



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
**МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ (МАДИ)**
Кафедра «Автоматизированные системы управления»

Н.Г. КУФТИНОВА, Ц.Б. ПРОНИН, А.В. ОСТРОУХ

БОЛЬШИЕ ДАННЫЕ

УЧЕБНОЕ ПОСОБИЕ

(лабораторный практикум)

по направлению подготовки, входящих в укрупненную группу

09.00.00 – «Информатика и вычислительная техника»



КРАСНОЯРСК – 2026

УДК 004.382.7:681.3

ББК 32.973.26

К95

- Куфтинова, Наталья Григорьевна.**
К95 Большие данные: учебное пособие (лабораторный практикум) по направлению подготовки, входящих в укрупненную группу 09.00.00 – «Информатика и вычислительная техника» / Н.Г. Куфтинова, Ц.Б. Пронин, А.В. Остроух; МАДИ. – Красноярск: Научно-инновационный центр, 2026. – 88 с.

<https://doi.org/10.12731/978-5-907608-60-3>

ISBN 978-5-907608-60-3

В настоящем учебном пособии изложены необходимые теоретические сведения и требования по выполнению лабораторных работ по дисциплине «Большие данные». Особое внимание уделено практическому применению современных технологий и инструментов для обработки и анализа больших объемов информации. Целью данного пособия является формирование у студентов компетенций, необходимых для работы с большими данными, включая сбор, хранение, обработку, анализ и визуализацию данных. Представленные лабораторные работы позволят закрепить теоретические знания и приобрести практические навыки, необходимые для решения реальных задач в области больших данных.

Данное пособие предназначено для студентов по направлению подготовки, входящих в укрупненную группу 09.00.00 – «Информатика и вычислительная техника» МАДИ, а также может быть предназначено для студентов всех форм обучения.

ISBN 978-5-907608-60-3

© [Авторы](#), 2026

© [МАДИ](#), 2026

ОГЛАВЛЕНИЕ

Введение	7
Экосистема больших данных	10
Лабораторная работа № 1. Кластеризация методом k-средних.....	16
1.1. Теоретическая часть	16
1.2. Задание лабораторной работы	20
1.3. Порядок выполнения лабораторной работы	20
1.4. Результат работы программы	23
1.5. Выводы.....	25
1.6. Вопросы для проверки	26
Лабораторная работа № 2. Классификация методом случайных лесов	28
2.1. Теоретическая часть	28
2.2. Задание лабораторной работы	35
2.3. Порядок выполнения лабораторной работы	35
2.4. Результат работы программы	40
2.5. Выводы.....	40
2.6. Вопросы для проверки	41
Лабораторная работа № 3. Регрессия методом опорных векторов	43
3.1. Теоретическая часть	43
3.2. Задание лабораторной работы	51
3.3. Порядок выполнения лабораторной работы	52
3.4. Результат работы программы	55
3.5. Выводы.....	55
3.6. Вопросы для проверки	55
Лабораторная работа № 4. Векторные базы данных и их программная интеграция	57
4.1. Теоретическая часть	57
4.2. Задание лабораторной работы	60

4.3. Порядок выполнения лабораторной работы	61
4.4. Результат работы программы	67
4.5. Выводы.....	69
4.6. Вопросы для проверки	69
Лабораторная работа № 5. Симбиоз машинного обучения и математических методов обработки больших данных	71
5.1. Теоретическая часть	71
5.2. Задание лабораторной работы	74
5.3. Порядок выполнения лабораторной работы	75
5.4. Результат работы программы в консоли.....	80
5.5. Выводы.....	82
5.6. Вопросы для проверки	83
Список информационных источников.....	85

ВВЕДЕНИЕ

Настоящее учебное пособие разработано для обучающихся по направлениям укрупнённой группы **09.04.00 «Информатика и вычислительная техника» (магистратура)** и адаптировано к специфике их профессиональной деятельности в сфере транспорта, логистики и дорожного хозяйства. Оно направлено на формирование у магистров глубокого понимания роли дата-центричных технологий в разработке и внедрении интеллектуальных систем для транспортного комплекса.

Дисциплина «Большие данные» занимает ключевое место в образовательной программе, обеспечивая формирование у будущих инженеров, проектировщиков и управленцев транспортной отрасли компетенций в области цифровой обработки, анализа и интерпретации массивов разнородных данных. В условиях цифровой трансформации транспортно-дорожного комплекса (ТДК) владение технологиями больших данных становится неотъемлемым элементом профессиональной подготовки, позволяющим решать задачи повышения безопасности, эффективности и устойчивости транспортных систем.

Предметом изучения дисциплины являются:

- архитектура, методы и инструменты сбора, хранения, обработки и анализа больших данных;
- информационные процессы и потоки данных в цифровой экосистеме транспортной отрасли;
- технологии визуализации и интерактивной аналитики для поддержки принятия управленческих решений.

Целью освоения дисциплины является формирование у обучающихся системных знаний и практических навыков применения

технологий больших данных для решения актуальных задач транспортно-дорожного комплекса, таких как:

- интеллектуальное управление транспортными потоками и оптимизация логистики;
- мониторинг и прогнозирование состояния дорожной инфраструктуры;
- анализ данных телематики, систем ГЛОНАСС/GPS и IoT-устройств;
- моделирование пассажиропотоков и грузоперевозок;
- повышение безопасности дорожного движения на основе данных видеонаблюдения и датчиков.

Задачи дисциплины включают:

- Освоение принципов работы с распределёнными хранилищами данных, потоковой обработкой и облачными платформами.
- Развитие навыков применения современных инструментов (Numpy, Pandas для чтения CSV-файла и фильтрации данных; PySpark; Scikit-Learn, платформы визуализации) в контексте отраслевых задач.
- Формирование умения проектировать архитектуру данных для систем управления транспортом и дорожным хозяйством.
- Подготовку к использованию методов машинного обучения и прогнозной аналитики для оптимизации работы ТДК.

В результате освоения дисциплины студент должен:

знать основные концепции, методы и технологии экосистемы больших данных, а также их применимость в транспортной сфере;

уметь проектировать процессы обработки данных, создавать аналитические отчёты и дашборды, применять инструменты для решения практических задач ТДК;

владеть навыками работы с современными инструментами анализа больших данных, проведения сквозного анализа данных — от сбора до визуализации и интерпретации результатов.

Дисциплина напрямую способствует формированию **профессиональных компетенций**, востребованных в таких областях, как:

- «Умный транспорт» и интеллектуальные транспортные системы (ИТС);
- Цифровое моделирование и управление дорожным движением;
- Логистика и управление цепями поставок;
- Мониторинг и диагностика дорожных покрытий;
- Анализ аварийности и разработка предиктивных моделей безопасности.

Таким образом, дисциплина «Большие данные» обеспечивает необходимую **цифровую грамотность** и **аналитический инструментарий**, позволяя выпускникам МАДИ эффективно решать комплексные инженерно-управленческие задачи в условиях цифровизации транспортной отрасли.

ЭКОСИСТЕМА БОЛЬШИХ ДАННЫХ

Экосистема больших данных (Big Data Ecosystem) — это многослойная структура, которая включает в себя источники информации, технологии для её обработки, аналитические инструменты, механизмы визуализации и методы обеспечения безопасности. Цель — преобразовывать большие объёмы разрозненной информации в ценные инсайты.



Рис. 1. Интеллект-карта экосистемы Больших данных

Большие данные (Big Data) — это структурированные и неструктурированные данные огромных объемов и разнообразия, а также методы их обработки, которые позволяют распределенно анализировать информацию.

Само по себе наличие огромного количества данных мало что дает. Именно Искусственный Интеллект и, в частности, Машинное Обучение позволяют извлекать из этих данных ценную информацию, выявлять закономерности и делать прогнозы.

Искусственный Интеллект (ИИ) – это широкая концепция, направленная на создание машин, способных выполнять задачи, которые обычно требуют человеческого интеллекта.

Машинное Обучение (МО) является областью искусственного интеллекта, занимающейся разработкой алгоритмов, позволяющих компьютерным системам извлекать знания и совершенствовать свои характеристики на основе анализа данных, не требуя явного программирования для каждой конкретной задачи. Алгоритмы машинного обучения способны автоматически адаптироваться и повышать точность своих предсказаний или решений по мере поступления новых данных.

Среди ключевых задач машинного обучения, активно применяемых в работе с большими данными, выделяют: кластеризацию, классификацию и регрессию.

Большие данные – это не просто большой объем информации. Это сложные наборы данных, которые требуют специальных подходов к обработке и анализу. Характеристики больших данных, известные как "5V", определяют эти особенности и диктуют выбор технологий и методологий.

Объем (Volume)

Большие данные характеризуются огромным объемом, который выходит за рамки возможностей традиционных систем обработки данных. Речь идет о терабайтах, петабайтах и даже эксабайтах информации.

Примеры

Социальные сети генерируют терабайты данных ежедневно (посты, лайки, комментарии, изображения, видео).

Интернет вещей (IoT) – миллионы устройств, постоянно отправляющих данные (датчики, камеры, сенсоры).

Онлайн-торговля – данные о транзакциях, поведении пользователей, истории покупок.

Влияние на обработку: Требуется масштабируемых систем хранения и обработки данных, таких как Hadoop, Spark, облачные хранилища и базы данных.

Скорость (Velocity)

Скорость относится к темпу поступления данных. Большие данные часто генерируются в режиме реального времени или близком к нему. Важно не только собрать данные, но и обработать их достаточно быстро, чтобы извлечь ценную информацию.

Примеры

Финансовые рынки – данные о котировках акций, изменениях валютных курсов поступают непрерывно.

Системы обнаружения мошенничества – требуют анализа транзакций в режиме реального времени для предотвращения несанкционированных действий.

Потоковое видео – анализ данных о просмотре для персонализации рекомендаций и оптимизации контента.

Влияние на обработку: Требуется технологий потоковой обработки данных (Stream Processing) таких как Apache Kafka, Apache Flink, Apache Storm.

Разнообразие (Variety)

Большие данные поступают в самых разных форматах. Это могут быть структурированные данные (таблицы баз данных), неструктурированные данные (тексты, изображения, видео, аудио) и полуструктурированные данные (JSON, XML).

Примеры

Социальные сети – текстовые сообщения, изображения, видео, геолокационные данные.

Интернет вещей – данные от различных датчиков (температура, давление, влажность, скорость).

Данные о клиентах – демографические данные, история покупок, отзывы, активность в социальных сетях.

Влияние на обработку: Требуется инструментов, способных работать с различными форматами данных, а также интеграции данных из разных источников. NoSQL базы данных, схемы-на-чтении (schema-on-read) подходы.

Достоверность (Veracity)

Данные могут быть неточными, неполными, противоречивыми или содержать ошибки. Оценка и обеспечение достоверности данных – критически важная задача.

Примеры

Ошибки при вводе данных, дубликаты данных, неверно интерпретированные данные от датчиков, ложная информация в социальных сетях.

Влияние на обработку: Требуется методов очистки данных, обнаружения аномалий, проверки качества данных и управления метаданными.

Ценность (Value)

В конечном итоге, главная цель работы с большими данными – извлечение ценной информации и получение практической пользы. Определение ценности данных и разработка методов ее извлечения – ключевая задача.

Примеры

Прогнозирование спроса на продукцию, персонализация маркетинговых кампаний, выявление мошеннических операций, оптимизация производственных процессов.

Влияние на обработку: Требуется использования методов анализа данных, машинного обучения, статистического моделирования и визуализации данных.

Понимание этих характеристик ("5V") больших данных позволяет правильно сформулировать задачи анализа, выбрать подходящие технологии и инструменты, а также оценить потенциальную ценность извлекаемой информации. Обработка больших данных – это комплексный процесс, требующий междисциплинарного подхода и глубокого понимания предметной области (см. рис. 1).

Важность Python для науки о данных в первую очередь обусловлена большой и активной экосистемой библиотек: *NumPy* для обработки однородных данных на основе массивов, *Pandas* для обработки неоднородных и помеченных данных, *SciPy* для общих научных вычислительных задач, *Matplotlib* для визуализаций качества публикации, *IPython* для интерактивного выполнения и обмена кодом, *Scikit-Learn* для машинного обучения.

В настоящее время используется значительное количество платформ и систем Больших Данных. Системы обработки больших данных являются фреймворками, то есть каркасами, для состыковки и адаптации в рамках конкретного решения. Рынок больших данных переживает беспрецедентный рост и расширение размера рынка до 2030 года (рис. 2).

Основные прогнозы по динамике и объему рынка формируются на основе официальной Стратегии развития рынка больших данных до 2030 года, представленной Ассоциацией больших данных (АБД) на конференции AI Journey 2025.

В стратегии определены три ключевых сценария развития в зависимости от внешних условий и принимаемых государством и бизнесом мер.

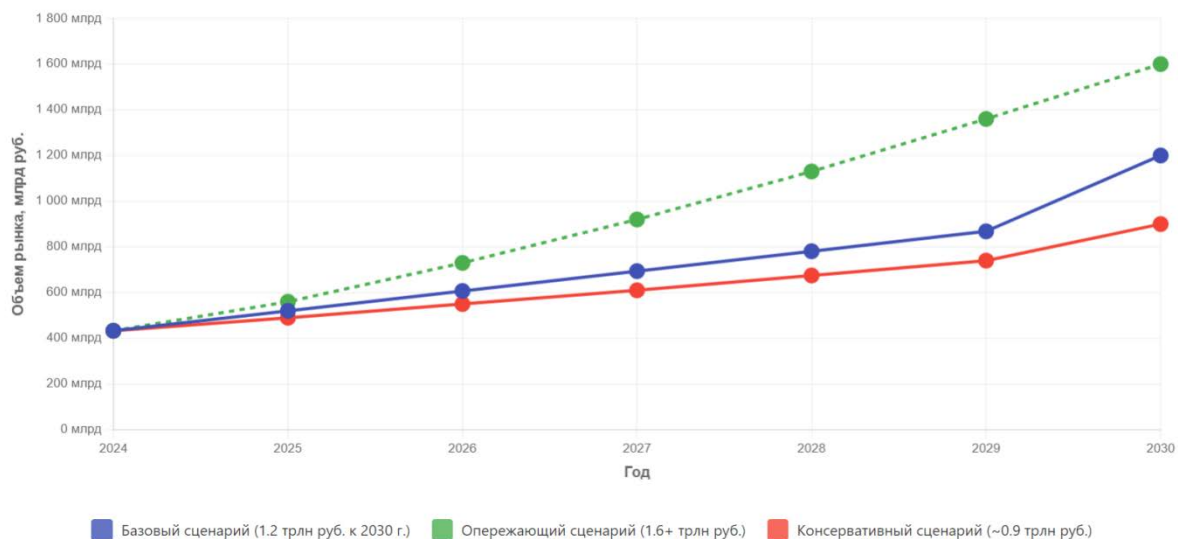


Рис. 2. Прогноз развития рынка больших данных в России до 2030 года

Прогноз объема рынка Big Data в России — не менее 1,2 трлн рублей к 2030 году (для сравнения в 2024 году — 433 млрд рублей). Это следует из исследования Ассоциации больших данных (АБД).

Прогноз мирового рынка Big Data — по оценкам Fortune Business Insights, мировой рынок Big Data & Analytics вырастет до 962 млрд долларов к 2032 году.

Тренд на слияние рынков ИИ и рынка больших данных — формируется единая экосистема (см. рис. 1), где большие данные являются «топливом» для ИИ.

В связи с развитием областей машинного обучения и методов обработки больших данных активно развивается новый тип баз данных - векторные базы данных.

Векторные базы данных – это современный тип баз данных, который произвел революцию в области повышения точности поиска, обработки и анализа данных, особенно в контексте искусственного интеллекта и машинного обучения.

ЛАБОРАТОРНАЯ РАБОТА № 1. КЛАСТЕРИЗАЦИЯ МЕТОДОМ К-СРЕДНИХ

Цель – освоение процесса кластеризации методом k-средних для выявления закономерностей (кластеров) в произвольном наборе данных.

1.1. Теоретическая часть

Задача кластеризации - разделить по признакам набор данных на группы (кластеры) таким образом, чтобы объекты внутри каждого кластера были более похожи друг на друга, чем объекты из разных кластеров.

Для кластеризации используется **обучение без учителя (Unsupervised Learning)**, то есть для кластеризации не используются размеченные данные (данные с известными категориями или значениями). **Задача алгоритмов кластеризации самим находить закономерности в наборах данных.**

Кластеризация часто используется для исследовательского анализа данных, чтобы выявить скрытые закономерности и сегменты.

К примерам задач, которые можно решить с помощью кластеризации можно отнести:

Сегментацию клиентов. Разделение клиентов на группы на основе их покупательского поведения.

Выявление аномалий. Например, обнаружение мошеннических транзакций, которые выделяются из общего паттерна.

Анализ социальных сетей. Выявление сообществ пользователей со схожими интересами.

К основным алгоритмам кластеризации относятся: k-means (метод k-средних), DBSCAN (пространственная кластеризация для приложений с шумами), иерархическая кластеризация.

Алгоритм k -средних ищет predetermined количество кластеров в немаркированном многомерном наборе данных. Он достигает этого, с помощью концепции оптимальной кластеризации, в которой:

«Центр кластера» — это среднее арифметическое всех точек, входящих в кластер.

Каждая точка находится ближе к центру своего кластера, чем к центрам других кластеров.

Основная идея метода — итеративное повторение двух шагов:

1. Распределение объектов выборки по кластерам;
2. Пересчет центров кластеров.

В начале работы алгоритма выбираются K случайных центров в пространстве признаков. Каждый объект выборки относят к тому кластеру, к центру которого объект оказался ближе. Далее центры кластеров пересчитывают как среднее арифметическое векторов признаков всех вошедших в этот кластер объектов (то есть центр масс кластера). Как только мы обновили центры кластеров, объекты заново перераспределяются по ним, а затем можно снова уточнить положение центров. Процесс продолжается до тех пор, пока центры кластеров не перестанут меняться.

Оба шага алгоритма работают на уменьшение среднего квадрата евклидова расстояния от объектов до центров их кластеров:

Рассмотрим данные, мерой близости которых является евклидово расстояние. Для нашей целевой функции, которая измеряет качество кластеризации, мы используем сумму квадратов отклонений (Sum of Squared Errors, SSE), которая также известна как разброс.

Другими словами, мы вычисляем отклонение каждой точки данных, т. е. ее евклидово расстояние до ближайшего центроида, а затем вычисляем

общую сумму квадратов отклонений. Учитывая два разных набора кластеров, которые получены двумя разными проходами алгоритма К-средних, мы предпочитаем тот, у которого наименьшее квадратичное отклонение, поскольку это означает, что прототипы (центроиды) этой кластеризации являются лучшим представлением точек в своем кластере.

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i - c_j\|^2$$

Где x_i – элемент набора данных, c_j – центроид кластера j ,

$\|x_i - c_j\|$ – функция расстояния, J – целевая функция.

Чтобы проиллюстрировать, что алгоритм К-средних не ограничивается данными в евклидовом пространстве, мы рассмотрим данные о документах и меру сходства косинуса.

Косинус двух ненулевых векторов **A** и **B** можно вывести, используя формулу скалярного произведения векторов (для евклидова пространства):

$$\bar{A} \cdot \bar{B} = \|\bar{A}\| \|\bar{B}\| \cdot \cos(\theta)$$

Для двух n -мерных векторов с атрибутами (признаками) **A** и **B** косинусное сходство $\cos(\theta)$, представлено в виде частного скалярного произведения этих векторов и произведения их длин, следующим образом:

$$\text{similarity} = \cos(\theta) = \frac{\bar{A} \cdot \bar{B}}{\|\bar{A}\| \|\bar{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

где A_i и B_i – i -тые компоненты векторов **A** и **B** соответственно.

Результирующее сходство находится в рамках значений от -1 (абсолютно не схожи) до $+1$ (абсолютно схожи), при этом 0 означает ортогональность или декорреляцию данных. Все промежуточные результаты обозначают сходство или различие.

Для сравнения текстов вектора атрибутов **A** и **B** обычно представляют собой частотные вектора этих текстов. Косинусное подобие может быть рассмотрено как метод нормализации длины текста при сравнении.

В случае извлечения информации, например, с помощью метода TF-IDF, который используется для векторизации текстов во второй лабораторной работе, косинусное подобие будет ограничено диапазоном от 0 до 1, так как частота встречаемости слов не может быть отрицательной.

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — 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.

TF (term frequency — частота слова) — отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова t_i в пределах отдельного документа.

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

где n_t число вхождений слова t в документ, а в знаменателе — общее число слов в данном документе.

IDF (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции. Основоположником данной концепции является Карен Спарк Джонс. Учёт IDF уменьшает вес широкоупотребительных слов. Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

$$idf(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}$$

где $|D|$ - число документов в коллекции. $|\{d_i \in D \mid t \in d_i\}|$ - число документов из коллекции D , в которых встречается t (когда $n_t \neq 0$). Выбор основания логарифма в формуле не имеет значения, поскольку изменение

основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$TFIDF(t,d,D)=tf(t,d)\times idf(t,D)$$

Большой вес в TF-IDF получают слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

Существуют различные формулы, основанные на методе TF-IDF. Они отличаются коэффициентами, нормировками, использованием логарифмированных шкал.

Одной из наиболее популярных формул является формула **BM25**.

1.2. Задание лабораторной работы

На основе выбранного набора данных из открытых источников (<https://huggingface.co/datasets> / <https://www.kaggle.com/datasets>) выполнить кластеризацию методом к-средних.

1.3. Порядок выполнения лабораторной работы

Основой для выполнения лабораторных в данном пособии является создание и использование виртуальных сред python на основе пакета Miniconda (Anaconda). Виртуальные среды python нужны для работы с проектами, использующими несовместимые зависимости на одном устройстве.

При необходимости, создаваемая виртуальная среда может иметь необходимую версию Python и библиотек (зависимостей) под конкретный проект, не затрагивая другие среды и системную версию Python.

Альтернативой Anaconda, с менее интуитивным синтаксисом, является встроенный в Python менеджер виртуальных сред `venv`.

Создание виртуальной среды в Miniconda (Anaconda).

- 1) Найти в системном поиске Anaconda Prompt.
- 2) Запустить ярлык Anaconda Prompt (`miniconda3`).
- 3) Создать среду командой среду с названием `skl`:

```
conda create -n skl python=3.11
```

- 4) Активировать среду командой:

```
conda activate skl
```

5) Установить библиотеку `pandas` и `scikit-learn`, через `pip`, командой:

```
pip install pandas scikit-learn
```

- 6) Посмотреть список созданных сред можно командой:

```
conda env list
```

Рассмотрим кластеризацию методом K-средних датасета Iris с помощью пакета `scikit-learn`.

Датасет Iris (Ирисы Фишера) - это один из самых известных и часто используемых наборов данных в области машинного обучения. Он идеально подходит для начинающих, поскольку является простым, понятным и позволяет практиковаться с различными алгоритмами классификации.

Происхождение и Цель

Источник. Первоначально данные были собраны Рональдом А. Фишером в 1936 году.

Цель. Изначально датасет использовался для демонстрации статистических методов, включая линейную дискриминантную функцию. В машинном обучении он широко используется как тестовый пример для алгоритмов классификации.

Задача. Определить вид цветка Ириса (*Iris*) на основе измерений его частей.

Структура Данных

Датасет состоит из таблицы, где каждая строка представляет отдельный цветок, а столбцы содержат информацию о нем. В целом, датасет содержит 150 образцов, разделенных на три класса:

***Iris setosa*:** 50 образцов

***Iris versicolor*:** 50 образцов

***Iris virginica*:** 50 образцов

Признаки (Features)

Каждый образец описывается четырьмя признаками (атрибутами):

sepal length (cm): Длина чашелистика в сантиметрах.

sepal width (cm): Ширина чашелистика в сантиметрах.

petal length (cm): Длина лепестка в сантиметрах.

petal width (cm): Ширина лепестка в сантиметрах.

Задача программы кластеризовать ирисы по признакам, оценить качество метрикой коэффициента силуэта и вывести график кластеризации по одному из признаков, например, длина и ширина чашелистика.

Код программы

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Загрузка датасета iris
iris = load_iris()
X = iris.data
y = iris.target

# Масштабирование признаков для лучшей кластеризации
scaler = StandardScaler()
```

```

X_scaled = scaler.fit_transform(X)

# Создаем модель k-средних с k=3 кластерами
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

# Получаем предсказанные метки кластеров
labels = kmeans.labels_

# Выводим информацию о центрах кластеров
print("Центры кластеров:")
for i, center in enumerate(kmeans.cluster_centers_):
    print(f"\nКластер {i}:")
    print(f"Длина чашелистика: {center[0]:.2f}")
    print(f"Ширина чашелистика: {center[1]:.2f}")
    print(f"Длина лепестка: {center[2]:.2f}")
    print(f"Ширина лепестка: {center[3]:.2f}")

# Оценка качества кластеризации
print("\nОценка качества кластеризации:")
print(f"Коэффициент силуэта (Silhouette index) – внутренний показатель
оценки кластеризации: {kmeans.score(X_scaled):.3f}")

# Визуализация результатов
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels,
map='viridis')
plt.colorbar(scatter)
plt.title('Кластеризация датасета Iris методом k-средних')
plt.xlabel('Масштабированная длина чашелистика')
plt.ylabel('Масштабированная ширина чашелистика')
plt.show()
plt.close()

```

1.4. Результат работы программы

Центры кластеров

Кластер 0:

Длина чашелистика: 0.57

Ширина чашелистика: -0.37

Длина лепестка: 0.69

Ширина лепестка: 0.66

Кластер 1:

Длина чашелистика: -0.82

Ширина чашелистика: 1.32

Длина лепестка: -1.29

Ширина лепестка: -1.22

Кластер 2:

Длина чашелистика: -1.33

Ширина чашелистика: -0.37

Длина лепестка: -1.14

Ширина лепестка: -1.11

Оценка качества кластеризации

Коэффициент силуэта (Silhouette index, рис. 3) — внутренний показатель оценки кластеризации: -191.025.

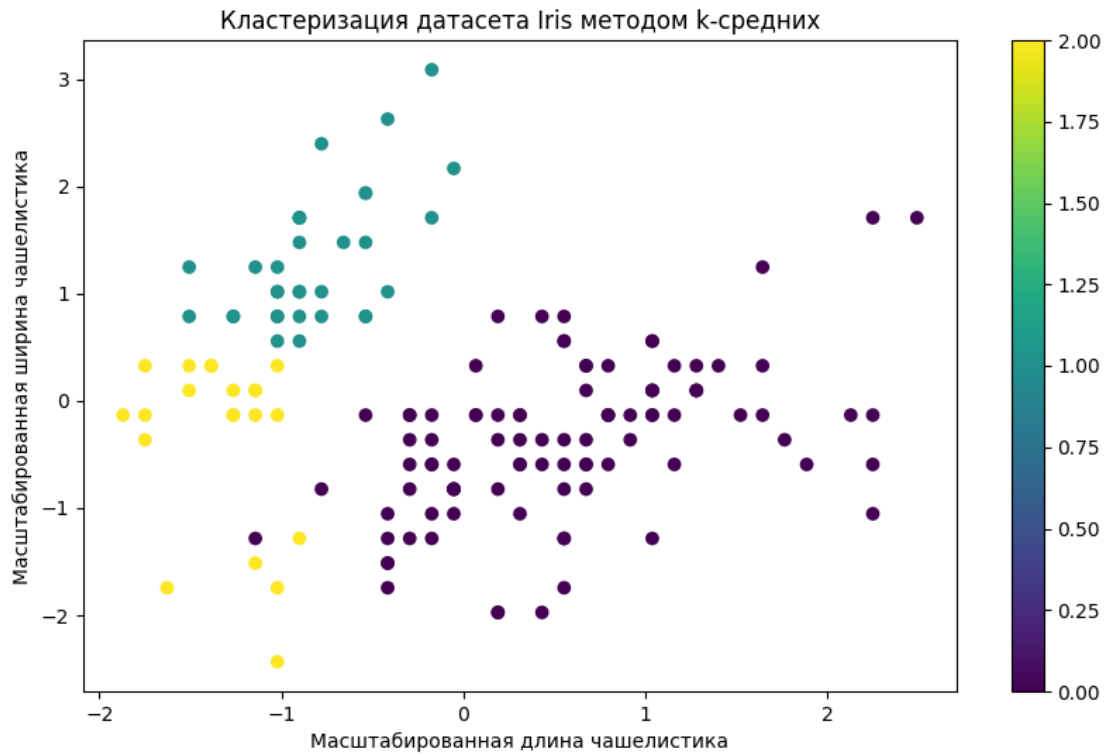


Рис. 3. Результат кластеризации цветов ириса по длине и ширине чашелистика

1.5. Выводы

В ходе выполнения лабораторной работы по кластеризации методом k-средних был получен практический опыт применения данного алгоритма для решения задач группировки данных. Реализация на Python с использованием библиотеки Scikit-learn позволила детально изучить основные этапы алгоритма, включая инициализацию центроидов, присвоение точек ближайшему центроиду и пересчет центроидов до сходимости. Эксперименты с различными значениями параметра k и оценкой качества кластеризации с помощью коэффициента силуэта подтвердили важность правильного выбора количества кластеров для получения значимых результатов. Важным этапом предварительной обработки данных оказалось масштабирование признаков, что позволило

избежать доминирования признаков с большими значениями и повысить эффективность алгоритма.

Основным выводом является то, что кластеризация методом k -средних является эффективным, но чувствительным к параметрам и предварительной обработке данных инструментом. Выбор оптимального количества кластеров (k) и функции расстояния требует анализа данных и оценки результатов с использованием различных метрик. Алгоритм k -средних может быть успешно применен для решения широкого круга задач, таких как сегментация клиентов, выявление аномалий и организация данных. Дальнейшее изучение альтернативных алгоритмов кластеризации и их сравнение с k -средним позволит расширить возможности анализа данных и выбора наиболее подходящего метода для конкретной задачи.

1.6. Вопросы для проверки

1. Что такое кластеризация, и почему она полезна?
2. Что такое метод k -средних (k -means)?
3. Опишите шаги алгоритма k -средних.
4. Что такое центроид кластера? Как он вычисляется?
5. Что такое функция стоимости (loss function) для k -средних?
6. Какова основная цель задачи кластеризации в машинном обучении?
7. Что такое обучение без учителя (Unsupervised Learning), и как оно используется в кластеризации?
8. Перечислите примеры задач, которые можно решить с помощью кластеризации.
9. Какой алгоритм кластеризации был рассмотрен в лабораторной работе, и как он работает?
10. Что такое "центр кластера" в контексте алгоритма k -средних?
11. Как определяется оптимальная кластеризация в алгоритме k -средних?

12. Какой набор данных был использован для демонстрации кластеризации методом k-средних в лабораторной работе?
13. Какова задача программы, представленной в лабораторной работе?
14. Что такое коэффициент силуэта (Silhouette index), и как он используется для оценки качества кластеризации?
15. Как масштабируются признаки перед кластеризацией, и почему это необходимо?
16. Как определяется количество кластеров в алгоритме k-средних?
17. Что такое виртуальная среда Python, и зачем она нужна для выполнения лабораторных работ?
18. Как установить библиотеки pandas и scikit-learn через pip?
19. Как вывести график кластеризации по одному из признаков с помощью matplotlib?
20. Что такое StandardScaler, и как он используется для масштабирования признаков?
21. Как получить предсказанные метки кластеров после выполнения алгоритма k-средних?

ЛАБОРАТОРНАЯ РАБОТА № 2.

КЛАССИФИКАЦИЯ МЕТОДОМ СЛУЧАЙНЫХ ЛЕСОВ

Цель работы - освоение процесса классификации методом случайных лесов на произвольном наборе данных.

2.1. Теоретическая часть

Задачей классификации является предсказание дискретных меток (классов) для новых данных на основе уже известных примеров с метками.

Наборы данных для классификации обычно содержат пары из признаков объекта и класса, к которому он относится. При этом количество признаков, по которым определяется класс может быть достаточно много.

Особенности

Для кластеризации используется **обучение с учителем (Supervised Learning)** - для обучения модели используются данные с известными метками классов.

Алгоритмы. Популярные алгоритмы включают логистическую регрессию, деревья решений, случайный лес, градиентный бустинг и нейронные сети.

Основные метрики классификации

Accuracy (Точность) - доля правильных предсказаний среди всех предсказаний модели:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

TP (True Positive) - верно предсказанные положительные случаи.

TN (True Negative) - верно предсказанные отрицательные случаи.

FP (False Positive) - отрицательные случаи, ошибочно предсказанные как положительные.

FN (False Negative) - положительные случаи, ошибочно предсказанные как отрицательные.

Precision (Точность в узком смысле) - показывает, какая доля объектов, выделенных как положительные, действительно являются положительными:

$$Precision = TP / (TP + FP)$$

Recall (Полнота) - показывает, какая доля положительных объектов была правильно идентифицирована моделью:

$$Recall = TP / (TP + FN)$$

F1-score — гармоническое среднее между Precision и Recall, позволяющее сбалансировать их значения:

$$F1 = 2 * (Precision * Recall) / (Precision + Recall)$$

Деревья решений

Дерево решений (Decision Tree, рис. 4) - это простой и интуитивный алгоритм машинного обучения, используемый для классификации и регрессии. Он представляет собой древовидную структуру, где каждый узел соответствует признаку или классу, а ребра представляют собой решения на основе значений признаков.

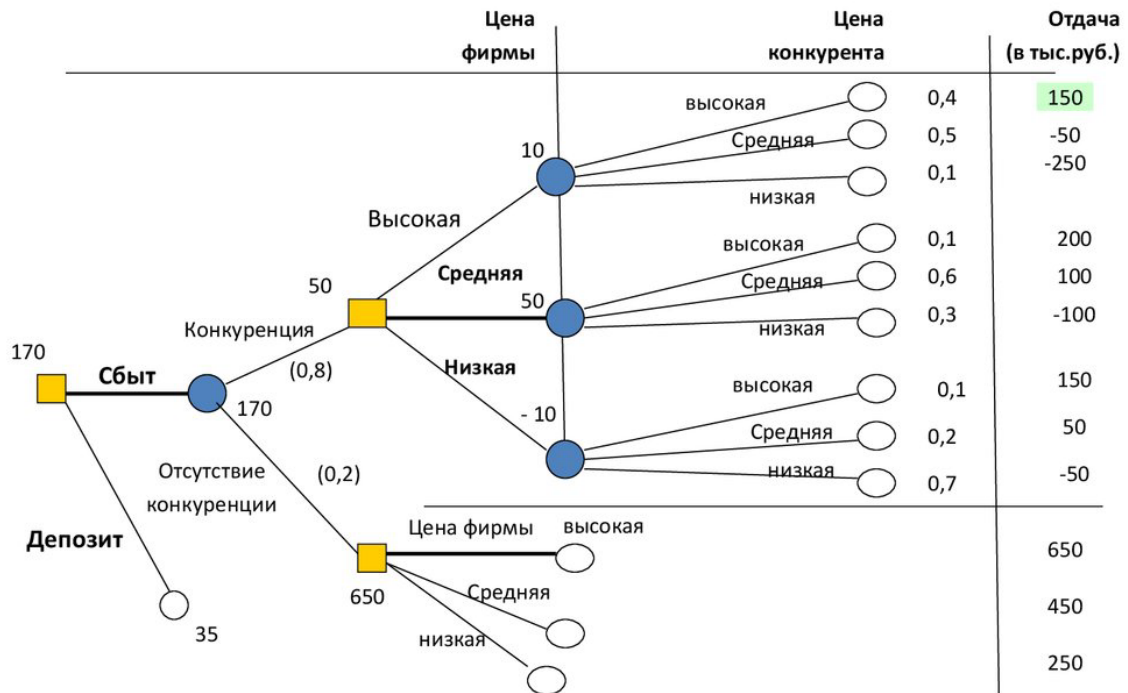


Рис. 4. Пример дерева решений

Структура дерева решений

1. **Корень:** начальный узел дерева, который содержит всю тренировочную выборку.
2. **Внутренние узлы:** каждый внутренний узел представляет собой признак и значение разделения, которое максимизирует чистоту классов в левой и правой частях узла.
3. **Листья:** конечные узлы дерева, которые содержат предсказанный класс или значение.
4. **Ребра:** ребра соединяют внутренние узлы с листьями и представляют собой решения на основе значений признаков.

Как работает дерево решений

Обучение: тренировочная выборка подается в корень дерева, и каждый образец проходит через дерево, пока не достигнет листа.

Разделение: в каждом внутреннем узле выбирается признак и значение разделения, которое максимизирует чистоту классов в левой и правой частях узла.

Прогноз: когда образец достигает листа, он предсказывает класс или значение на основе значения признака.

Типы деревьев решений

1. **Классификационные деревья:** используются для задач классификации, где цель состоит в том, чтобы предсказать класс объекта.

2. **Регрессионные деревья:** используются для задач регрессии, где цель состоит в том, чтобы предсказать числовое значение.

Параметры

1. **Максимальная глубина:** максимальная глубина дерева.

2. **Минимальное количество образцов для разделения:** минимальное количество образцов, необходимое для разделения узла.

3. **Минимальное количество образцов в листе:** минимальное количество образцов, необходимое для нахождения в листе.

Классификация методом случайных лесов

RandomForestClassifier - это один из наиболее популярных алгоритмов машинного обучения, используемых для задач классификации. Он реализован в библиотеке **scikit-learn** и основан на комбинации нескольких деревьев решений.

Идея **RandomForestClassifier** заключается в том, чтобы создать множество простых моделей (деревьев решений) и объединить их прогнозы для получения более точного результата. Каждое дерево решений обучается на подмножестве данных и предсказывает класс объекта на основе его признаков. Предполагается, что ансамбль деревьев решений позволяет получить более точный прогноз, чем одно дерево.

Принцип работы

1. **Бутстрэп-семплирование:** из исходных данных берется случайное подмножество (бутстрэп-выборка) размером **n_samples**. Это техника, используемая в ансамблевых методах, таких как Random Forest, для создания разнообразных моделей и повышения общей точности предсказаний.

2. **Создание дерева решений:** на основе бутстрэп-выборки создается дерево решений. Для каждого узла дерева выбирается один признак и одно значение разделения, которое максимизирует чистоту классов в левой и правой частях узла.

3. **Построение случайного подмножества признаков:** для каждого узла дерева выбирается случайное подмножество признаков (размером **max_features**) из всех доступных признаков. Это делается для того, чтобы предотвратить переобучение и повысить разнообразие деревьев.

4. **Обучение дерева решений:** дерево решений обучается на бутстрэп-выборке с использованием выбранных признаков и значения разделения.

5. **Предсказание класса:** каждое дерево решений предсказывает класс объекта на основе его признаков.

6. **Голосование:** все деревья решений голосуют за один из классов, и класс с наибольшим количеством голосов выбирается в качестве окончательного прогноза.

Параметры:

n_estimators: количество деревьев решений в ансамбле.

max_depth: максимальная глубина каждого дерева решений.

min_samples_split: минимальное количество образцов, необходимое для разделения узла.

min_samples_leaf: минимальное количество образцов, необходимое для нахождения в листе.

max_features: размер подмножества признаков, выбираемого случайным образом для каждого узла.

Подготовка обучающих данных

В данной лабораторной работе **CountVectorizer** и **TfidfTransformer** выполняют ключевые роли в процессе преобразования текстовых данных в числовой формат, который понятен алгоритму машинного обучения **RandomForestClassifier**.

CountVectorizer

CountVectorizer - преобразует коллекцию текстовых документов в матрицу числовых значений, представляющих частоту каждого слова в каждом документе. По сути, он создает "словарь" всех уникальных слов в корпусе текстов и подсчитывает, сколько раз каждое слово встречается в каждом документе.

Параметры в коде:

max_features=1500: Ограничивает словарь 1500 наиболее часто встречающимися словами. Это помогает уменьшить размерность данных и избежать переобучения.

min_df=5: Игнорирует слова, которые встречаются менее 5 раз в корпусе текстов. Это помогает удалить редкие слова, которые могут быть шумом.

max_df=0.7: Игнорирует слова, которые встречаются в более чем 70% документов. Это помогает удалить слишком распространенные слова, которые не несут большой смысловой нагрузки (например, предлоги, союзы).

stop_words=stopwords.words('english'): Удаляет из текста так называемые "стоп-слова" (например, "the", "a", "is", "are"), которые обычно не несут смысловой нагрузки и могут только ухудшить качество модели.

Результат: Создает разреженную матрицу, где строки соответствуют документам, столбцы соответствуют словам в словаре, а значения соответствуют частоте каждого слова в каждом документе.

TfidfTransformer

TfidfTransformer - преобразует матрицу частот слов, созданную **CountVectorizer**, в матрицу TF-IDF (Term Frequency-Inverse Document Frequency) весов (подробное описание в разделе 1.1). TF-IDF – это мера, которая учитывает как частоту слова в документе (TF), так и редкость слова во всем корпусе текстов (IDF).

TF (Term Frequency): Показывает, насколько часто слово встречается в документе. В данной реализации используется неявное вычисление TF, так как **CountVectorizer** уже предоставляет частоты.

IDF (Inverse Document Frequency): Показывает, насколько редко слово встречается во всем корпусе текстов. Слова, которые встречаются во многих документах, имеют низкий IDF, а слова, которые встречаются в небольшом количестве документов, имеют высокий IDF.

Результат: Создает матрицу, где значения представляют собой TF-IDF веса для каждого слова в каждом документе. Эти веса отражают важность каждого слова для данного документа в контексте всего корпуса текстов.

Вместе **CountVectorizer** и **TfidfTransformer** работают последовательно, чтобы преобразовать текстовые данные в числовой формат, который учитывает, как частоту слов в документах, так и их редкость во всем корпусе текстов. Это позволяет модели **RandomForestClassifier** лучше понимать смысл текстов и более точно классифицировать их. TF-IDF веса помогают выделить наиболее важные слова для каждого документа, игнорируя часто встречающиеся, но не несущие большой смысловой нагрузки слова.

2.2. Задание лабораторной работы

На основе выбранного набора данных из открытых источников (<https://huggingface.co/datasets> / <https://www.kaggle.com/datasets>) выполнить классификацию методом случайных лесов.

2.3. Порядок выполнения лабораторной работы

Задачей кода, приведённого ниже является классификация методом случайных лесов текстового датасета «dair-ai/Emotion». Этот набор данных представляет собой пары текстов и их эмоционального окраса. Следовательно, обучением модели классификатора решается задача определения тональности текста.

Так как модели машинного обучения работают с числовыми данными, для работы с текстами в них обычно применяются различные методы перевода текстовых последовательностей в числовые (векторизация, токенизация, создание эмбеддингов). В нашем примере, после приведения текстовых последовательностей к равномерному виду (очистки данных), применяется метод векторизации CountVectorizer («мешок слов»). Данный метод превращает текст в вектора, следующим образом: 1500 самых часто встречаемых слов заменяются на их индексы по частоте, при этом остальные слова заменяются на 0. Затем, для улучшения качества выборки применяется TfidfTransformer для преобразования результата CountVectorizer в TFIDF веса. Несмотря на относительную простоту и очевидные потери информации в результате подобного преобразования, данный метод является подходящим для решения задачи классификации на основе данного датасета.

Код программы

```
import pandas as pd
# Загрузка датасета в виде Pandas DataFrame
```

```
df = pd.read_parquet("hf://datasets/dair-ai/emotion/unsplit/train-00000-
of-00001.parquet")
print(df.head())
```

	text	label
0	i feel awful about it too because it s my job ...	0
1	im alone i feel awful	0
2	ive probably mentioned this before but i reall...	1
3	i was feeling a little low few days back	0
4	i beleive that i am much more sensitive to oth...	2

Рис. 5. Пример первых 5 строк из набора данных

```
# Удаление дубликатов строк
df = df.drop_duplicates().reset_index(drop=True)

# Выбор 10000 строк из датасета (для быстроедействия)
df = df[:10000]

import numpy as np
import re
import nltk

nltk.download('stopwords')
nltk.download('wordnet')

import pickle
from nltk.corpus import stopwords

# Задача классификатора - сопоставление текстов из столбца X, ярлыкам из
столбца y
X, y = df['text'], df['label']

# Очистка данных перед обучением
documents = []

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()
for sen in range(0, len(X)):
```

```

# Удаление спецсимволов
document = re.sub(r'\W', ' ', str(X[sen]))

# Удаление одиночных букв
document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

# Удаление одиночных букв с начала фраз
document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

# Замена повторяющихся пробелов одиночными
document = re.sub(r'\s+', ' ', document, flags=re.I)

# Удаление префиксных 'b'
document = re.sub(r'^b\s+', '', document)

# Конверсия текста в строчные буквы
document = document.lower()

# Лемматизация
document = document.split()
document = [stemmer.lemmatize(word) for word in document]
document = ' '.join(document)
documents.append(document)

# Векторизация очищенных данных
# 1500 Самых часто встречаемых слов заменяются на их индексы по частоте
# Остальные слова заменяются на 0
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7,
stop_words=stopwords.words('english'))
X = vectorizer.fit_transform(documents).toarray()

from sklearn.feature_extraction.text import TfidfTransformer
tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()

# Разбиение данных на обучающую и тестовую выборку
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

# Обучение модели классификатора на данных из обучающей выборки
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
classifier.fit(X_train, y_train)

```

```
# Прямой проход обученной модели на случайных данных
r1="""It was an terrible turn of events.
"""

r2="""The event organization was lovely
"""

r3="""This is cool
"""
docs_new = [r1, r2, r3]
X_new_counts = vectorizer.transform(docs_new)
X_new_tfidf = tfidfconverter.transform(X_new_counts)

predicted = classifier.predict(X_new_tfidf)
print(predicted)

# Результат тестов [0 2 1]
# Основные ярлыки выбранного датасета
#0 (sadness)
#1 (joy)
#2 (love)
#3 (anger)
#4 (fear)
#5 (surprise)

# Прямой проход обученной модели на данных из тестовой выборки
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Получение основных метрик для обученной модели классификатора
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test, y_pred))
```

```

[[531  23   6  16  18   4]
 [ 17 614  17   6   8   7]
 [   1  44 105   1   3   1]
 [ 13  13   1 215  15   5]
 [   8   9   2  14 193  14]
 [   0   4   0   1   9  62]]

```

	precision	recall	f1-score	support
0	0.93	0.89	0.91	598
1	0.87	0.92	0.89	669
2	0.80	0.68	0.73	155
3	0.85	0.82	0.83	262
4	0.78	0.80	0.79	240
5	0.67	0.82	0.73	76
accuracy			0.86	2000
macro avg	0.82	0.82	0.82	2000
weighted avg	0.86	0.86	0.86	2000

0.86

Рис. 6. Получение основных метрик для обученной модели классификатора

```

# Запись обученной модели в файл
with open('text_classifier', 'wb') as picklefile:
    pickle.dump(classifier,picklefile)

# Чтение обученной модели из файла
with open('text_classifier', 'rb') as training_model:
    model = pickle.load(training_model)

# Получение метрики для модели из файла
y_pred2 = model.predict(X_test)

# Метрика совпадает с версией до сохранения
print(confusion_matrix(y_test, y_pred2))
print(classification_report(y_test, y_pred2))
print(accuracy_score(y_test, y_pred2))

```

2.4. Результат работы программы

```

[[531  23   6  16  18   4]
 [ 17 614  17   6   8   7]
 [   1  44 105   1   3   1]
 [ 13  13   1 215  15   5]
 [   8   9   2  14 193  14]
 [   0   4   0   1   9  62]]
      precision    recall  f1-score   support

     0       0.93     0.89     0.91     598
     1       0.87     0.92     0.89     669
     2       0.80     0.68     0.73     155
     3       0.85     0.82     0.83     262
     4       0.78     0.80     0.79     240
     5       0.67     0.82     0.73     76

 accuracy                   0.86     2000
 macro avg                   0.82     2000
 weighted avg                 0.86     2000

0.86

```

Рис. 7. Метрики обученной модели классификации после загрузки её из файла

2.5. Выводы

В ходе выполнения лабораторной работы был успешно реализован процесс классификации текстовых данных методом случайных лесов. Использование библиотеки `scikit-learn` позволило эффективно обработать и подготовить данные, включая очистку текста, векторизацию с использованием "мешка слов" и TF-IDF преобразования. Обученная модель продемонстрировала способность корректно классифицировать эмоциональную окраску текстов, что подтверждается метриками, полученными на тестовой выборке: `accuracy`, `precision`, `recall` и `F1-score`. Возможность сохранения и загрузки обученной модели обеспечивает её

повторное использование без необходимости повторного обучения, что особенно важно для практических приложений.

Лабораторная работа позволила освоить ключевые этапы процесса классификации с использованием метода случайных лесов, включая предобработку данных, обучение модели, оценку её качества и сохранение для дальнейшего использования. Выбранный подход продемонстрировал свою эффективность для решения задачи классификации эмоциональной окраски текстов, а полученные результаты позволяют сделать вывод о практической применимости данного метода в задачах анализа тональности и обработки естественного языка. Несмотря на простоту используемой векторизации, модель показала приемлемую точность, что указывает на потенциал для дальнейшего улучшения результатов за счет использования более продвинутых методов представления текста, таких как word embeddings или transformer-based модели.

2.6. Вопросы для проверки

1. Что такое задача классификации и чем она отличается от задачи регрессии?
2. Какие основные метрики используются для оценки качества классификатора? Объясните их значение.
3. Что такое дерево решений и как оно работает?
4. В чем суть метода случайных лесов и чем он отличается от одиночного дерева решений?
5. Какие преимущества и недостатки имеет метод случайных лесов?
6. Что такое бутстрэп-семплирование и как оно используется в случайных лесах?
7. Зачем в случайных лесах выбирается случайное подмножество признаков для каждого узла дерева?

8. Какие параметры можно настроить в `RandomForestClassifier` и как они влияют на работу модели?
9. Какой набор данных использовался в лабораторной работе и какова его основная задача?
10. Какие этапы предобработки текста были выполнены в коде? Зачем они нужны?
11. Что такое векторизация и какой метод векторизации использовался в лабораторной работе?
12. Зачем используются `CountVectorizer` и `TfidfTransformer` в коде? В чем их разница?
13. Как были разделены данные на обучающую и тестовую выборки? Зачем это нужно?
14. Как обучается модель `RandomForestClassifier` в коде?
15. Как оценивается качество обученной модели?
16. Как были сделаны предсказания для новых текстовых данных?
17. Зачем сохраняется обученная модель в файл?
18. Как можно улучшить качество модели, изменив параметры `RandomForestClassifier`?
19. Какие альтернативные методы векторизации текста можно использовать вместо "мешка слов"? Какие у них преимущества и недостатки?
20. Для чего служит разделение датасета на обучающую и тестовую выборки в этом проекте?

ЛАБОРАТОРНАЯ РАБОТА № 3.

РЕГРЕССИЯ МЕТОДОМ ОПОРНЫХ ВЕКТОРОВ

Цель – освоение процесса регрессии методом опорных векторов для предсказания следующего элемента временной последовательности для произвольного набора данных.

3.1. Теоретическая часть

Регрессия – предсказание новых точек непрерывной (дискретной) числовой последовательности путём установления зависимости в наборе данных. Обычно для регрессии используются наборы данных содержащие некую временную зависимость, например, зависимость стоимости актива от даты/времени. В таком случае ставится цель предсказания последующих точек на временной шкале.

Постановка задачи регрессии - по обучающей выборке $\{(x_i, y_i)\}_{i=1}^N$, построить модель $f(x)$, где x_i – значение по оси X (оси времени), а y_i – соответствующее значение по оси Y (в примере цена акции), N – количество строк в наборе данных.

Величину $e_i = f(x_i) - y_i$ называют ошибкой на объекте или регрессионным остатком.

Одной из основных метрик качества регрессии является **средняя квадратичная ошибка (Mean Squared Error, MSE)** - это метрика оценки качества математической модели или алгоритма. Она рассчитывается как среднее значение квадрата разности между прогнозируемыми и фактическими значениями.

$$MSE(y^{true}, y^{pred}) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2$$

N – количество строк в наборе данных, y_i – реальное значение из тестового набора данных, $f(x_i)$ – результат предсказания обученной моделью для точки x_i из тестового набора данных.

Одним из методов регрессии является регрессия на основе метода опорных векторов.

Метод опорных векторов. Основные понятия

Гиперплоскость – это линия, которая используется для предсказания значений в последовательности как линия, наиболее точно описывающая точки данных в пределах определённой погрешности. В задачах классификации — это граница, используемая для разделения точек данных разных классов в пространстве признаков.

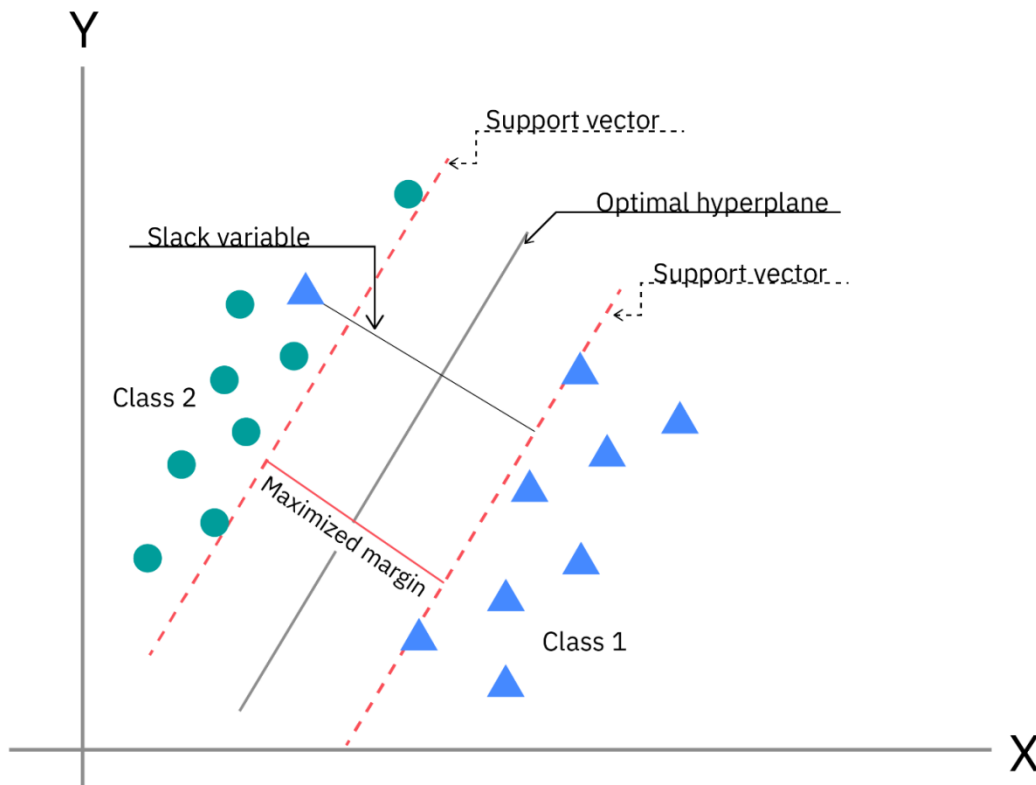


Рис. 7. Пример классификации методом опорных векторов

Любая гиперплоскость может быть описана как набор точек, удовлетворяющих условию:

$$w^T x - b = 0$$

Параметр b (bias) в SVM (методе опорных векторов) — это смещение, которое представляет расстояние гиперплоскости от начала координат вдоль нормального вектора.

Опорные Векторы - Ближайшие точки данных к гиперплоскости, которые играют решающую роль при выборе гиперплоскости и зазора.

Зазор - Расстояние между опорными векторами и гиперплоскостью. Основная цель SVM — максимизировать этот зазор для лучшей классификации.

Метод опорных векторов. Классификация

Метод опорных векторов (SVM) представляет собой мощный алгоритм машинного обучения, широко применяемый для задач линейной и нелинейной классификации, регрессии и обнаружения аномальных данных. Разработанный советско-американским математиком Владимиром Вапником и его коллегами, SVM основан на принципе нахождения оптимальной гиперплоскости, которая разделяет данные на различные классы с максимальным зазором.

Для классификации, набор данных выглядит следующим образом:

$$\{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\}$$

Где c_i принимает значение 1 или -1 в зависимости от того, какому классу принадлежит точка x_i . Каждое x_i — это p -мерный вещественный вектор (признаков p), обычно нормализованный значениями $[0,1]$ или $[-1,1]$. Если точки не будут нормализованы, то точка с большими отклонениями от средних значений координат точек слишком сильно повлияет на классификатор. Это можно рассматривать как обучающую выборку, в которой для каждого элемента уже задан класс, к которому он принадлежит. Мы хотим, чтобы алгоритм метода опорных векторов классифицировал их таким же образом. Для этого мы строим разделяющую гиперплоскость, которая имеет вид:

$$\bar{w} \cdot \bar{x} - b = 0$$

Вектор w – перпендикулярен к разделяющей гиперплоскости.

Параметр $\frac{b}{\|w\|}$ равен по модулю расстоянию от гиперплоскости до начала координат. Если параметр b равен нулю, гиперплоскость проходит через начало координат, что ограничивает решение.

Для оптимального разделения, выбираются опорные вектора и гиперплоскости, параллельные оптимальной и ближайшие к опорным векторам двух классов. Можно показать, что эти параллельные гиперплоскости могут быть описаны следующими уравнениями (с точностью до нормировки).

$$\bar{w} \cdot \bar{x} - b = 1$$

$$\bar{w} \cdot \bar{x} - b = -1$$

Если обучающая выборка линейно разделима (два множества точек могут быть полностью отделены единственной прямой. Для p -мерного пространства два набора точек линейно разделимы, если они могут быть отделены $(p-1)$ -мерной гиперплоскостью), то мы можем выбрать гиперплоскости таким образом, чтобы между ними не лежала ни одна точка обучающей выборки и затем максимизировать расстояние между гиперплоскостями. Ширина полосы между ними равна $\frac{2}{\|w\|}$, таким образом наша задача минимизировать $\|w\|$. Чтобы исключить все точки из полосы, мы должны убедиться для всех i , соблюдаются условия:

$$\begin{cases} \bar{w} \cdot \bar{x}_i - b \geq 1, & c_i = 1 \\ \bar{w} \cdot \bar{x}_i - b \leq -1, & c_i = -1 \end{cases}$$

Это может быть также записано в виде:

$$c_i(\bar{w} \cdot \bar{x}_i - b) \geq 1, \quad 1 \leq i \leq n$$

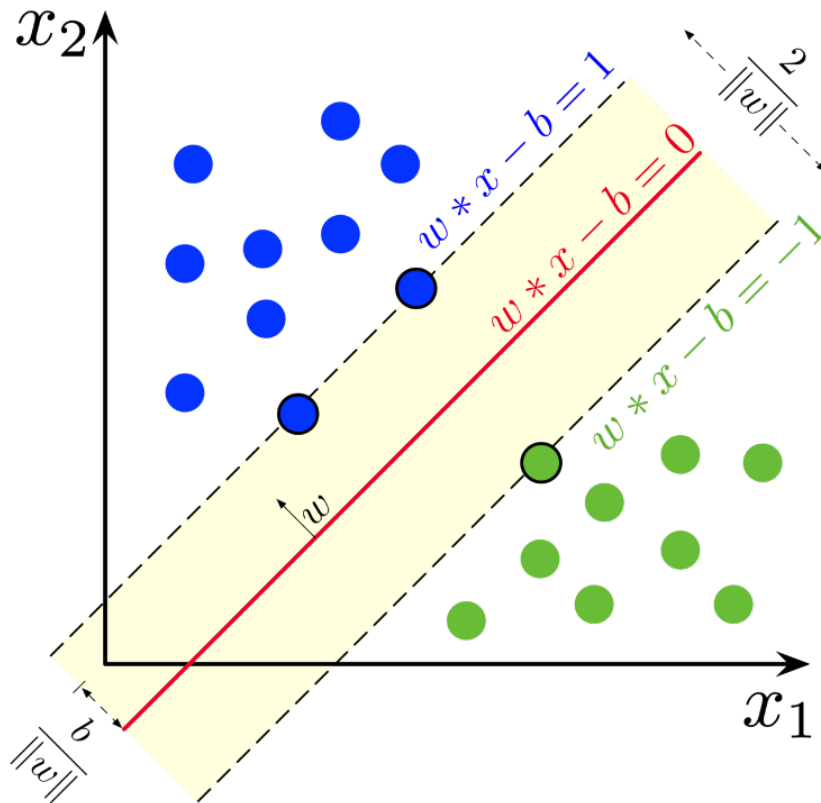


Рис. 9. Пример построения гиперплоскостей для метода опорных векторов

Метод опорных векторов. Регрессия

Регрессия опорных векторов (Support Vector Regression, SVR) — это метод машинного обучения, основанный на идеях метода опорных векторов (SVM), но адаптированный к задаче регрессии, вместо классификации.

Вместо поиска разделяющей гиперплоскости между классами (как в классификации), SVR старается найти функцию, которая игнорирует небольшие отклонения внутри некоторого допустимого порога ε (эпсилон) и акцентирует внимание на точках, которые лежат вне этой “трубы”, — это и есть опорные векторы.

В этом заключается отличие от обычной регрессии, которая старается проложить прямую, наиболее близкую ко всем точкам и “наказывает” любое отклонение.

Таким образом, SVR тоже строит линию (или кривую в случае нелинейного ядра), но с другим подходом, где функция SVR удовлетворена, если предсказание находится в пределах допустимой ошибки ε (эпсилон) от настоящего значения (рис. 10).

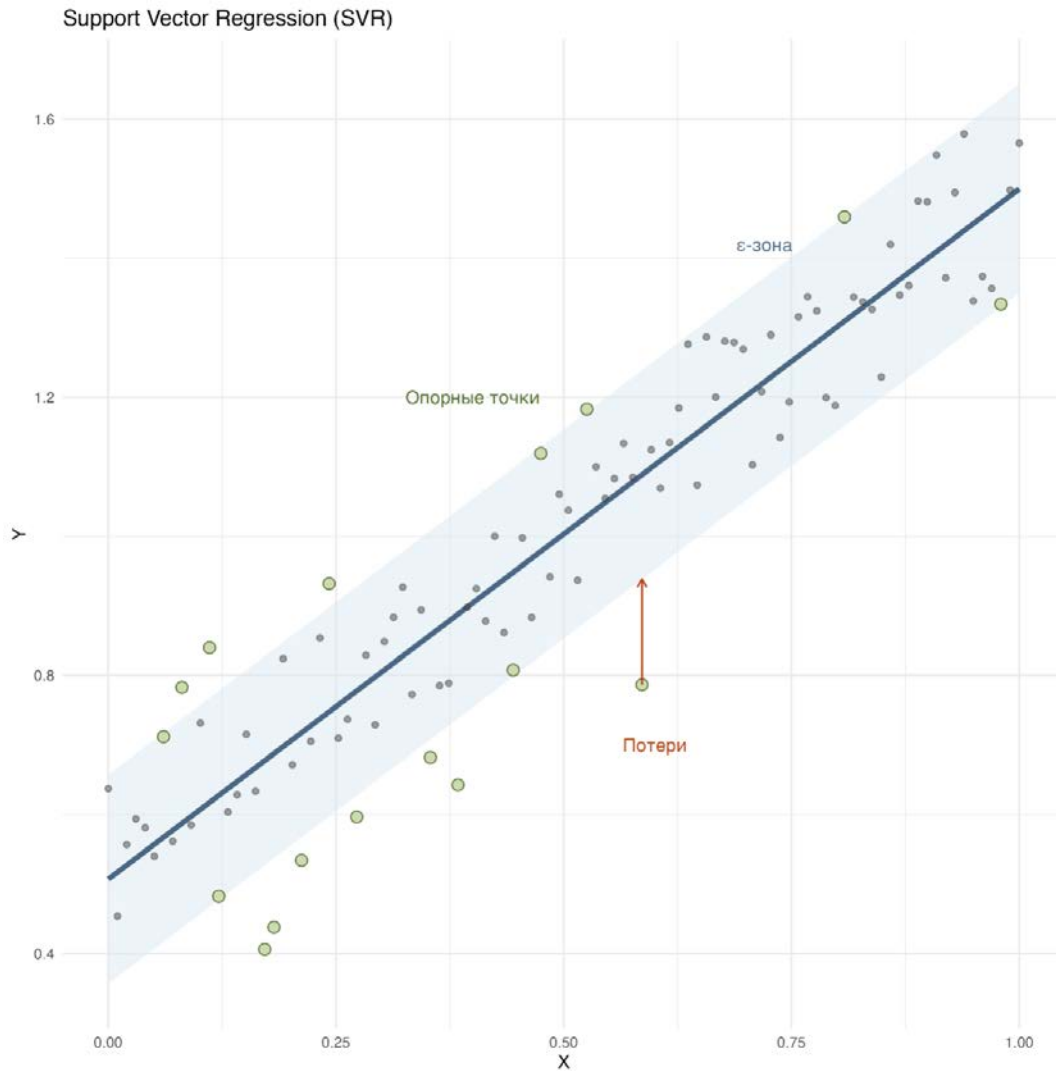


Рис. 10. Пример регрессии методом опорных векторов

SVR концентрируется только на тех точках, что выходят за эту “зону безразличия” (ε -зону) или лежат на ее границе — они называются опорными векторами. Именно они определяют форму и положение модели. Остальные точки (в пределах ε) никак не влияют на модель. Функцию потерь можно выразить следующим образом:

$$L(y, f(x)) = \begin{cases} 0, & \text{при } |y - f(x)| \leq \varepsilon \\ |y - f(x)| - \varepsilon, & \text{при } |y - f(x)| \geq \varepsilon \end{cases}$$

ε - максимально допустимая ошибка

y - фактическое значение

$f(x)$ - предсказанное значение

Ядровые функции

Алгоритм построения оптимальной разделяющей гиперплоскости, предложенный в 1963 году Владимиром Вапником и Алексеем Червоненкисом это алгоритм линейной классификации. Однако, в 1992 году Бернхард Босер, Изабель Гийон и Вапник предложили способ создания нелинейного классификатора, в основе которого лежит переход от скалярных произведений к произвольным ядрам, так называемый kernel trick, позволяющий строить нелинейные разделители.

Результирующий алгоритм похож на алгоритм линейной классификации, с той лишь разницей, что каждое скалярное произведение в приведённых выше формулах заменяется нелинейной функцией ядра (скалярным произведением в пространстве с большей размерностью). В этом пространстве уже может существовать оптимальная разделяющая гиперплоскость. Так как размерность получаемого пространства может быть больше размерности исходного, то преобразование, сопоставляющее скалярные произведения, будет нелинейным, а значит функция, соответствующая в исходном пространстве оптимальной разделяющей гиперплоскости, будет также нелинейной.

Если исходное пространство имеет достаточно высокую размерность, то выборка может быть линейно разделимой.

Ядро - Математическая функция, используемая для отображения исходных точек входных данных в более высокоразмерное пространство признаков, что позволяет находить гиперплоскость даже при нелинейной разделимости данных. Существует множество ядровых функций, они

определены для пар точек из обучающей выборки x_i и x_j и могут содержать дополнительные параметры для эмпирической подстройки. Основные ядровые функции приведены ниже:

Полиномиальная (однородная):

$$k(x_i, x_j) = (x_i \cdot x_j)^d$$

При $d = 1$ функция становится линейной.

Полиномиальная (неоднородная):

$$k(x_i, x_j) = (x_i \cdot x_j + r)^d$$

Радиальная базисная функция:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \text{ для } \gamma > 0$$

Иногда выполняется замена:

$$\gamma = \frac{1}{2\sigma^2}$$

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Сигмоида:

$$k(x_i, x_j) = \tanh(kx_i \cdot x_j + c)$$

для некоторых $k > 0$ и $c < 0$

Процесс классификации с SVM

1. **Сбор и Подготовка Данных** - Выбор объектов для представления данных, масштабирование или преобразование данных для единого масштаба.

2. **Обучение Модели** - SVM получает обучающий набор данных с уже помеченными категориями и находит гиперплоскость с максимальным зазором между классами.

3. **Применение Модели** - Новые данные преобразуются в то же пространство признаков, что и обучающие, и определяется метка класса на основе положения точки относительно гиперплоскости.

Типы SVM

Линейный SVM - Для задач двоичной классификации с линейно разделимыми данными.

Нелинейный SVM - Использует нелинейную функцию ядра для преобразования данных в многомерное пространство.

SVM с Мягкими Границами - Допускает ошибки классификации и может быть более надежным, чем SVM с жесткими границами.

SVM для регрессии (SVR) - Прогнозирует непрерывные значения вместо меток класса.

Многоклассовая SVM - Решает задачи классификации с более чем двумя классами.

Преимущества SVM

Высокая точность - Способность находить оптимальную гиперплоскость для разделения классов.

Устойчивость к шумам - Может обрабатывать данные с большим количеством шума и выбросов.

Простота реализации - Легко понимается, реализуется и используется без глубоких знаний об оптимизации.

3.2. Задание лабораторной работы

Выполните регрессию методом опорных векторов на произвольном наборе данных, представляющих собой временную зависимость (в приведённом примере стоимость акций компании Apple).

Обучите три модели регрессии: линейную, полиномиальную и радиально-базисную. Сделайте предсказание новых точек с датой/временем, следующим за последней записью в наборе данных. Визуализируйте результаты обучения и предсказания новых точек с помощью библиотеки matplotlib.

Представьте результаты в виде графика, на котором изображены данные – значения, предсказанные линейной, полиномиальной и радиально-базисной моделями.

3.3. Порядок выполнения лабораторной работы

Рассмотрим регрессию методом опорных векторов на примере датасета стоимости акций компании Apple (AAPL.csv). Набор данных представляет собой зависимость стоимости акций от времени. На графике стоимость отображается по оси ординат (Y), а время по оси абсцисс (X). Задача модели регрессии, обученной на данном наборе, – при получении на вход значения времени (даты), следующего за последней записью в наборе, предсказать для него стоимость акции.

В примере используются три функции – линейная, полиномиальная, которые позволяют предсказать общий тренд и радиально-базисная функция, которая позволяет более точно предсказать следующие точки. Все даты конвертируется в целочисленное значение.

1. Для начала работы необходимо скачать произвольный набор данных, представляющий собой временную зависимость. В примере используется датасет стоимости акций компании Apple (AAPL.csv).

2. Импортировать набор данных с помощью библиотеки pandas и отобразить последние 100 строк.

3. Даты в наборе данных необходимо преобразовать в целочисленные значения. Это те значения, которые будут отображены на оси X.

4. Создать функцию для предсказания новых точек с помощью обученных моделей.

5. Предсказать новые точки с целочисленными значениями даты (в примере от 620 до 630 с шагом 3).

6. Визуализировать результаты обучения и предсказания новых точек с помощью библиотеки `matplotlib`.

Код программы

```
import pandas as pd
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt

df = pd.read_csv("AAPL.csv")
df = df.tail(100) #last 100 rows
print(df)

def predict_value(svr_lin: SVR, svr_poly: SVR, svr_rbf: SVR, x):
    x = np.reshape(x, (-1, 1))
    return svr_lin.predict(x)[0], svr_poly.predict(x)[0],
svr_rbf.predict(x)[0]

def train(dates, prices):
    """
    SVM - Support Vector Machine (Метод опорных векторов)
    SVR - Support Vector Regression (Регрессия опорных векторов)
    Подробнее о функциях: https://scikit-learn.org/stable/modules/svm.html
    """
    dates = np.reshape(dates, (len(dates), 1))
    # Linear function SVR (Линейная функция)
    svr_lin = SVR(kernel='linear', C=1e3)
    # Polynomial function SVR (Полиномиальная функция)
    svr_poly = SVR(kernel='poly', C=1e3, degree=2)
    # Radial basis function SVR (Радиальная базисная функция)
    svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
    # Обучение моделей регрессии
    svr_lin.fit(dates, prices)
    svr_poly.fit(dates, prices)
    svr_rbf.fit(dates, prices)

plt.scatter(dates, prices, color='black', label='Data')
plt.plot(dates, svr_lin.predict(dates), color='red', label='Linear
model')
```

```

plt.plot(dates, svr_poly.predict(dates), color='green',
label='Polynomial model')
plt.plot(dates, svr_rbf.predict(dates), color='blue', label='RBF
model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Support Vector Regression')
plt.legend()

return svr_lin, svr_poly, svr_rbf

# 3 способа кодирования 'X'
# Кодирование 'X' частью даты
df['Date_int'] = df['Date'].apply(lambda x: int(x.split('-')[1] +
x.split('-')[2]))
# Кодирование 'X' полной датой
#df['Date'] = pd.to_numeric(df['Date'].str.replace('-', ''))
# Кодирование 'X' с помощью индекса pandas dataframe
#df['index_col'] = df.index

svr_lin, svr_poly, svr_rbf = train(df['Date_int'], df['High'])

newdate = 618
# Предсказание новой точки с целочисленным значением даты 618
new_pred = predict_value(svr_lin, svr_poly, svr_rbf, newdate)
plt.scatter(newdate, new_pred[0], color='red', label='Data')
plt.scatter(newdate, new_pred[1], color='green', label='Data')
plt.scatter(newdate, new_pred[2], color='blue', label='Data')

from_date = 620
to_date = 630
# Предсказание новых точек с целочисленными значениями даты от 620 до
630 с шагом 3
for i in range(newdate, to_date, 3):
    new_pred = predict_value(svr_lin, svr_poly, svr_rbf, i)
    plt.scatter(i, new_pred[0], color='red', label='Data')
    plt.scatter(i, new_pred[1], color='green', label='Data')
    plt.scatter(i, new_pred[2], color='blue', label='Data')
plt.show()

```

3.4. Результат работы программы

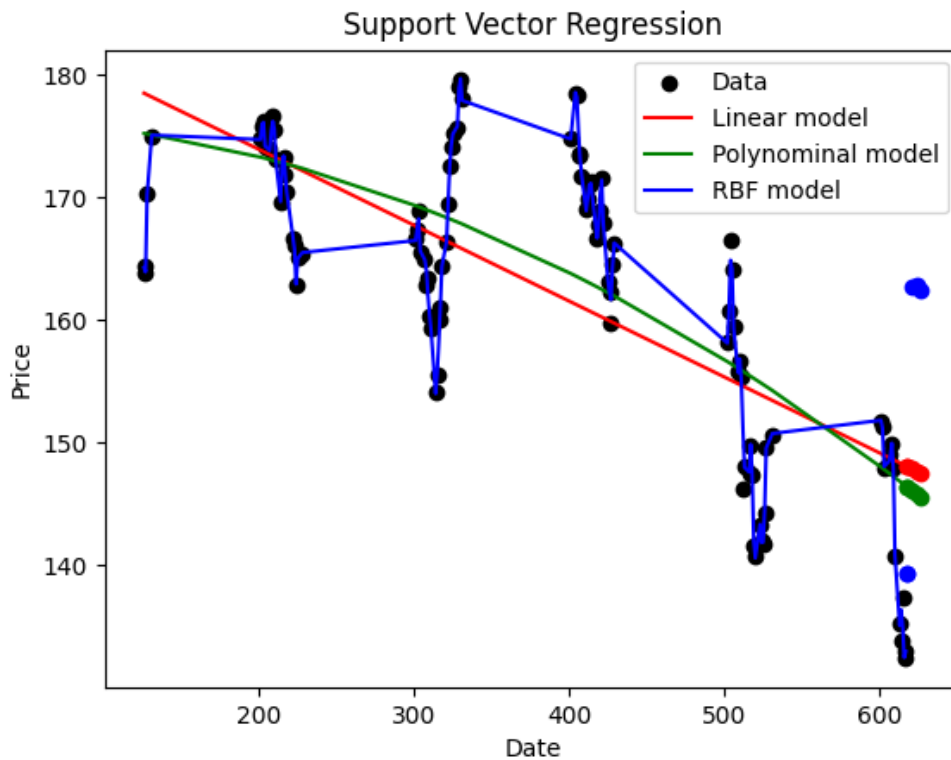


Рис. 11. Результат регрессии и предсказанные точки (не соединены)

3.5. Выводы

SVM является мощным инструментом для задач классификации и регрессии, предлагающим высокую точность, устойчивость к шумам и простоту использования. Его универсальность и эффективность делают его пригодным в различных приложениях машинного обучения.

3.6. Вопросы для проверки

1. Что такое регрессия методом опорных векторов?
2. Какие ядра используются в регрессии методом опорных векторов?
3. Какая библиотека используется для реализации регрессии в этой работе?
4. Как преобразуются даты в целочисленные значения в этой работе?
5. Что такое линейная модель регрессии методом опорных векторов?

6. Какая функция используется для обучения моделей регрессии методом опорных векторов в `scikit-learn`?
7. Что такое полиномиальная модель регрессии методом опорных векторов?
8. Как определяется оптимальное значение параметра C в модели SVR?
9. Как представляются результаты регрессии методом опорных векторов на графике?
10. Что такое радиально-базисная модель регрессии методом опорных векторов?
11. Какой тип данных используется для обучения моделей регрессии методом опорных векторов в этой работе?
12. Как предсказываются новые точки с помощью обученных моделей регрессии методом опорных векторов?
13. Что такое параметр γ в радиально-базисной модели регрессии методом опорных векторов?
14. Для чего используется функция `predict` в `scikit-learn`?
15. Как оценивается качество предсказания моделей регрессии методом опорных векторов?
16. Как определяется оптимальное значение параметра `degree` в полиномиальной модели SVR?
17. Для чего используется метод `fit` в `scikit-learn`?

ЛАБОРАТОРНАЯ РАБОТА № 4.

ВЕКТОРНЫЕ БАЗЫ ДАННЫХ И ИХ ПРОГРАММНАЯ ИНТЕГРАЦИЯ

Цель – освоение процесса создания интерактивных поисковых систем на основе векторных баз данных для произвольного набора данных.

4.1. Теоретическая часть

Векторные базы данных – это тип баз данных, который опирается на принципы векторного сходства и методы поиска ближайшего соседа (Approximate Nearest Neighbor, ANN) для операций поиска данных.

Традиционные реляционные базы данных хранят данные в структурированном виде (таблицы, строки, столбцы).

Главной особенностью векторных баз данных, заключается в том, что они дополняют каждую запись (строку) одним или несколькими векторами.

В контексте семантического поиска эта особенность позволяет отойти от обязательного использования ключевых слов или фраз в операциях поиска в пользу извлечения текстов схожих по векторному представлению смысла.

Векторные базы в зависимости от имплементации могут быть как самостоятельными системами, так и расширением существующих реляционных баз данных.

Вектор или эмбединг – это численная характеристика признаков хранимого объекта, которая представляет объект в многомерном пространстве. В векторной базе данных, вектора обычно имеют фиксированную длину, заранее ограниченную методом их генерации.

Эмбединги могут быть созданы для различных типов данных – текста, изображений, аудио, видео, и.т.д.

Ключевая идея векторных баз данных заключается в том, что схожие объекты представлены близкими векторами в этом пространстве. Следовательно, поиск основан на поиске ближайших соседей (Approximate Nearest Neighbor, ANN) – векторов, наиболее близких к заданному вектору запроса.

Существует множество способов оценить расстояние между двумя векторами. В обработке естественного языка (NLP) чаще используются скалярное произведение и косинусное расстояние.

Евклидово расстояние (рис. 12) - это длина кратчайшего пути между двумя точками или векторами.

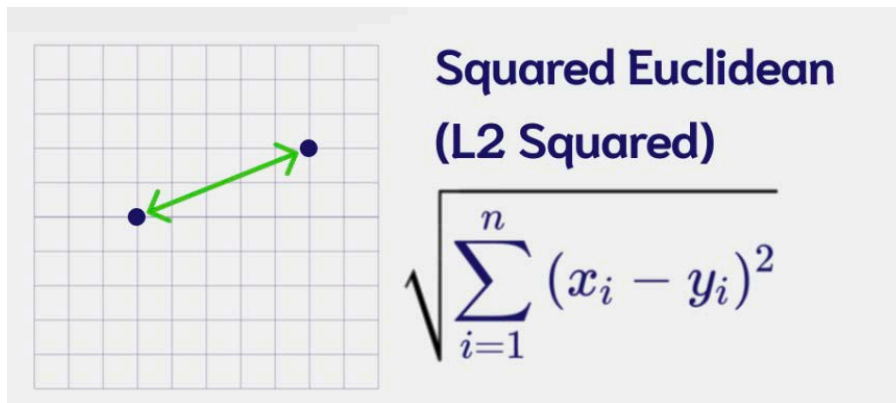


Рис. 12. Евклидово расстояние (L2)

Манхэттенское расстояние (рис. 13), или расстояние городских кварталов равно сумме модулей разностей их координат.

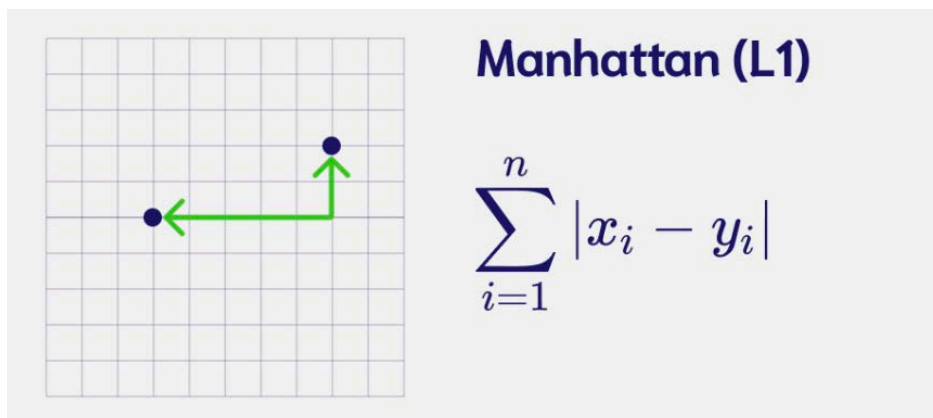


Рис. 13. Манхэттенское расстояние (L1)

Скалярное произведение (обычно нормализованное методом векторизации) (рис. 14) равно сумме произведений соответствующих координат векторов.

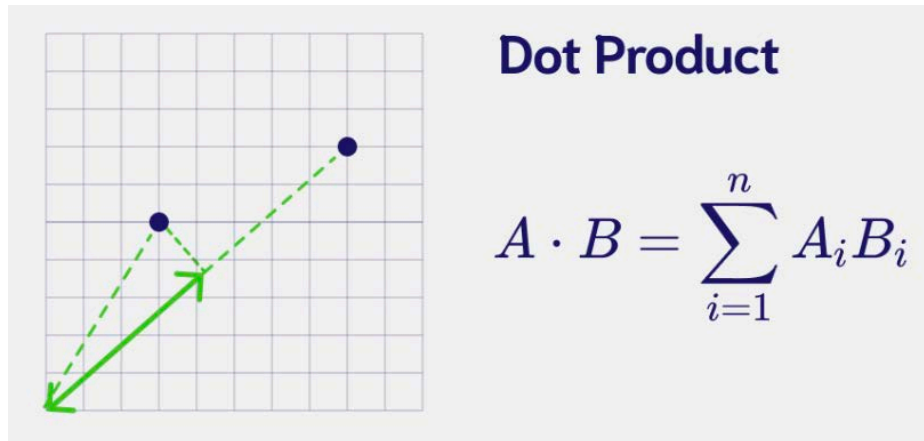


Рис. 14. Скалярное произведение

Косинусное расстояние (рис. 15) представляет собой разницу в направлении векторов.

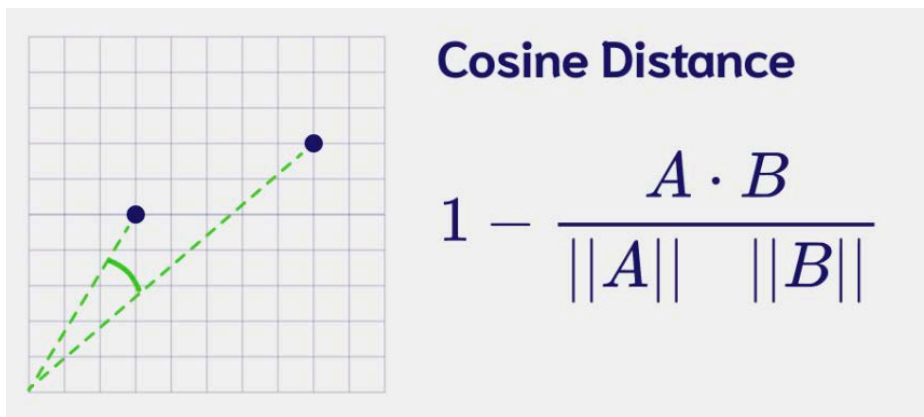


Рис. 15. Косинусное расстояние

Преимущества векторных баз данных

Семантический поиск. Более точный и релевантный поиск, основанный на извлечённом смысле данных, а не только на ключевых словах или фразах. Отсюда следует возможность находить похожие объекты, даже если они не содержат одинаковых ключевых слов.

Масштабируемость. Векторные базы данных сохраняют возможность эффективно обрабатывать большие объемы данных.

Гибкость. Подходят для различных типов данных (текст, изображения, аудио, видео и т.д.).

Поддержка машинного обучения. Интеграция с моделями машинного обучения для создания и использования эмбедингов.

Недостатки векторных баз данных

Могут проявляться в зависимости от их имплементации

Сложность. Векторные базы данных могут быть сложнее в настройке и обслуживании, чем традиционные базы данных.

Стоимость. Хранение и обработка векторов может быть дорогостоящей, особенно для больших объемов данных.

Выбор модели эмбедингов. Выбор подходящей модели эмбедингов для конкретной задачи или языка может быть сложным.

Размер векторов. Размер векторов может быть большим, что повышает требования к оперативной памяти и хранилищу.

Ограниченная поддержка SQL. Векторные базы данных обычно не поддерживают SQL, что может затруднить интеграцию с существующими системами.

4.2. Задание лабораторной работы

Создание системы поиска с графическим или веб-интерфейсом на основе поиска в векторной базе данных по двум (или более) векторам. Должна быть реализована возможность применения дальнейшей фильтрации к полученным данным (из векторной БД), например, по ключевому слову.

Для реализации векторной базы данных предложен пакет Qdrant. Для создания веб-интерфейса визуализации данных с предложена библиотека Flet (Flutter для Python).

Для создания эмбеддингов предложена модель SentenceTransformer «all-MiniLM-L6-v2».

4.3. Порядок выполнения лабораторной работы

Подготовка окружения: Установите необходимые библиотеки, включая `flet`, `qdrant_client`, `datasets`, `scikit-learn`, `embetter`, и `sentence_transformers`. Также убедитесь, что у вас установлен Python и подходящий редактор или IDE.

Выбор модели векторизации: Выберите модель для преобразования текста в векторы. В данном случае используется `SentenceTransformer("all-MiniLM-L6-v2")`.

Загрузка датасета: Загрузите набор данных, который будет использоваться для демонстрации поиска. Здесь выбрано `"somosnlp-hackathon-2022/scientific_papers_en"` с разделением на обучающую выборку (`split='train'`).

Объявление соединения с векторной БД: Создайте соединение с векторной базой данных, используя `QdrantClient`. Если коллекция не существует, создайте ее с необходимыми конфигурациями для хранения векторов.

Векторизация текстовых данных: Примените пайплайны векторизации к столбцам датасета, чтобы преобразовать текстовые данные в векторы.

Загрузка данных в векторную БД: Загрузите векторизованные данные в векторную базу данных, используя метод `upsert` для обновления или вставки точек в коллекцию.

Создание интерфейса поиска: Разработайте интерфейс пользователя с помощью `flet`, который включает текстовое поле для ввода запроса, выпадающий список для выбора типа вектора, поле для ввода ключевого слова и кнопку поиска.

Обработка запроса: Реализуйте функцию, которая обрабатывает ввод пользователя (текст запроса, тип вектора и ключевое слово), выполняет поиск в векторной БД (в примере `query_points`) и отображает результаты в таблице.

Фильтрация результатов: Реализуйте механизм фильтрации результатов с соответствующими элементами интерфейса. Фильтрация может быть по ключевому слову, фразе, категории, по длине ответа и т.п.

Запуск и тестирование приложения: Запустите приложение с помощью `ft.app(main)` и протестируйте функциональность поиска, вводя запросы, выбирая вектор поиска и применяя алгоритм фильтрации результатов.

Код программы

```
import flet as ft
from qdrant_client import QdrantClient
from datasets import load_dataset
from sklearn.pipeline import make_pipeline
from embetter.grab import ColumnGrabber
from embetter.text import SentenceEncoder
from qdrant_client.http.models import PointStruct, VectorParams,
Distance
from sentence_transformers import SentenceTransformer

# Выбор модели векторизации
encoder = SentenceTransformer("all-MiniLM-L6-v2")
# Загрузка датасета
ds = load_dataset("somosnlp-hackathon-2022/scientific_papers_en",
split='train')
# Выбор первых 100 строк из датасета (для быстрогодействия)
ds = ds.select(range(0, 100))

# Объявление соединения с векторной БД
client = QdrantClient(":memory:")

if not client.collection_exists("sci_papers"):
    client.create_collection(
        # Название коллекции
```

```

collection_name="sci_papers",
# Конфигурация векторов в коллекции
vectors_config={
    "abstract_v": VectorParams(
        # Размерность вектора
        size=encoder.get_sentence_embedding_dimension(),
        # Метрика расстояния
        distance=Distance.COSINE,
    ),
    "text_no_abstract_v": VectorParams(
        size=encoder.get_sentence_embedding_dimension(),
        distance=Distance.COSINE,
    ),
},
)

pipeline_encoder = SentenceEncoder("all-MiniLM-L6-v2")
# Пайплайн для векторизации столбца "abstract"
abstract_pipeline = make_pipeline(
    ColumnGrabber("abstract"),
    pipeline_encoder,
)
# Пайплайн для векторизации столбца "text_no_abstract"
text_no_abstract_pipeline = make_pipeline(
    ColumnGrabber("text_no_abstract"),
    pipeline_encoder,
)

ds_df = ds.to_pandas()
sample_df = ds_df.sample(n=100, random_state=643)

abstract_vectors = abstract_pipeline.transform(sample_df)
text_no_abstract_vectors =
text_no_abstract_pipeline.transform(sample_df)
sample_df["abstract_vectors"] = abstract_vectors.tolist()
sample_df["text_no_abstract_vectors"] =
text_no_abstract_vectors.tolist()

for index, row in sample_df.iterrows():
    client.upsert(
        collection_name="sci_papers",

```

```

points=[
    PointStruct(
        # Уникальный идентификатор точки
        id=index,
        vector={
            # Векторное представление "abstract"
            "abstract_v": row["abstract_vectors"],
            # Векторное представление "text_no_abstract"
            "text_no_abstract_v":
row["text_no_abstract_vectors"],
        },
        payload={
            # Исходный текст "abstract"
            "abstract": row["abstract"],
            # Исходный текст "text_no_abstract"
            "text_no_abstract": row["text_no_abstract"],
            # Идентификатор записи
            "id": row["id"],
        }
    )
]

def main(page: ft.Page):
    def make_row(id, score, abstract, text_no_abstract):
        return ft.DataRow(
            cells=[
                # Ячейка с идентификатором
                ft.DataCell(ft.Text(id, overflow=ft.TextOverflow.FADE,
max_lines=5, expand=True)),
                # Ячейка с оценкой
                ft.DataCell(ft.Text(score,
overflow=ft.TextOverflow.FADE, max_lines=5, expand=True)),
                # Ячейка с текстом abstract
                ft.DataCell(ft.Text(abstract,
overflow=ft.TextOverflow.FADE, max_lines=5, expand=True)),
                # Ячейка с текстом text_no_abstract
                ft.DataCell(ft.Text(text_no_abstract,
overflow=ft.TextOverflow.FADE, max_lines=5, expand=True)),
            ]
        )

```

```

def submit_query(e):
    # Получение текста запроса из текстового поля
    query = text_field.value
    # Получение выбранного типа вектора из выпадающего списка
    vector_type = dropdown.value
    # Получение значения фильтра по идентификатору
    filter_keyword = keywd.value

    # Если запрос пустой, ничего не делаем
    if not query:
        return

    results = client.query_points(
        collection_name="sci_papers",
        # Преобразование текста запроса в вектор
        query=encoder.encode(query),
        # using - Выбор вектора, по которому идёт поиск
        using=vector_type,
        # Ограничение на количество результатов
        limit=100,
        # Не возвращать векторы в результатах
        with_vectors=False,
        # Возвращать полезную нагрузку
        with_payload=True,
    ).points

    if filter_keyword:
        try:
            # Фильтрация результатов по ключевому слову
            results = [hit for hit in results if filter_keyword in
hit.payload["abstract"]]
        except ValueError:
            # Показ сообщения об ошибке
            page.snack_bar = ft.SnackBar(ft.Text("Введите корректное
ключевое слово"))
            page.snack_bar.open()
            return

    rows = []
    for hit in results:

```

```

        # Заполнение строк результатом
        rows.append(make_row(hit.payload['id'], hit.score,
hit.payload['abstract'], hit.payload['text_no_abstract']))

    table.rows = rows
    # Обновление страницы
    page.update()

page.title = "Поиск по двум векторам"
page.horizontal_alignment = ft.CrossAxisAlignment.CENTER
page.scroll = ft.ScrollMode.ADAPTIVE

text_field = ft.TextField(hint_text="Введите ваш запрос",
on_submit=submit_query, expand=True)

dropdown = ft.Dropdown(
    label="Выберите вектор",
    options=[
        # Опция для вектора "abstract_v"
        ft.dropdown.Option(key="abstract_v", text="Abstract"),
        # Опция для вектора "text_no_abstract_v"
        ft.dropdown.Option(key="text_no_abstract_v",
text="Text_no_abstract"),
    ],
    value="abstract_v",
)
keywd = ft.TextField(
    label="Введите ключевое слово",
    hint_text="Ключевое слово",
    expand=False,
)

search_button = ft.FloatingActionButton(
    #Кнопка поиска
    icon=ft.Icons.SEARCH, on_click=submit_query
)
table = ft.DataTable(
    columns=[
        # Столбец для "id"
        ft.DataColumn(ft.Text('id')),
        # Столбец для "score"

```

```

ft.DataColumn(ft.Text('score')),
# Столбец для "abstract"
ft.DataColumn(ft.Text('abstract')),
# Столбец для "text_no_abstract"
ft.DataColumn(ft.Text('text_no_abstract')),
],
rows=[
    make_row("", "", "", "")
],
# Снять ограничение на высоту строк
data_row_max_height=float('inf')
)

lv = ft.ListView(expand=1, spacing=10, padding=20, auto_scroll=True)
lv.controls = [table]

page.add(
    ft.Row(
        controls=[
            text_field, dropdown, keywd, search_button
        ],
    ),
    ft.Row(controls=[lv])
)

ft.app(main)

```

4.4. Результат работы программы

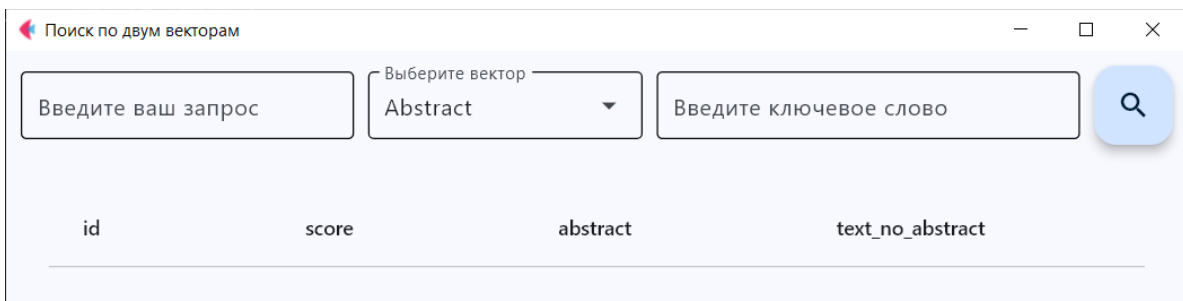


Рис. 16. Интерфейс программы для поиска по двум векторам

Поиск по двум векторам

Euclid

Выберите вектор: Abstract

Введите ключевое слово

id	score	abstract	text_no_abstract
704.0059	0.25125226689241875	We derive masses and radii for both components in the single-lined eclipsing binary HAT-TR-205-013, which consists of a F7V primary and a	Introduction, we assume that the spin axes of both stars have been aligned with the orbital normal and that the rotation of both stars has been synchronized to the orbital period. This allows us to use the observed rotational line broadening of the primary to solve for the radius of the primary in linear units, which in turn allows us to convert the orbital size and secondary radius into
704.0023	0.16858060196073116	The very nature of the solar chromosphere, its structuring and dynamics, remains far from being properly understood, in spite of	Introduction The chromosphere remains the least understood layer of the solar atmosphere, with the very basics of its structure being hotly debated: is it better described by the classical picture of a steady temperature rise as a func-
704.0026	0.16803263840677954	Zero-divisors (ZDs) derived by Cayley-Dickson Process (CDP) from N-dimensional hypercomplex numbers (N a power of 2, at	Introduction By Way of Reprise: From Box-Kites to ETs The creation of 2N-dimensional analogues of Complex Numbers (and it was not a trivial insight of 19th Century algebra that legitimate analogs always have dimension a power of 2) is handled by a now well-known algorithm called the Cayley-
704.006	0.15712417717124563	We investigate the Coulomb excitation of low-lying states of unstable nuclei in intermediate energy collisions ($5E \text{ (lab)} \sim 10 \text{--} 5005$	Coulomb excitation of unstable nuclei at intermediate energies C.A. Bertulani ¹ *, G. Cardella ² , M. De Napoli ^{2,3} , G. Raciti ^{2,3} , and E. Rapisarda ^{2,3} 1 Department of Physics and Astronomy, University of Tennessee, Knoxville, Tennessee 37996, USA 2 Istituto Nazionale di Fisica Nucleare, Sezione di Catania,
704.0004	0.14437744165826444	We show that a determinant of Stirling cycle numbers counts unlabeled acyclic single-source automata. The proof involves a bijection from	Introduction The chief purpose of this paper is to show bijectively that a determinant of Stirling cycle numbers counts unlabeled acyclic single-source automata. Specifically, let $A_k(n)$ denote the $k \times k$ matrix with (i, j) entry [L i-1 L i-1
		Partial cubes are isometric	Introduction

Рис. 17. Результат поиска по фразе «Euclid»

Поиск по двум векторам

Euclid

Выберите вектор: Abstract

Введите ключевое слово: distance

id	score	abstract	text_no_abstract
704.0017	0.11006713666157575	Results from spectroscopic observations of the Intermediate Polar (IP) EX Hya	Introduction Observations And Data Reduction 1991 Observations 2001 Observations The Radial Velocities
704.0003	0.04490583845239833	The evolution of Earth-Moon system is described by the dark matter field fluid model	Introduction The popularly accepted theory for the formation of the Earth-Moon system is that the Moon was formed from debris of a strong impact by a giant planetesimal with the Earth at the close of the planet-forming period (Hartmann and Davis 1975). Since the formation of the Earth-Moon system, it has been evolving at all time scale. It is well
704.008	-0.0021871633388557653	We show that the globular cluster mass function (GCMF) in the Milky Way	THE ASTROPHYSICAL JOURNAL, 679:1272-1287, 2008 JUNE 1 Preprint typeset using LATEX style emulateajp v. 08/22/09 SHAPING THE GLOBULAR CLUSTER MASS FUNCTION BY STELLAR-DYNAMICAL EVAPORATION DEAN E. MCLAUGHLIN ^{1,2} AND S. MICHAEL FALLS ^{3,4} The Astrophysical Journal, 679:1272-1287, 2008 June 1
704.0089	-0.030451127970450357	Statistical modeling of experimental physical laws is based on the	Introduction Fundamentals of nonparametric modeling Description of kernel function Nonparametric estimation of PDF pertaining to experimental data Estimation of a physical law

Рис. 18. Результат поиска по фразе «Euclid» с фильтром по ключевому слову «distance»

4.5. Выводы

В ходе выполнения лабораторной работы была успешно реализована система поиска на основе векторной базы данных Qdrant, использующая эмбединги, сгенерированные моделью SentenceTransformer.

Был освоен процесс загрузки данных, их векторизации и сохранения в векторной базе данных. Созданный интерфейс с использованием Flet позволил реализовать удобный поиск по двум векторам (abstract и text_no_abstract) с возможностью фильтрации результатов по ключевому слову. Практическая реализация подтвердила преимущества векторных баз данных в семантическом поиске, где результаты выдаются на основе смыслового сходства, а не простого совпадения ключевых слов.

Данная лабораторная работа позволила получить практический опыт работы с векторными базами данных и их интеграцией с инструментами для разработки пользовательского интерфейса. Применение Qdrant и SentenceTransformer оказалось эффективным для реализации системы, способной находить релевантные документы на основе семантического понимания текста.

Возможность фильтрации результатов по ключевому слову дополнительно повысила точность поиска и удобство использования системы. Полученные навыки могут быть применены для разработки более сложных и специализированных систем поиска и анализа данных в различных областях, таких как интеллектуальный поиск, анализ научных публикаций, чат-боты и системы рекомендаций.

4.6. Вопросы для проверки

1. Как векторные базы данных могут быть использованы для эффективного хранения и обработки информации?

2. Какие технические решения необходимы для успешной интеграции семантического поиска с векторными базами данных?
3. Какая библиотека и методы используются для работы с наборами данных в этой лабораторной работе?
4. Как в Вашей работе называется метод, который используется для поиска точек в векторной базе данных и как он реализован?
5. Что происходит, если коллекция в векторной базе данных не существует при попытке соединения?
6. Что происходит, если коллекция в векторной базе данных существует при попытке соединения?
7. Для чего служит пайплайн векторизации в этом проекте?
8. Какая библиотека используется для создания интерфейса пользователя в данном приложении поиска?
9. Как реализовано соединение с векторной базой данных в этой лабораторной работе?
10. Какая модель используется для векторизации текстовых данных?
11. Как называется метод, используемый для операции «обновления или вставки» точек в коллекцию векторной базы данных?
12. Какие типы данных можно закодировать в виде векторов для хранения в векторной базе данных?

ЛАБОРАТОРНАЯ РАБОТА № 5.

СИМБИОЗ МАШИННОГО ОБУЧЕНИЯ И МАТЕМАТИЧЕСКИХ МЕТОДОВ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Цель – интеграция методологии генерации с дополняющей выборкой (Retrieval-Augmented Generation, RAG) на основе контекстной векторной базы данных и большой языковой модели (LLM).

5.1. Теоретическая часть

Генерация с дополняющей выборкой (RAG, Retrieval-Augmented Generation) - это мощная методика, которая комбинирует возможности больших языковых моделей (LLM) с внешним источником знаний, позволяя LLM генерировать более точные, контекстуально релевантные и обоснованные ответы. В последние годы RAG стала одним из основных подходов для преодоления ограничений LLM, особенно в отношении, актуализации информации, заложенной в них на этапе обучения и достижения большей прозрачности в получаемых от LLM ответов.

Для чего нужна RAG

Большие языковые модели, такие как LLaMA, Mistral, GPT-4, Gemma, Gemini и другие, обучены на огромных объемах текстовых данных, но имеют ряд ограничений.

Ограниченные знания. Знания LLM ограничены данными, на которых они были обучены. Они не располагают информацией о событиях, произошедших после даты обучения, и могут не обладать специфическими знаниями в узких областях.

Галлюцинации. LLM могут генерировать правдоподобные, но неверные или выдуманные ответы ("галлюцинации").

Обновление знаний. Переобучение LLM для включения новых знаний - это дорогостоящий и трудоемкий процесс.

Отсутствие прозрачности. Сложно понять, почему был получен тот или иной ответ.

RAG решает эти проблемы, позволяя LLM получать доступ к внешнему источнику знаний во время генерации ответов.

Преимущества RAG

Повышение точности. Уменьшение галлюцинаций за счет опоры на проверенные данные.

Актуальность информации. Легкость обновления знаний за счет обновления внешнего источника.

Контекстуальность. Возможность настройки внешнего источника для конкретных задач и пользователей.

Объяснимость. Возможность отслеживания источников информации, использованных для генерации ответа.

Экономичность. Не требует переобучения LLM для включения новых знаний.

Адаптивность. Может использоваться с различными LLM и источниками данных.

Недостатки RAG

Сложность реализации. Требует настройки и управления несколькими компонентами (индексация, поиск, генерация).

Качество данных. Качество ответов зависит от качества данных во внешнем источнике. Неточные или неполные данные могут привести к неточным ответам.

Chunking strategy. Выбор оптимальной стратегии разбиения текста на фрагменты - непростая задача.

Performance. Время ответа может быть увеличено из-за поиска информации во внешнем источнике.

Semantic drift. Со временем, значения слов и фраз могут меняться, что может привести к тому, что модель не сможет найти релевантные фрагменты текста.

Retrieval failures. Иногда система может не найти релевантные фрагменты текста, даже если они существуют.

Как работает RAG

1. Индексация данных (Indexing)

Загрузка данных. Внешние данные (документы, базы знаний, веб-страницы, статьи, и т.д.) загружаются в систему.

Разделение текста (Chunking). Текст разбивается на более мелкие фрагменты (chunks). Размер фрагментов критичен - слишком маленькие фрагменты могут не содержать достаточно контекста, слишком большие - могут снизить эффективность поиска.

Создание векторных представлений (Embeddings). Каждый фрагмент текста преобразуется в векторное представление (embedding) с помощью модели машинного обучения (например, Sentence Transformers, OpenAI Embeddings). Эти векторы отражают семантическое значение фрагмента.

Хранение векторных представлений (Vector Store). Векторные представления хранятся в специализированной векторной базе данных (например, Qdrant, Chroma, Pinecone, Weaviate). Векторные базы данных оптимизированы для быстрого и эффективного поиска по семантическому сходству.

2. Получение информации (Retrieval)

Пользовательский запрос. Пользователь задает вопрос или делает запрос.

Преобразование запроса в вектор. Запрос пользователя преобразуется в векторное представление с помощью той же модели, что и для индексации данных.

Поиск по семантическому сходству. Вектор запроса сравнивается с векторами, хранящимися в векторной базе данных. Ищется K наиболее похожих фрагментов текста. K - это параметр, определяющий количество фрагментов, извлекаемых из базы данных.

Извлечение фрагментов. Найденные фрагменты текста извлекаются из векторной базы данных.

3. Генерация ответа (Generation)

Объединение запроса и фрагментов: Запрос пользователя и извлеченные фрагменты текста объединяются в один промпт (prompt). Промпт может быть сформирован разными способами, например, можно добавить инструкции для LLM, чтобы она использовала информацию из фрагментов для ответа на вопрос.

Генерация ответа LLM: LLM получает промпт и генерирует ответ. LLM использует информацию из извлеченных фрагментов, чтобы сгенерировать ответ, который является более точным, контекстуально релевантным и обоснованным.

5.2. Задание лабораторной работы

Реализовать методологию генерации с дополняющей выборкой (Retrieval-Augmented Generation, RAG) для генерации ответов на сложные запросы с использованием больших языковых моделей и векторных баз данных. В рамках этой работы необходимо изучить теоретические основы RAG, включая принципы работы больших языковых моделей, индексацию данных и получение информации из векторной базы данных. Далее, необходимо подготовить среду для выполнения лабораторной работы, загрузить и обработать набор данных, создать векторную базу данных с помощью Qdrant и реализовать функции для поиска документов и генерации ответа на основе контекста.

5.3. Порядок выполнения лабораторной работы

1. **Изучение теоретической части.** Перед началом работы необходимо тщательно изучить теорию, лежащую в основе методологии генерации с дополняющей выборкой (Retrieval-Augmented Generation, RAG). Это включает понимание принципов и ограничений больших языковых моделей (LLM), преимуществ и недостатков RAG, а также основные этапы реализации RAG: индексация данных, получение информации и генерация ответа.

2. **Подготовка среды.** Для выполнения лабораторной работы необходимо подготовить соответствующую anaconda среду. Это включает установку необходимых библиотек и модулей Python, таких как torch, transformers, sentence-transformers, qdrant_client, и другие, указанные в коде программы.

3. **Загрузка и предварительная обработка данных.** Далее необходимо загрузить набор данных (в примере, «somosnlp-hackathon-2022/scientific_papers_en») и выполнить его предварительную обработку. Это включает выбор первых 100 строк из датасета для ускорения процесса отладки, объявление соединения с векторной базой данных (Qdrant), создание пайплайнов для векторизации столбцов с текстом (в примере "abstract" и "text_no_abstract"), и трансформация этих столбцов в векторные представления.

4. **Заполнение векторной базы данных.** После предварительной обработки данных необходимо ими заполнить векторную базу данных Qdrant. Это включает отправку запросов на добавление точек (документов) в коллекцию (в примере "sci_papers") с их соответствующими векторными представлениями и полезной нагрузкой.

5. **Реализация функций для поиска документов и генерации ответа.** Необходимо реализовать две ключевые функции:

retrieve_documents для поиска ближайших документов в Qdrant по заданному запросу, и generate_answer для формирования контекста из найденных документов и последующей генерации ответа на основе этого контекста с помощью большой языковой модели (LLM).

6. **Тестирование RAG-пайплайна.** После реализации всех необходимых функций необходимо протестировать весь RAG-пайплайн на подобном запросе (в примере "Answer according to the context What is Numerical solution of shock and ramp compression for general material properties?"). Это тестирование должно включать методы поиска соответствующих документов, генерации ответа и вывода результата.

7. **Анализ результатов.** Наконец, необходимо проанализировать сгенерированный ответ на предмет его релевантности, точности и полноты в контексте заданного запроса. Следует сравнить выдачу языковой модели с существующими источниками информации или экспертными мнениями для оценки качества генерируемых ответов.

Код программы

```
import torch
from datasets import load_dataset
from sklearn.pipeline import make_pipeline
from embetter grab import ColumnGrabber
from embetter.text import SentenceEncoder
from qdrant_client import QdrantClient
from qdrant_client.http.models import VectorParams, Distance,
PointStruct
from sentence_transformers import SentenceTransformer
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline,
BitsAndBytesConfig

# Инициализация соединения с векторной БД (замените на ваши параметры
подключения)
#qd_client = qdrant_client.QdrantClient(host='localhost', port=6333)
# Инициализация соединения с векторной БД в памяти
qd_client = QdrantClient(":memory:")
# Загрузка модели для векторизации текста
```

```

vectorizer_model = SentenceTransformer('all-MiniLM-L6-v2')
# Загрузка LLM в RAM/VRAM
model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen3-1.7B",
torch_dtype="auto")
# Опциональное квантование, для ускорения работы LLM (вместо предыдущей
строки)
#quantization_config = BitsAndBytesConfig(load_in_4bit=True,
bnb_4bit_compute_dtype=torch.bfloat16)
#model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen3-1.7B",
torch_dtype="auto", quantization_config=quantization_config)
# Загрузка токенизатора LLM в RAM/VRAM
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen3-1.7B")

# Загрузка модели Mistralai для генерации ответов
generation_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer
)
encoder = SentenceEncoder("all-MiniLM-L6-v2")

ds = load_dataset("somosnlp-hackathon-2022/scientific_papers_en",
split='train')
# Выбор первых 100 строк из датасета (для быстрогодействия)
ds = ds.select(range(0, 100))

# Объявление соединения с векторной БД
if not qd_client.collection_exists("sci_papers"):
    qd_client.create_collection(
        # Название коллекции
        collection_name="sci_papers",
        # Конфигурация векторов в коллекции
        vectors_config={
            "abstract_v": VectorParams(
                # Размерность вектора

size=vectorizer_model.get_sentence_embedding_dimension(),
                # Метрика расстояния
                distance=Distance.COSINE,
            ),
            "text_no_abstract_v": VectorParams(

```

```

size=vectorizer_model.get_sentence_embedding_dimension(),
        distance=Distance.COSINE,
    ),
},
)

# Пайплайн для векторизации столбца "abstract"
abstract_pipeline = make_pipeline(
    ColumnGrabber("abstract"),
    encoder,
)

# Пайплайн для векторизации столбца "text_no_abstract"
text_no_abstract_pipeline = make_pipeline(
    ColumnGrabber("text_no_abstract"),
    encoder,
)

ds_df = ds.to_pandas()
sample_df = ds_df.sample(n=100, random_state=643)

abstract_vectors = abstract_pipeline.transform(sample_df)
text_no_abstract_vectors =
text_no_abstract_pipeline.transform(sample_df)
sample_df["abstract_vectors"] = abstract_vectors.tolist()
sample_df["text_no_abstract_vectors"] =
text_no_abstract_vectors.tolist()

# Заполнение векторной БД
for index, row in sample_df.iterrows():
    qd_client.upsert(
        collection_name="sci_papers",
        points=[
            PointStruct(
                # Уникальный идентификатор точки
                id=index,
                vector={
                    # Векторное представление "abstract"
                    "abstract_v": row["abstract_vectors"],
                    # Векторное представление "text_no_abstract"

```

```

        "text_no_abstract_v":
row["text_no_abstract_vectors"],
    },
    payload={
        # Исходный текст "abstract"
        "abstract": row["abstract"],
        # Исходный текст "text_no_abstract"
        "text_no_abstract": row["text_no_abstract"],
        # Идентификатор записи
        "id": row["id"],
    }
    )
]
)

```

```

def retrieve_documents(query):
    # Поиск ближайших документов в Qdrant
    results = qd_client.query_points(
        collection_name="sci_papers",
        # Преобразование текста запроса в вектор
        query=vectorizer_model.encode(query),
        # using - Выбор вектора, по которому идёт поиск
        using="abstract_v",
        # Ограничение на количество результатов
        limit=3,
        # Не возвращать векторы в результатах
        with_vectors=False,
        # Возвращать полезную нагрузку
        with_payload=True,
    ).points

    #filter_keyword = "temperatures" # опциональная фильтрация по
ключевому слову
    #results = [hit for hit in results if filter_keyword in
hit.payload["abstract"]]
    return results

def generate_answer(query, retrieved_docs):
    # Формирование контекста из найденных документов
    context = ""
    for doc in retrieved_docs:

```

```

        context += doc.payload["abstract"].replace('\n', '
').replace('\t', ' ').replace('\r', ' ')

        # Создание полного запроса для большой языковой модели (LLM)
        full_prompt = f"Context: {context}\nAnswer according to the context
{query}"

        # Генерация ответа на основе контекста и вопроса
        response = generation_pipeline(full_prompt, max_new_tokens=150)
        return response[0]['generated_text']

def rag_pipeline(query):
    # Поиск документов в векторизованной базе данных
    retrieved_docs = retrieve_documents(query)

    # Генерация ответа на основе найденных документов
    answer = generate_answer(query, retrieved_docs)

    return answer

# Пример использования
query = """What is Numerical solution of shock and ramp compression for
general material properties"""
answer = rag_pipeline(query)
print("Answer:", answer)

```

5.4. Результат работы программы в консоли

Текст запроса

Context: A general formulation was developed to represent material models for applications in dynamic loading. Numerical methods were devised to calculate response to shock and ramp compression, and ramp decompression, generalizing previous solutions for scalar equations of state. The numerical methods were found to be flexible and robust, and matched analytic results to a high accuracy. The basic ramp

and shock solution methods were coupled to solve for composite deformation paths, such as shock-induced impacts, and shock interactions with a planar interface between different materials. These calculations capture much of

the physics of typical material dynamics experiments, without requiring spatially-resolving simulations. Example calculations were made of loading histories in metals, illustrating the effects of plastic work on the temperatures induced in quasi-isentropic and shock-release experiments, and the effect of a phase transition. We construct a system of nonequilibrium entropy limiters for the lattice Boltzmann methods (LBM). These limiters erase spurious oscillations without blurring of shocks, and do not affect smooth solutions. In general, they do the same work for LBM as flux limiters

do for finite differences, finite volumes and finite elements methods, but for LBM the main idea behind the construction of nonequilibrium entropy limiter schemes is to transform a field of a scalar quantity - nonequilibrium entropy. There are two families of limiters: (i) based on restriction of nonequilibrium entropy (entropy "trimming") and (ii) based on filtering of nonequilibrium entropy (entropy filtering). The physical properties of LBM provide some additional benefits: the control of entropy production and accurate estimate of introduced artificial dissipation are possible. The constructed limiters are tested on classical numerical examples: 1D athermal shock tubes with an initial density ratio 1:2 and the 2D lid-driven cavity for Reynolds numbers Re between 2000 and 7500 on a coarse 100×100 grid. All limiter constructions are applicable for both entropic and non-entropic quasiequilibria. We propose an extrapolation technique that allows accuracy improvement of the discrete dipole approximation

computations. The performance of this technique was studied empirically based on extensive simulations for 5 test cases using many different discretizations. The quality of the extrapolation improves with refining discretization reaching extraordinary performance especially for cubically shaped particles. A two order of magnitude decrease of error was demonstrated. We also propose estimates of the extrapolation error, which were proven to be

reliable. Finally we propose a simple method to directly separate shape and discretization errors and illustrated this for one test case.

Answer according to the context What is Numerical solution of shock and ramp compression for general material properties?

Сгенерированный текст

The numerical solution of shock and ramp compression for general material properties was developed through the creation of a general formulation that allows for the modeling of material behaviors under dynamic loading conditions. This formulation was built upon the development of numerical methods that can compute the response of materials to various types of loading, such as shock and ramp compression, and ramp decompression. These methods are designed to generalize previous solutions for scalar equations of state, which are fundamental in describing the behavior of materials under extreme conditions.

The numerical methods were found to be both flexible and robust, and they were able to match analytical results with high accuracy. This flexibility and robustness enable the methods to handle a wide range of material properties and loading conditions, including composite deformation paths such as shock-induced impacts

5.5. Выводы

Данная лабораторная работа продемонстрировала успешную реализацию пайплайна извлечения информации с использованием Retrieval-Augmented Generation (RAG). В ходе выполнения работы была создана система, которая эффективно находит релевантные документы в базе знаний, представленной в виде векторной базы данных, и использует их как контекст для генерации ответа на заданный вопрос.

Результаты работы показали, что модель способна генерировать связные и информативные ответы, опирающиеся на извлеченную

информацию. Использование векторной базы данных и предварительно обученной LLM позволило создать систему, способную выдавать более точные и контекстуально релевантные ответы, чем простые поисковые системы или LLM без контекста.

5.6. Вопросы для проверки

1. Что такое RAG и для чего используется?
2. Какие основные этапы включает в себя реализация RAG?
3. В чем заключается принцип работы большого языкового моделирования (LLM)?
4. Каковы преимущества использования RAG по сравнению с традиционными методами генерации текста?
5. Что такое Qdrant и как он используется в лабораторной работе?
6. Как выполняется предварительная обработка данных для загруженного набора данных?
7. Для чего используются пайплайны в коде программы?
8. Что такое векторное представление текста и как оно используется в RAG?
9. Какие библиотеки Python используются для реализации лабораторной работы и для чего каждая из них предназначена?
10. Опишите функцию `retrieve_documents` и ее назначение в пайплайне RAG.
11. Как выполняется поиск ближайших документов в Qdrant по заданному запросу?
12. Что такое контекст в генерации ответа и как он формируется?
13. Опишите функцию `generate_answer` и ее роль в пайплайне RAG.
14. Как используется большая языковая модель в пайплайне RAG?
15. Приведите пример запроса, который может быть обработан с помощью лабораторной работы по RAG.

16. Как выполняется анализ результатов и оценка качества сгенерированных ответов?
17. В чем заключается важность предварительной обработки данных для эффективности RAG-пайплайна?
18. Можно ли использовать другие векторные базы данных вместо Qdrant в лабораторной работе? Если да, то какие и почему?
19. Какие потенциальные ограничения или проблемы могут возникнуть при реализации RAG-пайплайна для обработки естественного языка?

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. ["ГОСТ Р 59853-2021 Информационные технологии \(ИТ\). Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения" \(Источник: ИСС "ТЕХЭКСПЕРТ"\)](#)
2. ["Об информации, информационных технологиях и о защите информации \(с изменениями на 23 ноября 2024 года\) \(редакция, действующая с 1 января 2025 года\)" \(Источник: ИСС "ТЕХЭКСПЕРТ"\)](#)
3. Формализация информации и big data: учеб, пособие / авт. кол.: В. П. Часовских, М. П. Воронов, В. Г. Лабунец [и др.]; М-во науки и высш, образования Рос. Федерации, Урал. гос. экон, ун-т - Екатеринбург: Изд-во Урал. гос. экон, ун-та, 2021. — 218 с. ISBN 978-5-9656-0312- 1.
4. Анналин Ын, Кеннет Су. Теоретический минимум по Big Data. Всё, что нужно знать о больших данных. - СПб.: Питер, 2019. - 208 с. ISBN 978-5-4461-1040-7.
5. Радченко И.А, Николаев И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018. – 52 с.
6. Davy Cielen, Arno D. B. Meysman, and Mohamed Ali. Introducing Data Science. Big data, machine learning, and more, using Python tools [Электронный ресурс] / <https://www.manning.com/books/introducing-data-science> (дата обращения 11.08.2025)
7. Программирование на Python для науки о данных (PY4DS) (Python Programming for Data Science) [Электронный ресурс] / <https://www.tomasbeuzen.com/python-programming-for-data-science/chapters/chapter1-basics.html> (дата обращения 11.08.2025)
8. Learning Data Science By Sam Lau, Joey Gonzalez, and Deb Nolan [Электронный ресурс] / <https://learningds.org/intro.html> (дата обращения 11.08.2025)

9. Книга Python для анализа данных, 3-издание (Wes McKinney) (Python for Data Analysis, 3E) [Электронный ресурс] / <https://wesmckinney.com/book> (дата обращения 11.08.2025)
10. Наука о данных в командной строке (Data Science at the Command Line) [Электронный ресурс] / <https://jeroenjanssens.com/dsatcl> (дата обращения 11.08.2025)
11. Техэксперт - Справочная система [Электронный ресурс] / <https://cntd.ru/>
12. OpenML Всемирная лаборатория машинного обучения [Электронный ресурс] / <https://www.openml.org/>
13. Google Colab [Электронный ресурс] / <https://colab.research.google.com/>
14. Kaggle [Электронный ресурс] / <https://www.kaggle.com/>
15. Hugging Face [Электронный ресурс] / <https://huggingface.co/datasets>
16. Visual Studio Code (свободная лицензия) [Электронный ресурс] / <https://code.visualstudio.com/download>
17. Python (Pandas для чтения CSV-файла и фильтрации данных; PySpark; Scikit-Learn) [Электронный ресурс] / <https://www.python.org/downloads/release/python-31017/>
18. Метрики моделей машинного обучения [Электронный ресурс] / <https://sky.pro/wiki/analytics/accuracy-precision-recall-f1-metriki-otsenki-modelej-mashinnogo-obucheniya/>
19. Кластеризация методом k-средних [Электронный ресурс] / <https://fritz.ai/mathematics-behind-k-means-clustering/>
20. Кластеризация. Яндекс образование. [Электронный ресурс] / <https://education.yandex.ru/handbook/ml/article/klasterizaciya>

21. Текстовая классификация Scikit-Learn. Stackabuse. [Электронный ресурс] / <https://stackabuse.com/text-classification-with-python-and-scikit-learn/>
22. Текстовая классификация Scikit-Learn. [Электронный ресурс] / https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
23. Метрики классификации и регрессии. Яндекс образование. [Электронный ресурс] / <https://education.yandex.ru/handbook/ml/article/metriki-klassifikacii-i-regressii>
24. Регрессионные модели с tidymodels. [Электронный ресурс] / https://locusclassicus.github.io/text_analysis_2024/multivar.html
25. Эмбединги и метрики расстояния в обработке естественного языка. [Электронный ресурс] / <https://readmedium.com/embeddings-and-distance-metrics-in-nlp-7a000c96d7db>

Учебное издание

Куфтинова Наталья Григорьевна

Пронин Цезарь Борисович

Остроух Андрей Владимирович

БОЛЬШИЕ ДАННЫЕ

УЧЕБНОЕ ПОСОБИЕ

(лабораторный практикум)

по направлению подготовки, входящих в укрупненную группу

09.00.00 – «Информатика и вычислительная техника»

Доступ – свободный.

<https://nkras.ru/arhiv/2026/978-5-907608-60-3.pdf>

<https://doi.org/10.12731/978-5-907608-60-3>

Сборник содержится в едином файле PDF.

Дата выхода в свет 31.03.2026.

Заказ БД3103/26.

По вопросам приобретения и издания литературы обращаться по адресу:

Издательство «Научно-инновационный центр»

ул. 9 Мая, 5/192, г. Красноярск, 660127 Россия

тел. +7 (995) 080-90-42

Электронная почта: monography@nkras.ru

Дополнительная информация на сайте: www.nkras.ru