широкому распределению и отражают неопределенность и отсутствие предварительных знаний. С другой стороны, большие значения α и β дают распределение с резким пиком около определенного значения. Это означает, что мы можем подтвердить наше утверждение о том, что $P(p) = 0,7$.

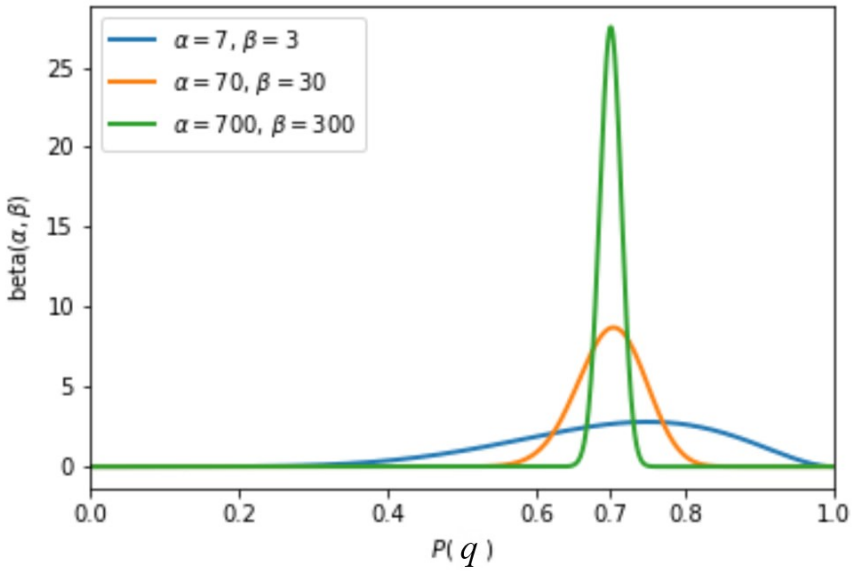

Рисунок 1 – Иллюстрация бета-распределения.

Таким образом, имеется гипотеза, – апроприорное убеждение, – что $P(p) = 0,7$. Согласно формуле Байеса можно описать вероятность события на основе априорных знаний, которая показывает, как с учетом некоторой вероятности, гипотезы и доказательств можно получить апостериорную вероятность:

$$P(H|A) = \frac{P(H)P(A|H)}{P(A)}, \tag{6}$$

где $P(H)$ – предварительная вероятность, вероятность гипотезы, $P(A|H)$ – вероятность доказательства A, при условии, что гипотеза H верна, $P(A)$ – априорная вероятность, что само доказательство истинно, $P(H|A)$ – апостериорная вероятность H, учитывая доказательство.

Априорная вероятность определяет прежние убеждения о событии. Идея состоит в том, что предполагается некоторое априорное распределение, наибо-

лее разумное с учетом наших лучших знаний. В примере выше она равна 0.7. Апостериорная вероятность – вероятность априорной вероятности с учетом, что наступило событие A. Вероятность свидетельства – свидетельством являются наблюдаемые данные/результат эксперимента. Другими словами, наличие высокой вероятности свидетельства ведет к малой апостериорной вероятности и наоборот. Высокая вероятность свидетельства отражает то, что альтернативные гипотезы так же совместимы с данными, как и текущая.

В методе LDA утверждается, что каждый документ представляет собой смесь небольшого количества тем и что появление каждого слова связано с одной из тем документа.

Согласно LDA, каждый документ генерируется независимо:

1. Случайно выбрать для документа его распределение по темам θ_d
2. Для каждого слова в документе:
   а) Случайно выбрать тему из распределения θ_d, полученного на первом шаге
   б) Случайно выбрать слово из распределения слов в выбранной теме φ_k

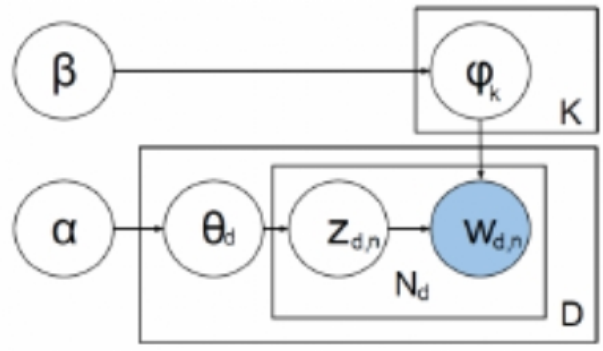

Рисунок 2 – Графическое представление модели латентного размещения Дирихле.

Таким образом, модель LDA примитивно считает, что все документы генерируются следующим образом: случайным образом выбрается тема, затем выбирается слово из этой темы и помещается в документ, после чего процесс повторяется. Согласно теореме Байеса, темы и документы представимы в виде формулы:

$$P(w|d) = \sum_z P(w|\varphi_k)P(z|\theta_d), \tag{7}$$

где w – определенное слово взятое из документа d, каждый документ состоит из смеси тем, где каждая тема z взята из полиномиального распределения

$z\ Mult(\theta)$, θ – распределение тем данного документа, то есть вероятность появления темы в документе. Чтобы определить значение θ, выбирается тематическое распределение из распределения Дирихле. Каждый документ имеет разное распределение тем, поэтому требуется построить распределения тем θ каждого документа из распределения Дирихле. Распределение Дирихле с использованием α (гипотезы) в качестве входного параметра для создания тематического распределения θ, то есть $\theta\ Dir(\alpha)$. Значение α – предварительная информация о смесях тем для этого документа. Затем используется θ, сгенерированное распределением Дирихле, в качестве параметров полиномиального распределения $z\ Mult(\theta)$ для генерации темы следующего слова в документе. Каждая тема z состоит из смеси слов, где каждое слово взято из полиномиального распределения $w\ Mult(\varphi)$. Назовем φ распределением слов по каждой теме, то есть вероятностью появления каждого слова в словаре в данной темы z. Чтобы определить значение φ, выбирается распределение слов по заданной теме из распределения Дирихле $\varphi z\ Dir(\beta)$, используя в качестве входного параметра β — априорную информацию о частоте слов в теме. Затем используется φ, сгенерированный из $Dir(\beta)$, в качестве параметра полиномиального распределения для выборки следующего слова в документе – при условии, что мы уже знали тему следующего слова [13].

Весь процесс генерации LDA для каждого документа выглядит следующим образом:

$$P(w, z, \theta, \varphi|\alpha, \beta) = P(\theta|\alpha)P(z|\theta)P(\varphi|\beta)P(w|\varphi) \tag{8}$$

Пусть имеется набор из D документов. Каждый документ состоит из N_d слов, w_{dn} соответствует наблюдаемым переменным – словам в документе. Остальные переменные скрытые. Переменная z_{dn} принимает значение темы, выбранной на шаге 2а для слова w_{dn}. Для каждого документа d пременная θ_d представляет собой распределение тем в этом документе.

В классической модели LDA количество тем фиксировано изначально и задается в явно виде параметром K. φ_k – распределение слов в теме k. Можно подобрать оптимальное значение K, варьируя его и измеряя способность модели предсказывать неизвестные данные.

Как правило, все компоненты параметров α и β распределения Дирихле берутся равными, поскольку отсутствует априорная информация о распреде-

лении слов в темах и тем в документах. На практике обычно используются значения, наиболее характерные для конкретных данных. К примеру, значение параметра, близкое к нулю, позволяет после оценивания параметров модели получить мультиномиальное распределение, в котором большая часть плотности вероятности сосредоточена на небольшом наборе значений [2].

2.3.3 Неотрицательное матричное разложение

Неотрицательная матричная факторизация также является контролируемым методом обучения, который выполняет кластеризацию, а также уменьшение размерности. Он может быть использован в сочетании со схемой TF-IDF для выполнения тематического моделирования.

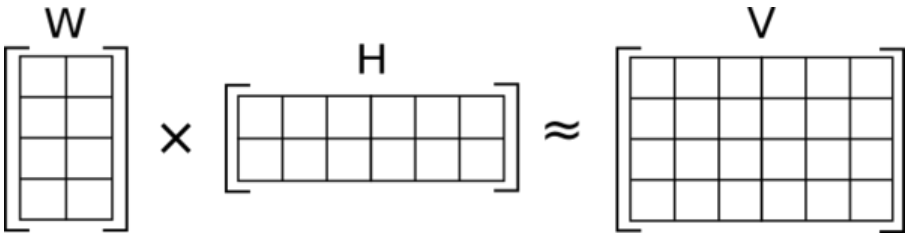

Рисунок 3 – Иллюстрация приближённого неотрицательного матричного разложения: матрица V представлена двумя меньшими матрицами W и H, которые при умножении приблизительно воспроизводят V.

Неотрицательное матричное разложение (NMP) – это группа алгоритмов в линейной алгебре, в которых матрица V разлагается на две матрицы W и H, со свойством, что все три матрицы имеют неотрицательные элементы. Эта неотрицательность делает получившиеся матрицы более простыми для исследования.

Говоря о тематическом моделировании, неотрицательная матричная факторизация – это статистический метод, позволяющий уменьшить размерность входных корпусов. Он использует метод факторного анализа, чтобы придать сравнительно меньший вес словам с меньшей связностью.

Для общего случая рассмотрим, что у нас есть входная матрица V формы mxn. Этот метод разлагает V на две матрицы W и H так, что размерность W равна $m \times k$, а размерность H равна $n \times k$. В нашей ситуации V представляет собой матрицу документ-терм, каждая строка матрицы H представляет собой вложение слова, а каждый столбец матрицы W представляет вес каждого слова в каждом предложении [14].

Предположим, имеется некоторый набор данных. В матрице терминов документа (матрица ввода) у нас есть отдельные документы по строкам матрицы

и каждый уникальный термин по столбцам. В этом методе учитывается каждое отдельное слово в матрице терминов документа. При разложении на множители каждому слову присваивается вес, основанный на семантической взаимосвязи между словами. Но тема с наибольшим весом рассматривается как тема для набора слов, а процесс представляет собой взвешенную сумму различных слов, присутствующих в документах.

НМР имеет внутреннее свойство кластеризации, т.е. он автоматически кластеризует столбцы входных данных $V = (v_1, ..., v_n)$. Приближение V посредством $V \simeq WH$ достигается минимизацией функции ошибок

$$\min \|V - WH\|_F, \tag{9}$$

при условиях $W \geq 0$, $H \geq 0$.

Более того, вычисленная матрица H даёт индикатор кластеров, т.е. если $H_{kj} > 0$, этот факт показывает, что входные данные v_j принадлежат k-му кластеру. Вычисленная же матрица W даёт центры кластеров, т.е. k-ый столбец задаёт центр k-го кластера.

Если ортогональность $HH^T = E$ не указана явно, ортогональность выполняется достаточно сильно и свойство кластеризации также имеет место. Если в качестве функции ошибки используется расстояние Кульбака—Лейблера, НМР идентично вероятностному латентно-семантическому анализу.

3 Практическая часть

3.1 Обзор инструментов

Все модели и код написаны на языке Python.

По ходу исследования будут использованы следующие библиотеки:

— pandas и matplotlib для представления и визуализации данных;
— wordcloud для представления слов по тематикам в качестве облака слов;
— sklearn для преобразования данных, в частности sklearn.metrics для подсчета точности и инструмент sklearn.CountVectorizer для векторазации данных;
— для реализации моделей LDA, LSA использована библиотека gensim.

Для практической части работы был выбран датасет «Topic modelling for research articles» [15], содержащий набор текстов на английском языке, каждый из которых относится к одной из четырех категорий:

— компьютерные науки;
— математика;
— физика;
— статистика.

В колонках соответствующих категорий напротив каждого текста стоит «1», если текст относится к данной тематике и «0» в ином случае.

3.2 Построение LDA модели

После загрузки данных используется инструмент библиотеки sklearn для векторизации текстов, расположенных в таблице колонке ABSTRACT таблицы papers. Векторизатор преобразовывает тексты документа в матрицу количества вхождений токенов. Здесь же происходит отбор слов по шаблону регулярного выражения и очистка текста от стоп-слов – слов, которые не несут смысловую нагрузку и служат для связи слов в предложении для читаемости. К таким стоп-словам относятся местоимения, предлоги, артикли, вводные слова, союзы и частицы. Требуется очистить текст от подобных слов, так как модель будет считать количество вхождений слов в документ, а так как подобные части речи встречаются встречаются в любых текстах в больших количествах, это может повлиять на точность модели. Так, если не произвести очистку от стоп-слов, на выходе из модели все тематики будут состоять исключительно из них. Помимо этого, параметры min_df=20, max_df=0.2 задают игнорирование терминов,

встречающихся слишком редко, или наоборот, слишком часто в тексте. Часто встречающиеся слова, как правило, не несут значительной семантической нагрузки, но и не относятся к стоп-словам, вследствие чего от них также требуется избавиться. Слова с низким числом вхождений также отбрасываются. После чего происходит формирование матрицы документ-терм.

```
#преобразование текста
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
                       token_pattern='(?u)\\b\\w\\w\\w+\\b')
X = vect.fit_transform(papers.ABSTRACT)

#формирование мешка слов и списка идентификаторов
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())

#построение модели
ldamodel = gensim.models.LdaMulticore(corpus=corpus, id2word=id_map, passes=2, random_state=5,
                       num_topics=4, workers=2)
```

Мешок слов и список идентификаторов – необходимые параметры для создания модели. Мешок слов (bag of words) – упрощенное представление текста в виде мультимножества слов без учета грамматики и семантического порядка, но с сохранением информации об их количестве. Частотность вхождения каждого слова используется моделью как признак (параметр) для построения распределений и подсчета вероятностей. В списке идентификаторов же каждому слову из текста присваивается целочисленное неотрицательное число – идентификатор. Мешок слов состоит из пар (идентификатор, частотность) для каждого слова в документе.

Другой важный параметр для модели – число тематик. На текущий момент нет эффективного способа определения количества тематик в документе, если не знать его изначально. Неверный выбор количества тем приведет к меньшей точности модели. Однако существуют механизмы, позволяющие оценить наиболее подходящее число тем, например, модель CoherenceModel, которая обучает несколько моделей на разном исходном количестве тем и выдает соответствующие оценки когерентности. Когерентность, или показатель согласованности используется в тематическом моделировании, чтобы измерить, насколько темы понятны людям. В этом случае темы представлены в виде N первых слов с наибольшей вероятностью принадлежности к конкретной теме. Вкратце, оценка когерентности измеряет, насколько похожи эти слова друг на друга. Так как

изначально количество тем в документах известно, выбрано число 4.

После обучения модели имеются сгенерированные тематики и принадлежащие к ним слова. Их можно получить с помощью метода print_topics() библиотеки gensim модели Lda, но для наглядности будут использованы облака слов, представляющие часто встречающиеся слова из каждой тематики по правилу: чем чаще встречается слово, тем больше его размер в облаке.

```
#цикл для вывода облака слов
for t in range(ldamodel.num_topics):
    plt.figure()
    plt.imshow(WordCloud(background_color="white").fit_words(dict(ldamodel.show_topic(t, 100))))
    plt.axis("off")
    plt.title("Topic #" + str(t))
    plt.show()
```

Получим следующие результаты:

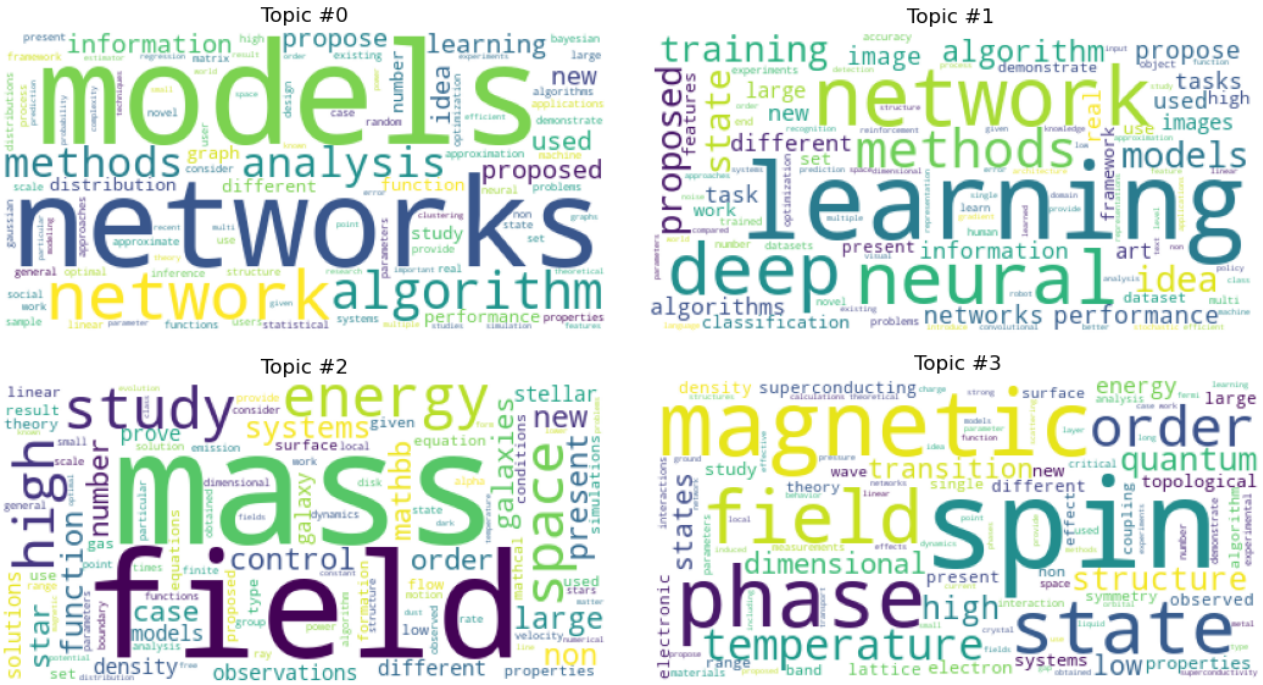

Рисунок 4 – Темы, выявленные LDA

Здесь и в дальнейшем важно помнить, что темы, построенные моделью, являются лишь абстракцией. Модель не способна описать тему одним емким словом, категорией. Для нее это лишь список терминов и их встречаемость в каждой теме.

Как можно наблюдать, возвращаясь к тому факту, что все тексты относятся к одной из четырех заданных тем, тема №0 может определять категорию «Статистика», тема №1 скорее относится к категории «Компьютерные науки»,

тема №2 может определять как «Математику», так и «Физику», а в теме №3 четко прослеживается категория «Физика».

Модель была обучена на тренировочных данных, и теперь можно подать ей на вход тестовые данные и запустить процесс предсказания – пусть модель скажет, к какой из сгенерированных ей ранее категорий относятся новые тексты. Соответствующая функция topic_prediction() принимает на вход очередной текст, очищает и векторизует его, после чего модель определяет, на какую из тем больше всего похожа тема этого текста. Функция вернет число в диапазоне от 0 до 3, соответствующее теме из приведенной выше нумерации.

```
#функция предсказания
def topic_prediction(my_document):
    string_input = [my_document]
    X = vect.transform(string_input)
    corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
    output = list(ldamodel[corpus])[0]
    topics = sorted(output,key=lambda x:x[1],reverse=True)
    return topics[0][0]
```

Тестовые данные также имеют четыре колонки для определения реальной темы текста, функция target() соотнесет текст с его исходной темой и вернет число от 0 до 3 согласно описанной выше нумерации тем.

```
#соотнесение текста и числа
def target(x):
    if tests["Computer Science"][x] == 1:
        return 1
    elif tests["Mathematics"][x] == 1:
        return 2
    elif tests["Physics"][x] == 1:
        return 3
    elif tests["Statistics"][x] == 1:
        return 0
```

После добавления к исходной таблице данных колонок target и prediction соответствующих реальной и предсказанной тематике текста, останется сравнить колонки и подсчитать точность модели.

```
#подсчет точности
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(tests['target'], tests['prediction']))
print(classification_report(tests['target'], tests['prediction']))
```

	precision	recall	f1-score	support
0	0.32	0.61	0.42	826
1	0.80	0.59	0.68	2529
2	0.45	0.77	0.57	1031
3	0.97	0.46	0.63	1616
accuracy			0.59	6002
macro avg	0.63	0.61	0.57	6002
weighted avg	0.72	0.59	0.61	6002

Рисунок 5 – Точность LDA

```
[[ 506  307    8    5]
 [ 854 1499  165   11]
 [ 187   43  794    7]
 [  52   22  794  748]]
```

Рисунок 6 – Матрица путаницы LDA

Как можно наблюдать, модель определила темы верно с точностью 61%. На рисунке 6 изображена матрица путаницы: каждый столбец соответсвует действительной теме документа, значение в строках – сколько документов из данной тематики были определены моделью в другие темы.

3.3 Построение LSA модели

Так как реализации моделей LSA и LDA взяты из одной библиотеки, процесс подготовки документов и обучения модели полностью идентичен описанию в предыдущем разделе.

```
#построение LSA
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
                       token_pattern='(?u)\\b\\w\\w\\w+\\b')
X = vect.fit_transform(papers.ABSTRACT)
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())
lsamodel = gensim.models.LsiModel(corpus, id2word=id_map,
                       num_topics=4)
```

Моделью был сгенерирован следующий набор тем:

Стоит отметить, что уже на данном этапе можно говорить о преимуществах LDA перед LSA. При повторном запуске LDA, слова в выявленных тематиках практически не меняются, при повторном же запуске LSA компоненты тем могут разительно отличаться друг от друга, и для достижения представленного результата пришлось перезапустить модель несколько раз. Говоря о минусах LSA, стоит также отметить, что слово «learning» появилось в трех из четырех темах, а судя по пустотам в облаке слов темы №1 можно говорить, что модель определила в эту тему гораздно меньше слов, чем во все остальные – меньше ста.

Можно заметить следующее: тема №0 предположительно относится к категории «Компьютерные науки», тема №1 больше похожа на «Статистику», тема

Рисунок 7 – Темы, выявленные LSA

№2 скорее всего является «Математикой» и тема №3 сочетает в себе слова из категории «Физика».

После осуществления загрузки тестовых данных и описания функций target() и topic_prediction(), аналогичных соответствующим функциям в предыдущем разделе, получен следующий результат оценки точности модели:

	precision	recall	f1-score	support
0	0.43	0.97	0.60	2529
1	0.25	0.00	0.00	826
2	0.47	0.08	0.14	1031
3	0.99	0.10	0.18	1616
accuracy			0.45	6002
macro avg	0.54	0.29	0.23	6002
weighted avg	0.56	0.45	0.33	6002

Рисунок 8 – Точность LSA

```
[[2441    3   83    2]
 [ 810    1   15    0]
 [ 944    0   87    0]
 [1453    0    0  163]]
```

Рисунок 9 – Матрица путаницы LSA

Модель удачно справилась с определением тем текста с точностью 33%.

Можно осуществить переход в семантическое пространство с помощью модели SVD библиотеки sklearn. Семантическое пространство – условная система координат, где каждый текст представлен в виде точки с определенными координатами. В семантическом пространстве можно наглядно увидеть, насколько один текст далеко расположен от другого. Визуализация семантического пространства выполнена с помощью средств библиотеки upam – методов уменьшения размерности.

```
from sklearn.decomposition import TruncatedSVD
# SVD represent documents and terms in vectors
svd_model = TruncatedSVD(n_components=4, algorithm='randomized', n_iter=100, random_state=122)
svd_model.fit(X)
import umap.umap_ as umap
X_topics = svd_model.fit_transform(X)
embedding = umap.UMAP(n_neighbors=150, min_dist=0.5, random_state=12).fit_transform(X)
plt.figure(figsize=(7,5))
scatter = plt.scatter(embedding[:, 0], embedding[:, 1],
c = papers.target,
s = 10, # size
edgecolor='none')
plt.colorbar()
plt.show()
```

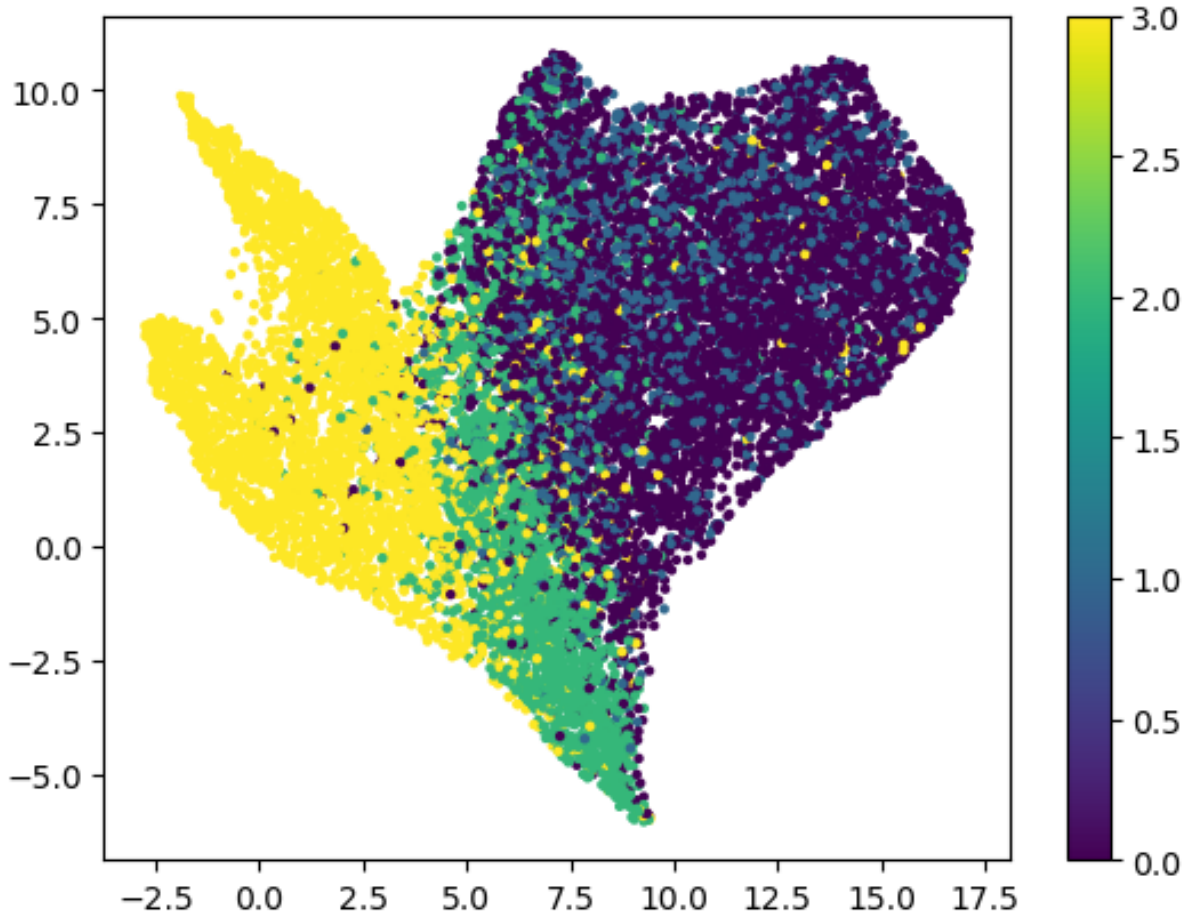

Рисунок 10 – Семантическое пространство LSA

Каждая точка – текст – окрашена в один из четырех цветов, согласно цветовой шкале справа, где нужно обратить внимание лишь на целочисленные значения – число соответствует нумерации тем. Как можно заметить, тема №3 – «Физика», наиболее далека от темы №0 – «Компьютерные науки», а тема №1 – «Статистика», почти незаметна на фоне темы №0. Последний факт также

является объяснением, почему в таблице точности модели тема №1 не была определена вовсе – 0% точности.

3.4 Построение матриц NMF

Для реализации NMF использована библиотека sklearn, вследствии чего и процесс создания модели немного отличается.

```
vectorizer = CountVectorizer(max_features=1500, min_df=10, stop_words='english',
             token_pattern='(?u)\\b\\w\\w+\\b')
X = vectorizer.fit_transform(text_data)
#words - мешок слов
words = np.array(vectorizer.get_feature_names_out())
#построение NMF
nmf = NMF(n_components=4, solver="mu")
#solver - параметр для оптимизации модели
W = nmf.fit_transform(X)
H = nmf.components_
```

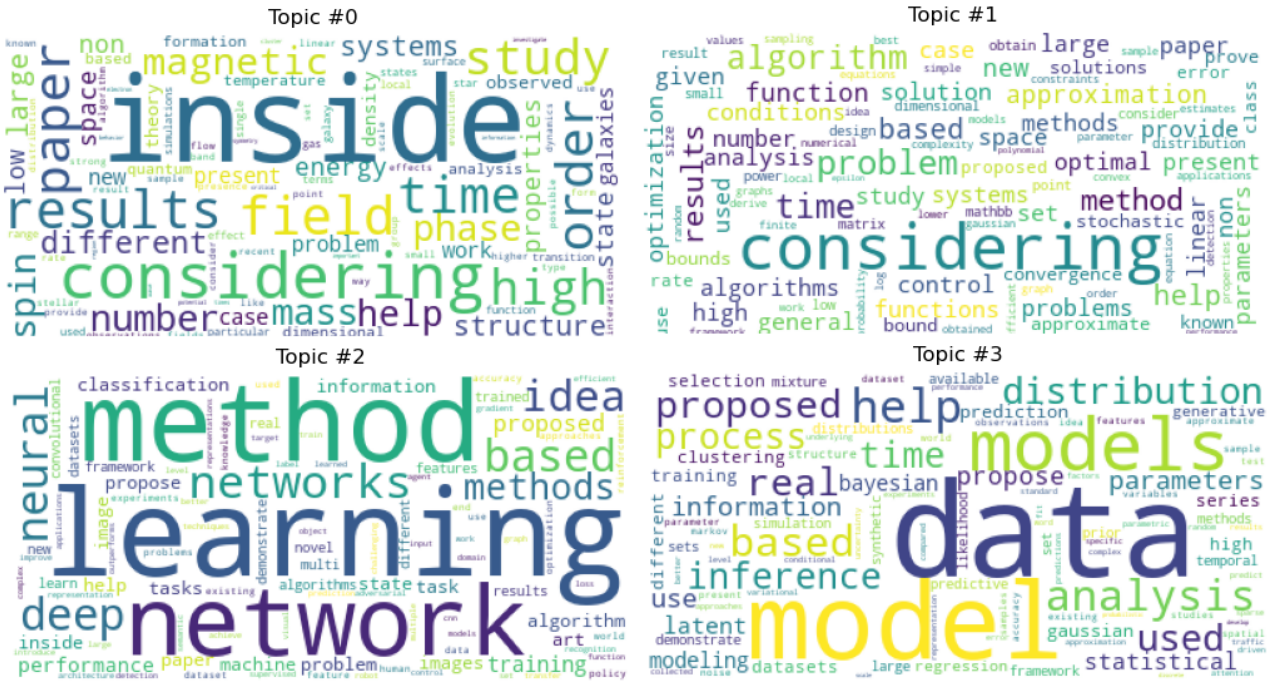

Рисунок 11 – Темы, выявленные NMF

Можно заметить следующее: тема №0 соответствует категории «Физика», тема №1 – «Математика», тема №2 – «Компьютерные науки», тема №3 – «Статистика».

Также отличается и процесс предсказания новых текстов:

```
#подготовка новых данных
new_text = [x for x in tests.ABSTRACT]
vect_new = vectorizer.transform(new_text)
```

```
4    X_new = nmf.transform(vect_new)
5    #предсказание
6    predicted_topics = [np.argsort(each)[::-1][0] for each in X_new]
7    tests['prediction'] = predicted_topics
```

Результат после оценки точности оказался лучше, чем у LSA – 44% верно предсказанных тем:

	precision	recall	f1-score	support
0	0.43	0.65	0.52	1616
1	0.29	0.43	0.35	1031
2	0.74	0.34	0.46	2529
3	0.31	0.33	0.32	826
accuracy			0.44	6002
macro avg	0.44	0.44	0.41	6002
weighted avg	0.52	0.44	0.44	6002

Рисунок 12 – Точность NMF

```
[[1058  350   35  173]
 [ 467  444   54   66]
 [ 733  578  852  366]
 [ 194  146  215  271]]
```

Рисунок 13 – Матрица путаницы NMF

Подводя итог, можно сказать, что наихудший результат показала модель LSA: как видно из матрицы путаницы на рисунке 9 в тему №1 был верно определен лишь один документ. Это можно объяснить тем, что в семантическом пространстве (рисунок 10) тема №1 почти полностью сливается с темой №0, что делает определение этой темы затруднительным. Также следует подчеркнуть, что LSA хорошо подходит для решения задачи семантического сходства, в то время как LDA лучше справляется с тематическим моделированием. NMF-модель показала средние результаты: это объясняется тем, что она более универсальна и подходит для решения самых различных задач машинного обучения: от кластеризации до тематического моделирования.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены различные методы построения тематических моделей. В теоретической части были представлены латентное распределение Дирихле, латнетно-семантический анализ и неотрицательное матричное разложение, а также проанализирован процесс кластеризации текстовых документов по темам, включающий в себя алгоритм метода k-средних и алгомеративную кластеризацию.

В практической части исследования были построены соответствующие модели и проведена оценка точности выявления тематик каждой модели.

В результате исследования было выявлено, что наилучшие результаты показало латентное размещение Дирихле, с оценкой точности в 61%, а латентно-семантический анализ наоборот, показал худший результат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 А. Коршунов, Тематическое моделирование текстов на естественном языке / А. Коршунов // Труды Института системного программирования РАН. — 2012. — Т. 23. — Pp. 215–243.

2 D.M. Blei A.Y. Ng, M. J. Latent dirichlet allocation / M. J. D.M. Blei, A.Y. Ng // Journal of Machine Learning Research. — 2003. — Т. 3. — Pp. 993–1022.

3 Тематическое моделирование в действии. LDA [Электронный ресурс]. — URL: https://lambda-it.ru/post/tematicheskoe-modelirovanie-v-deistvii-lda (Дата обращения 06.12.2022). Загл. с экр. Яз. рус.

4 Latent Dirichlet Allocation [Электронный ресурс]. — URL: https://medium.com/analytics-vidhya/latent-dirichelt-allocation-1ec8729589d4 (Дата обращения 06.12.2022). Загл. с экр. Яз. англ.

5 Латентно-семантический анализ [Электронный ресурс]. — URL: https://habr.com/ru/post/110078/ (Дата обращения 06.12.2022). Загл. с экр. Яз. рус.

6 Милкова, Тематические модели как инструмент «дальнего чтения» / Милкова // Цифровая экономика. — 2019. — Т. 5. — Pp. 57–70.

7 Коэльо, Л. П.. Построение систем машинного обучения на языке Python / Л. П.. Коэльо, В. Ричарт. — Москва : ДМК Пресс, 2016. — 302 с.

8 Грасс, Дж. Data Science. Наука о данных с нуля / Дж. Грасс. — Санкт-Петербург : БХВ-Петербург, 2019. — 411 с.

9 Мюллер, А.П. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными / А.П. Мюллер, С. Гвидо. — Москва : Вильямс, 2017. — 343 с.

10 Бенгфорт, Б.. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка / Б.. Бенгфорт, Р. Билбро, Т. Охеда. — Санкт-Петербург : Питер, 2018. — 368 с.

11 Краснов, Оценка оптимального количества тематик в тематической модели: подход на основе качества кластеров / Краснов // International Journal of Open Information Technologies. — 2019. — Т. 7. — Pp. 8–15.

12 Тематическое моделирование на пути к разведочному информационному поиску [Электронный ресурс]. — URL: https://habr.com/ru/company/yandex/blog/313340/ (Дата обращения 06.12.2022). Загл. с экр. Яз. рус.

13 Тематическое моделирование с использованием LDA [Электронный ресурс]. — URL: https://dev-gang.ru/article/tematiczeskoe-modelirovanie-s-ispolzovaniem-lda-map81iwksb/ (Дата обращения 06.12.2022). Загл. с экр. Яз. рус.

14 Topic modeling articles with NMF [Электронный ресурс]. — https://towardsdatascience.com/topic-modeling-articles-with-nmf-8c6b2a227a45 (Дата обращения 06.12.2022). Загл. с экр. Яз. англ.

15 Topic modeling for research articles [Электронный ресурс]. — https://www.kaggle.com/datasets/abisheksudarshan/topic-modeling-for-research-articles (Дата обращения 06.12.2022). Загл. с экр. Яз. англ.

Приложение А
Построение LDA модели

```
#используемые библиотеки
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import gensim
from sklearn.feature_extraction import text
from gensim.models import LdaMulticore
from gensim.corpora import Dictionary
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.metrics import confusion_matrix, classification_report

#загрузка данных
papers = pd.read_csv("temp/nlp/Train.csv")

#подготовка данных и векторизация
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
            token_pattern='(?u)\\b\\w\\w\\w+\\b')
X = vect.fit_transform(papers.ABSTRACT)

#мешок слов и список идентификаторов
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())

#построение модели
ldamodel = gensim.models.LdaMulticore(corpus=corpus, id2word=id_map, passes=2, random_state=5,
            num_topics=4, workers=2)

#вывод слов по темам
for idx, topic in ldamodel.print_topics(-1):
    print("Topic: {} \nWords: {}".format(idx, topic))
    print("\n")

#вывод облака слов
for t in range(ldamodel.num_topics):
    plt.figure()
    plt.imshow(WordCloud(background_color="white").fit_words(dict(ldamodel.show_topic(t, 100))))
    plt.axis("off")
    plt.title("Topic #" + str(t))
    plt.show()

#загрузка тестовых данных
tests = pd.read_csv("temp/nlp/Test.csv")

```

```
#функция для разметки по выявленным темам
def target(x):
    if tests["Computer Science"][x] == 1:
        return 1
    elif tests["Mathematics"][x] == 1:
        return 2
    elif tests["Physics"][x] == 1:
        return 3
    elif tests["Statistics"][x] == 1:
        return 0

#разметка
tests["target"] = [target(x) for x in range(tests.shape[0])]

#функция предсказания
def topic_prediction(my_document):
    string_input = [my_document]
    X = vect.transform(string_input)
    corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
    output = list(ldamodel[corpus])[0]
    topics = sorted(output,key=lambda x:x[1],reverse=True)
    return topics[0][0]

#предсказание
tests["prediction"] = [topic_prediction(test_text) for test_text in tests.ABSTRACT]

#сравнение двух колонок - оценка точности
print(confusion_matrix(tests['target'], tests['prediction']))
print(classification_report(tests['target'], tests['prediction']))
```

Приложение Б
Построение LSA модели

```
#используемые библиотеки
import pandas as pd
import gensim
from gensim.models import LsiModel
from gensim.corpora import Dictionary
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.decomposition import TruncatedSVD
import umap.umap_ as umap
from sklearn.metrics import confusion_matrix, classification_report

#загрузка данных
papers = pd.read_csv("temp/nlp/Train.csv")

#подготовка данных и векторизация
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
        token_pattern='(?u)\\b\\w\\w+\\b')
X = vect.fit_transform(papers.ABSTRACT)

#мешок слов и список идентификаторов
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())

#построение модели
lsamodel = gensim.models.LsiModel(corpus, id2word=id_map, num_topics=4)

#вывод слов по темам
print(lsamodel.print_topics(num_topics=4, num_words=10))

#функция для разметки по выявленным темам
def target(x, table):
    if table["Computer Science"][x] == 1:
        return 0
    if table["Mathematics"][x] == 1:
        return 2
    elif table["Physics"][x] == 1:
        return 3
    elif table["Statistics"][x] == 1:
        return 1

#разметка
papers["target"] = [target(x, papers) for x in range(papers.shape[0])]

#вывод облака слов
```

```
for t in range(lsamodel.num_topics):
    plt.figure()
    plt.imshow(WordCloud(background_color="white").fit_words(dict(lsamodel.show_topic(t, 100))))
    plt.axis("off")
    plt.title("Topic #" + str(t))
    plt.show()

#модель SVD для перехода в семантическое пространство
svd_model = TruncatedSVD(n_components=4, algorithm='randomized', n_iter=100, random_state=122)
svd_model.fit(X)
X_topics = svd_model.fit_transform(X)

#визуализация семантического пространства
embedding = umap.UMAP(n_neighbors=150, min_dist=0.5, random_state=12).fit_transform(X)
plt.figure(figsize=(7,5))
scatter = plt.scatter(embedding[:, 0], embedding[:, 1],
    c = papers.target,
    s = 10, # size
    edgecolor='none')
plt.colorbar()
plt.show()

#загрузка тестовых данных
tests = pd.read_csv("temp/nlp/Test.csv")

#разметка
tests["target"] = [target(x, tests) for x in range(tests.shape[0])]

#функция предсказания
def topic_prediction(my_document):
    string_input = [my_document]
    X = vect.transform(string_input)
    corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
    output = list(lsamodel[corpus])[0]
    topics = sorted(output,key=lambda x:x[1],reverse=True)
    return topics[0][0]

#предсказание
tests["prediction"] = [topic_prediction(test_text) for test_text in tests.ABSTRACT]

#сравнение двух колонок - оценка точности
print(confusion_matrix(tests['target'], tests['prediction']))
print(classification_report(tests['target'], tests['prediction']))
```

Приложение В
Построение матриц NMF

```
#используемые библиотеки
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.metrics import confusion_matrix, classification_report

#загрузка данных
papers = pd.read_csv("temp/nlp/Train.csv")

#подготовка данных и векторизация
text_data = [x for x in papers.ABSTRACT]
vectorizer = CountVectorizer(max_features=1500, min_df=10, stop_words='english',
            token_pattern='(?u)\\b\\w\\w\\w+\\b')
X = vectorizer.fit_transform(text_data)
words = np.array(vectorizer.get_feature_names_out())

#построение модели NMF
nmf = NMF(n_components=4, solver="mu")
W = nmf.fit_transform(X)
H = nmf.components_

#вывод облака слов
for t, topic in enumerate(H):
    x = dict(zip(words, topic))
    #myDict = {key:val for key, val in x.items() if val != 0}
    a = dict(sorted(x.items(), key= lambda item: item[1], reverse=True))
    lst = []
    for key, value in a.items():
        lst.append([key, value])
    plt.figure()
    plt.imshow(WordCloud(background_color="white").fit_words(dict(lst[0:100])))
    plt.axis("off")
    plt.title("Topic #" + str(t))
    plt.show()

    #print("Topic {}: {}".format(i + 1, ",".join([str(x) for x in words[topic.argsort()[-10:]]]))

#загрузка тестовых данных
tests = pd.read_csv("temp/nlp/Test.csv")

```

```
#функция для разметки по выявленным темам
def target(x, table):
    if table["Computer Science"][x] == 1:
        return 2
    elif table["Mathematics"][x] == 1:
        return 1
    elif table["Physics"][x] == 1:
        return 0
    elif table["Statistics"][x] == 1:
        return 3

#разметка
tests["target"] = [target(x, tests) for x in range(tests.shape[0])]

#подготовка новых данных и векторизация
new_text = [x for x in tests.ABSTRACT]
vect_new = vectorizer.transform(new_text)
X_new = nmf.transform(vect_new)

#предсказание
predicted_topics = [np.argsort(each)[::-1][0] for each in X_new]
tests['prediction'] = predicted_topics

#сравнение двух колонок - оценка точности
print(confusion_matrix(tests['target'], tests['prediction']))
print(classification_report(tests['target'], tests['prediction']))
```

Приложение В
CD-диск с отчётом о выполненной работе

На приложенном диске содержится полный код исследования.
model.zip — исходный код исследования.