МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

Кафедра №43 «Компьютерных технологий и программной инженерии»

ОТЧЁТ ПО ПРАКТИКЕ

ЗАЩИЩЁН С ОЦЕНКОЙ

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | преддипломная | |
| на тему индивидуального задания | | Разработка алгоритма машинного обучения для |
| классификации текстов по жанру литературы | | | |

|  |  |
| --- | --- |
| выполнен | Бобрович Николаем Сергеевичем |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 02.03.03 |  | Математическое обеспечение и администрирование информационных систем |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 02 |  | Системный анализ в информационных технологиях |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4136 |  | Подпись 05.05.2025 |  | Н.С. Бобрович |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2025

# Содержание

[1. Содержание 1](#_Toc19108)

[2. Перечень сокращений 2](#_Toc29739)

[3. Введение 3](#_Toc24359)

[3.1. Анализ предметной области 3](#_Toc28203)

[3.2. Постановка задачи 4](#_Toc32034)

[3.3. Представление этапов решения задачи 6](#_Toc22634)

[3.4. Сравнение с аналогами 7](#_Toc14716)

[4. Основная часть 9](#_Toc23347)

[4.1. Исследование и сбор данных 9](#_Toc3639)

[4.2. Проектирование системы 12](#_Toc14304)

[4.3. Реализация и оценка качества МК 14](#_Toc4407)

[4.4. Оптимизация АК 17](#_Toc26349)

[4.5. Повторная оценка качества МК 28](#_Toc21129)

[4.6. Интеграция МК в ПО 29](#_Toc21422)

[4.7. Документация 32](#_Toc9665)

[4.8. Способы возможной модификации ПО 36](#_Toc2734)

[5. Заключение 40](#_Toc30565)

[6. Список использованных источников 41](#_Toc6140)

[7. Приложения 44](#_Toc5751)

# Перечень сокращений

1. МК - модель классификации.
2. ММО - метод машинного обучения.
3. АК - алгоритм классификации.
4. НД - набор данных.
5. МО - машинное обучение.
6. ПО - программное обеспечение.
7. ГП - гиперпараметр(ы).

# Введение

# Анализ предметной области

Традиционно классификация текстов осуществляется вручную экспертами, однако этот подход требует значительных временных затрат и подвержен субъективному фактору. Автоматизация данного процесса позволяет ускорить обработку большого объёма данных и повысить объективность оценки. К тому же, в большинстве случаев МО как раз и применяется для больших данных, для сокращения трудовых и материальных ресурсов.

Классификация текстов по жанрам литературы представляет собой важную задачу в области обработки естественного языка (NLP). Жанр текста определяет его стилистические особенности, тематику и структуру, что делает автоматическое распознавание жанра полезным инструментом для организации больших коллекций текстов, например, для электронных библиотек или книжных онлайн-магазинов.

# Постановка задачи

Целью данной работы является разработка системы классификации текстов по жанру литературы на основе модели алгоритма машинного обучения.

1. **Формулировка задачи:**

Задача заключается в создании ПО на основе МК (на основе АК МО), способного автоматически определять жанр литературного произведения на основе его текста.

Это позволит упростить процесс каталогизации и поиска книг в библиотеках, электронных магазинах и других информационных системах. Также решение этой задачи - важный шаг на пути к созданию системы персональных рекомендаций литературы. На сегодняшний день на эту тему написано множество статей [1, 5, 7, 14, 19, 24, 26],книг [9] и исследований [2], что подчёркивает её актуальность.

Система должна обеспечивать высокую точность классификации и возможность интеграции в существующие информационные системы (электронные библиотеки, книжные онлайн-магазины).

1. **Основные требования:** 
   1. Функциональные требования.

Классификация текстов: Система должна уметь классифицировать тексты по следующим жанрам:

1. Роман
2. Повесть
3. Рассказ
4. Поэма
5. Пьеса
6. Статья
7. Очерк

Интерфейс ввода: Пользовательский интерфейс должен позволять загружать текстовые файлы в формате .txt, .docx, .pdf.

Интерфейс вывода: Результаты классификации должны отображаться в виде списка с указанием жанра для каждого загруженного текста.

Модуль обучения: Система должна иметь возможность переобучения на новых данных для адаптации к изменениям в литературе.

Язык обрабатываемых текстов: Русский язык.

2.2. Нефункциональные требования:

Производительность: Время обработки одного текста должно составлять не более 10 секунд.

Надёжность: Система должна быть устойчива к ошибкам ввода и правильно реагировать на некорректные данные.

Масштабируемость: Архитектура системы должна позволять легко добавлять новые жанры и типы документов.

Документация: Проект должен сопровождаться подробной документацией, включающей руководство пользователя и техническое описание архитектуры.

1. **Оценка качества:**

Качество работы системы будет оцениваться по следующим критериям:

1) Точность классификации: Процент правильно классифицированных текстов.

2) Скорость обработки: Среднее время обработки одного текста.

3) Устойчивость к ошибкам: Способность системы корректно обрабатывать некорректные данные.

# Представление этапов решения задачи

Для решения поставленной задачи необходимо пройти следующие этапы:

1. Исследование и сбор данных:

Сбор базы текстов различных жанров, предварительная обработка данных.

1. Проектирование системы:

Определение структуры системы, ММО.

1. Реализация и оценка качества работы МК:

Создание МК и её оценка.

1. Оптимизация:

Настройка ГП ММО, экспериментирование с различными методами предобработки данных.

1. Повторная оценка качества работы МК:

При недостаточной эффективности МК должна происходить повторная оптимизация.

1. Интеграция МК в ПО:

Разработка ПО, подготовка системы к эксплуатации, интеграции в целевую информационную среду.

1. Документация:

Составление руководства пользователя и технического описания всей системы.

1. Способы возможной модификации ПО:

Рассмотрение возможных направлений дальнейшего развития системы.

# Сравнение с аналогами

Уже существует ряд решений и подходов, направленных на автоматизацию классификации текстов по жанрам литературы. Рассмотрим наиболее значимые аналоги и сравним их с моим решением:

**1. Библиотеки NLP общего назначения [2, 6]:**

Примеры: spaCy, NLTK, Stanford CoreNLP

Преимущества: Широкий спектр возможностей для анализа текстов, поддержка разных языков, простота интеграции.

Недостатки: Ограниченная функциональность для узконаправленных задач, таких как определение жанра литературы, отсутствие специализированных моделей для конкретных типов текстов.

**2. Коммерческие сервисы облачной аналитики [11, 12, 23]:**

Примеры: Google Cloud Natural Language API, IBM Watson Natural Language Understanding

Преимущества: Высокая производительность, интеграция с облачными сервисами, наличие готовых моделей для различных задач.

Недостатки: Высокие затраты на использование, необходимость передачи данных третьим лицам, ограниченность кастомизации.

1. **Специализированные научные проекты [1, 19, 28]:**

Примеры: исследования от российских вузов и научных центров

Преимущества: Глубокая специализация на конкретной задаче, высокая эффективность благодаря использованию современных методов МО.

Недостатки: Отсутствие готовой инфраструктуры для развёртывания, сложность внедрения в коммерческие продукты.

**Преимущества моего решения:**

Специализированная модель, ориентированная именно на литературу русского языка.

Возможность настройки и оптимизации модели под конкретные нужды пользователей.

Простота интеграции в существующие информационные системы (библиотеки, магазины электронной литературы).

Поддержка основных форматов файлов (.txt, .docx, .pdf).

Открытая архитектура, позволяющая расширять систему новыми жанрами и типами документов.

Таким образом, предложенное решение выгодно отличается от аналогов своей специализированностью и адаптивностью, что делает его оптимальным выбором для автоматизации процессов классификации текстов по жанрам русской литературы.

# Основная часть

# Исследование и сбор данных

Чтобы создать высококачественную МК текстов по жанрам литературы, необходимо собрать репрезентативный НД, охватывающий разнообразные литературные жанры. Процесс сбора данных включает следующие ключевые этапы:

1. **Определение источников данных:**

Основными источниками сбора данных послужили интернет-библиотеки: Алексея Комарова [29] и Максима Мошкова [30].

С помощью скриптов (приложение 1, 2, 3), которые спарсили все произведения с сайтов, было получено свыше 19 тысяч произведений, принадлежащих семи основным жанрам: роман, повесть, рассказ, поэма, пьеса, очерк и статья.

1. **Форматирование данных:** (преобразование текстов в единый формат; удаление избыточных символов, разметки и прочих элементов, не относящихся непосредственно к содержанию текста)

С помощью тех же скриптов (приложение 1, 2, 3) и методов форматирования [21, 22], весь корпус был преобразован в стандартный текстовый формат .txt, нормализован и очищен от лишних метаданных и шумовых элементов.

1. **Аннотирование данных:** (назначение каждому тексту соответствующего жанра; проверка аннотаций на согласованность и качество)

С помощью тех же скриптов (приложение 1, 2, 3), Каждое произведение получило уникальное имя файла, соответствующее следующей схеме:

<жанр>\_<автор>\_<название>\_<номер>.txt.

Эта схема облегчает последующую категоризацию и обработку.

При таком аннотировании очень легко произвести сортировку произведений по жанру. В связи с тем, что скрипт брал жанр напрямую из библиотеки проверка аннотаций на согласованность и качество может считаться успешной.

1. **Обеспечение сбалансированности НД:** (утверждение, что количество текстов в каждом жанре примерно одинаково, чтобы избежать дисбаланса в обучении МК)

Изначальное распределение произведений по жанрам таково:

1. Очерк - 1975
2. Повесть - 2166
3. Поэма - 2020
4. Пьеса - 1868
5. Рассказ - 2015
6. Роман - 2562
7. Статья - 6713

Дабы избежать дисбаланса модели и сохранить как можно большее количество произведений часть произведений будет удалена из начального набора данных случайным образом (для сохранения разнообразности элементов внутри класса).

Конечное распределение произведений по жанрам таково:

1. Очерк - 1975
2. Повесть - 2050
3. Поэма - 2020
4. Пьеса - 1868
5. Рассказ - 2015
6. Роман - 2050
7. Статья - 2050

При таком распределении разница между наименьшим и наибольшим количеством элементов в классе < 10%, что позволяет считать такой НД сбалансированным.

В итоговом НД 14028 произведений, равномерно распределённых по семи различным жанрам.

1. **Обеспечение масштабируемости НД:** (возможность изменять размер НД, сохраняя его сбалансированность)

С помощью скрипта (приложение 4) был проведён контроль качества собранных данных, в первую очередь удалялись дубликаты и низкокачественные записи. Стало возможным масштабирование данных с сохранением их сбалансированности.

# Проектирование системы

Сначала будут проанализированы и определены основные АК и ММО (в будущем в общем случае АК): (информация об АК взята из [3, 5, 8, 9, 16, 17, 28])

1. Naive Bayes (NB) — ММО, основанный на теореме Байеса и предположении о независимости признаков (признаки считаются независимыми друг от друга даже если на самом деле это не так). Подходит для простых и быстрых решений задач классификации. Используется там, где признаки условно независимы.
2. Support Vector Machine (SVM) — мощный алгоритм линейной и нелинейной классификации, особенно эффективен для больших объёмов данных. Основная идея заключается в поиске оптимальной границы разделения данных, называемой гипер-плоскостью, которая максимизирует расстояние до ближайших точек данных (опорных векторов).
3. Decision Tree (DT) — АК, основанный на построении дерева решений, каждая ветвь соответствует условию проверки признака, а листья — итоговым классам. Может переобучаться на сложных данных.
4. Feedforward Neural Network (FFBP) — ММО нейронной сети прямого распространения. Основной особенностью является использование метода обратного распространения ошибки (backpropagation) для оптимизации весов сети посредством градиентного спуска.
5. Recurrent Neural Network (RNN) — ММО, основанный на нейронных сетях, специально предназначенный для анализа последовательных данных (текстов, временных рядов и т.д.) путём хранения информации о предыдущих состояниях.
6. Deep Averaging Network (DAN2) — ММО, использующий предварительную обработку текста через векторные представления слов (embeddings).
7. Convolutional Neural Network (CNN) — ММО, основанный на нейронных сетях. Отличительной чертой являются свёрточные слои, позволяющие обнаруживать локальные паттерны (например, края, углы, формы) независимо от положения на изображении. Эффективно работает с текстовыми данными благодаря своей способности находить локальные паттерны в тексте.

Теперь необходимо спроектировать ПО, определяющее жанр литературы загруженного пользователем текста или документа(ов), на основе полученной МК.

Планируемое веб-приложение будет представлять собой удобный инструмент для автоматического определения жанра литературного произведения на основе анализа текста. Оно будет разработано с учётом современных стандартов проектирования пользовательских интерфейсов и обладать простой, интуитивно понятной структурой.

Технические аспекты:

Серверная сторона будет построена на Flask — лёгком и мощном веб-фреймворке Python, обеспечивающем быстрый старт и простую реализацию REST API. (приложение 13)

Клиентская сторона будет выполнена с использованием стандартных инструментов HTML, CSS и JavaScript, обеспечивая лёгкость поддержки и расширения функциональности. (приложение 14, 15)

Модель классификации: в основе будет лежать предварительно обученная МК, использующая подход TF-IDF для представления текста и оптимальный ММО, который будет найден позже, для принятия решений.

Работа с текстовыми файлами: для удобства обработки документов будут использоваться специализированные библиотеки PyPDF2 и python-docx, поддерживающие чтение содержимого из .pdf и .docx соответственно.

# Реализация и оценка качества МК

После завершения этапа проектирования, каждый из АК был запущен несколько раз на разных объёмах НД, чтобы выявить зависимость точности от размера обучающего массива.

С помощью скрипта (приложение 5) был получен следующий результат запусков каждого АК на разных размерах исходного НД:

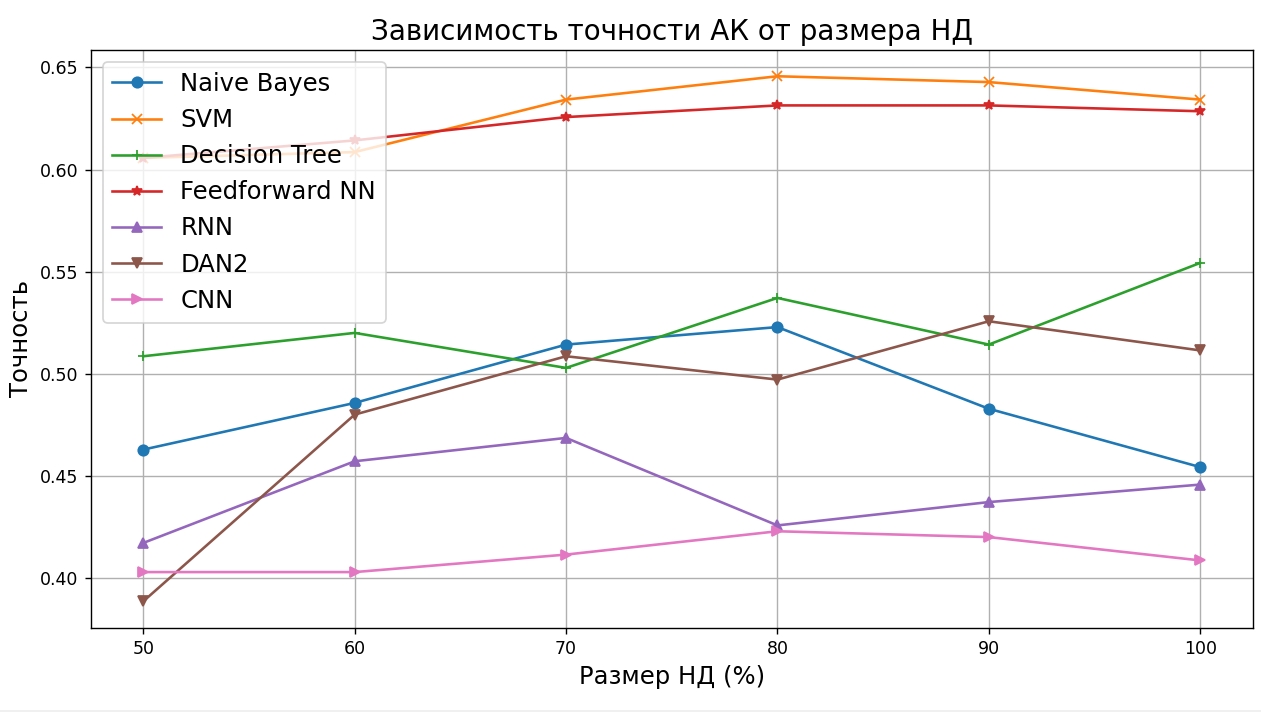


Рис. 1 - график зависимости точности АК от размера НД

Где по вертикали указана точность модели для каждого АК (при ГП по умолчанию), а по горизонтали указан процент от максимального сбалансированного НД, полученного ранее.

Или в числовом представлении:

NB

Accuracy: 0.3915

Accuracy: 0.4112

Accuracy: 0.4207

Accuracy: 0.4121

Accuracy: 0.4277

Accuracy: 0.4388

SVM

Accuracy: 0.6764

Accuracy: 0.6770

Accuracy: 0.6827

Accuracy: 0.6840

Accuracy: 0.6885

Accuracy: 0.6910

DT

Accuracy: 0.5784

Accuracy: 0.5746

Accuracy: 0.6012

Accuracy: 0.5996

Accuracy: 0.6110

Accuracy: 0.6212

FFBP

Accuracy: 0.6865

Accuracy: 0.6992

Accuracy: 0.7046

Accuracy: 0.6999

Accuracy: 0.6964

Accuracy: 0.7075

RNN

Accuracy: 0.4946

Accuracy: 0.5127

Accuracy: 0.4997

Accuracy: 0.5114

Accuracy: 0.5393

Accuracy: 0.5493

DAN2

Accuracy: 0.5936

Accuracy: 0.5949

Accuracy: 0.5761

Accuracy: 0.5688

Accuracy: 0.5739

Accuracy: 0.5746

CNN

Accuracy: 0.5536

Accuracy: 0.5451

Accuracy: 0.5435

Accuracy: 0.5520

Accuracy: 0.5514

Accuracy: 0.5555

Оценка показала, что ни одна из МК пока не достигла желаемого уровня точности (даже с учётом погрешности в 5%, обусловленной тем, что данные для неполного НД выбираются случайным образом из полного НД), необходимого для практической эксплуатации. Следовательно, необходим дополнительный этап оптимизации каждой из АК.

# Оптимизация АК

Далее будет рассмотрен каждый АК по отдельности и методы оптимизации его работы для увеличения его точности (с помощью информации из [14, 15, 18, 23, 25, 26, 27, 28]):

**NB:**

1. ГП:

У алгоритма Naive Bayes есть несколько ГП:

1. alpha - параметр сглаживания Лапласа, который используется для предотвращения проблемы нулевых вероятностей. Чем больше значение alpha, тем сильнее влияние сглаживания (по умолчанию оно равно 1.0).
2. fit\_prior - логический параметр, определяющий, нужно ли априорные вероятности класса оценивать на основе обучающих данных (True) или считать равновероятными классы (False)(по умолчанию установлено True).
3. class\_prior - пользовательские априорные вероятности классов. (по умолчанию равен None, что означает использование эмпирических частот встречаемости классов в обучающем наборе).

Значительное влияние на точность в данном случае будет оказывать только ГП alpha. Будет произведено несколько запусков с использованием различных значений alpha.

1. Архитектура:

Архитектурные изменения для улучшения точности очень незначительно влияют на результаты, поскольку Naive Bayes уже предполагает независимость признаков, что делает его менее чувствительным к архитектуре обработки признаков. Поэтому изменений в архитектуре для этого АК не будет.

1. Признаки:

Использование би- или триграмм вместо униграмм может лучше отразить контекст и улучшить точность классификации.

Будет произведено несколько запусков с использованием n-грамм.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

Bagging. Создание нескольких экземпляров модели на случайных подвыборках данных и усреднение результатов.

Boosting. Алгоритмы типа AdaBoost, которые последовательно строят слабые классификаторы и объединяют их в один сильный.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 6), в котором был оптимизирован метод классификации NB, была получена оптимальная точность (+- 0.05):

**Accuracy NB: 0.5492**

**SVM:**

1. ГП:

У алгоритма Support Vector Machine есть несколько ГП:

C: Регуляризационный параметр, который контролирует баланс между точностью классификации и обобщённой способностью модели. Большие значения C делают модель более склонной к переобучению, тогда как меньшие значения C могут приводить к недоучиванию.

kernel: Тип ядра, используемого для преобразования признаков. Возможные варианты включают:

linear: Линейное ядро, подходящее для линейно разделяемых данных.

poly: Полиномиальное ядро, которое может справляться с нелинейностью.

rbf: Радиальная базисная функция (Gaussian kernel), которая часто применяется для нелинейных данных.

gamma: Параметр ядра, важный для ядер rbf и poly. Управляет степенью влияния отдельных точек на результат. Высокие значения gamma приводят к тому, что точки ближе к границе решения имеют большее влияние, тогда как низкие значения увеличивают радиус влияния.

degree: Степень полинома для полиномиальных ядер.

Значительное влияние на точность в данном случае будут оказывать только ГП C, kernel. Будет произведено несколько запусков с использованием различных значений этих ГП.

1. Архитектура:

Поскольку SVM уже является мощным классификатором, особенно для линейно разделимых данных, значительные архитектурные изменения могут не принести большого выигрыша.

Поэтому изменений в архитектуре для этого МК не будет.

1. Признаки:

Для SVM качество признаков играет ключевую роль, но так как используемый НД уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), а дополнительные признаки в данном случае не помогут, изменений в признаках для этого МК не будет.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

Bagging. Объединение нескольких экземпляров SVM, обучаемых на случайно выбранных поднаборах данных, может снизить дисперсию модели.

Stacking. Использование SVM в составе стекинга, где его выходы комбинируются с результатами других классификаторов.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 7), в котором был оптимизирован метод классификации SVM, была получена оптимальная точность (+- 0.05):

**Accuracy SVM: 0.7991**

**DT:**

1. ГП:

У алгоритма Decision Trees есть несколько ГП:

max\_depth: Максимальная глубина дерева. Ограничивая глубину, можно избежать переобучения.

min\_samples\_split: Минимальное количество образцов, необходимое для расщепления узла.

min\_samples\_leaf: Минимальное количество образцов, требуемое для создания листа.

criterion: Критерий для выбора лучшего расщепления (например, gini или entropy).

max\_features: Число признаков, рассматриваемых при поиске наилучшего расщепления.

Значительное влияние на точность в данном случае будут оказывать только ГП max\_depth, min\_samples\_split, min\_samples\_leaf, criterion. Будет произведено несколько запусков с использованием различных значений этих ГП.

1. Архитектура:

Поскольку DT уже является мощным классификатором, особенно для линейно разделимых данных, значительные архитектурные изменения могут не принести большого выигрыша.

1. Признаки:

Для DT качество признаков играет ключевую роль, но так как используемый НД уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), а дополнительные признаки в данном случае не помогут, изменений в признаках для этого МК не будет.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.   
Также будет добавлена проверочная выборка для настройки ГП и предотвращения переобучения.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

Random Forest: Этот метод создает лес из множества деревьев решений, каждое из которых обучено на случайной подвыборке признаков и данных. Результат определяется голосованием деревьев.

Gradient Boosting Machines (GBM): Метод градиентного бустинга создает последовательность деревьев, каждое из которых исправляет ошибки предыдущего.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 8), в котором был оптимизирован метод классификации DT, была получена оптимальная точность (+- 0.05):

**Accuracy DT: 0.6343**

**FFBP:**

1. ГП:

У алгоритма Feed Forward Back Propagation есть несколько ГП:

hidden\_layer\_sizes: Размеры скрытых слоев. Определите количество нейронов в каждом слое.

activation: Функция активации для скрытых слоев. Популярные варианты: relu, sigmoid, tanh.

solver: Алгоритм оптимизации. Варианты: sgd, adam, lbfgs.

alpha: Коэффициент регуляризации L2.

learning\_rate\_init: Начальная скорость обучения.

max\_iter: Максимальное количество итераций обучения.

Значительное влияние на точность в данном случае будут оказывать только ГП hidden\_layer\_sizes, activation, alpha, learning\_rate\_init. Будет произведено несколько запусков с использованием различных значений этих ГП.

1. Архитектура:

Добавление дополнительных скрытых слоев: Увеличение глубины сети может улучшить способность моделирования сложных взаимосвязей.

Использование функций активации: Будут опробованы разные функции активации, такие как ReLU, sigmoid или tanh.

Регуляризация: Добавление регуляризации, такой как dropout или L2-нормализацию, может помочь предотвратить переобучение.

1. Признаки:

Для FFBP качество признаков играет ключевую роль, но так как используемый НД уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), а дополнительные признаки в данном случае не помогут, изменений в признаках для этого МК не будет.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

Bagging. Создание нескольких экземпляров модели на случайных подвыборках данных и усреднение результатов.

Boosting. Алгоритмы типа AdaBoost, которые последовательно строят слабые классификаторы и объединяют их в один сильный.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 9), в котором был оптимизирован метод классификации FFBP, была получена оптимальная точность (+- 0.05):

**Accuracy FFBP: 07391**

**RNN:**

1. ГП:

У алгоритма Recurrent neural network есть несколько ГП:

units: Количество нейронов в рекуррентном слое (например, LSTM или GRU).

optimizer: Оптимизатор для обучения сети (например, Adam, RMSprop).

epochs: Количество эпох обучения.

batch\_size: Размер мини-пакетов данных, подаваемых на обучение.

dropout: Уровень dropout для предотвращения переобучения.

recurrent\_dropout: Уровень dropout для рекуррентных связей.

Значительное влияние на точность в данном случае будут оказывать только ГП units, epochs, batch\_size, dropout. Будет произведено несколько запусков с использованием различных значений этих ГП.

1. Архитектура:

Word embeddings: Cпособы представления слов в виде плотных векторов, содержащие смысловую близость между ними. Они используются для эффективного извлечения признаков из текста.

Будет произведено несколько запусков с использованием Word embeddings.

1. Признаки:

Одним из эффективных способов повышения точности RNN является использование более продвинутых типов рекуррентных блоков, таких как:

Long Short-Term Memory (LSTM): Этот блок способен запоминать долгосрочные зависимости, что идеально подходит для задач обработки естественного языка.

Gated Recurrent Units (GRU): Упрощённая версия LSTM, которая также способна обрабатывать длинные временные ряды.

В данном случае лучше выбрать LSTM, так как GRU лучше подходит для коротких данных или в случаях, когда ресурсоемкость важнее точности.

Будет произведено несколько запусков с использованием LSTM.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

Bagging. Объединение нескольких экземпляров RNN, обучаемых на случайно выбранных поднаборах данных, может снизить дисперсию модели.

Stacking. Использование RNN в составе стекинга, где его выходы комбинируются с результатами других классификаторов.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 10), в котором был оптимизирован метод классификации RNN, была получена оптимальная точность (+- 0.05):

**Accuracy RNN: 0.4943**

**DAN2:**

1. ГП:

У алгоритма Dynamic Artificial Neural Network есть несколько ГП:

embedding\_dims: Размерность эмбеддингов слов.

hidden\_units: Количество нейронов в среднем слое.

dropout\_rate: Вероятность отключения нейронов для предотвращения переобучения.

Будет произведено несколько запусков с использованием различных значений всех этих ГП.

1. Архитектура:

Добавление дополнительного среднего слоя: Добавление промежуточного полного связного слоя может помочь сети лучше усваивать сложную структуру данных.

Регуляризация: Использование регуляризации L2 или dropout для контроля переобучения.

1. Признаки:

Использование предварительно обученных эмбеддингов: Замена случайных эмбеддингов на предварительно обученные (например, GloVe или Word2Vec) поможет сети сразу воспользоваться богатством готового представления слов.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Следующие техники помогают уменьшить дисперсию и смещение модели, улучшая общую точность предсказания:

* Stacking: Можно использовать DAN2 в стеке с другими моделями (например, SVM или Random Forest), предоставляющими дополнительные признаки.
* Voting Ensemble: Соединить DAN2 с другими нейронными сетями (например, CNN или RNN) и агрегировать их выводы посредством голосования.

Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 11), в котором был оптимизирован метод классификации DAN2, была получена оптимальная точность (+- 0.05):

**Accuracy DAN2: 0.5329**

**CNN:**

1. ГП:

У алгоритма есть несколько ГП:

filters: Количество фильтров в свёртке.

kernel\_size: Размер окна свертки.

pool\_size: Размер максимального пулинга.

dropout: Вероятность отбрасывания нейронов для предотвращения переобучения.

batch\_size: Размер пакета.

epochs: Количество эпох обучения.

Будет произведено несколько запусков с использованием различных значений всех этих ГП.

1. Архитектура:

Для CNN качество признаков играет ключевую роль, но так как используемый НД уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), а дополнительные признаки в данном случае не помогут, изменений в признаках для этого МК не будет.

1. Признаки:

Для CNN качество признаков играет ключевую роль, но так как используемый НД уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), а дополнительные признаки в данном случае не помогут, изменений в признаках для этого МК не будет.

1. Валидация:

Будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок.

1. Ансамблирование:

Ансамбли могут значительно улучшить точность CNN:

* Bagging: Объединение нескольких CNN, обученных на разных подмножествах данных.
* Stacking: Использование CNN в качестве базовой модели в ансамбле с другими моделями (например, SVM или Random Forest).

Эти ансамбли помогут повысить точность и устойчивость модели. Будет произведено несколько запусков с использованием этих техник.

С помощью скрипта (приложение 12), в котором был оптимизирован метод классификации CNN, была получена оптимальная точность (+- 0.05):

**Accuracy CNN: 0.5100**

# Повторная оценка качества МК

Результаты оптимизации в графическом представлении:

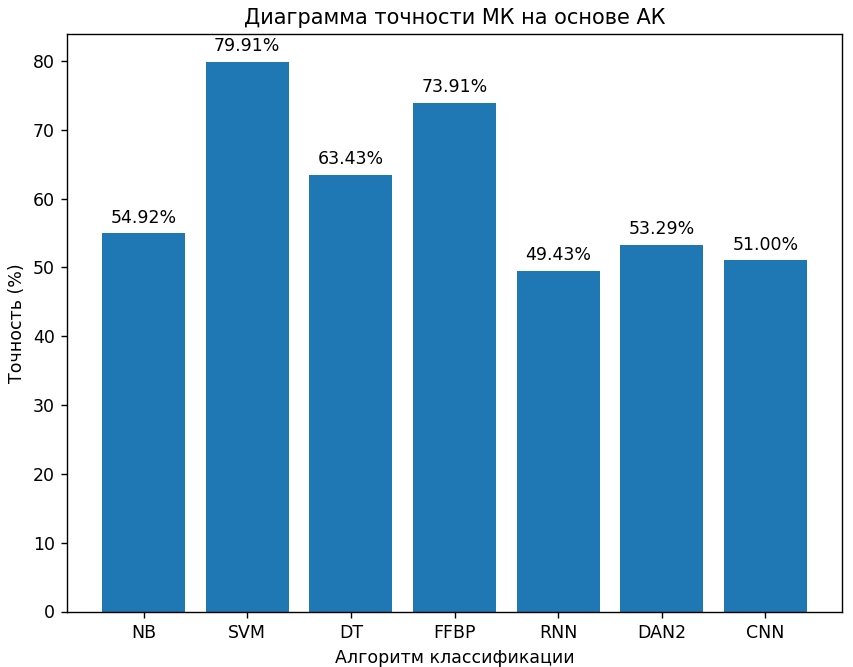


Рис. 2 - Диаграмма точности МК на основе АК

Теперь (особенно с учётом погрешности в 5%, обусловленной тем, что данные для неполного НД выбираются случайным образом из полного НД), МК на основе SVM является достаточно точной для классификации текстов по жанрам литературы.

# Интеграция МК в ПО

Теперь необходимо разработать ПО, определяющее жанр литературы загруженного пользователем текста или документа(ов), на основе полученной МК. (приложения 13, 14, 15)

**Внешний вид и структура приложения:**

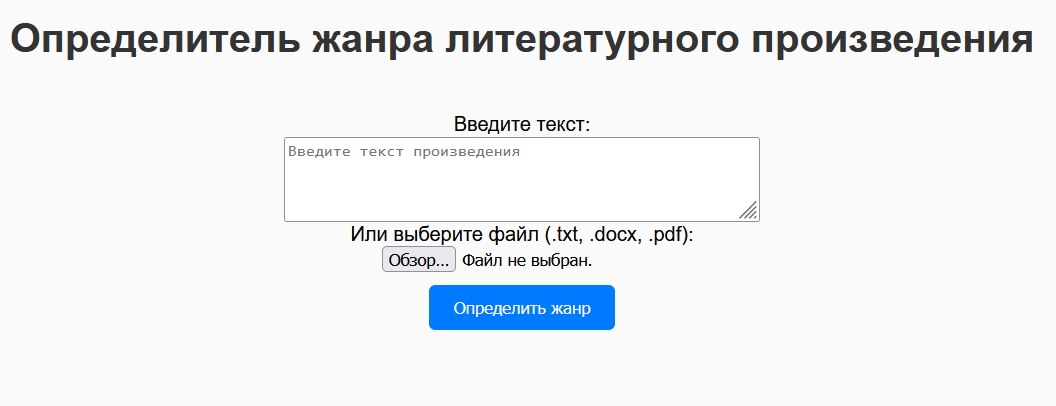


Рис. 3 - Вид главной страницы приложения

Главная страница: представлена простым, лаконичным оформлением с двумя основными элементами взаимодействия:

Поле для ввода текста: позволяет пользователям ввести фрагмент произведения непосредственно в специальное текстовое поле.

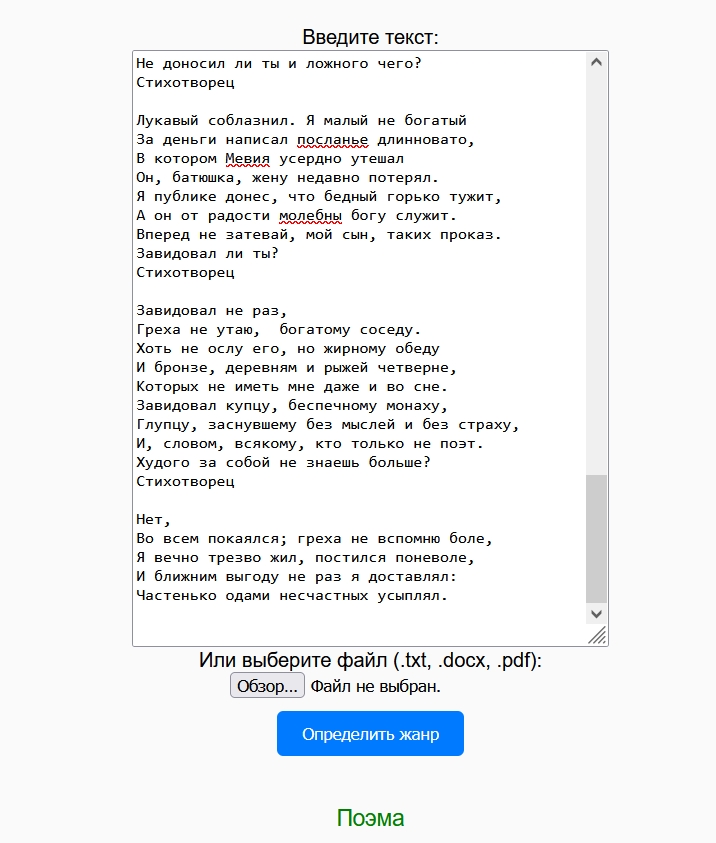


Рис. 4 - Определение жанра произведения по введённому тексту

Кнопка выбора файла: даёт возможность загрузить готовый текстовый файл в форматах .txt, .docx или .pdf. Эта опция полезна для тех случаев, когда произведение представлено в готовом документе.

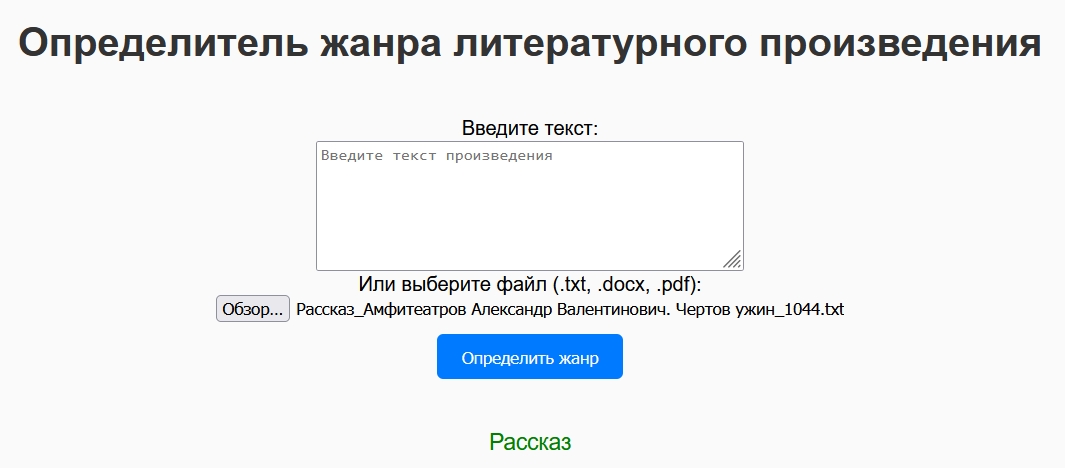


Рис. 5 - Определение жанра произведения по выбранному файлу

Логика поведения: приложение вначале проверяет, заполнено ли текстовое поле. Если да, текст берётся оттуда. Если поле пустое, начинается обработка выбранного файла.

Результат: после обработки пользователь получает мгновенный ответ с указанием предполагаемого жанра произведения, будь то роман, повесть, рассказ, поэма, пьеса, статья или очерк.

**Особенности реализации:**

Удобство использования: отсутствие сложных элементов управления делает взаимодействие быстрым и комфортным. Пользователю достаточно заполнить одно из полей и нажать кнопку "Определить жанр".

Поддерживаемые форматы: поддержка популярных форматов текстовых файлов - .txt, .docx, .pdf обеспечивает максимальную совместимость с разными источниками данных.

Автоматическая адаптация: приложение способно автоматически определять наиболее подходящую кодировку текста, устраняя возможные ошибки декодирования.

Быстрая реакция: благодаря использованию AJAX-запросов и асинхронному взаимодействию с сервером, пользователю не приходится ждать перезагрузки страницы после каждой операции.

# Документация

Ниже приведена полная документация по приложению для классификации жанров литературы. Она включает руководство пользователя и подробное техническое описание архитектуры.

**Документация приложения «Классификация жанров литературы».**

**Описание приложения:**

Приложение предназначено для автоматического определения жанра литературного произведения путём анализа его текста. Используя продвинутые методы машинного обучения, оно позволяет пользователям определить жанр любого текста, введённого вручную или загруженного в виде файла.

**Руководство пользователя.**

1. Назначение приложения:

«Классификация жанров литературы» предназначена для помощи пользователям в определении жанра конкретного произведения на основании его содержания. Программа способна распознавать следующие жанры:

* Роман
* Повесть
* Рассказ
* Поэма
* Пьеса
* Статья
* Очерк

1. Начало работы:

Перейдите на главную страницу приложения.

Выберите один из способов ввода текста:

Напишите текст произведения вручную в специальном текстовом поле.

Или загрузите файл в одном из поддерживаемых форматов: .txt, .docx, .pdf.

Нажмите кнопку «Определить жанр».

Результат появится ниже формы.

1. Примечания:

Если текст задан вручную, файл игнорируется.

Форматирование текста не влияет на итоговую оценку.

Рекомендуемый минимальный объем текста для точного определения жанра — около 500–1000 знаков.

**Архитектура приложения.**

1. Общая архитектура:

Архитектура приложения состоит из трёх ключевых компонентов:

Клиентская сторона:

Представляет собой веб-интерфейс, построенный на HTML, CSS и JavaScript.

Позволяет пользователю вводить текст или выбирать файл.

Отправляет запросы на сервер через AJAX, гарантируя быструю реакцию без перезагрузок страниц.

Серверная сторона:

Реализован на платформе Flask — легковесном Python-фреймворке.

Отвечает за прием и обработку HTTP-запросов, отправленных клиентом.

Содержит бизнес-логику для разбора файлов и формирования векторов признаков.

Машинное обучение (МК):

Модель классификации реализована на основе подхода TF-IDF (частота термина-обратная частота документа) и Support Vector Machine (метода опорных векторов).

Используется библиотека Scikit-learn для предобработки текста и обучения модели.

1. Детали технического устройства:

Клиентская сторона:

Формат ввода: пользовательские данные принимаются в виде строки (введённой вручную) или бинарного потока (загружаемого файла).

Интерфейс: строится с применением стандартного набора HTML-тегов и стилей CSS, улучшенных скриптами jQuery для асинхронного взаимодействия с сервером.

Отображение результата: результат возвращается в виде простого сообщения поверх формы ввода.

Серверная сторона:

Обработчики маршрутов: главные маршруты контролируются с помощью функций декоратора Flask (@app.route).

Алгоритм анализа: последовательность операций следующая:

Получение данных от клиента (текст или файл).

Анализ типа данных (рукописный текст или загружаемый файл).

Конвертирование текста в вектор признаков (TF-IDF).

Применение ML-модели для предсказания жанра.

Возвращение результата клиенту.

Машинное обучение:

Модуль предобработки: библиотека Scikit-learn используется для создания вектора признаков текста методом TF-IDF.

Классификатор: применяется SVM-классификатор, выбранный за высокую производительность и стабильность на множестве текстовых задач.

Гиперпараметры: настройка оптимальной производительности достигается за счёт подбора количества слоев, оптимизатора и скорости обучения.

1. Процесс обработки:

Процесс обработки запроса пользователя делится на три этапа:

Сбор данных: получение текста (либо из рукописного ввода, либо из файла).

Предобработка: создание вектора признаков на основе метода TF-IDF.

Классификация: выполнение предсказания с помощью обученной SVM-модели.

1. Безопасность:

Входящие данные проходят проверку безопасности с целью предотвращения атак типа SQL-инъекции или Cross-site scripting (XSS).

Все данные сохраняются временно и удаляются после завершения сессии.

**Заключение.**

Документ описывает полное руководство пользователя и детальное техническое описание архитектуры приложения для классификации жанров литературы. Следуя инструкциям, любой пользователь сможет эффективно применять этот инструмент для анализа своего материала.

# Способы возможной модификации ПО

Предлагаемые способы модернизации ПО позволяют расширить функциональные возможности приложения, адаптироваться к новым требованиям и улучшать качество классификации.

Ключевые направления возможной модификации:

1. **Добавление новых жанров.**

В настоящий момент система поддерживает 7 жанров. Расширив набор жанров, можно сделать систему более универсальной и применимой для большего числа ситуаций.

Метод реализации:

Собрать новый корпус данных для каждого из новых жанров.

Отформатировать новые НД и сбалансировать их со старыми НД.

Дополнить общий НД новыми примерами.

Повторно обучить МК с увеличенным набором жанров.

Протестировать новую модель на валидных данных.

Пример:

В моей реализации всё выглядит очень просто:



Рис. 6 - вид НД

Есть набор папок, в каждой из которых лежат произведения отдельных жанров. Скрипт для создания МК проходит по каждой и с последующей обработкой:

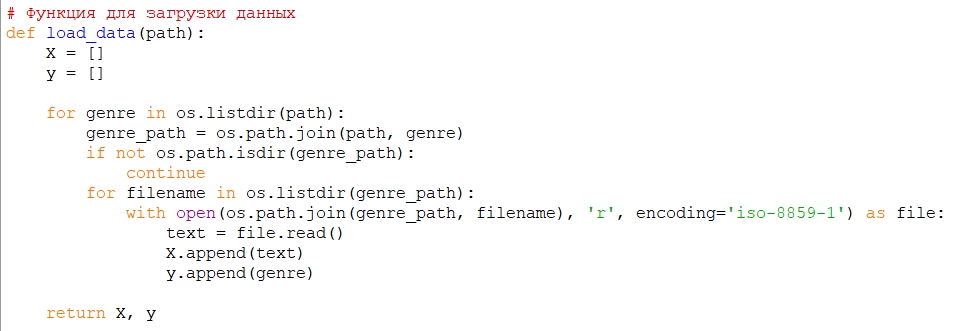


Рис. 7 - функция для загрузки данных (приложение 7)

То есть, чтобы добавить новый жанр нужно добавить в директорию с остальными папками новую и пересоздать МК.

1. **Добавление новых расширений текстовых файлов.**

В текущей версии поддерживается работа с файлами .txt, .docx и .pdf. Можно добавить поддержку других распространённых форматов, таких как .epub, .odt, .rtf.

Метод реализации:

Установить соответствующие библиотеки для обработки нужных форматов (например, ebooklib для .epub).

Интеграция новых механизмов парсинга файлов в существующую структуру.

Адаптация механизма выбора файла на стороне клиента.

Пример:

Для расширения файла .epub:

Шаги для интеграции:

Установить библиотеку ebooklib:

pip install ebooklib

Добавить в код (приложение 13) специальный обработчик для формата .epub:

elif filename.endswith('.epub'):

from ebooklib import epub

book = epub.read\_epub(file.stream)

all\_text = []

for item in book.get\_items\_of\_type(epub.ITEM\_DOCUMENT):

all\_text.append(item.get\_body\_content().decode('utf-8'))

return ''.join(all\_text)

1. **Возможность переобучения на новых данных.**

Переобучение позволяет регулярно обновлять модель, учитывая современные тенденции в литературе и новых авторов. Система должна поддерживать процедуру обновления без полной замены текущего решения.

Метод реализации:

Регулярно проводить переобучение с добавлением новых примеров.

Хранить предыдущие модели для возможного отката в случае неудачи.

Пример:

В моей реализации всё выглядит очень просто:



Рис. 8 - вид НД

Есть набор папок, в каждой из которых лежат произведения отдельных жанров. Скрипт для создания МК проходит по каждой и с последующей обработкой:

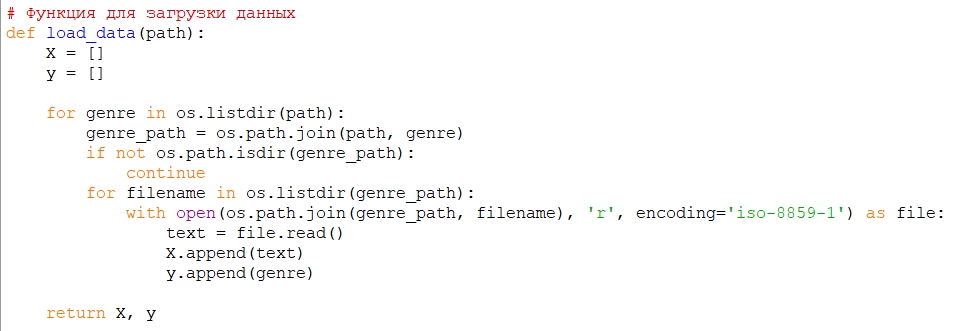


Рис. 9 - функция для загрузки данных (приложение 7)

То есть, чтобы обновить НД нужно добавить новые файлы произведений каждого жанра (с сохранением сбалансированности) в директорию в каждую папку жанров и пересоздать МК.

1. **Другие потенциальные улучшения.**

Кроме перечисленных выше направлений, возможны и другие полезные усовершенствования:

Многопоточная обработка: ускорение работы с большим количеством запросов.

Мониторинг и сбор статистики: отслеживание успешности определений жанров, частоты обращений и проблемных сценариев.

Оптимизация памяти: уменьшение нагрузки на память при работе с большими наборами данных.

Использование облачных сервисов: масштабирование вычислительных ресурсов для обработки большого объёма данных.

Предлагаемые подходы обеспечивают дальнейшее развитие и совершенствование ПО.

# Заключение

Данная работа посвящена актуальной и востребованной задаче — автоматической классификации текстов по жанрам литературы с применением ММО. Созданная система решает проблему эффективной и быстрой классификации большого количества текстов, избавляя сотрудников библиотек, книжных магазинов и исследователей от трудоёмкой ручной работы.

Работа проведена поэтапно, начиная с анализа предметной области и постановки задачи, через разработку дизайна системы и реализацию самих МК, до их оценки и интеграции в готовое ПО. Исследования показали, что разработанная МК обеспечивает высокий уровень точности и быстродействие, позволяя качественно решать поставленную задачу.

Важнейшие результаты работы заключаются в следующем:

* разработана надёжная система классификации текстов по жанрам,
* обеспечена возможность быстрого внесения изменений и переобучения МК,
* созданы условия для удобной интеграции в цифровые библиотеки и аналогичные системы.

Проведённые исследования подтвердили обоснованность и целесообразность применения предложенных методов и показали значительный потенциал для практического применения системы. Её достоинства выражаются в простоте эксплуатации, поддержке множественных форматов файлов и хорошей масштабируемости.

В дальнейшем предстоит развивать данную систему, увеличивая число поддерживаемых жанров, повышая точность и ускоряя процессы обработки.

# Список использованных источников

1. Ермакова Л.М., Абашев М.А., Никитин Р.В., Ушаков Р.И.: Методы автоматической классификации текстов по функциональным стилям. - 6 стр. - Журнал: Вестник Пермского университета - 2014 г.
2. Панг П., Ли Л.: Opinion Mining and Sentiment Analysis. - 135 стр. - Foundation and Trends in Information Retrieval - 2008 г.
3. Браславский П.: Морфологический строй функциональных стилей. - 8 стр. - Журнал: Известия Уральского государственного университета - 2001 г.
4. Емашова О.А., Мальковский М.Г.: Функциональные стили русского языка и их влияние на задачу автоматического реферирования текста 5 - стр. - Журнал: Диалог - 2007 г.
5. Лифшиц Ю.: Классификация текстов. - Internet Resource - 2005 г.
6. Statamatos E., Fakotakis N., Kokkinakis G.: Automatic text categorization in terms of genre and author. - 25 стр. - Computational Linguistics - 2000 г.
7. Petrenz P., Webber B.: Stable classification of text genres. - 8 стр. - Computational Linguistics - 2008 г.
8. Шевелев О.Г., Петраков А.В.: Классификация текстов с помощью деревьев решений и нейронных сетей прямого распространения. - 290 стр. - Журнал: Вестник Томского государственного университета - 2006 г.
9. Дементьев В.Е., Киреев С.Х.: Выбор алгоритмов машинного обучения для классификации текстовых документов. - 31 стр. - Журнал: Техника средств связи - 2022 г.
10. Дементьев В.Е., Чулков А.А.: Метод автоматизированной идентификации признаков протоколов сетей передачи данных. - 8 стр. - Журнал: Информация и космос - 2021 г.
11. Дементьев В.Е., Чулков А.А.: Модель протокола сети передачи данных в условиях деструктивных кибернетических воздействий. Часть 1. - 7 стр. - Журнал: Защита информации. Инсайд. - 2021 г.
12. Дементьев В.Е., Чулков А.А.: Модель протокола сети передачи данных в условиях деструктивных кибернетических воздействий. Часть 2. - 9 стр. - Журнал: Защита информации. Инсайд. - 2021 г.
13. Дементьев В.Е., Чулков А.А.: Методика оценки защищенности протоколов сети передачи данных в условиях деструктивных кибервоздействий. - 11 стр. - Журнал: Известия Тульского государственного университета. Технические науки. - 2007 г.
14. Sebastiani F.: Machine learning in automated text categorization. - 47 стр. - ACM Computing Surveys - 2002 г.
15. Andreas H.: A Brief Survey of Text Mining. - Internet - 2005 г.
16. Cheng H., Soon Cheol P.: Combination of modified BPNN algorithms and an efficient feature selection method for text categorization.. - 12 стр. - Information Processing and Management 45 - 2009 г.
17. Ming Li, Hang Li, Zhi-Hua Z.,: Semi-supervised document retrieval. - 15 стр. - Information Processing and Management 45 - 2009 г.
18. Monica Rogati, Yiming Yang: High-Performing Feature Selection for Text Classification. - 9 стр. - CIKM'02 - 2009 г.
19. Т.В. Батура: Методы автоматической классификации текстов. - 14 стр. - Международный журнал: Программные продукты и системы - 2017 г.
20. Н.Н. Буйлова: Автоматизация обработки текста. - 5 стр. - Журнал: Информационные процессы и системы - 2018 г.
21. Голубева И.Б.: Стилистика русского языка. - 448 стр. - Рольф - 1999 г.
22. Мангалова Е.С., Агафонов Е.Д.: О проблеме выделения информативных признаков в задаче классификации текстовых документов. - Журнал: Вестник Том. гос. ун-та. Управление, вычислительная техника и информатика. - 2013 г.
23. Скляренко Н.С.: Обзор алгоритмов машинного обучения, решающих задачу обнаружения спама. - 5 стр. - Журнал: Новые информационные технологии в автоматизированных системах. - 2017 г.
24. Епрев А.С.: Автоматическая классификация текстовых документов. - 5 стр. - Журнал: Математические структуры и моделирование. - 2010 г.
25. Борисов Л.А., Орлов Ю.Н., Осминин К.П.: Идентификация автора текста по распределению частот буквосочетаний. - 13 стр. - Журнал: Прикладная информатика - 2013 г.
26. Сухарева А.В., Царьков С.В.: Классификация научных текстов по отраслям знаний. - 3 стр. - Журнал: Машинное обучение и анализ данных. - 2014 г.
27. Кубарев А.И., Кукушкина О.В., Поддубный В.В. и др.: Построение таблиц стилей текстовых произведений с использованием алгоритмов классификации на основе деревьев решений. - 10 стр. - Журнал: Вестн. Томск. гос. ун-та. Сер. Управление, вычислительная техника и информатика. - 2012 г.
28. Веретенников И., Карташев Е., Царегородцев А.: Оценка качества классификации текстовых материалов с использованием алгоритма машинного обучения «Случайный лес». - 11 стр. - Журнал: Известия АлтГУ - 2017 г.
29. Интернет-библиотека Алексея Комарова - https://ilibrary.ru/
30. Интернет-библиотека Максима Мошкова - https://lib.ru/

# Приложения

1. **Приложение 1:**

For\_diploma\_script\_collector\_1.py:

# Импортируем необходимые библиотеки

import os

import re

import requests

# Шаблоны для извлечения текста из HTML-документа

text\_pattern = re.compile("<div id=\"text\".\*?>(.\*?)<div id=\"tbd\"", re.DOTALL)

span\_pattern2 = re.compile("<\/span>([^<].+?)<\/span")

span\_pattern = re.compile("<\/span>(.+)<\/span")

strip\_pattern = re.compile("&.\*?;")

strip\_pattern2 = re.compile("<.\*?>")

# Регулярное выражение для поиска всех специальных символов

special\_chars\_pattern = re.compile(r'[^\w\s-]')

# Функция для очистки текста книги от специальных символов

def sanitize\_text(text):

sanitized\_text = special\_chars\_pattern.sub('', text)

sanitized\_text = ' '.join(sanitized\_text.split())

return sanitized\_text

# Шаблон для поиска заголовка книги

title\_pattern = re.compile(r'<div class="title">\s\*<h1>(.\*?)</h1>', re.DOTALL)

# Шаблон для поиска автора книги

author\_pattern = re.compile(r'<div class="author">(.\*?)</div>', re.DOTALL)

# Функция для очистки строк от HTML-сущностей и тегов

def strip\_of\_shit(lines):

for i in range(0, len(lines)):

lines[i] = "".join(strip\_pattern.split(lines[i]))

lines[i] = "".join(strip\_pattern2.split(lines[i]))

lines[i] = lines[i].lstrip()

return lines

# Функция для получения общей информации о книге (заголовок и автор)

def get\_info(book, sess):

r = sess.get(f"https://ilibrary.ru/text/{book}/p.1/index.html")

try:

author = strip\_of\_shit(author\_pattern.findall(r.text))[0]

except IndexError:

print(f"Не удалось найти автора {book}. Пропускаем...")

try:

title = strip\_of\_shit(title\_pattern.findall(r.text))[0]

except IndexError:

print(f"Не удалось найти название книги {book}. Пропускаем...")

return (title, author)

# Функция для получения содержимого одной страницы книги

def get\_page(book, pnum, sess):

r = sess.get(f"https://ilibrary.ru/text/{book}/p.{pnum}/index.html")

# Получение всего текста страницы

page\_text = r.text

# Обрабатываем первый вариант структуры

match\_first\_variant = re.findall('<z><o>(.+?)</o>(.+?)</z>', page\_text, flags=re.DOTALL)

if match\_first\_variant:

text\_lines = []

for match in match\_first\_variant:

text\_lines.extend(match[0].splitlines())

text\_lines.extend(match[1].splitlines())

# Обрабатываем второй и третий варианты структуры

elif '<v>' in page\_text:

text\_lines = []

# Разделяем страницу на строки, содержащие теги <v> и </v>

v\_tags = re.finditer('<v>(.\*?)</v>', page\_text, flags=re.DOTALL)

for tag\_match in v\_tags:

line = tag\_match.group(1)

# Удаляем любые внутренние теги (<s5>, <s8>, <sc> и т.п.)

cleaned\_line = re.sub(r'</?\w+>', '', line)

text\_lines.append(cleaned\_line.strip())

else:

raise ValueError("Неизвестная структура страницы.")

# Очищаем строки от HTML-сущностей и лишних символов

clean\_lines = strip\_of\_shit(text\_lines)

return clean\_lines

# Основная функция для скачивания всей книги

def get\_book(book, session):

try:

info = get\_info(book, session)

except Exception as e:

print(f"Error getting info for book {book}, {e}")

return

title = info[0]

title = sanitize\_text(title)

if title == "" or title == " ":

title = "-"

author = info[1]

author = sanitize\_text(author)

print(f"Book {book}: {title} - {author}")

# Определяем, является ли книга поэмой

is\_poem = False

i = 1

while True:

try:

p = get\_page(book, i, session)

except:

break

for s in p:

if isinstance(s, str) and 'poems.push(ge("pmt1"));' in s:

is\_poem = True

break

if is\_poem:

break

i += 1

# Формируем правильное название файла

prefix = "Поэма\_" if is\_poem else "Повесть\_"

filestring = f"books/{prefix}{author}\_{title}\_{book}.txt"

os.makedirs(os.path.dirname(filestring), exist\_ok=True)

f = open(filestring, "w")

i = 1

while True:

try:

p = get\_page(book, i, session)

except:

break

# Здесь фиксируем запись построчно

for s in p:

if isinstance(s, str):

f.write(s + "\n")

else:

for item in s:

f.write(item + "\n")

f.write("\n")

i += 1

f.close()

# Создаем сессию для запросов

session = requests.Session()

# Цикл для обработки книг с номерами от 1 до 4588

for i in range(1,4589):

get\_book(i, session)

1. **Приложение 2:**

For\_diploma\_script\_collector\_3.py:

import requests

import re

from bs4 import BeautifulSoup

import os

import time

def clean\_filename(text):

# Убираем все недопустимые символы

invalid\_chars = r'[\\/\*:?<>|"]'

cleaned\_text = re.sub(invalid\_chars, '', text)

if len(cleaned\_text) <= 220:

return cleaned\_text

else:

return 'Без названия'

# Функция для получения содержимого страницы

def get\_page\_content(url):

response = requests.get(url)

if response.status\_code == 200:

return response.text

else:

print(f"Ошибка загрузки страницы {url}. Код ошибки: {response.status\_code}")

return None

# Функция для извлечения ссылок на авторов

def extract\_author\_links(html):

soup = BeautifulSoup(html, 'html.parser')

links = []

count = 0

for link in soup.find\_all('a'):

href = link.get('href')

if href is not None and href.startswith('http://az.lib.ru/'):

if count == 0:

count = count + 1

else:

author\_link = f'{href}'

#print(author\_link)

links.append(author\_link)

return links

# Функция для извлечения ссылок на произведения конкретного автора

def extract\_work\_links(author\_link, author\_html):

soup = BeautifulSoup(author\_html, 'html.parser')

works = []

for work\_link in soup.find\_all('a', href=True):

href = work\_link.get('href')

if href.endswith('.shtml') and href.startswith('text'):

work\_url = f'{author\_link}{href}'

#print(work\_url)

works.append(work\_url)

return works

# Функция для сохранения текста произведения в файл

def save\_text\_to\_file(text, filename):

with open(filename, 'w', encoding='utf-8') as file:

file.write(text)

# Функция для извлечения жанра произведения

def extract\_genre(soup):

# Находим первую ссылку, содержащую путь, начинающийся с '/type/index\_type\_...'

genre\_link = soup.find('a', href=lambda x: x and x.startswith('/type/index\_type\_'))

if genre\_link:

genre = genre\_link.get\_text(strip=True)

return genre

return None

# Функция для извлечения названия произведения

def extract\_title(soup):

# Находим тег <title> и извлекаем текст после второй точки

title\_tag = soup.find('title')

if title\_tag:

title\_text = title\_tag.text.strip()

parts = title\_text.split(':', maxsplit=1)

if len(parts) > 1:

return parts[1].strip()

return None

# Проверка соединения с сервером

def check\_server\_availability(url):

try:

response = requests.get(url, timeout=5)

if response.status\_code == 200:

return True

else:

return False

except requests.exceptions.RequestException:

return False

# Основная функция парсинга

def parse\_authors\_and\_works():

# URL главной страницы с авторами

base\_url = 'https://lib.ru/LITRA/'

# Проверяем доступность сервера перед началом парсинга

if not check\_server\_availability(base\_url):

print("Сервер недоступен. Парсинг невозможен.")

return

# Получаем содержимое главной страницы

main\_page\_html = get\_page\_content(base\_url)

# Извлекаем ссылки на авторов

author\_links = extract\_author\_links(main\_page\_html)

# Создаем общую папку для всех книг

os.makedirs('books', exist\_ok=True)

j = 1

# Проходимся по каждому автору

for i, author\_link in enumerate(author\_links):

author\_name = author\_link.split('/')[-1].replace('\_', ' ')

# Получаем страницу автора

author\_html = get\_page\_content(author\_link)

# Извлекаем ссылки на произведения автора

work\_links = extract\_work\_links(author\_link, author\_html)

# Проходимся по каждому произведению

for work\_link in work\_links:

while True:

# Проверяем доступность сервера перед каждым произведением

if check\_server\_availability(base\_url):

break

else:

print("Сервер временно недоступен. Жду 10 секунд...")

time.sleep(10)

# Получаем содержание произведения

work\_html = get\_page\_content(work\_link)

# Проверяем, что страница загружена успешно

if work\_html is None:

print(f"Произведение не найдено: {work\_link}")

continue # Переходим к следующему произведению

# Парсим название произведения

soup = BeautifulSoup(work\_html, 'html.parser')

title = extract\_title(soup)

if not title:

title = ""

cleaned\_title = clean\_filename(title)

# Извлекаем жанр произведения

genre = extract\_genre(soup)

# Если жанр не найден, ставим 'Без жанра'

if not genre:

genre = 'Без жанра'

# Извлекаем текст произведения

text\_div = soup.find('dd')

if text\_div is not None:

text = text\_div.get\_text().strip()

else:

text = "" # Если тег <dd> не найден, присваиваем пустую строку

# Формируем имя файла

filename = f"books/{genre}\_{cleaned\_title}\_{j}.txt"

j = j + 1

# Сохраняем текст в файл

save\_text\_to\_file(text, filename)

print(f"Сохранено произведение: {filename}")

if \_\_name\_\_ == "\_\_main\_\_":

parse\_authors\_and\_works()

1. **Приложение 3:**

For\_diploma\_script\_collector\_4.py:

import requests

import re

from bs4 import BeautifulSoup

import os

import time

def clean\_filename(text):

# Убираем все недопустимые символы

invalid\_chars = r'[\\/\*:?<>|"]'

cleaned\_text = re.sub(invalid\_chars, '', text)

if len(cleaned\_text) <= 220:

return cleaned\_text

else:

return 'Без названия'

# Функция для получения содержимого страницы

def get\_page\_content(url):

response = requests.get(url)

if response.status\_code == 200:

return response.text

else:

print(f"Ошибка загрузки страницы {url}. Код ошибки: {response.status\_code}")

return None

# Функция для получения авторов и извлечения ссылок их произведения

def extract\_authors\_and\_works(page\_html):

soup = BeautifulSoup(page\_html, 'html.parser')

authors = []

dl\_tags = soup.find\_all('dl')

for dl in dl\_tags:

dt\_tag = dl.find('dt')

if dt\_tag:

# Извлечение ФИО автора

font\_tag = dt\_tag.find('font', attrs={'color': '#555555'})

if font\_tag:

author\_name = font\_tag.text.strip()

#print(author\_name)

else:

author\_name = ''

# Извлечение ссылки на произведение и его названия

first\_a\_tags = dt\_tag.find\_all('a')

if first\_a\_tags:

work\_link = first\_a\_tags[1]['href'] # Вторая ссылка <a>

#print(work\_link)

b\_tag = first\_a\_tags[1].find('b') # Поиск тега <b> внутри второй ссылки <a>

if b\_tag:

work\_title = b\_tag.text.strip() # Извлечение текста из тега <b>

#print(work\_title)

else:

work\_title = '' # Если тег <b> не найден

else:

work\_link = ''

work\_title = ''

# Добавление данных в список

if author\_name and work\_link and work\_title:

authors.append((author\_name, work\_link, work\_title))

return authors

# Функция для извлечения текста произведения

def extract\_prose\_text(prose\_html):

soup = BeautifulSoup(prose\_html, 'html.parser')

text\_div = soup.find('dd')

if text\_div is not None:

text = text\_div.get\_text().strip()

else:

text = "" # Если тег <dd> не найден, присваиваем пустую строку

return text

# Функция для сохранения текста произведения в файл

def save\_prose\_to\_file(text, filename):

with open(filename, 'w', encoding='utf-8') as file:

file.write(text)

# Проверка соединения с сервером

def check\_server\_availability(url):

try:

response = requests.get(url, timeout=5)

if response.status\_code == 200:

return True

else:

return False

except requests.exceptions.RequestException:

return False

# Основная функция парсинга

def parse\_pages():

# URL главной страницы с авторами

base\_url = 'http://az.lib.ru/type/index\_type\_9-{num}.shtml'

# Создаем папку для хранения файлов

os.makedirs('Пьеса', exist\_ok=True)

# Проходимся по каждой странице

file\_count = 21000 # Начальный порядковый номер

for page\_num in range(1, 12):

current\_base\_url = base\_url.format(num=page\_num)

print(f"Парсим страницу: {current\_base\_url}")

# Проверяем доступность сервера перед началом парсинга

if not check\_server\_availability(current\_base\_url):

print("Сервер недоступен. Парсинг невозможен.")

return

# Получаем содержимое текущей страницы

main\_page\_html = get\_page\_content(current\_base\_url)

# Извлекаем ссылки на авторов и их произведения

authors = extract\_authors\_and\_works(main\_page\_html)

# Проходимся по каждому автору и его произведению

for author\_name, work\_link, work\_title in authors:

# Получаем текст произведения

full\_work\_link = f"http://az.lib.ru/{work\_link}"

work\_html = get\_page\_content(full\_work\_link)

prose\_text = extract\_prose\_text(work\_html)

cleaned\_author\_name = clean\_filename(author\_name)

cleaned\_work\_title = clean\_filename(work\_title)

# Формируем имя файла

filename = f"Пьеса/Пьеса\_{cleaned\_author\_name}\_{cleaned\_work\_title}\_{file\_count}.txt"

file\_count += 1

# Сохраняем текст произведения в файл

save\_prose\_to\_file(prose\_text, filename)

print(f"Сохранено произведение: {filename}")

if \_\_name\_\_ == "\_\_main\_\_":

parse\_pages()

1. **Приложение 4:**

For\_diploma\_collector\_helper\_2.py:

import os

import random

def delete\_random\_txt\_files(folder\_path, num\_to\_delete):

# Получаем список всех .txt файлов в папке

files = [f for f in os.listdir(folder\_path) if f.endswith('.txt')]

# Проверяем, есть ли файлы для удаления

if len(files) == 0:

print("Нет .txt файлов в указанной папке.")

return

# Проверяем, достаточно ли файлов для удаления

if len(files) <= num\_to\_delete:

print(f"Удалено {len(files)} файла(ов), так как больше нет.")

files\_to\_delete = files

else:

# Выбираем случайные файлы для удаления

files\_to\_delete = random.sample(files, num\_to\_delete)

# Удаление выбранных файлов

for file\_name in files\_to\_delete:

file\_path = os.path.join(folder\_path, file\_name)

try:

os.remove(file\_path)

print(f"Файл '{file\_name}' успешно удалён.")

except Exception as e:

print(f"Произошла ошибка при удалении файла '{file\_name}': {e}")

# Пример использования

folder\_path = r'C:\Users\User\AppData\Local\Programs\Python\Python310\books\_1\Статья-2100' # Укажите путь к вашей папке

num\_to\_delete = 50 # Количество файлов для удаления

delete\_random\_txt\_files(folder\_path, num\_to\_delete)

1. **Приложение 5:**

For\_diploma\_CMs.py:

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.naive\_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense, Conv1D, MaxPooling1D, GlobalAveragePooling1D

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from keras.layers import Flatten

# Функция для загрузки данных

def load\_data(path):

X = [] # список всех текстов

y = [] # список меток жанров

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

y = le.fit\_transform(y)

return X, y

# Подготовка данных и разделение на тренировочные/тестовые наборы

X, y = load\_data('books\_max\_balanced')

# Разделение на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Реализация функций для тренировки и оценки моделей

def fit\_and\_evaluate(model, X\_train, y\_train, X\_test, y\_test, vectorizer=None):

if vectorizer is None:

vectorizer = TfidfVectorizer()

# Преобразование текстов в числовые признаки

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Обучение модели

model.fit(X\_train\_vec, y\_train)

# Прогнозирование на тестовом наборе

y\_pred = model.predict(X\_test\_vec)

# Оценка точности

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.4f}')

return accuracy

# Проведение серии экспериментов для изменения размера тренировочного набора

def run\_experiments(model, vectorizer=None):

results = []

sizes = np.arange(0.5, 1.01, 0.1)

for size in sizes:

X\_train\_subset, \_, y\_train\_subset, \_ = train\_test\_split(

X\_train, y\_train, train\_size=size, random\_state=42

)

accuracy = fit\_and\_evaluate(model, X\_train\_subset, y\_train\_subset, X\_test, y\_test, vectorizer)

results.append(accuracy)

return sizes \* 100, results

#NB

nb\_model = MultinomialNB(alpha=0.001)

sizes\_nb, accuracies\_nb = run\_experiments(nb\_model)

#SVM

svm\_model = LinearSVC(max\_iter=20000)

sizes\_svm, accuracies\_svm = run\_experiments(svm\_model)

#DT

dt\_model = DecisionTreeClassifier(random\_state=42)

sizes\_dt, accuracies\_dt = run\_experiments(dt\_model)

#FFBP

ffbp\_model = MLPClassifier(hidden\_layer\_sizes=(64, 32), activation='relu', solver='adam', random\_state=42, max\_iter=1000)

sizes\_ffbp, accuracies\_ffbp = run\_experiments(ffbp\_model)

#RNN

def prepare\_rnn\_data(X\_train, X\_test, tokenizer, max\_length=100):

tokenizer.fit\_on\_texts(X\_train + X\_test)

sequences = tokenizer.texts\_to\_sequences(X\_train + X\_test)

padded\_sequences = pad\_sequences(sequences, maxlen=max\_length)

return padded\_sequences[:len(X\_train)], padded\_sequences[len(X\_train):]

def prepare\_dataset(X\_train\_padded, y\_train):

dataset = tf.data.Dataset.from\_tensor\_slices((X\_train\_padded, y\_train))

dataset = dataset.batch(32)

return dataset

tokenizer = Tokenizer(num\_words=5000)

X\_train\_padded, X\_test\_padded = prepare\_rnn\_data(X\_train, X\_test, tokenizer)

rnn\_model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

LSTM(units=64),

Dense(len(np.unique(y)), activation='softmax')

])

rnn\_model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

def run\_rnn\_experiments():

results = []

sizes = np.arange(0.5, 1.01, 0.1)

for size in sizes:

X\_train\_subset, \_, y\_train\_subset, \_ = train\_test\_split(

X\_train\_padded, y\_train, train\_size=size, random\_state=42

)

dataset = prepare\_dataset(X\_train\_subset, y\_train\_subset)

rnn\_model.fit(dataset, epochs=10, verbose=0)

y\_pred = rnn\_model.predict(X\_test\_padded) # Используем predict

y\_pred = np.argmax(y\_pred, axis=-1) # Получаем индексы классов

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.4f}')

results.append(accuracy)

return sizes \* 100, results

sizes\_rnn, accuracies\_rnn = run\_rnn\_experiments()

#DAN2

dan2\_model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

GlobalAveragePooling1D(),

Dense(len(np.unique(y)), activation='softmax')

])

dan2\_model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

def run\_dan2\_experiments():

results = []

sizes = np.arange(0.5, 1.01, 0.1)

for size in sizes:

X\_train\_subset, \_, y\_train\_subset, \_ = train\_test\_split(

X\_train\_padded, y\_train, train\_size=size, random\_state=42

)

dataset = prepare\_dataset(X\_train\_subset, y\_train\_subset)

dan2\_model.fit(dataset, epochs=10, verbose=0)

y\_pred = dan2\_model.predict(X\_test\_padded) # Используем predict

y\_pred = np.argmax(y\_pred, axis=-1) # Получаем индексы классов

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.4f}')

results.append(accuracy)

return sizes \* 100, results

sizes\_dan2, accuracies\_dan2 = run\_dan2\_experiments()

#CNN

cnn\_model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

Conv1D(filters=64, kernel\_size=3, padding='same'),

MaxPooling1D(pool\_size=2),

Flatten(),

Dense(len(np.unique(y)), activation='softmax')

])

cnn\_model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

def run\_cnn\_experiments():

results = []

sizes = np.arange(0.5, 1.01, 0.1)

for size in sizes:

X\_train\_subset, \_, y\_train\_subset, \_ = train\_test\_split(

X\_train\_padded, y\_train, train\_size=size, random\_state=42

)

dataset = prepare\_dataset(X\_train\_subset, y\_train\_subset)

cnn\_model.fit(dataset, epochs=10, verbose=0)

y\_pred = cnn\_model.predict(X\_test\_padded) # Используем predict

y\_pred = np.argmax(y\_pred, axis=-1) # Получаем индексы классов

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.4f}')

results.append(accuracy)

return sizes \* 100, results

sizes\_cnn, accuracies\_cnn = run\_cnn\_experiments()

# Постройка графиков

plt.figure(figsize=(12, 6))

plt.plot(sizes\_nb, accuracies\_nb, label='Naive Bayes', marker='o')

plt.plot(sizes\_svm, accuracies\_svm, label='SVM', marker='x')

plt.plot(sizes\_dt, accuracies\_dt, label='Decision Tree', marker='+')

plt.plot(sizes\_ffbp, accuracies\_ffbp, label='Feedforward NN', marker='\*')

plt.plot(sizes\_rnn, accuracies\_rnn, label='RNN', marker='^')

plt.plot(sizes\_dan2, accuracies\_dan2, label='DAN2', marker='v')

plt.plot(sizes\_cnn, accuracies\_cnn, label='CNN', marker='>')

plt.xlabel('Размер тренировочного набора данных (%)', fontsize=14)

plt.ylabel('Точность', fontsize=14)

plt.title('Зависимость точности классификатора от размера набора данных', fontsize=16)

plt.legend(fontsize=14)

plt.grid(True)

plt.show()

1. **Приложение 6:**

[For\_diploma\_CM\_1.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#NB

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

import numpy as np

import joblib

from sklearn.preprocessing import LabelEncoder

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

path = 'books\_max\_balanced'

X, y = load\_data(path)

y = le.fit\_transform(y)

# Использовать n% от общего числа примеров

n = 0.95

total\_samples = len(X)

num\_samples\_to\_use = int(total\_samples \* n)

best\_accuracy = 0

best\_model = None

best\_vectorizer = None

for iteration in range(100):

# Установка фиксированного seed для каждой итерации

np.random.seed(iteration + 42)

# Случайная выборка заданного количества образцов

indices = np.random.choice(total\_samples, num\_samples\_to\_use, replace=False)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.25, random\_state=iteration+42)

# Векторизация текста

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Обучаем модель Naive Bayes

nb\_model = MultinomialNB(alpha=0.001)

nb\_model.fit(X\_train\_vec, y\_train)

# Прогнозируем класс для тестового набора

y\_pred = nb\_model.predict(X\_test\_vec)

# Рассчитываем точность модели

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Iteration {iteration}: Accuracy: {accuracy:.4f}')

# Если данная модель показывает лучшую точность, запоминаем её

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_model = nb\_model

best\_vectorizer = vectorizer

# Сохраняем лучшую модель и векторизатор после завершения всех итераций

if best\_model is not None:

joblib.dump(best\_model, 'best\_nb\_model.pkl')

joblib.dump(best\_vectorizer, 'best\_tfidf\_nb\_vectorizer.pkl')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 7:**

[For\_diploma\_CM\_2.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#SVM

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import numpy as np

import joblib

from sklearn.preprocessing import LabelEncoder

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

path = 'books\_max\_balanced'

X, y = load\_data(path)

y = le.fit\_transform(y)

# Параметры выбора части данных

n = 0.8

total\_samples = len(X)

num\_samples\_to\_use = int(total\_samples \* n)

best\_accuracy = 0

best\_model = None

best\_vectorizer = None

for iteration in range(100):

# Фиксируем случайность для каждого шага (чтобы была возможность воспроизвести эксперимент)

np.random.seed(iteration + 42)

# Выбор подмножества данных

indices = np.random.choice(total\_samples, num\_samples\_to\_use, replace=False)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.2, random\_state=iteration+42)

# Создание векторизатора TF-IDF

vectorizer = TfidfVectorizer()

# Преобразуем тексты в числовые признаки

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Создаем и обучаем модель SVM

svm\_model = SVC(C=10, kernel='poly', max\_iter=5000)

svm\_model.fit(X\_train\_vec, y\_train)

# Получение предсказания на тестовых данных

y\_pred = svm\_model.predict(X\_test\_vec)

# Вычисляем точность модели

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Iteration {iteration}: Accuracy: {accuracy:.4f}')

# Если текущая модель лучше предыдущей, запоминаем её

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_model = svm\_model

best\_vectorizer = vectorizer

# После завершения циклов сохраняем лучшую модель и соответствующий ей векторизатор

if best\_model is not None:

joblib.dump(best\_model, 'best\_svm\_model.pkl')

joblib.dump(best\_vectorizer, 'best\_tfidf\_svm\_vectorizer.pkl')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 8:**

[For\_diploma\_CM\_3.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#DT

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

import joblib

from sklearn.preprocessing import LabelEncoder

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

path = 'books\_max\_balanced'

X, y = load\_data(path)

y = le.fit\_transform(y)

# Используем полный набор данных (n = 1.0)

n = 1.0

total\_samples = len(X)

num\_samples\_to\_use = int(total\_samples \* n)

best\_accuracy = 0

best\_model = None

best\_vectorizer = None

for iteration in range(100):

# Устанавливаем фиксированный seed для каждой итерации

np.random.seed(iteration + 42)

# Случайная выборка заданного количества образцов

indices = np.random.choice(total\_samples, num\_samples\_to\_use, replace=False)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.3, random\_state=iteration+42)

# Векторизация текста

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Обучаем дерево решений

dt\_model = DecisionTreeClassifier(max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, criterion="entropy")

dt\_model.fit(X\_train\_vec, y\_train)

# Прогнозируем класс для тестового набора

y\_pred = dt\_model.predict(X\_test\_vec)

# Рассчитываем точность модели

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Iteration {iteration}: Accuracy: {accuracy:.4f}')

# Если данная модель показывает лучшую точность, запоминаем её

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_model = dt\_model

best\_vectorizer = vectorizer

# Сохраняем лучшую модель и векторизатор после завершения всех итераций

if best\_model is not None:

joblib.dump(best\_model, 'best\_dt\_model.pkl')

joblib.dump(best\_vectorizer, 'best\_tfidf\_dt\_vectorizer.pkl')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 9:**

[For\_diploma\_CM\_4.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

import joblib

from sklearn.preprocessing import LabelEncoder

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

path = 'books\_max\_balanced'

X, y = load\_data(path)

y = le.fit\_transform(y)

# Количество используемого процента данных

n = 1.0

total\_samples = len(X)

num\_samples\_to\_use = int(total\_samples \* n)

best\_accuracy = 0

best\_model = None

best\_vectorizer = None

for iteration in range(100):

# Устанавливаем фиксированный seed для данной итерации

np.random.seed(iteration + 42)

# Случайная выборка заданного количества образцов

indices = np.random.choice(total\_samples, num\_samples\_to\_use, replace=False)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделяем данные на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.35, random\_state=iteration+42)

# Создание векторизатора TF-IDF

vectorizer = TfidfVectorizer()

# Преобразуем тексты в числовые признаки

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Создаем и обучаем модель FFNN (Feedforward Neural Network)

ffbp\_model = MLPClassifier(

hidden\_layer\_sizes=(64, 32),

activation='relu',

solver='adam',

alpha=0.0001,

learning\_rate\_init=0.001,

random\_state=42,

max\_iter=1000

)

ffbp\_model.fit(X\_train\_vec, y\_train)

# Предсказываем классы на тестовых данных

y\_pred = ffbp\_model.predict(X\_test\_vec)

# Вычисляем точность модели

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Iteration {iteration}: Accuracy: {accuracy:.4f}')

# Если данная модель точнее предыдущих, запоминаем её

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_model = ffbp\_model

best\_vectorizer = vectorizer

# Сохраняем лучшую модель и векторизатор после окончания всех итераций

if best\_model is not None:

joblib.dump(best\_model, 'best\_ffbp\_model.pkl')

joblib.dump(best\_vectorizer, 'best\_tfidf\_ffbp\_vectorizer.pkl')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 10:**

[For\_diploma\_CM\_5.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#RNN

import os

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from sklearn.metrics import accuracy\_score

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

X, y = load\_data('books\_max\_balanced')

y = le.fit\_transform(y)

# Подготовка данных для RNN

def prepare\_rnn\_data(X\_train, X\_test, tokenizer, max\_length=100):

tokenizer.fit\_on\_texts(X\_train + X\_test)

sequences = tokenizer.texts\_to\_sequences(X\_train + X\_test)

padded\_sequences = pad\_sequences(sequences, maxlen=max\_length)

return padded\_sequences[:len(X\_train)], padded\_sequences[len(X\_train):]

# Инициализация токенайзера

tokenizer = Tokenizer(num\_words=5000)

# Описание архитектуры RNN-модели

def create\_rnn\_model():

model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

LSTM(units=128, dropout=0.3, recurrent\_dropout=0.3),

Dense(units=len(np.unique(y)), activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

return model

# Переменные для хранения лучшей модели

best\_accuracy = 0

best\_model = None

for iteration in range(100):

# Установим случайное состояние для воспроизводимости

np.random.seed(iteration + 42)

# Перешивем индексные выборки для текущего прохода

total\_samples = len(X)

indices = np.random.permutation(total\_samples)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.25, random\_state=iteration+42)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# Токенизация и подготовка последовательности символов

X\_train\_padded, X\_test\_padded = prepare\_rnn\_data(X\_train, X\_test, tokenizer)

#print("Shape of training data:", X\_train\_padded.shape)

#print("Shape of labels:", y\_train.shape)

# Создаем новую модель

rnn\_model = create\_rnn\_model()

# Обучение модели

history = rnn\_model.fit(X\_train\_padded, y\_train, batch\_size=32, epochs=100, validation\_data=(X\_test\_padded, y\_test))

# Проверка качества модели

y\_pred = rnn\_model.predict(X\_test\_padded)

y\_pred\_classes = np.argmax(y\_pred, axis=-1)

test\_accuracy = accuracy\_score(y\_test, y\_pred\_classes)

print(f'Iteration {iteration}: Accuracy: {test\_accuracy:.4f}')

# Если текущая модель лучше прежней, запоминаем её

if test\_accuracy > best\_accuracy:

best\_accuracy = test\_accuracy

best\_model = rnn\_model

# Сохраняем лучшую модель

if best\_model is not None:

best\_model.save('best\_rnn\_model.keras')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 11:**

[For\_diploma\_CM\_6.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#DAN2

import os

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense

from sklearn.metrics import accuracy\_score

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

X, y = load\_data('books\_max\_balanced')

y = le.fit\_transform(y)

# Подготовка данных для DAN2

def prepare\_dan2\_data(X\_train, X\_test, tokenizer, max\_length=100):

tokenizer.fit\_on\_texts(X\_train + X\_test)

sequences = tokenizer.texts\_to\_sequences(X\_train + X\_test)

padded\_sequences = pad\_sequences(sequences, maxlen=max\_length)

return padded\_sequences[:len(X\_train)], padded\_sequences[len(X\_train):]

# Токенизатор

tokenizer = Tokenizer(num\_words=5000)

# Архитектура модели DAN2

def create\_dan2\_model(output\_units):

model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

GlobalAveragePooling1D(),

Dense(output\_units, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

return model

# Начальные условия для отслеживания лучшей модели

best\_accuracy = 0

best\_model = None

for iteration in range(100):

# Убедимся в воспроизводимости результатов

np.random.seed(iteration + 42)

# Случайная выборка данных

total\_samples = len(X)

indices = np.random.permutation(total\_samples)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.25, random\_state=iteration+42)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# Подготовим данные для DAN2

X\_train\_padded, X\_test\_padded = prepare\_dan2\_data(X\_train, X\_test, tokenizer)

# Создаем новую модель

dan2\_model = create\_dan2\_model(output\_units=len(np.unique(y)))

# Обучение модели

dan2\_model.fit(X\_train\_padded, y\_train, batch\_size=32, epochs=10, verbose=0)

# Оценка качества модели

y\_pred = dan2\_model.predict(X\_test\_padded)

y\_pred\_classes = np.argmax(y\_pred, axis=-1)

test\_accuracy = accuracy\_score(y\_test, y\_pred\_classes)

print(f'Iteration {iteration}: Accuracy: {test\_accuracy:.4f}')

# Если текущая модель лучше прежних, запоминаем её

if test\_accuracy > best\_accuracy:

best\_accuracy = test\_accuracy

best\_model = dan2\_model

# Сохраняем лучшую модель

if best\_model is not None:

best\_model.save('best\_dan2\_model.keras')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 12:**

[For\_diploma\_CM\_7.py](https://github.com/OskolockKoli/For-a-diploma-1/blob/main/other/For_diploma_CM_1.py" \o "For_diploma_CM_1.py):

#CNN

import os

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense

from sklearn.metrics import accuracy\_score

# Функция для загрузки данных

def load\_data(path):

X = []

y = []

for genre in os.listdir(path):

genre\_path = os.path.join(path, genre)

if not os.path.isdir(genre\_path):

continue

for filename in os.listdir(genre\_path):

with open(os.path.join(genre\_path, filename), 'r', encoding='iso-8859-1') as file:

text = file.read()

X.append(text)

y.append(genre)

return X, y

# Преобразование меток жанров в числовые значения

le = LabelEncoder()

# Загрузка данных

X, y = load\_data('books\_max\_balanced')

y = le.fit\_transform(y)

# Разделение на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Подготовка данных для CNN

def prepare\_cnn\_data(X\_train, X\_test, tokenizer, max\_length=100):

tokenizer.fit\_on\_texts(X\_train + X\_test)

sequences = tokenizer.texts\_to\_sequences(X\_train + X\_test)

padded\_sequences = pad\_sequences(sequences, maxlen=max\_length)

return padded\_sequences[:len(X\_train)], padded\_sequences[len(X\_train):]

# Токенизатор

tokenizer = Tokenizer(num\_words=5000)

# Архитектура модели CNN

def create\_cnn\_model(output\_units):

model = Sequential([

Embedding(input\_dim=5000, output\_dim=128),

Conv1D(filters=64, kernel\_size=3, padding='same'),

MaxPooling1D(pool\_size=2),

Flatten(),

Dense(output\_units, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

return model

# Настройки начальной точки

best\_accuracy = 0

best\_model = None

for iteration in range(100):

# Гарантируем воспроизводимость

np.random.seed(iteration + 42)

# Пересчёт индексов выборки

total\_samples = len(X)

indices = np.random.permutation(total\_samples)

X\_subset = [X[i] for i in indices]

y\_subset = [y[i] for i in indices]

# Разделение на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y\_subset, test\_size=0.25, random\_state=iteration+42)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# Готовим данные для CNN

X\_train\_padded, X\_test\_padded = prepare\_cnn\_data(X\_train, X\_test, tokenizer)

# Создаем новую модель

cnn\_model = create\_cnn\_model(output\_units=len(np.unique(y)))

# Обучение модели

cnn\_model.fit(X\_train\_padded, y\_train, batch\_size=32, epochs=10, verbose=0)

# Тестируем качество модели

y\_pred = cnn\_model.predict(X\_test\_padded)

y\_pred\_classes = np.argmax(y\_pred, axis=-1)

test\_accuracy = accuracy\_score(y\_test, y\_pred\_classes)

print(f'Iteration {iteration}: Accuracy: {test\_accuracy:.4f}')

# Если текущая модель лучше прошлых, запоминаем её

if test\_accuracy > best\_accuracy:

best\_accuracy = test\_accuracy

best\_model = cnn\_model

# Сохраняем лучшую модель

if best\_model is not None:

best\_model.save('best\_cnn\_model.keras')

print(f'\nBest Model Accuracy: {best\_accuracy:.4f}\nModel saved.')

else:

print('No models were trained successfully.')

1. **Приложение 13:**

app.py:

from flask import Flask, render\_template, request, jsonify

from joblib import load

import PyPDF2

import docx

import re

import chardet # Импортируем модуль для автоматической проверки кодировки

# Загружаем ранее обученную модель и векторизатор

vectorizer = load('models/best\_tfidf\_svm\_vectorizer.pkl')

ffbp\_model = load('models/best\_svm\_model.pkl')

# Создание экземпляра Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

try:

input\_type = request.form.get('input\_type') # Определяем метод передачи данных (текст или файл)

if input\_type == 'file':

uploaded\_file = request.files['file']

content = extract\_text(uploaded\_file)

elif input\_type == 'text':

content = request.form.get('text\_input')

else:

return jsonify({'error': 'Неверный тип ввода'})

# Преобразовываем текст в вектор признаков

feature\_vector = vectorizer.transform([content])

# Прогоняем предсказание

prediction = ffbp\_model.predict(feature\_vector)[0]

genres = ['Роман', 'Повесть', 'Рассказ', 'Поэма', 'Пьеса', 'Статья', 'Очерк']

result = f'Жанр произведения: {genres[prediction]}'

return jsonify({'result': result})

except Exception as e:

return jsonify({'error': str(e)})

#Функция для извлечения текста из различных форматов

def extract\_text(file):

filename = file.filename.lower()

raw\_bytes = file.read()

detected\_encoding = chardet.detect(raw\_bytes)['encoding'] or 'utf-8' # Автоматически определяем кодировку

if filename.endswith('.txt'):

return raw\_bytes.decode(detected\_encoding)

elif filename.endswith('.pdf'):

pdf\_reader = PyPDF2.PdfReader(file.stream)

pages = []

for page\_num in range(len(pdf\_reader.pages)):

pages.append(pdf\_reader.pages[page\_num].extract\_text())

return '\n'.join(pages)

elif filename.endswith('.docx'):

document = docx.Document(file.stream)

fullText = []

for para in document.paragraphs:

fullText.append(para.text)

return '\n'.join(fullText)

else:

raise ValueError("Неподдерживаемый формат файла")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **Приложение 14:**

index.html:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Определитель жанра литературного произведения</title>

<link rel="stylesheet" href="/static/style.css">

</head>

<body>

<h1>Определитель жанра литературного произведения</h1>

<form id="uploadForm" enctype="multipart/form-data" method="post">

<label for="manual\_text">Введите текст:</label><br>

<textarea name="manual\_text" rows="4" cols="50" placeholder="Введите текст произведения"></textarea><br>

<label for="file">Или выберите файл (.txt, .docx, .pdf):</label><br>

<input type="file" name="file" accept=".txt,.docx,.pdf"><br>

<button type="submit">Определить жанр</button>

</form>

<div id="result"></div>

<!-- Скрипт для отправки формы асинхронно -->

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>

$(document).ready(function(){

$('#uploadForm').on('submit', function(event){

event.preventDefault();

var formData = new FormData(this);

$.ajax({

url: '/predict',

type: 'POST',

data: formData,

processData: false,

contentType: false,

success: function(response){

$('#result').html('<p>'+response.result+'</p>');

},

error: function(error){

console.error('Ошибка:', error.responseJSON.error);

alert('Возникла ошибка!');

}

});

});

});

</script>

</body>

</html>

1. **Приложение 15:**

style.css:

body {

font-family: Arial, sans-serif;

background-color: #fafafa;

}

h1 {

color: #333;

margin-top: 50px;

text-align: center;

}

form {

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

padding: 20px;

}

button {

margin-top: 10px;

padding: 10px 20px;

border-radius: 5px;

cursor: pointer;

background-color: #007bff;

color: white;

border: none;

}

button:hover {

background-color: #0056b3;

}

#result {

text-align: center;

margin-top: 20px;

font-size: 18px;

color: green;

}