# РЕФЕРАТ

Бакалаврская работа состоит из 108 страниц, содержащих 28 источников, 9 рисунков. Объект исследования – разработка алгоритма машинного обучения для классификации текстов по жанру литературы.

Актуальность: упрощение процесса каталогизации и поиска книг в информационных системах; ускорение обработки большого объёма данных и повышение объективности оценки результата.

Цель данной работы заключается в разработке системы классификации текстов по жанру литературы на основе алгоритма машинного обучения.

Задачи работы: анализ предметной области, сравнение с аналогами, проектирование системы, исследование и сбор данных, реализация, оценка и оптимизация качества модели классификации, интеграция модели классификации в программное обеспечение.

Основные результаты: получена система классификации текстов по жанру литературы на основе модели алгоритма машинного обучения, обладающая высокой точностью оценки жанра, высокой скоростью обработки данных по сравнению с самыми близкими аналогами.

ФИО студента: Бобрович Николай Сергеевич.

Учебное заведение и группа: государственный университет аэрокосмического приборостроения, группа 4136.

# ABSTRACT

The bachelor's thesis consists of 108 pages containing 28 sources and 9 figures.The object of the study is the development of a machine learning algorithm for classifying texts by literary genre.

Relevance: simplifying the process of cataloging and searching for books in information systems; accelerating the processing of large amounts of data and increasing the objectivity of the result assessment.

The purpose of this work is to develop a system for classifying texts by literary genre based on a machine learning algorithm.

Tasks of the work: analysis of the subject area, comparison with analogues, system design, research and data collection, implementation, evaluation and optimization of the quality of the classification model, integration of the classification model into software.

Main results: a system for classifying texts by literary genre based on a machine learning algorithm model was obtained, which has a high accuracy of genre assessment, high data processing speed compared to the closest analogues.

Student's full name: Bobrovich Nikolay Sergeevich.

Educational institution and group: State University of Aerospace Instrumentation, group 4136.

**СОДЕРЖАНИЕ**

## ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

АК - алгоритм классификации

ГП - гиперпараметр(ы)

МК - модель классификации

ММО - метод машинного обучения

МО - машинное обучение

НД - набор данных

ПО - программное обеспечение

API - Application Programming Interface

CNN - Convolutional Neural Network

DAN2 - Deep Averaging Network

DT - Decision Tree

FFBP - Feedforward Neural Network

NB - Naive Bayes

RNN - Recurrent Neural Network

SVM - Support Vector Machine

# ВВЕДЕНИЕ

Создание системы для классификации текстов по жанру литературы 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.

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

Для решения поставленной задачи необходимо пройти следующие этапы:

- проектирование системы: определение методов машинного обучения и архитектуры ПО;

- исследование и сбор данных: сбор базы текстов различных жанров, предварительная обработка данных;

- реализация и оценка качества работы модели классификации: создание МК и её оценка;

- оптимизация: настройка гиперпараметров методов машинного обучения, экспериментирование с различными методами предобработки данных;

- повторная оценка качества работы модели классификации: при недостаточной эффективности МК должна происходить повторная оптимизация;

- интеграция модели классификации в программное обеспечеие: разработка ПО, подготовка системы к эксплуатации, интеграции в целевую информационную среду;

- документация: составление руководства пользователя и технического описания всей системы;

- способы возможной модификации ПО: рассмотрение возможных направлений дальнейшего развития системы.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

## 1.1 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 машинное обучение 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.

## 1.2 Постановка задачи

Задача заключается в создании программного обеспечения на основе модели классификации (на основе алгоритма машинного обучения), способного автоматически определять жанр литературного произведения на основе его текста.

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

**Функциональные требования:**

1. Классификация текстов: система должна уметь классифицировать тексты по следующим жанрам: Роман, Повесть, Рассказ, Поэма, Пьеса, Статья, Очерк.

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

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

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

5. Язык обрабатываемых текстов: Русский язык.

**Нефункциональные требования:**

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

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

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

4. Простота интеграции: система должна иметь возможность интеграции без изменений в её архитектуре.

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

**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.

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## 1.3 Проектирование программного обеспечения

**Платформа для реализации системы:**

- операционная система: Linux Ubuntu Server 20.04 (стабильная платформа с широким спектром пакетов для работы с языками программирования (Python в данном случае), развитой поддержкой виртуализации контейнеров (Docker в данном случае) и высокой производительностью);

- библиотека пользовательских интерфейсов: Bootstrap (HTML/CSS/JS) для клиентской стороны и Flask для серверной стороны (эти инструменты позволяют быстро создавать удобные и отзывчивые интерфейсы);

- аппаратная платформа: стандартный ПК или сервер с Intel Core i5/i7 и 8 ГБ оперативной памяти (требований к оборудованию особых нет ввиду небольшой сложности используемых моделей и небольшого объёма данных);

- необходимая конфигурация компьютера: необходимы базовые ресурсы CPU (≥ 4 ядра), RAM (≥ 8 GB), хранилище SSD (≥ 100 GB) (для установки необходимых библиотек и моделей).

**Средства автоматизированного проектирования, разработки, тестирования и сопровождения:**

- средствами проектирования и разработки являются: PyCharm для IDE разработки и отладки Python-кода и GitHub для ведения версий и совместной работы;

- средствами тестирования выступают: unit-тесты для проверки правильности функционирования отдельных модулей и Selenium для автотестирования UI-компонентов;

- инструментами сопровождения служат: Docker для деплоймента и изолированной среды исполнения.

**Описание разработки ПО:**

ПО будет построено на принципах RESTful API и микросервисной архитектуры. Архитектуру составляют следующие элементы:

- backend-сервис (Flask): реализует обработку запросов, вычисления и хранение данных;

- frontend (Bootstrap): формирует удобную оболочку для конечного пользователя;

- МК (Scikit-Learn/SVM): лежит в основе самой критичной части системы — классификации текстов.

Каждый элемент имеет чёткое разделение обязанностей и общается через стандартизированные JSON-вызовы.

**Описание разработки структур данных:**

данные будут хранится в двух видах:

- тексты произведений в виде отдельных файлов в папках соответствующих жанров;

- векторы признаков (TF-IDF) сохраняются отдельно для ускорения последующих вычислений.

Такая организация позволит гибко управлять и оперативно получать доступ к данным.

**Описание разработки интерфейса пользователя:**

пользовательский интерфейс будет построен с акцентом на удобство и простоту использования. Главная страница содержит два варианта подачи текста:

- прямой ввод текста пользователем;

- возможность загрузки файла (.txt/.docx/.pdf).

## 1.4 Определение методов машинного обучения

**Определение основных алгоритмов классификации и методов машинного обучения** (в будущем в общем случае АК [3, 5, 8, 9, 16, 17, 28]):

1. Naive Bayes (NB) — метод машинного обучения, 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) — алгоритм классификации, 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.

## 1.5 Архитектура информационной системы

На основе полученной информации необходимо спроектировать ПО, определяющее жанр литературы загруженного пользователем текста или документа(ов), на основе полученной модели классификации.

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:**

1. Серверная сторона будет построена на Flask — лёгком и мощном веб-фреймворке Python, обеспечивающем быстрый старт и простую реализацию REST API.
2. Клиентская сторона будет выполнена с использованием стандартных инструментов HTML, CSS и JavaScript, обеспечивая лёгкость поддержки и расширения функциональности.
3. 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.
4. Работа с текстовыми файлами: для удобства обработки документов будут использоваться специализированные библиотеки PyPDF2 и python-docx, поддерживающие чтение содержимого из .pdf и .docx соответственно.

## 1.6 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.

3. **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:**

1. Специализированная модель, ориентированная именно на литературу русского языка.

2. Возможность настройки и оптимизации модели под конкретные нужды пользователей.

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

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

5. 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.

# РЕАЛИЗАЦИЯ СИСТЕМЫ

## 1.7 Исследование и сбор данных

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

1. **Определение источников данных:**

с помощью скриптов (приложение А, Б, В), которые спарсили все произведения с сайтов, было получено свыше 19 тысяч произведений, принадлежащих семи основным жанрам: роман, повесть, рассказ, поэма, пьеса, очерк и статья. Основными источниками сбора данных послужили интернет-библиотеки: Алексея Комарова [29] и Максима Мошкова [30]).

2. **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 А, Б, В) и методов форматирования [21, 22], весь корпус был преобразован в стандартный текстовый формат .txt, нормализован и очищен от лишних метаданных и шумовых элементов.

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

с помощью тех же скриптов (приложение А, Б, В), 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).

**4.** Обеспечение сбалансированности набора данных (утверждение, что количество текстов в каждом жанре примерно одинаково, чтобы избежать дисбаланса в обучении модели классификации):

изначальное распределение выборки произведений по жанрам таково: очерк - 1975, повесть - 2166, поэма - 2020, пьеса - 1868, рассказ - 2015, роман - 2562, 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 выборки произведений по жанрам таково: очерк - 1975, повесть - 2050, поэма - 2020, пьеса - 1868, рассказ - 2015, роман - 2050, статья - 2050. При таком распределении разница между наименьшим и наибольшим количеством элементов в классе < 10%, что позволяет считать такой НД сбалансированным. В итоговом наборе данных 14028 произведений, равномерно распределённых по семи различным жанрам).

**5.** Обеспечение масштабируемости набора данных (возможность изменять размер НД, сохраняя его сбалансированность):

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

## 1.8 Реализация и оценка качества модели классификации

Каждый из алгоритмов классификации был запущен несколько раз на разных объёмах набора данных, чтобы выявить зависимость точности от размера обучающего массива.

С помощью скрипта (приложение Д) был получен следующий результат запусков каждого алгоритма классификации на разных размерах исходного набора данных (в соответствии с рисунком 1):

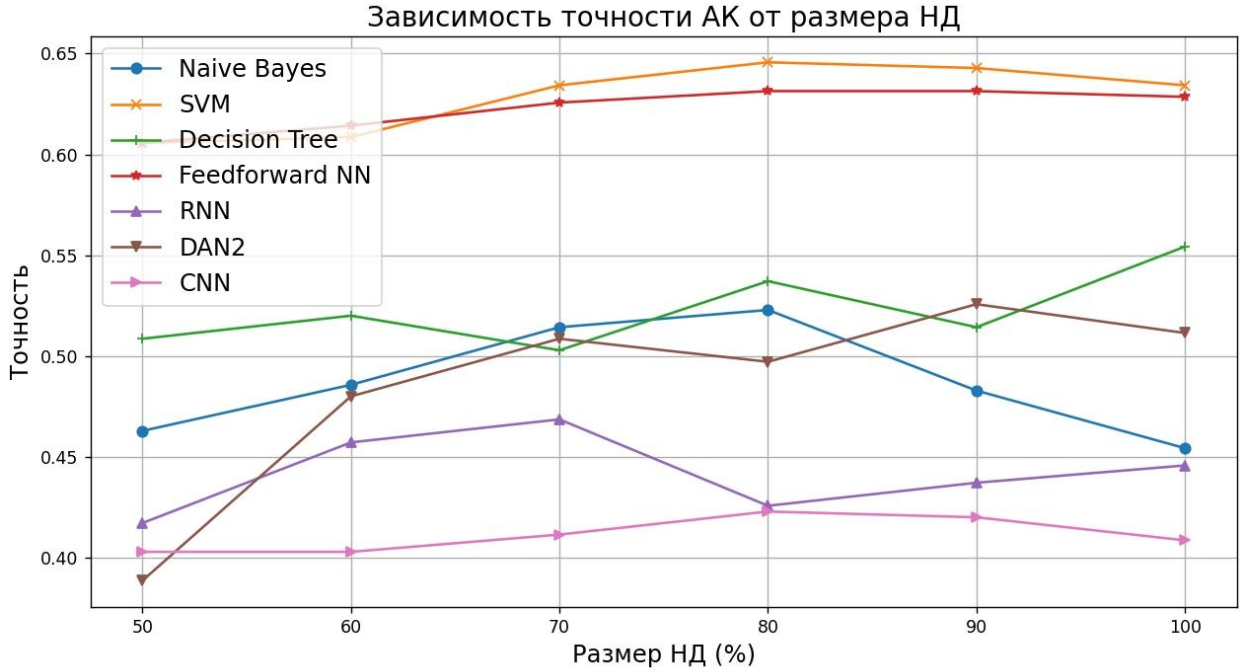


Рисунок 1 - График зависимости точности алгоритма классификации от размера набора данных

На рис. 1 по вертикали указана точность модели для каждого алгоритма классификации (при гиперпараметрах по умолчанию), а по горизонтали указан процент от максимального сбалансированного НД, полученного ранее.

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 набора данных выбираются случайным образом из полного НД), необходимого для практической эксплуатации. Следовательно, необходим дополнительный этап оптимизации каждой из АК.

## 1.9 Оптимизация алгоритмов классификации

Далее будет рассмотрен каждый алгоритм классификации по отдельности и методы оптимизации работы каждого алгоритма классификации для увеличения его точности

(с помощью информации из [14, 15, 18, 23, 25, 26, 27, 28]).

**1. NB:**

**- ГП:**

у алгоритма Naive Bayes есть несколько гиперпараметров:

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);

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);

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 гиперпараметров alpha.

Будет произведено несколько запусков с использованием различных значений alpha;

**- 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;

**- признаки:**

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

Будет произведено несколько запусков с использованием n-грамм;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 Е), в котором был оптимизирован метод машинного обучения NB, была получена оптимальная точность (+-0.05): accuracy NB: 0.5492

**2. SVM:**

**- ГП:**

у алгоритма Support Vector Machine есть несколько гиперпараметров:

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 гиперпараметров C, kernel.

Будет произведено несколько запусков с использованием различных значений этих гиперпараметров;

**- архитектура:**

поскольку SVM уже является мощным классификатором, особенно для линейно разделимых данных, значительные архитектурные изменения не принесут большого выигрыша.

Поэтому изменений в архитектуре для этого алгоритма классификации не будет;

**- признаки:**

для SVM качество признаков играет ключевую роль, но так как используемый набор данных уже оптимален (нормализован, сбалансирован, очищен и дискретизирован), дополнительные признаки в данном случае не помогут.

Изменений в признаках для этого АК не будет;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 Ж), в котором был оптимизирован метод машинного обучения SVM, была получена оптимальная точность (+- 0.05): accuracy SVM: 0.7991

**3. DT:**

**- ГП:**

у алгоритма Decision Trees есть несколько гиперпараметров:

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 гиперпараметров max_depth, min_samples_split, min_samples_leaf, criterion.

Будет произведено несколько запусков с использованием различных значений этих гиперпараметров;

**- архитектура:**

поскольку DT уже является мощным классификатором, особенно для линейно разделимых данных, значительные архитектурные изменения не принесут большого выигрыша.

Поэтому изменений в архитектуре для этого МК не будет;

**- признаки:**

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

Поэтому изменений в признаках для этого МК не будет;

**- валидация:**

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

**- 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 И), в котором был оптимизирован ММО DT, была получена оптимальная точность (+- 0.05): accuracy DT: 0.6343

**4. FFBP:**

**- ГП:**

у алгоритма Feed Forward Back Propagation есть несколько гиперпараметров:

hidden_layer_sizes: размеры скрытых слоев;

activation: функция активации для скрытых слоев;

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 гиперпараметры hidden_layer_sizes, activation, alpha, learning_rate_init.

Будет произведено несколько запусков с использованием различных значений этих гиперпараметров;

**- 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;

**- признаки:**

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

Изменений в признаках для этого МК не будет;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 К), в котором был оптимизирован ММО FFBP, была получена оптимальная точность (+- 0.05): accuracy FFBP: 07391

**5. RNN:**

**- ГП:**

у алгоритма Recurrent neural network есть несколько гиперпараметров:

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 гиперпараметры units, epochs, batch_size, dropout.

Будет произведено несколько запусков с использованием различных значений этих гиперпараметров;

**- архитектура:**

Word embeddings: способы 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;

**- 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;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 Л), в котором был оптимизирован метод классификации RNN, была получена оптимальная точность (+- 0.05): accuracy RNN: 0.4943

**6. DAN2:**

**- ГП:**

у алгоритма Dynamic Artificial Neural Network есть несколько гиперпараметров:

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 гиперпараметров;

**- 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;

**- признаки:**

использование предварительно обученных эмбеддингов: Замена случайных эмбеддингов на предварительно обученные (GloVe и Word2Vec) поможет сети сразу воспользоваться богатством готового представления слов.

Будет произведено несколько запусков с использованием GloVe и Word2Vec;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 М), в котором был оптимизирован метод классификации DAN2, была получена оптимальная точность (+- 0.05): accuracy DAN2: 0.5329

**7. CNN:**

**- ГП:**

у алгоритма  есть несколько гиперпараметров:

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 гиперпараметров;

**- архитектура:**

поскольку CNN уже является мощным классификатором, особенно для линейно разделимых данных, значительные архитектурные изменения не принесут большого выигрыша.

Поэтому изменений в архитектуре для этого МК не будет;

**- 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;

**- валидация:**

будет произведено несколько запусков с использованием различных соотношений тестовой и обучающей выборок;

**- 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 Н), в котором был оптимизирован метод классификации CNN, была получена оптимальная точность (+- 0.05): accuracy CNN: 0.5100

### 1.10 Повторная оценка качества модели классификации

Результаты оптимизации в графическом представлении в соответствии с рисунком 2:

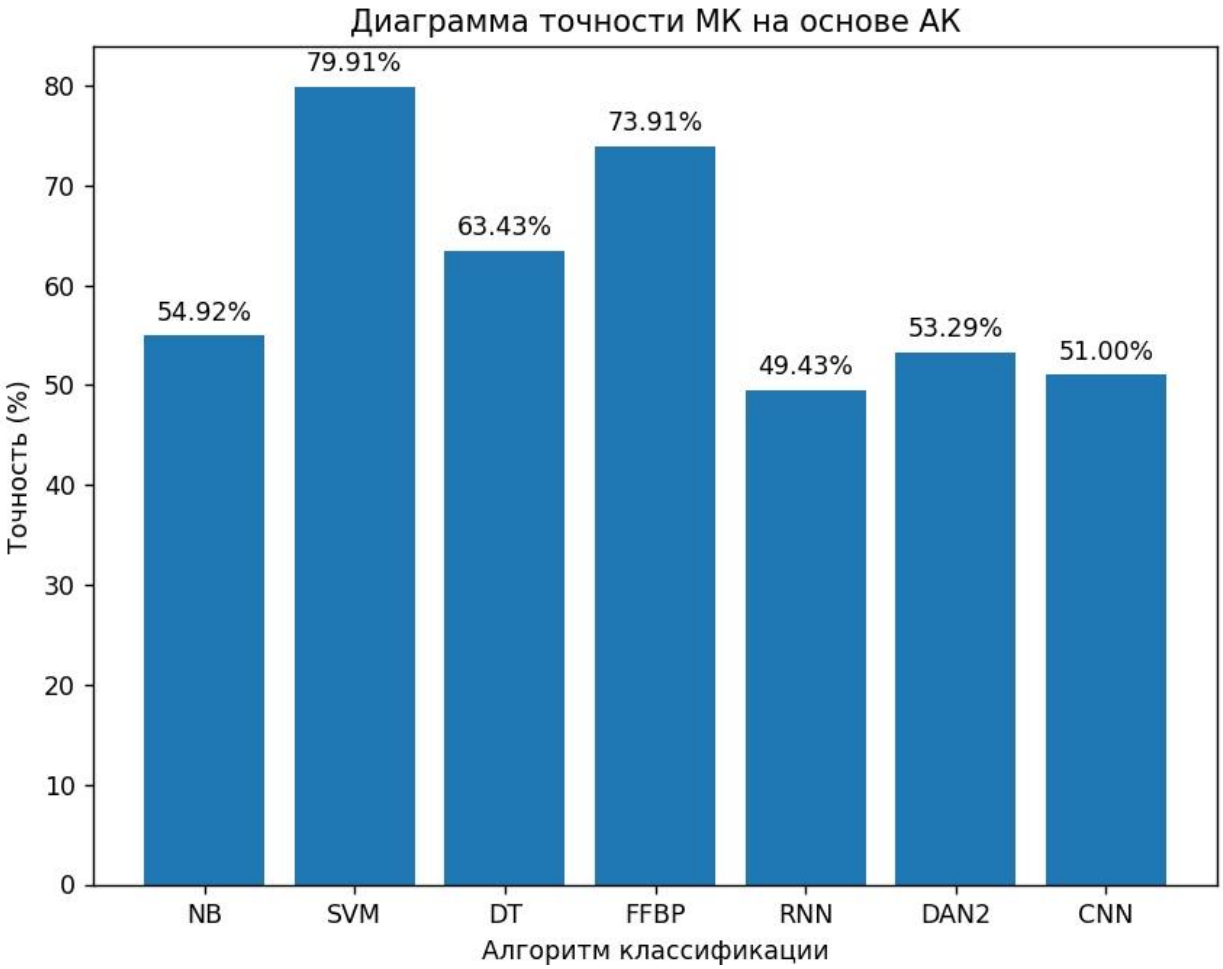


Рисунок 2 - Диаграмма точности модели классификации на основе алгоритма классификации

на рис. 2 видно (особенно с учётом погрешности в 5%, обусловленной тем, что данные для неполного набора данных выбираются случайным образом из полного НД), что модель классификации на основе SVM является достаточно точной для классификации текстов по жанрам литературы.

Конечный выбор МК на основе SVM обуславливается не только его высокой точностью, но и тем, что это наиболее подходящий метод машинного обучения для работы с текстом (на основе [1], [9]).

### 1.11 Интеграция модели классификации в программное обеспечение

Разработано программное обеспечение, определяющее жанр литературы загруженного пользователем текста или документа(ов), на основе полученной модели классификации. (приложения П, Р, С)

Внешний вид и структура приложения (в соответствии с рисунком 3):

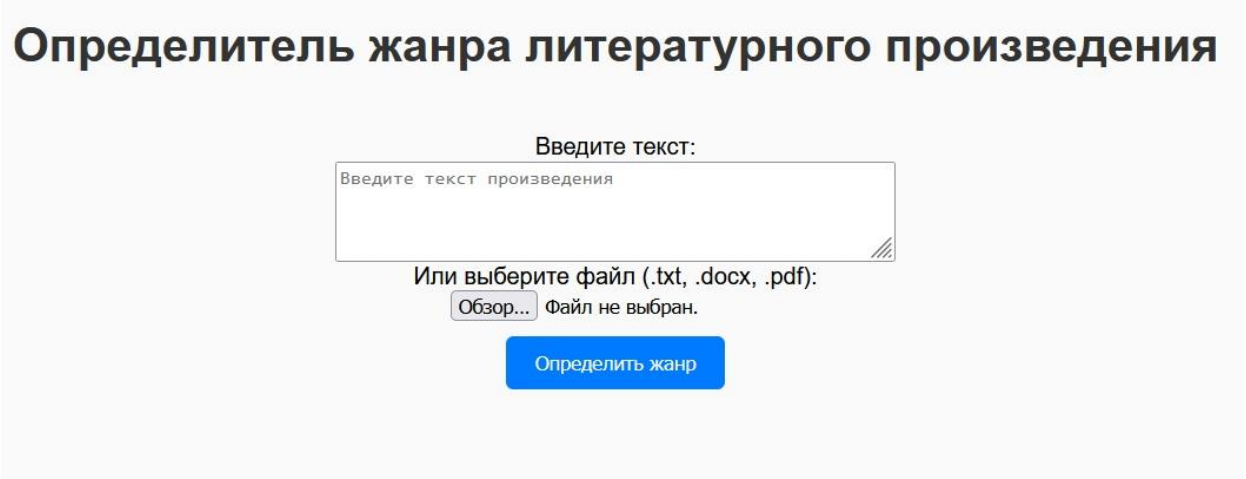


Рисунок 3 - Вид главной страницы приложения

на рис. 3 показано, что главная страница представлена простым, лаконичным оформлением с двумя основными элементами взаимодействия:

1. Поле для ввода текста:

позволяет пользователям ввести фрагмент произведения непосредственно в специальное текстовое поле (в соответствие с рисунком 4).

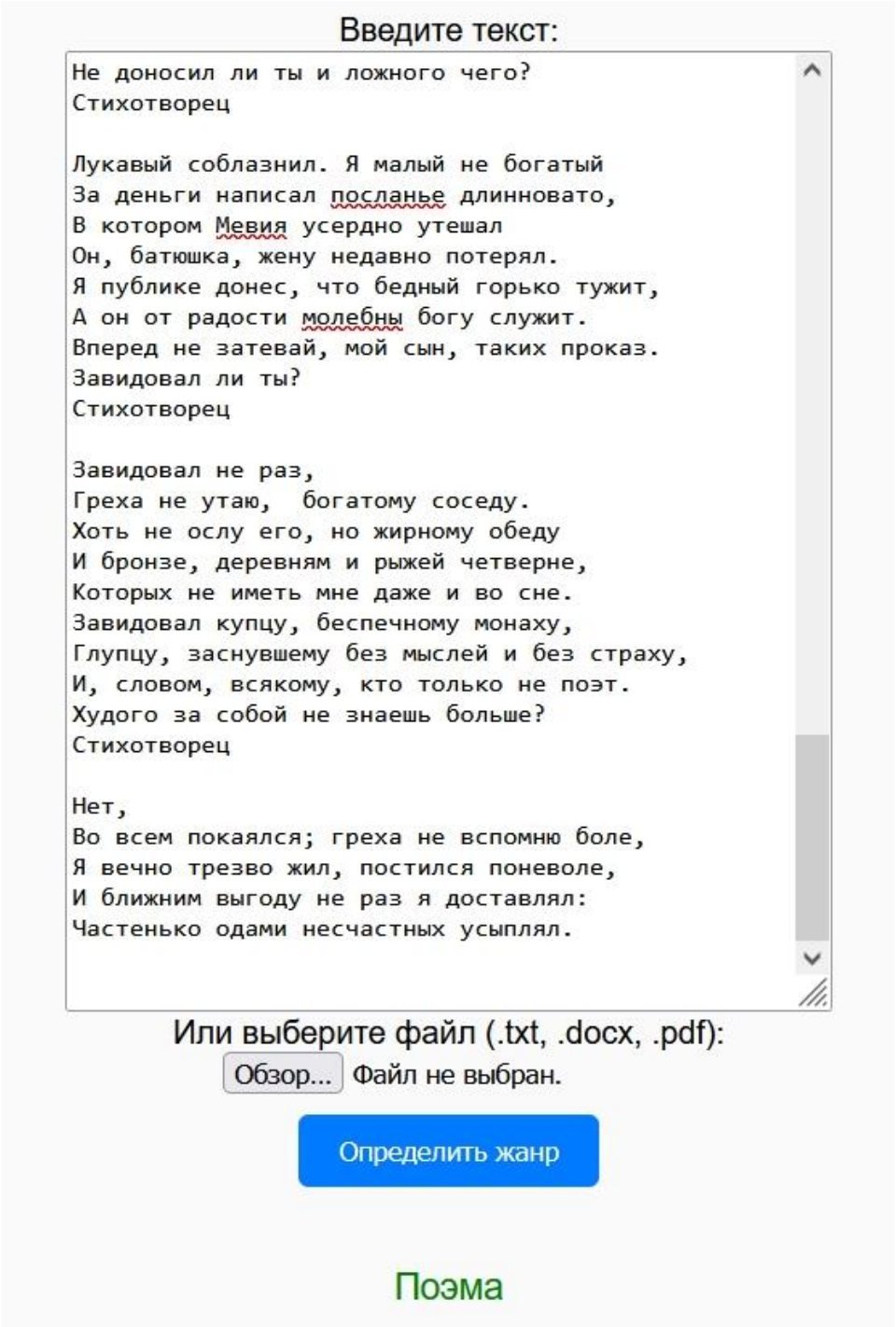


Рисунок 4 - Определение жанра произведения по введённому тексту

2. 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 (в соответствие с рисунком 5).

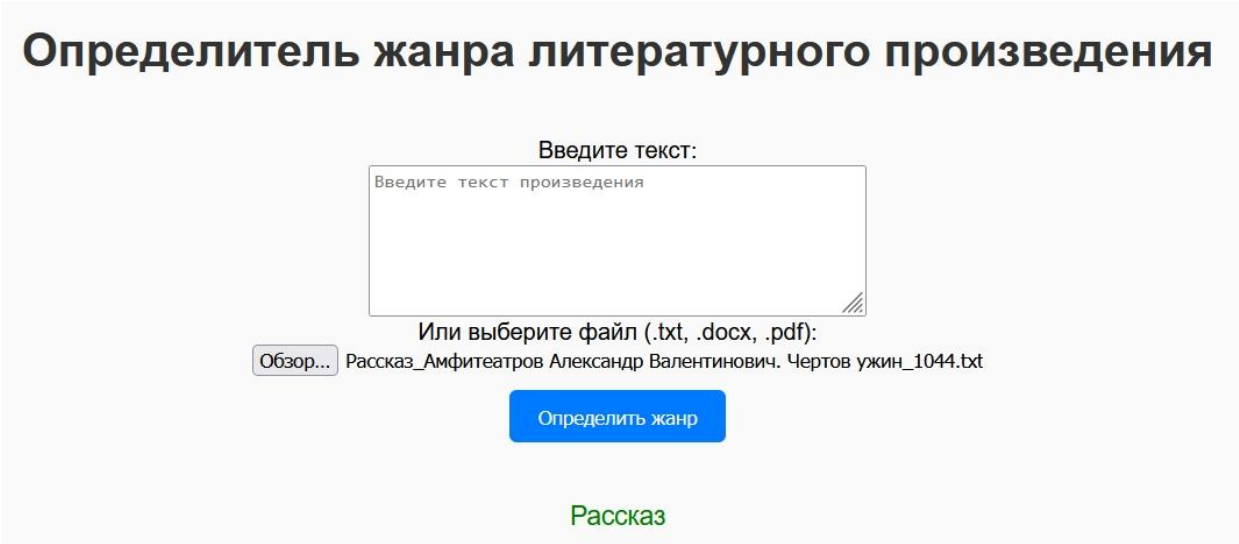


Рисунок 5 - Определение жанра произведения по выбранному файлу

**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.

## 1.12 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».**

**1. 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.

**2. Руководство пользователя.**

Назначение приложения: система 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.

Начало работы:

перейдите на главную страницу приложения;

выберите один из способов ввода текста:

- напишите текст произведения вручную в специальном текстовом поле;

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

нажмите кнопку «Определить жанр»;

получите результат ниже формы.

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;

**3. Архитектура приложения.**

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:

Реализована 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;

- модель классификации:

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;

**4. 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;

Модель классификации:

- 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 слоёв, оптимизатора и скорости обучения;

**5. 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;

**6. 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;

**7. 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.

## 1.13 Способы возможной модификации программного обеспечения

Предлагаемые способы модернизации программного 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 наборы данных и сбалансировать их со старыми наборами данных.

- дополнить общий НД новыми примерами.

- повторно обучить модель классификации с увеличенным набором жанров.

- протестировать новую модель на валидных данных.

Пример:

в моей реализации набор данных представлен довольно простым способом (в соответствие рисунку 6):


Рисунок 6 - Вид набора данных

на рис. 6 видно, что есть набор папок, в каждой из которых лежат произведения отдельных жанров. Скрипт для создания модели классификации проходит по каждой и с последующей обработкой (в соответствие с рисунком 7):

```
# 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 f
                text = file.read()
                X.append(text)
                y.append(genre)

    return X, y
```

Рисунок 7 - Функция для загрузки данных (приложение Ж)

на рис. 7 видно, что чтобы добавить новый жанр нужно добавить в директорию с остальными папками новую и пересоздать модель классификации;

**2. 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 П) 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)
```

**3. 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 НД представлен довольно простым способом (в соответствие с рисунком 8):


Рисунок 8 - Вид набора данных

на рис. 8 видно, что есть набор папок, в каждой из которых лежат произведения отдельных жанров. Скрипт для создания МК проходит по каждой и с последующей обработкой (в соответствие с рисунком 9):

```
# 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 f
                text = file.read()
                X.append(text)
                y.append(genre)

    return X, y
```

Рисунок 9 - Функция для загрузки данных (приложение Ж)

на рис. 9 видно, что чтобы обновить НД нужно добавить новые файлы произведений каждого жанра (с сохранением сбалансированности) в директорию в каждую папку жанров и пересоздать МК.

**4. 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 методов машинного обучения. 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 моделей классификации, до их оценки и интеграции в готовое программное обеспечение. Исследования показали, что разработанная модель классификации 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 // Вестник Пермского университета - 2014. - с. 12-18.
2. Панг П., Ли Л.: Opinion Mining and Sentiment Analysis. - Foundation and Trends in Information Retrieval, 2008 г. - 135 с.
3. Браславский П. Морфологический строй функциональных стилей. // Известия Уральского государственного университета - 2001. - с. 10-18.
4. Емашова О.А., Мальковский М.Г. Функциональные стили русского языка и их влияние на задачу автоматического реферирования текста // Диалог - 2007. - с. 7-12.
5. Statamatos E., Fakotakis N., Kokkinakis G. Automatic text categorization in terms of genre and author. // Computational Linguistics - 2000. - с. 1-26.
6. Petrenz P., Webber B. Stable classification of text genres. // Computational Linguistics - 2008. - с. 10-18.
7. Шевелев О.Г., Петраков А.В. Классификация текстов с помощью деревьев решений и нейронных сетей прямого распространения. - Вестник Томского государственного университета, 2006. - 290 с.
8. Дементьев В.Е., Киреев С.Х. Выбор алгоритмов машинного обучения для классификации текстовых документов. // Техника средств связи - 2022. - с. 21-52.
9. Дементьев В.Е., Чулков А.А. Метод автоматизированной идентификации признаков протоколов сетей передачи данных. // Информация и космос - 2021. - с. 16-24.
10. Дементьев В.Е., Чулков А.А. Модель протокола сети передачи данных в условиях деструктивных кибернетических воздействий. Часть 1. // Защита информации. Инсайд. - 2021. - с. 8-15.

11. Дементьев В.Е., Чулков А.А. Модель протокола сети передачи данных в условиях деструктивных кибернетических воздействий. Часть 2. // Защита информации. Инсайд. - 2021. - с. 10-19.

12. Дементьев В.Е., Чулков А.А. Методика оценки защищенности протоколов сети передачи данных в условиях деструктивных кибервоздействий. // Известия Тульского государственного университета. Технические науки. - 2007. - с. 11-22.

13. Sebastiani F. Machine learning in automated text categorization. - ACM Computing Surveys, 2002. - 47 с.

14. Cheng H., Soon Cheol P. Combination of modified BPNN algorithms and an efficient feature selection method for text categorization. // Information Processing and Management 45 - 2009. - с. 5-17.

15. Ming Li, Hang Li, Zhi-Hua Z. Semi-supervised document retrieval. // Information Processing and Management 45 - 2009. - с. 17-32.

16. Monica Rogati, Yiming Yang. High-Performing Feature Selection for Text Classification. // CIKM'02 - 2009. - с. 11-20.

17. Т.В. Батура. Методы автоматической классификации текстов. // Программные продукты и системы - 2017. - с. 1-14.

18. Н.Н. Буйлова. Автоматизация обработки текста. // Информационные процессы и системы - 2018. - с. 12-17.

19. Голубева И.Б. Стилистика русского языка. - Рольф, 1999. - 448 с.

20. Мангалова Е.С., Агафонов Е.Д. О проблеме выделения информативных признаков в задаче классификации текстовых документов. // Вестник Том. гос. ун-та. Управление, вычислительная техника и информатика. - 2013. - с. 12-19.

21. Скляренко Н.С. Обзор алгоритмов машинного обучения, решающих задачу обнаружения спама. // Новые информационные технологии в автоматизированных системах. - 2017. - с. 12-17.

22. Епрев А.С. Автоматическая классификация текстовых документов. // Математические структуры и моделирование. - 2010. - с. 7-12.

23. Борисов Л.А., Орлов Ю.Н., Осминин К.П. Идентификация автора текста по распределению частот буквосочетаний. // Прикладная информатика - 2013. - с. 16-29.

24. Сухарева А.В., Царьков С.В. Классификация научных текстов по отраслям знаний. // Машинное обучение и анализ данных. - 2014. - с. 7-10.

25. Кубарев А.И., Кукушкина О.В., Поддубный В.В. и др. Построение таблиц стилей текстовых произведений с использованием алгоритмов классификации на основе деревьев решений. // Вестн. Томск. гос. ун-та. Сер. Управление, вычислительная техника и информатика. - 2012. - с. 23-33.

26. Веретенников И., Карташев Е., Царегородцев А. Оценка качества классификации текстовых материалов с использованием алгоритма машинного обучения «Случайный лес». // Известия АлтГУ - 2017. - с. 12-23

27. Интернет-библиотека Алексея Комарова - URL: https://ilibrary.ru/ (дата обращения 01.05.2025)

28. Интернет-библиотека Максима Мошкова - URL: https://lib.ru/ (дата обращения 01.05.2025)

## ПРИЛОЖЕНИЕ А

## 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)
```

## ПРИЛОЖЕНИЕ Б

## 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()
```

## ПРИЛОЖЕНИЕ В

## 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()
```

## ПРИЛОЖЕНИЕ Г

## 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)
```

## ПРИЛОЖЕНИЕ Д

## 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()
```

**ПРИЛОЖЕНИЕ Е**

**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.')
```

## ПРИЛОЖЕНИЕ Ж

## For_diploma_CM_2.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.')
```

**ПРИЛОЖЕНИЕ И**

**For_diploma_CM_3.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.')
```

**ПРИЛОЖЕНИЕ К**

**For_diploma_CM_4.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.')
```

**ПРИЛОЖЕНИЕ Л**

**For_diploma_CM_5.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.')
```

**ПРИЛОЖЕНИЕ М**

**For_diploma_CM_6.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.')
```

## ПРИЛОЖЕНИЕ Н

## For_diploma_CM_7.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.')
```

## ПРИЛОЖЕНИЕ П

### 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)
```

## ПРИЛОЖЕНИЕ Р

### 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>
```

## ПРИЛОЖЕНИЕ С

### 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;
}
```