Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Кафедра анализа данных и машинного обучения**

**Пояснительная записка к курсовой работе**по дисциплине “Технологии анализа данных и машинное обучение” на тему:

«Машинное обучение в задачах нормализации слов текста»

**Выполнила:**   
студентка группы ЗБ-ПИ20-1  
Кичеева А.В.  
**Научный руководитель:**

старший преподаватель кафедры

анализа данных и машинного обучения

Добрина М. В.

2024 г

Оглавление

[Введение 3](#_Toc167555597)

[1. Теоретическая часть 5](#_Toc167555598)

[1.1 Основные методы нормализации текста 6](#_Toc167555599)

[1.2 Используемые технологии 8](#_Toc167555600)

[1.3 Описание алгоритмов машинного обучения 9](#_Toc167555601)

[2. Обработка данных 11](#_Toc167555602)

[2.1 Сбор и очистка данных 12](#_Toc167555603)

[2.2 Удаление стоп-слов 14](#_Toc167555604)

[2.3 Стемминг и лемматизация 15](#_Toc167555605)

[3. Разработка и обучение модели 17](#_Toc167555606)

[3.1 Применение алгоритмов машинного обучения 19](#_Toc167555607)

[3.2 Повышение эффективности модели 21](#_Toc167555608)

[3.3 Анализ результатов 24](#_Toc167555609)

[Заключение 26](#_Toc167555610)

[Список литературы и источники 27](#_Toc167555611)

[Приложение 28](#_Toc167555612)

# **Введение**

Современный объем информации, создаваемой и распространяемой каждый день, непрерывно растет. Этот приводит к необходимости разработки эффективных методов обработки и анализа текстовых данных для извлечения полезной информации. Одним из ключевых этапов такой обработки является нормализация текста.

**Нормализация текста** играет ключевую роль в процессе обработки и анализа текстовых данных. Этот этап позволяет стандартизировать текстовую информацию, упрощая ее дальнейшую обработку и анализ. Однако с увеличением объема и разнообразия текстовых данных возникают новые вызовы для существующих методов нормализации. Традиционные подходы, могут оказаться недостаточно эффективными или гибкими для обработки сложных и разнообразных текстов.

**Машинное обучение** становится все более популярным инструментом для решения задач нормализации текста. Алгоритмы машинного обучения позволяют автоматически извлекать закономерности из данных и адаптироваться к различным структурам текста. Это открывает новые возможности для разработки более гибких и эффективных методов нормализации, которые могут успешно справляться с разнообразными видами текстов.

**Целью** курсовой работы является обучение моделей для нормализации слов текста и оценка эффективности данных методов.

**Задачами** курсовой работы являются:

* Разработка методов нормализации текста.
* Разделение данных и обучение моделей
* Проведение экспериментов для оценки качества разработанных моделей на тестовой выборке данных
* Предоставление результатов в виде графиков

Достижение поставленных целей и выполнение задач позволят разработать эффективные методы нормализации текста, что в свою очередь способствует улучшению процессов обработки и анализа текстовых данных.

# **Теоретическая часть**

Современные технологии непрерывно генерируют огромные объемы текстовой информации, создавая необходимость в разработке эффективных методов обработки и анализа текстов для извлечения полезной информации. Одним из важных этапов такой обработки является нормализация текста.

**Нормализация текста** — процесс обработки естественного языка, который состоит из очистки и предварительной обработки текстовых данных, чтобы сделать их согласованными и пригодными для различных задач. Этот процесс включает в себя различные методы. Основные методы: нормализация регистра, удаление знаков препинания и стоп-слов, стемминг и лемматизация.

**Машинное обучение** — это подраздел ИИ, занимающийся разработкой определенных алгоритмов и моделей. Они могут извлекать важную информацию из датасета и принимать решения после обучения.

Главной идеей машинного является создание моделей, которые обучаются на определенных данных и в дальнейшем могут использовать полученный опыт для будущих прогнозов или принятия новых решений в различных задачах.

Основные виды обучения:

* Обучение с учителем.

Используется для решения задач классификации и регрессии. Классификация прогнозирует метку класса, регрессия прогнозирует числовое значение. В данном виде происходит обучение модели на размеченных данных, которые имеют соответствующую метку.

* Обучение без учителя.

Используется для решения задач кластеризации и снижения размерности. Кластеризация определяет группирование похожих объектов, а снижение размерности поиск компактного размещения данных. В данном обучении к модели применяются неразмеченные данных со скрытыми структурами.

* Обучение с подкреплением.

Этот тип обучения часто используется в задачах, где агент должен выбирать оптимальные действия для достижения определенной цели, например, в играх или робототехнике. Смысл обучения модели во взаимодействии с окружающей средой.

Машинное обучение становится все более популярным подходом к решению задач нормализации текста. Алгоритмы машинного обучения позволяют автоматически извлекать сложные закономерности из данных и адаптироваться к различным структурам текста. Нейронные сети, алгоритмы обучения с подкреплением и другие методы машинного обучения предоставляют новые возможности для разработки более гибких и эффективных методов нормализации.

## **Основные методы нормализации текста**

Предварительная обработка текста означает приведение текста в форму, предсказуемую и поддающуюся анализу для определенной задачи. Существуют разные способы предварительной обработки текста.

Методы нормализации текста:

1. **Нормализация пунктуации и регистра**

Нормализация пунктуации заключается в удалении или замене знаков пунктуации в тексте, что позволяет унифицировать его форматирование. Нормализация регистра включает приведение всех символов к одному регистру (например, нижнему), чтобы устранить различия в написании слов.

Перевод текстовых данных в нижний или верхний регистр – это достаточно простой способ предварительной обработки текста. Его можно использовать для решения большого количества различных задач и примеров. Особенно его хорошо использовать для небольшого набора данных.

**Нормализация пунктуации** включает следующие шаги:

* Удаление всех знаков пунктуации.
* Замена знаков пунктуации на пробелы или специальные маркеры.
* Замена повторяющихся знаков пунктуации на один знак.

**Нормализация регистра** обычно включает в себя приведение всех символов к нижнему регистру (lowercase), чтобы исключить различия в написании слов из-за использования разных регистров. (рис 1)

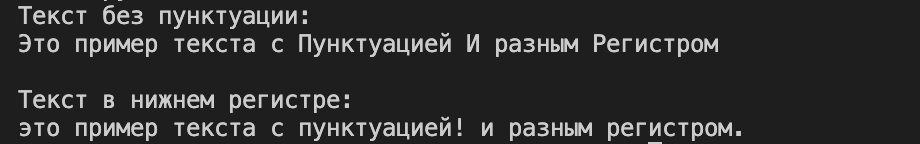


Рисунок 1 – пример выполнения нормализации регистра

1. **Стемминг**

Это процесс обрезки слов до их основы или корня. Основная цель стемминга состоит в том, чтобы убрать окончания и суффиксы, чтобы получить основу слова. Даже если это может привести к тому, что некоторые слова будут обрезаны неправильно и могут не существовать, стемминг — это простой и быстрый метод нормализации текста. (рис 2)



Рисунок 2 – пример выполнения стемминга

1. **Лемматизация**

Это процесс приведения слов к их базовой или словарной форме, называемой леммой. В отличие от стемминга, лемматизация учитывает грамматические формы слова и его семантический контекст, что делает её более точной и полезной для задач обработки текста. (рис 3)



Рисунок 3 – пример выполнения лемматизации

1. **Удаление стоп-слов**

Во всех текстах присутствуют слова, которые не несут смысловой нагрузки. Эти слова и являются стоп словами. Их необходимо удалить из общего набора слов, тк их наличие ничего не изменит, а только приведёт к аномалиям. Такие слова включают в себя предлоги, союзы, местоимения, артикли и другие части речи, которые часто встречаются в языке, но не несут полезной информации для анализа текста.

Стоп-слова могут варьироваться в зависимости от языка и конкретного приложения. Включение этих слов в анализ может привести к шуму и снижению качества результатов. Поэтому удаление стоп-слов является важным этапом предобработки текста в задачах обработки естественного языка.

Пример списка русских стоп-слов:

* предлоги: в, на, по, с, у
* союзы: и, но, а, как, что
* местоимения: он, она, оно, они

## **Используемые технологии**

В качестве языка разработки выбран язык **Python.**

**Python** – это высокоуровневый функциональный язык программирования, который пользуется большой популярность в силу лёгкой семантики и большого спектра применений. Значительным плюсом этого языка является то, что для решения многих типовых задач используется огромное количество библиотек.

**Jupyter Notebook** выбран в качестве основного инструмента для проведения анализа данных, разработки алгоритмов и представления результатов моего исследования.

**Jupyter Notebook** — это интерактивная среда, которая позволяет создавать и выполнять код в различных языках программирования, включая Python. Он предоставляет удобный способ интеграции кода, текста, изображений и результатов выполнения кода в единый документ.

## **Описание алгоритмов машинного обучения**

Алгоритм машинного обучения — это набор правил или процессов, используемых системой ИИ для выполнения задач — чаще всего для обнаружения новых данных и закономерностей или для прогнозирования выходных значений на основе заданного набора входных переменных. Алгоритмы позволяют машинному обучению (ML) учиться.

Существует четыре типа алгоритмов машинного обучения: контролируемые, неконтролируемые, полуконтролируемые и с подкреплением. В зависимости от разных факторов каждый вариант имеет свои преимущества. Для продвинутых алгоритмов машинного обучения требуется множество технологий, включая глубокое обучение, нейронные сети и обработку естественного языка, и они могут использовать как обучение без учителя, так и обучение с учителем.

Обучение с учителем делится на задачи классификации и регрессии.

Алгоритм классификации распределяет данные по соответствующим категория. Сначала он выделяет определенные объекты и формирует о них вывод, чтобы понимать, каким образом их маркировать.

Алгоритм регрессии применяется для анализа взаимосвязи между зависимыми и независимыми переменными. Его часто применяют в сферах прогнозирования, например, в продажах акций.

Алгоритмы обучения с учителем:

* **Логистическая регрессия.**

Логистическая регрессия применяется для решения проблем двоичной классификации. Ее применяют, когда зависимая переменная является категориальной, что означает наличие двоичных выходных данных, таких как 1 и 0 (да и нет).

* **Дерево решений.**

Деревья решений используют ветвящуюся последовательность связанных решений, которые могут быть представлены в виде древовидной диаграммы. Одним из преимуществ деревьев решений является то, что их легко проверять.

* **Случайный лес.**

В случайном лесу алгоритм машинного обучения прогнозирует значение или категорию путем объединения результатов нескольких деревьев решений.

* **Метод опорных векторов (SVM).**

Используется для обеих задач, но чаще для задач классификации. Этот метод формирует гиперплоскость, в которой расстояние между двумя классами точек данных максимально. Она называется границей решения, тк разделяет классы точек данных по разные стороны плоскости.

* **Наивный Байес.**

Данный метод основан на теореме Байеса, а именно на принципе условной независимости класса. Разные признаки и их наличие не влияют на вероятность результата, все имеют одинаковое влияние на конечный итог. Часто этот метод применяется для решения задач классификации текста и в рекомендательных системах.

* **K-ближайший соседей.**

Алгоритм KNN классифицирует точки данных на основе их близости и связи с другими доступными данными. В методе лежит идея, что схожие точки находятся рядом друг с другом. В итоге происходит вычисление расстояния между точками и назначается категория, которая чаще встречается или берется среднее значение.

* **Градиентный бустинг.**

Этот метод, также называемый адаптивным повышением, усиливает неэффективный алгоритм регрессии, объединяя его с более слабыми алгоритмами для создания более сильного алгоритма, который приводит к меньшему количеству ошибок. Бустинг сочетает в себе прогнозирующую силу нескольких базовых оценщиков.

# **Обработка данных**

Рассмотрим процессе обработки данных, который является важным этапом перед применением методов нормализации текста. Обработка данных включает в себя ряд шагов, направленных на подготовку текстовых данных для дальнейшего анализа и применения методов нормализации. В данной главе мы рассмотрим основные этапы обработки данных и обсудим их значимость в контексте работы с текстом.

Шаги обработки данных:

* **Удаление шума**

Очистка текста от нежелательных символов, таких как пунктуация, специальные символы и цифры. Это позволяет упростить текст и сделать его более однородным для последующего анализа.

* **Нормализация регистра**

Приведение всех символов к нижнему или верхнему регистру. Это позволяет избежать дублирования, которые написаны по-разному.

* **Удаление стоп-слов**

Удаление слов без смысловой нагрузки, например, предлоги, союзы и местоимения. Это позволяет сосредоточиться на более важных словах в тексте.

* **Стемминг и лемматизация**

Приведение слов к их базовой форме (стемминг) или к словарной форме (лемматизация). Это помогает сократить варианты написания слова и сделать анализ текста более точным.

Обработка данных играет ключевую роль в успешном анализе текста и применении методов нормализации. Чистые и однородные данные позволяют точнее и эффективнее извлекать информацию и понимать смысл текста. Неправильная обработка данных или пропуск этого этапа может привести к искажению результатов и неправильным выводам.

## **Сбор и очистка данных**

Рассмотрим процесс сбора и очистки данных, который является важным этапом в подготовке данных для последующего анализа и нормализации текста. Правильная обработка и очистка данных позволяет получить качественные и однородные данные, что способствует более точному и эффективному анализу текста.

В качестве исследуемых данных был выбран крупнейший русскоязычный датасет отзывов об организациях, опубликованных на Яндекс Картах.

Описание датасета:

* 500 000 уникальных отзывов
* Только отзывы на организации в России
* Доступны на Яндекс Картах
* Опубликованы с января по июль 2023 года
* Датасет не содержит коротких односложных отзывов
* Тексты очищены от персональных данных (номеров телефонов, адресов почты)

Датасет в формате tskv содержит следующую информацию:

* Адрес организации (address)
* Название организации (name\_ru)
* Список рубрик, к которым относится организация (rubrics)
* Оценка пользователя от 0 до 5 (rating)
* Текст отзыва (text)

Для анализа размер датасата был уменьшен, а также будут использованы

только тексты отзывов и их рейтинг. (рис 4)

def read\_tskv(file\_path, limit=50):

data = []

with open(file\_path, 'r', encoding='utf-8') as file:

count = 0

for line in file:

line\_data = {}

pairs = line.strip().split('\t')

for pair in pairs:

if '=' in pair:

key, value = pair.split('=', 1)

line\_data[key] = value

if line\_data:

data.append(line\_data)

count += 1

if count >= limit:

break

return [entry['text'] for entry in data]

file\_path = './geo-reviews-dataset-2023.tskv'

texts = read\_tskv(file\_path)

Рисунок 4 – сбор данных

Этот код читает строки из файла формата TSKV (табулятор-разделённые ключ-значения), извлекает значения по ключу 'text' из каждой строки и возвращает их в виде списка. Функция открывает файл, разбивает строки на пары ключ-значение, сохраняет данные в словари и формирует итоговый список текстов. (рис 5)

import string

import re

def clean\_text(text):

text = re.sub(r'\s+', ' ', text)

text = text.replace('\\n', ' ') # заменяем '\n' на пробел

# Убираем цифры

text = re.sub(r'\d+', '', text)

# Приводим текст к нижнему регистру

text = text.lower()

# Убираем пунктуацию

text = text.translate(str.maketrans('', '', string.punctuation))

return text

# Очистка текстов

cleaned\_texts = [clean\_text(text) for text in texts]

Рисунок 5 – очистка данных

Этот код определяет функцию **clean\_text**, которая принимает текст в качестве аргумента и возвращает очищенный текст, приведенный к нижнему регистру и без пунктуации. Эта функция применяется к каждому тексту из датасета, а результаты сохраняется в список **cleaned\_texts.**

## **Удаление стоп-слов**

**Стоп-слова** — это слова без смысловой нагрузки, которые часто встречаются в тексте, такие как предлоги, союзы и местоимения. Удаление стоп-слов позволяет сосредоточиться на более важных словах в тексте и улучшить качество последующего анализа. (рис 6)

from nltk.corpus import stopwords

russian\_stopwords = stopwords.words('russian')

# Функция для очистки текста

def clean\_text(text):

text = ' '.join([word for word in text.split() if word not in stopwords.words('russian')])

return text

cleaned\_texts = [clean\_text(text) for text in texts]

Рисунок 6 – удаление стоп-слов

Функция **clean\_text** принимает текст и возвращает его версию без стоп-слов. Затем мы применили эту функцию к каждому тексту в списке **texts**, результаты сохранены в списке **cleaned\_texts.**

## **Стемминг и лемматизация**

Рассмотрим процессы стемминга и лемматизации текста, которые направлены на приведение слов к их базовой форме. Стемминг и лемматизация помогают сократить варианты написания слова и улучшить точность анализа текста. (рис 7)

# Функция для лемматизации текста

def lemmatize\_text(text):

words = text.split()

lemmatized\_words = [morph.parse(word)[0].normal\_form for word in words]

return ' '.join(lemmatized\_words)

# Функция для стемминга текста

def stem\_text(text):

words = text.split()

stemmed\_words = [stemmer.stem(word) for word in words]

return ' '.join(stemmed\_words)

Рисунок 7 – лемматизация и стемминг

В этом коде мы определили функции **lemmatize\_text** и **stem\_text**, которые принимают текст и возвращают его версии после лемматизации и стемминга соответственно. Затем мы применили эти функции к каждому тексту в списке **cleaned\_texts**, результаты сохранены в списках **lemmatized\_texts** и **stemmed\_texts**, и каждый текст выводится на экран. (рис 8)

lemmatized\_text = lemmatize\_text(cleaned\_texts[2])

stemmed\_text = stem\_text(cleaned\_texts[2])

# Вывод результата

print("Очищенный текст:", cleaned\_texts[2])

print("Лемматизированный текст:", lemmatized\_text)

print("Стеммированный текст:", stemmed\_text)

Рисунок 8 – пример выполнения для 1 текста

**Результаты:**

Очищенный текст*: не знаю смутят ли когото данные правила но я была удивлена хочешь что бы твой шкаф замыкался купи замочек ты должен предоставить свой отпечаток пальца полнейшая дичь ставят подпись на договоре с клиентом по доверенности графу с номером доверенности оставляют пустой а на вопрос о номере доверенности говорят номер «» вы серьезно номер предоставить доверенность не могут но говорят что у них в клубе «свои» доверенности типа особенные какието цирк*

Лемматизированный текст: *не знать смутить ли коготь дать правило но я быть удивить хотеть что бы твой шкаф замыкаться купить замочка ты должный предоставить свой отпечаток палец полный дичь ставить подпись на договор с клиент по доверенность графа с номер доверенность оставлять пустой а на вопрос о номер доверенность говорить номер «» вы серьёзно номер предоставить доверенность не мочь но говорить что у они в клуб «свои» доверенность тип особенный какиеть цирк*

Стеммированный текст: *не зна смут ли когот дан прав но я был удивл хочеш что бы тво шкаф замыка куп замочек ты долж предостав сво отпечаток пальц полн дич став подп на договор с клиент по доверен граф с номер доверен оставля пуст а на вопрос о номер доверен говор номер «» вы серьезн номер предостав доверен не могут но говор что у них в клуб «свои» доверен тип особен какиет цирк*

Лемматизация в данном случае работает лучше, так как она сохраняет смысл и структуру текста, делая его более понятным и читабельным. Стемминг, несмотря на его полезность в некоторых задачах, обрезает слова слишком агрессивно, что значительно ухудшает читабельность и точность восприятия текста.

# **Разработка и обучение модели**

Разработка и обучение модели машинного обучения представляют собой многослойный процесс, включающий несколько ключевых этапов, каждый из которых имеет свои цели и методы. Этот процесс начинается с выбора подходящей архитектуры модели и заканчивается оценкой её производительности.

**Разработка модели включает в себя следующие шаги:**

* определении задачи, которую нужно решить. В нашем случае, это задача нормализации слов текста, например, приведение слов к их каноническим формам.
* Выбор архитектуры модели. Это может быть правило-основанный подход, статистическая модель, или нейронная сеть.
* Определение характеристик модели. Это включает выбор гиперпараметров, структуры слоёв (для нейронных сетей), функции активации, и других параметров, которые определяют, как модель будет работать.

**Обучение модели включает несколько этапов:**

* Сбор и подготовка данных. Тексты токенизируются, очищаются, и аннотируются (например, разметка лемм или стеммов).
* Разделение данных на обучающую, валидационную и тестовую выборки.
* Инициализация модели. Модель инициализируется с начальными значениями параметров.
* Обучение модели. Модель обучается на обучающей выборке с использованием различных алгоритмов
* Подбор гиперпараметров. На этом этапе производится настройка гиперпараметров, таких как размер скрытых слоёв, количество слоёв, параметры оптимизатора.
* Регуляризация. Применяются методы регуляризации для улучшения обобщающей способности модели и предотвращения переобучения.

**Оценка и анализ модели:**

* Оценка модели. Модель оценивается на тестовой выборке с использованием различных метрик, таких как точность, полнота, F1-мера. Кросс-валидация может использоваться для получения более стабильной оценки производительности модели.
* Анализ ошибок. Анализ ошибок помогает выявить типичные ошибки модели и определить, какие аспекты модели могут быть улучшены.
* Интерпретация и использование результатов. Интерпретация результатов позволяет понять, насколько хорошо модель справляется с поставленной задачей.

## **Применение алгоритмов машинного обучения**

Для классификации извлекаем рейтинги из датасета и преобразуем их в бинарные метки для дальнейшего использования в моделях машинного обучения. Отзывы с рейтингом 4 и выше обозначаются как положительные (1), а остальные — как отрицательные (0).

Разделяем данные на обучающие и тестовые выборки для двух типов текстов: лемматизированных и стеммированных. **train\_test\_split** используется для разбиения данных, где 80% данных используется для обучения, а 20% — для тестирования. (рис 9)

# Разделение данных для лемматизированных текстов

X\_train\_lem, X\_test\_lem, y\_train\_lem, y\_test\_lem = train\_test\_split(X\_lemmatized, y, test\_size=0.2, random\_state=0)

# Разделение данных для стеммированных текстов

X\_train\_stem, X\_test\_stem, y\_train\_stem, y\_test\_stem = train\_test\_split(X\_stemmed, y, test\_size=0.2, random\_state=0)

Рисунок 9 – разделение данных

Определяем различные модели машинного обучения. В данной работе мы решаем задачу классификации, поэтому будем использовать основные методы обучения с учителем.

Далее обучаем модель на лемматизированных и стеммированных данных.

Собираем результаты точности в словари и выводим. Далее эти результаты нам понадобятся для выявления наиболее эффективного метода. (рис 10)

models = {

'Logistic Regression': LogisticRegression(max\_iter=1000),

'Decision Tree': DecisionTreeClassifier(),

'Random Forest': RandomForestClassifier(),

'SVM': SVC(),

'Naive Bayes': MultinomialNB(),

'KNN': KNeighborsClassifier(n\_neighbors=3),

'Gradient Boosting': GradientBoostingClassifier()

}

results\_lemmatized = {}

results\_stemmed = {}

# Обучение и оценка моделей для лемматизированных данных

for model\_name, model in models.items():

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

accuracy\_lem = model.score(X\_test\_lem, y\_test\_lem)

results\_lemmatized[model\_name] = accuracy\_lem

# Обучение и оценка моделей для стеммированных данных

for model\_name, model in models.items():

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

accuracy\_stem = model.score(X\_test\_stem, y\_test\_stem)

results\_stemmed[model\_name] = accuracy\_stem

print('Results for Lemmatized Data:', results\_lemmatized)

print('Results for Stemmed Data:', results\_stemmed)

Рисунок 10 – обучение моделей

**Результаты:**

Results for Lemmatized Data: {'Logistic Regression': 0.905, 'Decision Tree': 0.85, 'Random Forest': 0.905, 'SVM': 0.905, 'Naive Bayes': 0.905, 'KNN': 0.87, 'Gradient Boosting': 0.915}

Results for Stemmed Data: {'Logistic Regression': 0.905, 'Decision Tree': 0.84, 'Random Forest': 0.905, 'SVM': 0.905, 'Naive Bayes': 0.905, 'KNN': 0.88, 'Gradient Boosting': 0.93}

Модели, обученные на лемматизированных данных, показали высокие показатели точности, преимущественно в диапазоне от 0.85 до 0.92. Логистическая регрессия, случайный лес, SVM и наивный байес демонстрируют высокую точность на уровне 0.905, что свидетельствует о их способности эффективно классифицировать тексты. Метод k-ближайших соседей имеет точность 0.87, что также является хорошим показателем. Дерево решений показывает немного более низкую точность, составляющую 0.85, в то время как градиентный бустинг демонстрирует наивысший результат с точностью 0.915.

Модели, обученные на стеммированных данных, также показывают высокие показатели точности, в основном в диапазоне от 0.84 до 0.93. Логистическая регрессия, случайный лес, SVM и наивный байес сохраняют высокую точность на уровне 0.905, что подтверждает их эффективность. Метод k-ближайших соседей имеет точность 0.88, а градиентный бустинг демонстрирует наивысший результат с точностью 0.93.

## **Повышение эффективности модели**

В качестве перспективной модели был выбран градиентный бустинг, который продемонстрировал наивысший результат.

Проводим поиск лучших гиперпараметров для модели градиентного бустинга с использованием метода Grid Search. Мы определяем набор гиперпараметров, который будет исследоваться, затем применяем Grid Search с кросс-валидацией для выбора наилучших параметров. Результаты выводятся на экран, показывая лучшие параметры для модели. (рис 11)

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import GridSearchCV

# Выбор модели (Gradient Boosting)

model = GradientBoostingClassifier()

# Определение гиперпараметров для поиска

param\_grid = {

'n\_estimators': [50, 100, 200],

'learning\_rate': [0.01, 0.1, 0.5],

'max\_depth': [3, 5, 7],

'min\_samples\_split': [2, 5, 10]

}

# Проведение Grid Search

grid\_search = GridSearchCV(model, param\_grid, cv=2, n\_jobs=-1, verbose=1)

grid\_search.fit(X\_train\_stem, y\_train\_stem)

# Лучшие параметры

best\_params = grid\_search.best\_params\_

print('Лучшие параметры:', best\_params)

Рисунок 11 – поиск лучших параметров

**Результат:**

Лучшие параметры: {'learning\_rate': 0.1, 'max\_depth': 3, 'min\_samples\_split': 10, 'n\_estimators': 50}

best\_model = GradientBoostingClassifier(\*\*best\_params)

best\_model.fit(X\_train\_stem, y\_train\_stem)

# Оценка модели

accuracy = best\_model.score(X\_test\_stem, y\_test\_stem)

print('Точность улучшенной модели:', accuracy)

Рисунок 12 – оценка модели

**Результат:**

Точность улучшенной модели: 0.93

Проводим последовательную обработку текстовых данных для подготовки их к классификации. Сначала мы применяем метод главных компонент (PCA) для уменьшения размерности данных до 95% сохраненной дисперсии. Затем мы обратно преобразуем данные из пространства главных компонент обратно в исходное пространство текстовых данных. После этого мы преобразуем данные в строки и применяем векторизацию с помощью TF-IDF векторизатора. Затем данные разделяются на обучающую и тестовую выборки, и модель градиентного бустинга обучается на данных, подготовленных с использованием лемматизации. Точность модели оценивается на тестовой выборке и выводится на экран. (рис 13)

# Применение PCA до векторизации

pca\_lem = PCA(n\_components=0.95)

pca\_stem = PCA(n\_components=0.95)

X\_reduced\_lem = pca\_lem.fit\_transform(X\_lemmatized)

X\_reduced\_stem = pca\_stem.fit\_transform(X\_stemmed)

# Обратное преобразование PCA в текстовые данные

X\_inverse\_lem = pca\_lem.inverse\_transform(X\_reduced\_lem)

X\_inverse\_stem = pca\_stem.inverse\_transform(X\_reduced\_stem)

# Преобразование массивов numpy в строки

X\_text\_lem = [' '.join(row) for row in X\_inverse\_lem.astype(str)]

X\_text\_stem = [' '.join(row) for row in X\_inverse\_stem.astype(str)]

# Векторизация

tfidf = TfidfVectorizer(stop\_words=russian\_stopwords, max\_features=5000)

X\_final\_lem = tfidf.fit\_transform(X\_text\_lem).toarray()

X\_final\_stem = tfidf.fit\_transform(X\_text\_stem).toarray()

# Разделение данных и обучение модели

X\_train\_final\_lem, X\_test\_final\_lem, y\_train, y\_test = train\_test\_split(X\_final\_lem, y, test\_size=0.2, random\_state=0)

X\_train\_final\_stem, X\_test\_final\_stem, y\_train, y\_test = train\_test\_split(X\_final\_stem, y, test\_size=0.2, random\_state=0)

best\_model = GradientBoostingClassifier(\*\*best\_params)

best\_model.fit(X\_train\_final\_lem, y\_train)

accuracy = best\_model.score(X\_test\_final\_lem, y\_test)

print('Accuracy with changed preprocessing order (Lemmatized Data):', accuracy)

Рисунок 13 – пример выполнения PCA

**Результат:**

Accuracy with changed preprocessing order (Lemmatized Data): 0.905

## **Анализ результатов**

Оба метода предобработки текста (лемматизация и стемминг) показали хорошие результаты, приводя к высоким показателям точности моделей. Однако стемминг продемонстрировал небольшое превосходство в точности по сравнению с лемматизацией. (рис 14)

Модель градиентного бустинга показала наивысшие результаты на обоих видах данных. Ее высокая точность и способность обобщения делают ее хорошим выбором для данной задачи классификации текста.

После применения Grid Search точность модели на тестовой выборке составила 0.93, что означает, что примерно 93% тестовых данных были правильно классифицированы моделью. Это высокий показатель точности и свидетельствует о хорошей способности модели к обобщению на новых данных. (рис 15)

Использование PCA уменьшило размерности данных до сохранения 95% исходной дисперсии. Это позволило уменьшить количество признаков, что может ускорить обучение модели и снизить риск переобучения.

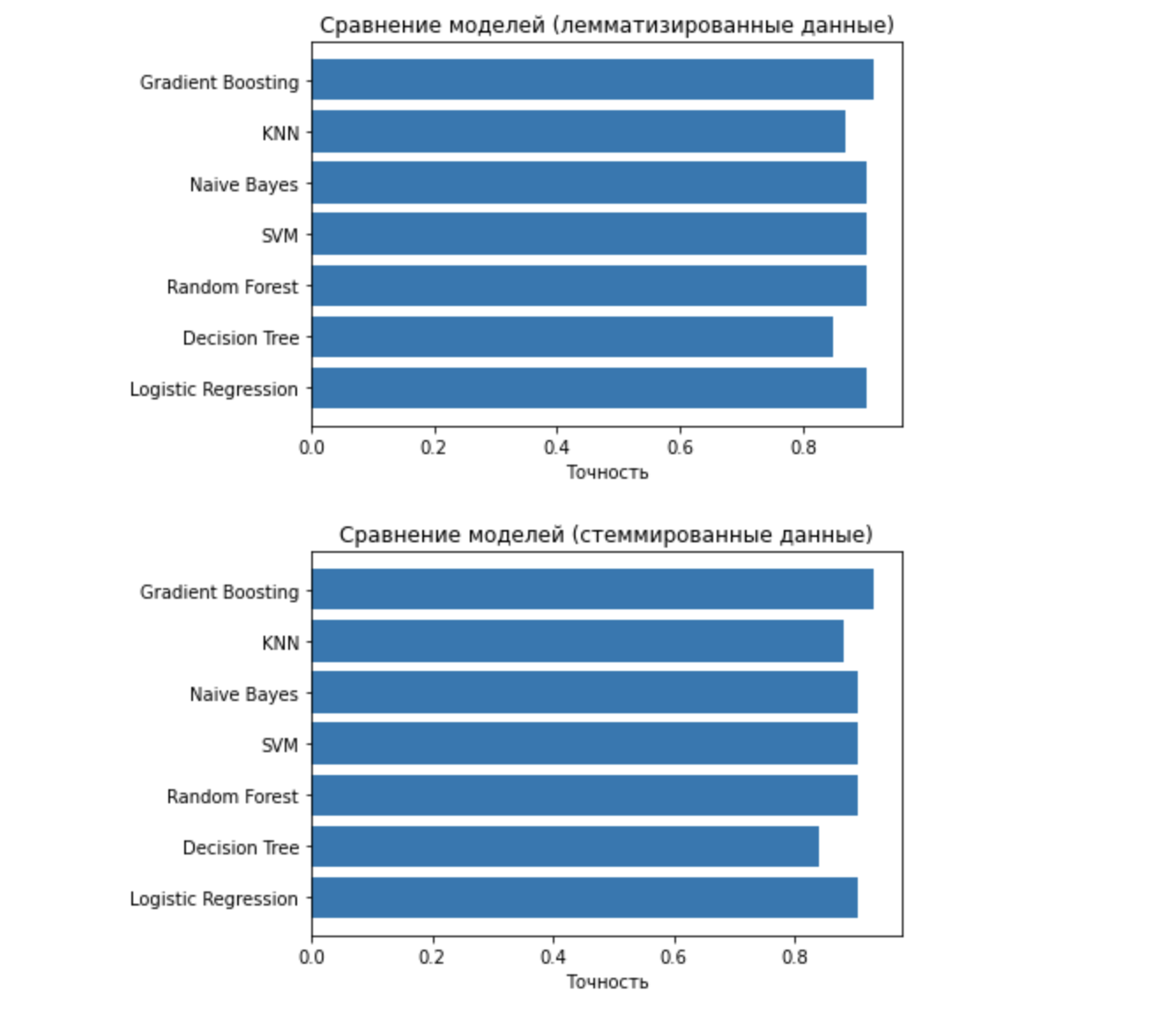


Рисунок 14 – сравнение моделей

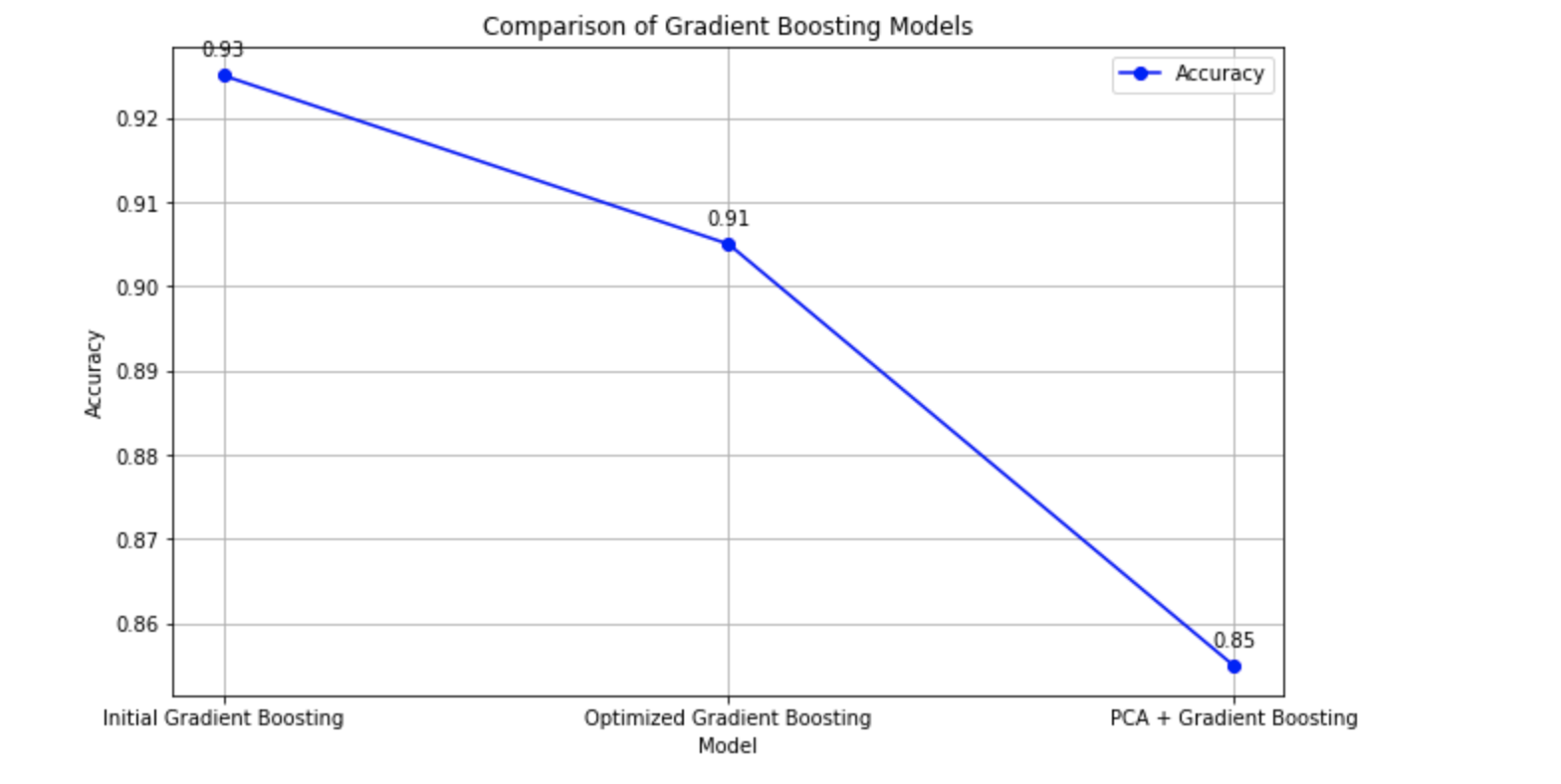


Рисунок 15 – сравнение Gradient Boosting

# **Заключение**

Исходя из результатов проведенной работы, можно сделать вывод о том, что модели машинного обучения, обученные на предварительно нормализованных текстовых данных с использованием методов лемматизации и стемминга, способны эффективно классифицировать тексты. В процессе работы были рассмотрены различные методы предобработки текста, и исследовано их влияние на производительность моделей машинного обучения.

Одним из ключевых шагов был выбор подходящей модели машинного обучения. Было проведено сравнение производительности нескольких моделей. Наиболее эффективными моделями в данной задаче оказались модели градиентного бустинга.

Для оптимизации производительности моделей был применен Grid Search для подбора оптимальных гиперпараметров. Также было исследовано влияние метода главных компонент (PCA) на производительность моделей.

Во время выполнения курсовой работы были выполнены **задачи**:

* Разработаны методы нормализации текста.
* Проведено разделение данных и обучение моделей
* Проведены экспериментов для оценки качества разработанных моделей на тестовой выборке данных
* Предоставлены результатов в виде графиков

# **Список литературы и источники**

1. М.В. Коротеев. Об основных задачах дескриптивного анализа данных.
2. М.В. Коротеев. Учебное пособие по дисциплине “Анализ данных и машинное обучение” - 2018.
3. Lane, Hobson, and Cole. Natural Language Processing in Action – 2019.
4. Sarkar. Text Analytics with Python – 2018.
5. A. Geron. Hand on Machine Learning with scikit-learn and Tensorflow – 2017
6. J. Grus. Data science from scratch - 2015
7. <https://github.com/yandex/geo-reviews-dataset-2023>
8. <https://github.com/vkosuri/CourseraMachineLearning>
9. <https://scikit-learn.org/stable/index.html>
10. https://www.python.org/

# **Приложение**

pip install pymorphy2 nltk

**def** read\_tskv(file\_path, limit**=**1000):

data **=** []

**with** open(file\_path, 'r', encoding**=**'utf-8') **as** file:

count **=** 0

**for** line **in** file:

line\_data **=** {}

pairs **=** line**.**strip()**.**split('\t')

**for** pair **in** pairs:

**if** '=' **in** pair:

key, value **=** pair**.**split('=', 1)

line\_data[key] **=** value

**if** line\_data:

data**.**append(line\_data)

count **+=** 1

**if** count **>=** limit:

**break**

**return** [entry['text'] **for** entry **in** data]

file\_path **=** './geo-reviews-dataset-2023.tskv'

texts **=** read\_tskv(file\_path)

**import** string

**import** re

**def** clean\_text(text):

text **=** re**.**sub(r'\s+', ' ', text) *# заменяем все последовательности пробельных символов на один пробел*

text **=** text**.**replace('\\n', ' ') *# заменяем '\n' на пробел*

*# Убираем цифры*

text **=** re**.**sub(r'\d+', '', text)

*# Приводим текст к нижнему регистру*

text **=** text**.**lower()

*# Убираем пунктуацию*

text **=** text**.**translate(str**.**maketrans('', '', string**.**punctuation))

**return** text

*# Очистка текстов*

cleaned\_texts **=** [clean\_text(text) **for** text **in** texts]

**import** pymorphy2

**from** nltk.stem.snowball **import** SnowballStemmer

**from** nltk.corpus **import** stopwords

*# Инициализация морфологического анализатора и стеммера*

morph **=** pymorphy2**.**MorphAnalyzer()

stemmer **=** SnowballStemmer("russian")

*# загрузка стоп слов*

russian\_stopwords **=** stopwords**.**words('russian')

*# Функция для лемматизации текста*

**def** lemmatize\_text(text):

words **=** text**.**split()

lemmatized\_words **=** [morph**.**parse(word)[0]**.**normal\_form **for** word **in** words]

**return** ' '**.**join(lemmatized\_words)

*# Функция для стемминга текста*

**def** stem\_text(text):

words **=** text**.**split()

stemmed\_words **=** [stemmer**.**stem(word) **for** word **in** words]

**return** ' '**.**join(stemmed\_words)

**def** tokenize\_text(text):

**return** ' '**.**join(text**.**split())

*# пример отработки стеммнга / токенизации и лематизации на 1 тексте*

lemmatized\_text **=** lemmatize\_text(cleaned\_texts[2])

stemmed\_text **=** stem\_text(cleaned\_texts[2])

tokenized\_text **=** tokenize\_text(cleaned\_texts[2])

*# Вывод результата*

print("Очищенный текст:", cleaned\_texts[2])

print("Лемматизированный текст:", lemmatized\_text)

print("Стеммированный текст:", stemmed\_text)

print("Токенизированный текст:", tokenized\_text)

Очищенный текст: не знаю смутят ли когото данные правила но я была удивлена хочешь что бы твой шкаф замыкался купи замочек ты должен предоставить свой отпечаток пальца полнейшая дичь ставят подпись на договоре с клиентом по доверенности графу с номером доверенности оставляют пустой а на вопрос о номере доверенности говорят номер «» вы серьезно номер предоставить доверенность не могут но говорят что у них в клубе «свои» доверенности типа особенные какието цирк

Лемматизированный текст: не знать смутить ли коготь дать правило но я быть удивить хотеть что бы твой шкаф замыкаться купить замочка ты должный предоставить свой отпечаток палец полный дичь ставить подпись на договор с клиент по доверенность графа с номер доверенность оставлять пустой а на вопрос о номер доверенность говорить номер «» вы серьёзно номер предоставить доверенность не мочь но говорить что у они в клуб «свои» доверенность тип особенный какиеть цирк

Стеммированный текст: не зна смут ли когот дан прав но я был удивл хочеш что бы тво шкаф замыка куп замочек ты долж предостав сво отпечаток пальц полн дич став подп на договор с клиент по доверен граф с номер доверен оставля пуст а на вопрос о номер доверен говор номер «» вы серьезн номер предостав доверен не могут но говор что у них в клуб «свои» доверен тип особен какиет цирк

Токенизированный текст: не знаю смутят ли когото данные правила но я была удивлена хочешь что бы твой шкаф замыкался купи замочек ты должен предоставить свой отпечаток пальца полнейшая дичь ставят подпись на договоре с клиентом по доверенности графу с номером доверенности оставляют пустой а на вопрос о номере доверенности говорят номер «» вы серьезно номер предоставить доверенность не могут но говорят что у них в клубе «свои» доверенности типа особенные какието цирк

*# Применение лемматизации к очищенным текстам*

lemmatized\_texts **=** [lemmatize\_text(text) **for** text **in** cleaned\_texts]

stemmed\_texts **=** [stem\_text(text) **for** text **in** cleaned\_texts]

**import** nltk

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

*# Инициализация векторизатора с использованием собственного списка стоп-слов*

tfidf **=** TfidfVectorizer(stop\_words**=**russian\_stopwords, max\_features**=**5000)

*# Преобразование текстов*

X\_lemmatized **=** tfidf**.**fit\_transform(lemmatized\_texts)**.**toarray()

X\_stemmed **=** tfidf**.**fit\_transform(stemmed\_texts)**.**toarray()

print(X\_lemmatized**.**shape)

print(X\_stemmed**.**shape)

(1000, 5000)

(1000, 5000)

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** pandas **as** pd

*# Построение корелляционной матрицы для лемматизированных данных*

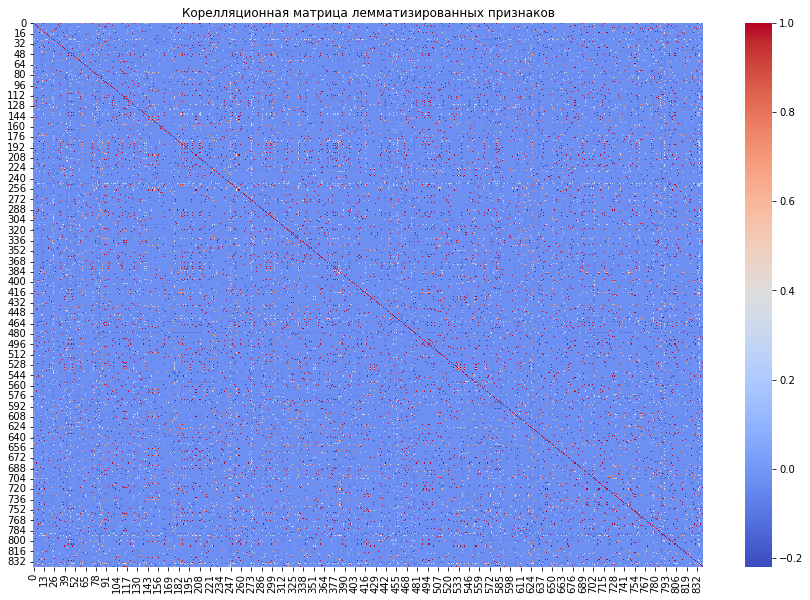
plt**.**figure(figsize**=**(15, 10)) *# Задаем размеры фигуры*

correlation\_matrix\_lemmatized **=** pd**.**DataFrame(X\_lemmatized)**.**corr()

sns**.**heatmap(correlation\_matrix\_lemmatized, cmap**=**'coolwarm')

plt**.**title('Корелляционная матрица лемматизированных признаков')

plt**.**show()



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

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.svm **import** SVC *# Пример альтернативной модели*

*# 1 - положительный отзыв, 0 - отрицательный отзыв*

**def** read\_ratings(file\_path, limit**=**1000):

data **=** []

**with** open(file\_path, 'r', encoding**=**'utf-8') **as** file:

count **=** 0

**for** line **in** file:

line\_data **=** {}

pairs **=** line**.**strip()**.**split('\t')

**for** pair **in** pairs:

**if** '=' **in** pair:

key, value **=** pair**.**split('=', 1)

line\_data[key] **=** value

**if** line\_data:

data**.**append(line\_data)

count **+=** 1

**if** count **>=** limit:

**break**

*# Преобразование рейтингов в целые числа и замена значений*

ratings **=** []

**for** entry **in** data:

rating **=** round(float(entry['rating']))

**if** rating **>=** 4:

ratings**.**append(1)

**else**:

ratings**.**append(0)

**return** ratings

*# Извлечение бинарных меток из файла*

y **=** read\_ratings(file\_path)

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

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.svm **import** SVC

**from** sklearn.naive\_bayes **import** MultinomialNB

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.ensemble **import** GradientBoostingClassifier

*# Разделение данных для лемматизированных текстов*

X\_train\_lem, X\_test\_lem, y\_train\_lem, y\_test\_lem **=** train\_test\_split(X\_lemmatized, y, test\_size**=**0.2, random\_state**=**0)

*# Разделение данных для стеммированных текстов*

X\_train\_stem, X\_test\_stem, y\_train\_stem, y\_test\_stem **=** train\_test\_split(X\_stemmed, y, test\_size**=**0.2, random\_state**=**0)

*# Определение моделей*

models **=** {

'Logistic Regression': LogisticRegression(max\_iter**=**1000),

'Decision Tree': DecisionTreeClassifier(),

'Random Forest': RandomForestClassifier(),

'SVM': SVC(),

'Naive Bayes': MultinomialNB(),

'KNN': KNeighborsClassifier(n\_neighbors**=**3),

'Gradient Boosting': GradientBoostingClassifier()

}

results\_lemmatized **=** {}

results\_stemmed **=** {}

*# Обучение и оценка моделей для лемматизированных данных*

**for** model\_name, model **in** models**.**items():

model**.**fit(X\_train\_lem, y\_train\_lem)

accuracy\_lem **=** model**.**score(X\_test\_lem, y\_test\_lem)

results\_lemmatized[model\_name] **=** accuracy\_lem

*# Обучение и оценка моделей для стеммированных данных*

**for** model\_name, model **in** models**.**items():

model**.**fit(X\_train\_stem, y\_train\_stem)

accuracy\_stem **=** model**.**score(X\_test\_stem, y\_test\_stem)

results\_stemmed[model\_name] **=** accuracy\_stem

print('Results for Lemmatized Data:', results\_lemmatized)

print('Results for Stemmed Data:', results\_stemmed)

Results for Lemmatized Data: {'Logistic Regression': 0.905, 'Decision Tree': 0.85, 'Random Forest': 0.905, 'SVM': 0.905, 'Naive Bayes': 0.905, 'KNN': 0.87, 'Gradient Boosting': 0.915}

Results for Stemmed Data: {'Logistic Regression': 0.905, 'Decision Tree': 0.84, 'Random Forest': 0.905, 'SVM': 0.905, 'Naive Bayes': 0.905, 'KNN': 0.88, 'Gradient Boosting': 0.93}

**from** sklearn.ensemble **import** GradientBoostingClassifier

**from** sklearn.model\_selection **import** GridSearchCV

*# Выбор модели (Gradient Boosting)*

model **=** GradientBoostingClassifier()

*# Определение гиперпараметров для поиска*

param\_grid **=** {

'n\_estimators': [50, 100, 200],

'learning\_rate': [0.01, 0.1, 0.5],

'max\_depth': [3, 5, 7],

'min\_samples\_split': [2, 5, 10]

}

*# Проведение Grid Search*

grid\_search **=** GridSearchCV(model, param\_grid, cv**=**2, n\_jobs**=-**1, verbose**=**1)

grid\_search**.**fit(X\_train\_stem, y\_train\_stem)

*# Лучшие параметры*

best\_params **=** grid\_search**.**best\_params\_

print('Лучшие параметры:', best\_params)

Fitting 2 folds for each of 81 candidates, totalling 162 fits

Лучшие параметры: {'learning\_rate': 0.1, 'max\_depth': 3, 'min\_samples\_split': 10, 'n\_estimators': 50}

best\_model **=** GradientBoostingClassifier(**\*\***best\_params)

best\_model**.**fit(X\_train\_stem, y\_train\_stem)

*# Оценка модели*

accuracy **=** best\_model**.**score(X\_test\_stem, y\_test\_stem)

print('Точность улучшенной модели:', accuracy)

Точность улучшенной модели: 0.93

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

**from** sklearn.decomposition **import** PCA

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

**from** sklearn.ensemble **import** RandomForestClassifier

*# Применение PCA до векторизации*

pca\_lem **=** PCA(n\_components**=**0.95)

pca\_stem **=** PCA(n\_components**=**0.95)

X\_reduced\_lem **=** pca\_lem**.**fit\_transform(X\_lemmatized)

X\_reduced\_stem **=** pca\_stem**.**fit\_transform(X\_stemmed)

*# Обратное преобразование PCA в текстовые данные*

X\_inverse\_lem **=** pca\_lem**.**inverse\_transform(X\_reduced\_lem)

X\_inverse\_stem **=** pca\_stem**.**inverse\_transform(X\_reduced\_stem)

*# Преобразование массивов numpy в строки*

X\_text\_lem **=** [' '**.**join(row) **for** row **in** X\_inverse\_lem**.**astype(str)]

X\_text\_stem **=** [' '**.**join(row) **for** row **in** X\_inverse\_stem**.**astype(str)]

*# Векторизация*

tfidf **=** TfidfVectorizer(stop\_words**=**russian\_stopwords, max\_features**=**5000)

X\_final\_lem **=** tfidf**.**fit\_transform(X\_text\_lem)**.**toarray()

X\_final\_stem **=** tfidf**.**fit\_transform(X\_text\_stem)**.**toarray()

*# Разделение данных и обучение модели*

X\_train\_final\_lem, X\_test\_final\_lem, y\_train, y\_test **=** train\_test\_split(X\_final\_lem, y, test\_size**=**0.2, random\_state**=**0)

X\_train\_final\_stem, X\_test\_final\_stem, y\_train, y\_test **=** train\_test\_split(X\_final\_stem, y, test\_size**=**0.2, random\_state**=**0)

best\_model **=** GradientBoostingClassifier(**\*\***best\_params)

best\_model**.**fit(X\_train\_final\_lem, y\_train)

accuracy **=** best\_model**.**score(X\_test\_final\_lem, y\_test)

print('Accuracy with changed preprocessing order (Lemmatized Data):', accuracy)

Accuracy with changed preprocessing order (Lemmatized Data): 0.905

**import** matplotlib.pyplot **as** plt

*# График точности моделей для лемматизированных данных*

model\_names **=** list(results\_lemmatized**.**keys())

accuracies\_lem **=** list(results\_lemmatized**.**values())

plt**.**barh(model\_names, accuracies\_lem)

plt**.**xlabel('Точность')

plt**.**title('Сравнение моделей (лемматизированные данные)')

plt**.**show()

*# График точности моделей для стеммированных данных*

accuracies\_stem **=** list(results\_stemmed**.**values())

plt**.**barh(model\_names, accuracies\_stem)

plt**.**xlabel('Точность')

plt**.**title('Сравнение моделей (стеммированные данные)')

plt**.**show()

