Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ национальный исследовательский  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ**

**Н. Г. ЧЕРНЫШЕВСКОГО»**

Факультет компьютерных наук и информационных технологий

УТВЕРЖДАЮ

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

**Практика - учебная (научно-исследовательская работа)**

студента 3 курса, 341 группы, факультета КНиИТ

**Подкидышевой Дарьи Александровны \_**

(ФИО полностью, в родительном падеже)

вид практики: учебная ("Научно-исследовательская работа")

кафедра: кафедра информатики и программирования

(наименование кафедры)

курс: 3 .

(номер курса)

семестр: 3 .

(номер семестра)

продолжительность: с 02.09.2024 по 27.12.2024 .

(дата начала) (дата окончания)

Руководитель практики:

зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Огнева М. В. .

(должность) (подпись, дата) (ФИО)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184483476)

[1. Обзор источников 5](#_Toc184483477)

[2. Анализ особенностей текстов 7](#_Toc184483478)

[3. Реализации программ для классификации и кластеризации текстовых данных 9](#_Toc184483479)

[3.1 Предобработка текста 10](#_Toc184483480)

[3.2 Выделение признаков 11](#_Toc184483481)

[3.3 Обучение моделей 13](#_Toc184483482)

[3.4 Анализ результатов 15](#_Toc184483483)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 18](#_Toc184483484)

[Приложение А Классификация методом SVM 20](#_Toc184483485)

[Приложение Б Классификация методом Байеса 27](#_Toc184483486)

[Приложение В Классификация методом Логистической регрессии 34](#_Toc184483487)

[Приложение Г Классификация методом Случайного леса 41](#_Toc184483488)

# **ВВЕДЕНИЕ**

Согласно некоторым сервисам, каждую секунду в Google делается более 40 тысяч поисковых запросов [1]. В интернете хранятся миллионы терабайтов памяти, среди которых не малую часть занимают текстовые данные. Такой объём текстовых данных требует серьёзной систематизации и фильтрации, поэтому была придумана технология для перевода естественного для человека языка на язык компьютера – NLP (Natural Language Processing). Обработка текстовых данных на естественном языке– это важная область исследований в области искусственного интеллекта, которая занимается разработкой методов и алгоритмов для обработки и анализа текстов на естественных языках.

NLP – это технология, которая используется повсюду в повседневной жизни обычного человека, например, поиск в интернете, сортировка комментариев на негативные и положительные, анализировать сценарии фильмов и тексты песен, обнаружение плагиата и даже прогноз выборов. Технология NLP успешнее справляется в работе с текстовыми данными, особенно в сложных задачах, поэтому её применяют для замены людей в некоторых профессиях. Такие системы могут заменять так называемых администраторов электронной почты, они ведут записи о планах, назначают встречи и могут общаться с другими людьми от имени человека. Современные системы NLP настолько хорошо умеет анализировать текст, что они с большей точностью определяют сарказм и юмор в тексте.

NLP автоматизирует бизнес-процессы, анализируя большие объёмы данных, поэтому используется во многих сферах: здравоохранение, юриспруденция, финансы, страхование, обслуживание клиентов.

Цель НИР: исследовать методы классификации текстов различных жанров и провести сравнительный анализ эффективности этих методов.

Задачи НИР:

1. Провести исследование уже существующих теоретических и практических подходов к классификации различных видов текстов
2. Анализ особенностей текстов разных жанров и выбор различных методов для построения моделей классификаторов
3. Разработать и обучить модели классификации текстов для каждого жанра с использованием выбранных алгоритмов.
4. Провести сравнительный анализ эффективности методов классификации и сделать выводы на основе полученных результатов.

# **1. Обзор источников**

Классификация текстов различных жанров (художественная, публицистическая и научная литература) требует применения методов для эффективного извлечения признаков, учитывающих содержание и стилистику текстов. Каждый из этих жанров требует применения специальных методов обработки текста, адаптированных к их особенностям и задачам классификации.

Художественные тексты характеризуются сложными лексическими структурами и многослойными смыслами. Для их классификации часто применяются подходы на основе TF-IDF и векторных представлений текста, таких как Word2Vec. Эти методы позволяют выявить ключевые термины и сохранять контекстные связи, что особенно важно для анализа стиля и тематики произведений. В исследованиях классификации поэзии, например, подчеркивается важность предобработки текста, включающей удаление стоп-слов и нормализацию, что значительно улучшает качество классификации [2]. Кроме того, использование более современных методов, таких как BERT, позволяет глубже анализировать семантические связи в художественных текстах, что повышает точность классификации [3].

Публицистические тексты, обладая более прямой и формализованной структурой, требуют менее сложных методов извлечения признаков, таких как мешок слов и TF-IDF. Эти методы позволяют эффективно извлекать ключевые термины, что делает их подходящими для анализа новостных статей и контента в социальных сетях [4]. Для классификации публицистических текстов также применяются методы, такие как логистическая регрессия и метод опорных векторов, которые демонстрируют высокую эффективность благодаря ясной структуре данных [4]. В то же время, наивный Байесовский классификатор показывает хорошие результаты, например, в классификации новостей и журналистских статей [7].

Научные тексты отличаются высокой степенью специализированной терминологии и требуют более углубленного анализа. Для их классификации успешно применяется такой метод, как Word2Vec, который учитывает контекстуальные взаимосвязи слов, что позволяет точно выделять ключевые концепции и терминологию, характерные для научной литературы [5]. Метод TF-IDF также помогает выделять частотные и специфичные термины для дальнейшего анализа, что делает его полезным для работы с научными статьями [6].

Далее, более подробно рассмотрим весь процесс классификации текстов.

# **2. Анализ особенностей текстов**

Тексты различных жанров обладают уникальными особенностями, которые определяют их лексику, структуру и стилистические приемы. Эти особенности необходимо учитывать при выборе методов классификации, таких как Word2Vec, TF-IDF и Bag of Words, поскольку каждый из них по-разному интерпретирует текстовые данные и выявляет признаки, значимые для классификации. Рассмотрим ключевые характеристики текстов художественной, публицистической и научной литературы и определим, какие из указанных методов наиболее подходят для их анализа.

Художественная литература отличается эмоционально насыщенной лексикой, образностью и сложной синтаксической структурой. В таких текстах преобладают описания, метафоры, эпитеты и другие выразительные средства языка, что создает богатый контекст [2]. Слова в художественной литературе часто приобретают дополнительный смысл в зависимости от окружения, что делает такие тексты трудными для анализа методами, которые ориентируются исключительно на частотность или независимость слов. Методы Word2Vec и TF-IDF оказываются наиболее подходящими для анализа художественных текстов, так как они формируют векторное представление слов, учитывающее их контекстное окружение. Например, слово "звезда" в художественном тексте может быть связано как с небесным телом, так и с понятием славы; контекст помогает модели разграничить эти значения. В то же время метод BoW менее эффективен для этого жанра, так как игнорирует последовательность слов и их взаимосвязи, что приводит к потере контекстной информации [3].

Публицистические тексты, напротив, ориентированы на информационность и воздействие на аудиторию. Они часто содержат ключевые слова, отражающие основную идею текста, а также риторические конструкции, такие как вопросы, лозунги и утверждения [4]. Такие тексты имеют тенденцию к упрощенной структуре и включают множество повторяющихся слов или фраз, связанных с темой публикации. Для классификации публицистических текстов метод TF-IDF является одним из наиболее эффективных. Он позволяет выделить значимые слова, снижая вес часто встречающихся, но малоинформативных, таких как "и", "в", "на". Например, в тексте о политике TF-IDF придаст больший вес словам "реформа", "закон", "экономика", чем словам общего употребления. Bag of Words также может быть полезен для обработки публицистических текстов, так как такие тексты часто имеют устойчивую лексику, характерную для конкретных тем. Однако этот метод не учитывает порядок слов, что может стать ограничением при работе с фразами или выражениями, которые имеют значение только в определенной последовательности. Word2Vec в случае публицистики полезен для определения тематической близости слов, но его сложность может быть избыточной [5].

Научная литература выделяется строгой структурой и насыщенностью специализированными терминами. Такие тексты редко используют эмоционально окрашенные слова, вместо этого фокусируясь на четкой аргументации и объективной подаче информации. Здесь важно учитывать высокую частотность терминов, которые играют ключевую роль в описании темы. Для научных текстов эффективными оказываются как TF-IDF, так и Bag of Words. TF-IDF позволяет выделять наиболее значимые термины, исключая общеупотребительные слова, которые не несут смысловой нагрузки [6]. Например, в тексте о биологии термины "ДНК", "митохондрия", "репликация" получат высокий вес, что облегчит их идентификацию моделью. Bag of Words также может быть применим, поскольку в научной литературе лексика достаточно стандартизирована, а взаимосвязь слов в предложении играет меньшую роль, чем их наличие [6]. Однако, если требуется учитывать взаимосвязь терминов или тематическую близость слов, Word2Vec становится наиболее предпочтительным. Он позволяет выявлять скрытые связи между терминами и улучшает понимание тематического содержания текста, например, распознавая, что "ген" и "хромосома" тесно связаны.

# **3. Реализации программ для классификации и кластеризации текстовых данных**

В данной курсовой работе для анализа текстов были выбраны три различных набора данных, каждый из которых представляет определённый жанр текстов: художественная литература, научная литература и публицистика.

1. **Датасет для художественной литературы [8]**:  
   Набор данных содержит аннотации и характеристики книг. Датасет включает в себя такие параметры, как название книги, её рейтинг, количество отзывов и подписчиков автора. Жанры классифицируются по содержанию книги, среди которых триллер, фантастика, романтика, ужас, история, психология, путешествовать, наука, спорт, научная фантастика.
2. **Датасет для научной литературы [9]**:  
   Этот набор данных содержит метаданные и аннотации для 50 000 академических статей. Включены ключевые метаданные для каждой статьи, такие как дата публикации, название, аннотация, автор(ы) и категория. Столбец категория содержит информацию о принадлежности каждой работы к одной из классификаций. Основные категории включают: Computer Science, Economics, Electrical Engineering and Systems Science, Mathematics, Physics, Quantitative Biology, Quantitative Finance и Statistics.
3. **Датасет для публицистической литературы [10]**:  
   Набор данных содержит 10 000 статей из различных категорий. Каждая статья включает заголовок, краткое описание, полный текст статьи и категорию. Датасет собран из материалов новостного издания The Indian Express. В столбце категория указана категория новости, которая может быть одной из пяти: Business, Technology, Sports, Education, Entertainment.

На основе данных из указанных наборов будут написаны программы для классификации текстов по жанрам и тематике. Для обработки и анализа текстов применяются методы **TF-IDF**, **Word2Vec** и **BOW (мешок слов)**, а для классификации и кластеризации будут использоваться алгоритмы: **SVM**, **наивный байесовский классификатор**, **случайный лес**, **логистическая регрессия**, **k-means** и **иерархическая кластеризация**.

## **3.1 Предобработка текста**

Для предобработки данных был использован набор инструментов на базе библиотеки NLTK для очистки текста от лишних символов и приведения его к нужному виду. Для этого была написана функция preprocess\_text, которая выполняет следующие шаги:

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text) tokens = word\_tokenize(text)

# Удаление стоп-слов и лемматизация

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words and word != '' and len(word) > 1

return ' '.join(tokens)

else:

return ''

Метод text.lower() используется для приведения всего текста к нижнему регистру, что позволяет избежать различий в регистре слов. Метод re.sub(r'[^a-z\s]', '', text) удаляет все неалфавитные символы, оставляя только буквы и пробелы. text.strip() удаляет лишние пробелы в начале и в конце строки. nltk.word\_tokenize(text) используется для токенизации текста, то есть для разбивки текста на отдельные слова (токены). Метод stopwords.words('english') загружает список стандартных стоп-слов для английского языка. Эти слова не несут полезной информации и исключаются из списка токенов. lemmatizer.lemmatize(word) используется для лемматизации каждого токена, приводя слова к их начальной форме (например, "running" -> "run"). Метод tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words and word != '' and len(word) > 1] исключает стоп-слова, пустые строки и короткие токены (например, однобуквенные слова), которые обычно не несут смысловой нагрузки. return ' '.join(tokens) объединяет обработанные токены обратно в строку.

Этот процесс помогает очистить и нормализовать текст, удаляя избыточные данные и улучшая качество дальнейшего анализа и классификации.

# **3.2 Выделение признаков**

1. Word2Vec

Для векторизации текста с использованием модели Word2Vec применяется библиотека gensim. Сначала обучается модель с заданными параметрами:

model\_w2v = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

Здесь sentences=[text.split() for text in X\_train] — это преобразование обучающих данных в список предложений, где каждое предложение представлено как список слов.  
Параметры: vector\_size=100 — размерность вектора для каждого слова, определяющая длину вектора признаков, window=5 — максимальное расстояние между текущим и прогнозируемым словом в предложении, min\_count=1 – минимальное количество раз, которое слово должно встречаться в обучающем корпусе, чтобы быть учтенным моделью, workers=4 – количество потоков, используемых для обучения модели.

Функция для преобразования предложения в вектор с использованием обученной модели Word2Vec:

def get\_sentence\_vector(sentence, model\_w2v):

valid\_words = [word for word in sentence.split() if word in model\_w2v.wv]

if valid\_words:

vectors = np.array([model\_w2v.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(model\_w2v.vector\_size)

Эта функция создает вектор документа, усредняя векторы всех слов, присутствующих в предложении, из обученной модели Word2Vec. Если в тексте нет подходящих слов, функция возвращает нулевой вектор.

Затем все предложения преобразуются в векторы:

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_test])

1. BOW (Bag of Words)

Для преобразования текстовых данных в матрицу признаков с использованием метода Bag of Words (BOW) используется CountVectorizer из библиотеки sklearn.feature\_extraction.text. Параметры:

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

max\_features=3500 – ограничивает количество признаков до 3500 наиболее часто встречающихся слов, min\_df=10 – игнорирует слова, которые встречаются в менее чем 10 документах, max\_df=0.7 – игнорирует слова, которые появляются в более чем 70% документов, ngram\_range=(1, 2) – включает униграммы (одиночные слова) и биграммы (пары слов) в качестве признаков.

После применения CountVectorizer создается матрица признаков:

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

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

1. TF-IDF

Для выделения признаков с использованием метода TF-IDF применяется TfidfVectorizer из библиотеки sklearn.feature\_extraction.text. Параметры:

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

Здесь параметры аналогичны тем, что использовались в методе BOW: max\_features=3500 — ограничивает количество признаков до 3500, min\_df=10 – игнорирует слова, встречающиеся в менее чем 10 документах, max\_df=0.7 – игнорирует слова, которые появляются в более чем 70% документов, ngram\_range=(1, 2) — включает униграммы и биграммы.

После применения TfidfVectorizer создается матрица признаков:

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

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

Все три метода преобразуют текстовые данные в числовые векторы, которые могут быть использованы для обучения моделей машинного обучения, таких как классификаторы.

## **3.3 Обучение моделей**

После предобработки данных, текст делится на тренировочные и тестовые данные для дальнейшего обучения моделей. Мы используем функцию train\_test\_split, чтобы разделить данные:

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

Здесь X — это обработанные текстовые данные, а y — метки классов. 20% данных выделяется для тестирования, а остальные 80% используются для обучения модели. Параметр stratify=y гарантирует, что распределение классов будет одинаковым в обучающем и тестовом наборах данных.

Далее инициализируются модели машинного обучения, которые будут обучаться на тренировочных данных:

Модель случайного леса:

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

Здесь создается модель случайного леса с 100 деревьями и фиксированным значением random\_state для воспроизводимости.

Модель Word2Vec:

Модель Word2Vec обучается с использованием текстовых данных:

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

Эта модель использует обучающие данные, разделенные на слова, для получения векторов для каждого слова. Параметры аналогичны предыдущему описанию.

Логистическая регрессия:

lr\_model = LogisticRegression(max\_iter=1000, random\_state=42)

Модель логистической регрессии с максимальным числом итераций 1000 и фиксированным значением random\_state.

Модель SVM:

svm\_model = SVC(kernel='linear', random\_state=42)

Это классификатор, использующий стохастический градиентный спуск с линейной функцией потерь ("hinge"), что делает модель аналогичной линейной поддерживающей векторной машине SVM.

Для оценки производительности каждой модели на разных методах векторизации данных выполняются следующие шаги:

1. Оценка для BOW:

svm\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_genres\_bow = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_genres\_bow))

1. Оценка для TF-IDF:

svm\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_genres\_tfidf = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_genres\_tfidf))

1. Оценка для Word2Vec:

svm\_model.fit(X\_train\_w2v, y\_train\_encoded)

y\_pred\_w2v = svm\_model.predict(X\_test\_w2v)

y\_pred\_genres\_w2v = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_genres\_w2v))

Таким образом, на каждом из этапов производится обучение моделей на соответствующих представлениях данных (BOW, TF-IDF, Word2Vec), после чего оценивается их производительность с использованием метрики классификации, такой как precision, recall, f1-score и другие, выводимые функцией classification\_report.

Полный код программ классификации методом SVM, Наивного Байеса, логистической регрессии и случайного леса с использованием BOW, TF-IDF и Word2Vec находятся в приложениях А, Б, В, Г соответственно.

## **3.4 Анализ результатов**

**1. Художественная литература**

В случае с датасетом, содержащим художественную литературу, методы BOW и TF-IDF продемонстрировали наилучшие результаты, что обусловлено их способностью эффективно выявлять ключевые слова, характерные для различных жанров художественной литературы. Метод TF-IDF, учитывающий важность слов в контексте, показал особенно высокую точность при классификации жанров с ярко выраженными тематическими особенностями, такими как фэнтези, спорт и путешествия. Эти жанры содержат много характерных терминов, что позволяет методам BOW и TF-IDF эффективно извлекать важные признаки.

Тем не менее, эти методы оказались менее эффективными для жанров, где важны контекст и глубина содержания, таких как психология и романтика. В этих случаях ключевые признаки, такие как эмоциональная окраска текста, не всегда хорошо отражаются в частоте или значимости слов, что снижает эффективность стандартных методов векторизации текста.

Метод Word2Vec, несмотря на свою теоретическую состоятельность, показал более слабые результаты. Это может быть связано с ограниченным объемом данных, который не позволил модели сформировать точные векторные представления слов, что привело к низкой эффективности этого метода при классификации.

Среди алгоритмов классификации наиболее стабильными показателями отличились методы SVM и Наивный Байес. Эти модели показали более высокую стабильность в условиях ограниченного объема данных по сравнению с случайным лесом, который оказался менее эффективным.

**2. научная литература**

Для датасета с научной литературой методы BOW и TF-IDF показали схожие результаты, достигая точности около 72-77%. Эти методы хорошо работают с дисциплинами, где часто встречаются ключевые слова, такие как физика и математика. В этих областях присутствие повторяющихся терминов способствует высокой эффективности представленных методов. Однако для более узких дисциплин, таких как экономика и статистика, BOW и TF-IDF продемонстрировали снижение точности, что может быть связано с особенностями словаря, который не всегда покрывает редкие или специфичные термины.

Метод Word2Vec продемонстрировал лучшие результаты в классификации крупных дисциплин с точностью около 77%, благодаря способности учитывать семантические связи между словами. Однако этот метод не смог адекватно обработать более узкие категории, что говорит о его зависимости от объема данных для формирования точных векторных представлений.

Наивный Байес показал хорошие результаты, особенно в таких дисциплинах, как математика, однако он страдал от низкой эффективности при обработке редких категорий. Методы SVM и логистическая регрессия показали стабильные результаты, но также пострадали от малочисленных классов. В целом для работы с большими датасетами и дисциплинами, где важна семантика, лучше использовать более сложные методы, такие как Word2Vec.

**3. публицистическая литература**

Для датасета с публицистической литературой методы BOW и TF-IDF показали отличные результаты с точностью около 97-98%, особенно при использовании алгоритмов SVM и логистической регрессии. Эти методы эффективно справляются с категориями, где часто повторяются ключевые слова, такими как бизнес и образование. Тем не менее, они менее эффективны для текстов с редкими или специфическими терминами, которые не так часто встречаются в таких текстах.

Метод Word2Vec показал точность около 96%, что объясняется его способностью учитывать семантические связи между словами. Однако он не смог продемонстрировать высокую эффективность в случае с малочисленными классами, такими как редкие категории, что, в свою очередь, привело к ухудшению его работы в таких случаях, как с Наивным Байесом (89%).

Случайный лес показал стабильные результаты около 95%, при этом не было значительных различий по методам представления текста. В случае с публицистической литературой использование методов SVM или логистической регрессии с векторизацией BOW или TF-IDF показало наилучшие результаты.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Основная статистика и факты поиска Google. — Текст : электронный // techblog.sdstudio.top : [сайт]. — URL: <https://techblog.sdstudio.top/osnovnaja-statistika-i-fakty-poiska-google-2020-g/#1_Kazduu_sekundu_v_Google_vypolnaetsa_40_000_zaprosov> (дата обращения: 26.11.2024).
2. Ян Л., Ван Г., Ван Х. Переосмысление литературного анализа: использование искусственного интеллекта для классификации модернистской французской поэзии // Информация. 2024. № 15 (2070). С. 1–18. (х)
3. Хаттак А., Асгар М. З., Халид Х. Классификация эмоций в поэтическом тексте с использованием глубокой нейронной сети // Multimed Tools Appl. 2022. № 81. С. 26223–26244.
4. Челышев Э. А., Оцоков Ш. А., Раскатова М. В., Щёголев П. Сравнение методов классификации русскоязычных новостных текстов с использованием алгоритмов машинного обучения // Вестник кибернетики. 2022. № 1 (42). С. 63–71. (п)
5. Лукачевич Н., Артемова Е., Батура Т., Браславский П., Иванов В., Манандар С., Пугачёв А., Рожков И., Шельманов А., Тутубалина Е., Яндутов А. NEREL: российский набор данных для извлечения информации с расширенной аннотацией вложенных сущностей, связей и ссылок на Wikidata // Language Resources and Evaluation. 2024. Том 58. № 2. С. 547–583.
6. Мишра, Р. Б. Классификация проблемных и решенческих строк в научных текстах: оценка эффективности классификаторов машинного обучения и глубоких нейронных сетей / Р. Б. Мишра, Х. Цзян. — Текст : непосредственный // Прикладные науки. — 2021. — № 11(9997). — С. 1-18.
7. Ковсари К., Джафари Мейманди К., Хейдарсафа М., Менду С., Барнс Л., Браун Д. Алгоритмы классификации текстов: обзор // Информация. 2019. № 10 (4150). С. 1–19.
8. TagMyBook. — Текст : электронный // Kaggle : [сайт]. — URL: https://www.kaggle.com/datasets/athu1105/tagmybook (дата обращения: 27.09.2024).
9. Papers by Subject — Текст : электронный // Kaggle : [сайт]. — URL: https://www.kaggle.com/datasets/arplusman/papers-by-subject (дата обращения: 27.09.2024).
10. News Articles Classification Dataset— Текст : электронный // Kaggle : [сайт]. — URL: https://www.kaggle.com/datasets/banuprakashv/news-articles-classification-dataset-for-nlp-and-ml (дата обращения: 27.09.2024).

# **Приложение А**

**Классификация методом SVM**

**Художественная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.svm import SVC # Используем SVM

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder

from gensim.models import Word2Vec

# Загружаем данные из Excel файла

file\_path = r'C:\Users\darap\Desktop\literature2.xlsx'

df = pd.read\_excel(file\_path)

# Очистка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip(): # Проверяем, является ли текст строкой и не пустой

# Приводим текст к нижнему регистру

text = text.lower()

# Убираем все неалфавитные символы

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return '' # Если строка пуста или NaN, возвращаем пустую строку

# Применяем очистку ко всем строкам

df['Processed\_Synopsis'] = df['synopsis'].apply(preprocess\_text)

# Проверка на пропущенные значения

print(df.isnull().sum()) # Проверим, есть ли NaN в столбцах

# Убираем строки с пропущенными значениями

df = df.dropna(subset=['synopsis', 'genre']) # Удаляем строки с пропущенными значениями

# Вывод первых 5 строк исходного датасета и обработанного текста для проверки

print("Примеры из исходного датасета:")

print(df[['title', 'synopsis', 'genre']].head(), "\n")

print("Примеры обработанных данных:")

print(df[['Processed\_Synopsis', 'genre']].head(), "\n")

# Подготовка данных для классификации

X = df['Processed\_Synopsis']

y = df['genre']

# Разделение на обучающую и тестовую выборки

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

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2)) # Используем CountVectorizer

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2)) # Используем TfidfVectorizer

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели word2vec

model\_w2v = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы с использованием word2vec

def get\_sentence\_vector(sentence, model\_w2v):

valid\_words = [word for word in sentence.split() if word in model\_w2v.wv]

if valid\_words:

vectors = np.array([model\_w2v.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(model\_w2v.vector\_size)

# Преобразуем все предложения в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_test])

# Кодируем целевые метки

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Обучение модели SVM и оценка для каждого метода векторизации ---

# Модель SVM

svm\_model = SVC(kernel='linear', random\_state=42)

# Для BOW

svm\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_genres\_bow = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_genres\_bow))

# Для TF-IDF

svm\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_genres\_tfidf = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_genres\_tfidf))

# Для Word2Vec

svm\_model.fit(X\_train\_w2v, y\_train\_encoded)

y\_pred\_w2v = svm\_model.predict(X\_test\_w2v)

y\_pred\_genres\_w2v = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_genres\_w2v))

**Публицистическая литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Пути к файлам

file\_paths = [

r'C:\Users\darap\Desktop\публицистика\education\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\entertainment\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\sports\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\business\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\technology.xlsx'

]

# Загрузка данных из всех файлов в один DataFrame

dataframes = []

for file\_path in file\_paths:

df = pd.read\_excel(file\_path)

dataframes.append(df)

df = pd.concat(dataframes, ignore\_index=True)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text) # Удаляем все символы, кроме букв и пробелов

return text

else:

return ''

# Очистка данных

df['Processed\_Content'] = df['content'].apply(preprocess\_text)

df = df.dropna(subset=['Processed\_Content', 'category'])

# Подготовка данных

X = df['Processed\_Content']

y = df['category']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(w2v\_model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели SVM для каждого метода векторизации ---

svm\_model = SVC(kernel='linear', random\_state=42)

# --- Векторизация BOW ---

svm\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

svm\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

svm\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = svm\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

**Научная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

from sklearn.linear\_model import SGDClassifier

# Загрузка данных

file\_path = r'C:\Users\darap\Desktop\science.xlsx'

df = pd.read\_excel(file\_path)

# Функция для маппинга подкатегорий на основные категории

def map\_to\_primary\_category(subcategory):

if subcategory.startswith("cs."):

return "Computer Science"

elif subcategory.startswith("econ."):

return "Economics"

elif subcategory.startswith("eess."):

return "Electrical Engineering and Systems Science"

elif subcategory.startswith("math."):

return "Mathematics"

elif subcategory.startswith("astro-ph.") or subcategory.startswith("cond-mat"):

return "Physics"

elif subcategory.startswith("q-bio."):

return "Quantitative Biology"

elif subcategory.startswith("q-fin."):

return "Quantitative Finance"

elif subcategory.startswith("stat."):

return "Statistics"

else:

return "Physics"

# Применение маппинга

df['Primary Category Mapped'] = df['Primary Category'].apply(map\_to\_primary\_category)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return ''

# Очистка данных

df['Processed\_Summary'] = df['Summary'].apply(preprocess\_text)

# Проверка и удаление пропущенных значений

df = df.dropna(subset=['Processed\_Summary', 'Primary Category Mapped'])

# Подготовка данных

X = df['Processed\_Summary']

y = df['Primary Category Mapped']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=1000, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=1000, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели Word2Vec

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=50, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(w2v\_model.vector\_size)

# Преобразование всех предложений в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели SVM для каждого метода векторизации ---

# SVM модель

svm\_model = SGDClassifier(loss="hinge", random\_state=42)

# --- Векторизация BOW ---

svm\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

svm\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

svm\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = svm\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

# **Приложение Б**

**Классификация методом Наивного Байеса**

**Художественная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.naive\_bayes import MultinomialNB # Используем наивный байесовский классификатор

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder

from gensim.models import Word2Vec

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# Загружаем данные из Excel файла

file\_path = r'C:\Users\darap\Desktop\literature21.xlsx'

df = pd.read\_excel(file\_path)

# Очистка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip(): # Проверяем, является ли текст строкой и не пустой

# Приводим текст к нижнему регистру

text = text.lower()

# Убираем все неалфавитные символы

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return '' # Если строка пуста или NaN, возвращаем пустую строку

# Применяем очистку ко всем строкам

df['Processed\_Synopsis'] = df['synopsis'].apply(preprocess\_text)

# Убираем строки с пропущенными значениями

df = df.dropna(subset=['synopsis', 'genre']) # Удаляем строки с пропущенными значениями

# Вывод первых 5 строк исходного датасета и обработанного текста для проверки

print("Примеры из исходного датасета:")

print(df[['title', 'synopsis', 'genre']].head(), "\n")

print("Примеры обработанных данных:")

print(df[['Processed\_Synopsis', 'genre']].head(), "\n")

# Подготовка данных для классификации

X = df['Processed\_Synopsis']

y = df['genre']

# Разделение на обучающую и тестовую выборки

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

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2)) # Используем CountVectorizer

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=1000, min\_df=10, max\_df=0.65, ngram\_range=(1, 4)) # Используем TfidfVectorizer

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели word2vec

model\_w2v = Word2Vec(sentences=X\_train, vector\_size=100, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы с использованием word2vec

def get\_sentence\_vector(sentence, model\_w2v):

# Игнорируем слова, которых нет в модели

valid\_words = [word for word in sentence if word in model\_w2v.wv]

if valid\_words:

# Усредняем векторы всех слов в предложении

vectors = np.array([model\_w2v.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

# Если нет подходящих слов, возвращаем нулевой вектор

return np.zeros(model\_w2v.vector\_size)

# Преобразуем все предложения в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_test])

# Масштабирование данных с использованием MinMaxScaler

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_scaled = scaler.transform(X\_test\_w2v)

# Кодируем целевые метки

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Обучение модели Наивного Байеса и оценка для каждого метода векторизации ---

# Модель Наивного Байеса

nb\_model = MultinomialNB()

# Для BOW

nb\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_genres\_bow = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_genres\_bow))

# Для TF-IDF

nb\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_genres\_tfidf = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_genres\_tfidf))

# Для Word2Vec

nb\_model.fit(X\_train\_scaled, y\_train\_encoded)

y\_pred\_w2v = nb\_model.predict(X\_test\_scaled)

y\_pred\_genres\_w2v = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_genres\_w2v))

**Публицистическая литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.naive\_bayes import MultinomialNB, GaussianNB

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Пути к файлам

file\_paths = [

r'C:\Users\darap\Desktop\публицистика\education\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\entertainment\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\sports\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\business\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\technology.xlsx'

]

# Загрузка данных из всех файлов в один DataFrame

dataframes = []

for file\_path in file\_paths:

df = pd.read\_excel(file\_path)

dataframes.append(df)

df = pd.concat(dataframes, ignore\_index=True)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text) # Удаляем все символы, кроме букв и пробелов

return text

else:

return ''

# Очистка данных

df['Processed\_Content'] = df['content'].apply(preprocess\_text)

df = df.dropna(subset=['Processed\_Content', 'category'])

# Подготовка данных

X = df['Processed\_Content']

y = df['category']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(w2v\_model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели наивного Байеса для каждого метода векторизации ---

# Модель MultinomialNB для BOW и TF-IDF

nb\_model = MultinomialNB()

# --- Векторизация BOW ---

nb\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

nb\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

# Используем GaussianNB для непрерывных данных Word2Vec

nb\_model\_w2v = GaussianNB()

nb\_model\_w2v.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = nb\_model\_w2v.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

**Научная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Загрузка данных

file\_path = r'C:\Users\darap\Desktop\science.xlsx'

df = pd.read\_excel(file\_path)

# Функция для маппинга подкатегорий на основные категории

def map\_to\_primary\_category(subcategory):

if subcategory.startswith("cs."):

return "Computer Science"

elif subcategory.startswith("econ."):

return "Economics"

elif subcategory.startswith("eess."):

return "Electrical Engineering and Systems Science"

elif subcategory.startswith("math."):

return "Mathematics"

elif subcategory.startswith("astro-ph.") or subcategory.startswith("cond-mat"):

return "Physics"

elif subcategory.startswith("q-bio."):

return "Quantitative Biology"

elif subcategory.startswith("q-fin."):

return "Quantitative Finance"

elif subcategory.startswith("stat."):

return "Statistics"

else:

return "Physics"

# Применение маппинга

df['Primary Category Mapped'] = df['Primary Category'].apply(map\_to\_primary\_category)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return ''

# Очистка данных

df['Processed\_Summary'] = df['Summary'].apply(preprocess\_text)

# Проверка и удаление пропущенных значений

df = df.dropna(subset=['Processed\_Summary', 'Primary Category Mapped'])

# Подготовка данных

X = df['Processed\_Summary']

y = df['Primary Category Mapped']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели Word2Vec

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(w2v\_model.vector\_size)

# Преобразование всех предложений в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели наивного Байеса для каждого метода векторизации ---

# Модель MultinomialNB

nb\_model = MultinomialNB()

# --- Векторизация BOW ---

nb\_model.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

nb\_model.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

nb\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = nb\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

# **Приложение В**

**Классификация методом Логистической регрессии**

**Художественная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# --- 1. Загрузка данных ---

file\_path = r'C:\Users\darap\Desktop\literature21.xlsx'

df = pd.read\_excel(file\_path)

# --- 2. Очистка текста ---

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return '' # Возвращаем пустую строку

df['Processed\_Synopsis'] = df['synopsis'].apply(preprocess\_text)

# --- 3. Убираем строки с пропущенными значениями ---

df = df.dropna(subset=['Processed\_Synopsis', 'genre'])

# Вывод первых 5 строк исходного датасета и обработанного текста для проверки

print("Примеры из исходного датасета:")

print(df[['title', 'synopsis', 'genre']].head(), "\n")

print("Примеры обработанных данных:")

print(df[['Processed\_Synopsis', 'genre']].head(), "\n")

# --- 4. Подготовка данных ---

X = df['Processed\_Synopsis']

y = df['genre']

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- 5. Word2Vec ---

# Обучение модели Word2Vec

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

# Преобразование предложений в векторы

def get\_sentence\_vector(sentence, model):

valid\_words = [word for word in sentence.split() if word in model.wv]

if valid\_words:

vectors = np.array([model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# Обучение и предсказание для Word2Vec

model\_w2v = LogisticRegression(max\_iter=1000, random\_state=42)

model\_w2v.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = model\_w2v.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

# --- 6. "Мешок слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# Обучение и предсказание для BOW

model\_bow = LogisticRegression(max\_iter=1000, random\_state=42)

model\_bow.fit(X\_train\_bow, y\_train\_encoded)

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

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- 7. TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=750, min\_df=15, max\_df=0.8, ngram\_range=(1, 2))

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

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

# Обучение и предсказание для TF-IDF

model\_tfidf = LogisticRegression(max\_iter=1000, random\_state=42)

model\_tfidf.fit(X\_train\_tfidf, y\_train\_encoded)

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

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

**Публицистическая литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Пути к файлам

file\_paths = [

r'C:\Users\darap\Desktop\публицистика\education\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\entertainment\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\sports\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\business\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\technology.xlsx'

]

# Загрузка данных из всех файлов в один DataFrame

dataframes = []

for file\_path in file\_paths:

df = pd.read\_excel(file\_path)

dataframes.append(df)

df = pd.concat(dataframes, ignore\_index=True)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text) # Удаляем все символы, кроме букв и пробелов

return text

else:

return ''

# Очистка данных

df['Processed\_Content'] = df['content'].apply(preprocess\_text)

df = df.dropna(subset=['Processed\_Content', 'category'])

# Подготовка данных

X = df['Processed\_Content']

y = df['category']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(w2v\_model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели логистической регрессии для каждого метода векторизации ---

log\_reg\_model = LogisticRegression(max\_iter=1000, random\_state=42)

# --- Векторизация BOW ---

log\_reg\_model.fit(X\_train\_bow, y\_train\_encoded)

y\_pred\_bow = log\_reg\_model.predict(X\_test\_bow)

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

log\_reg\_model.fit(X\_train\_tfidf, y\_train\_encoded)

y\_pred\_tfidf = log\_reg\_model.predict(X\_test\_tfidf)

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

log\_reg\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = log\_reg\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

**Научная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Загрузка данных

file\_path = r'C:\Users\darap\Desktop\science.xlsx'

df = pd.read\_excel(file\_path)

# Функция для маппинга подкатегорий на основные категории

def map\_to\_primary\_category(subcategory):

if subcategory.startswith("cs."):

return "Computer Science"

elif subcategory.startswith("econ."):

return "Economics"

elif subcategory.startswith("eess."):

return "Electrical Engineering and Systems Science"

elif subcategory.startswith("math."):

return "Mathematics"

elif subcategory.startswith("astro-ph.") or subcategory.startswith("cond-mat"):

return "Physics"

elif subcategory.startswith("q-bio."):

return "Quantitative Biology"

elif subcategory.startswith("q-fin."):

return "Quantitative Finance"

elif subcategory.startswith("stat."):

return "Statistics"

else:

return "Physics"

# Применение маппинга

df['Primary Category Mapped'] = df['Primary Category'].apply(map\_to\_primary\_category)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return ''

# Очистка данных

df['Processed\_Summary'] = df['Summary'].apply(preprocess\_text)

df = df.dropna(subset=['Processed\_Summary', 'Primary Category Mapped'])

# Подготовка данных

X = df['Processed\_Summary']

y = df['Primary Category Mapped']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(w2v\_model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели логистической регрессии для каждого метода векторизации ---

lr\_model = LogisticRegression(max\_iter=1000, random\_state=42)

# --- Векторизация BOW ---

lr\_model.fit(X\_train\_bow, y\_train\_encoded)

y\_pred\_bow = lr\_model.predict(X\_test\_bow)

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

lr\_model.fit(X\_train\_tfidf, y\_train\_encoded)

y\_pred\_tfidf = lr\_model.predict(X\_test\_tfidf)

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

lr\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = lr\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

# **Приложение Г**

**Классификация методом Случайного леса**

**Художественная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier # Случайный лес

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder

from gensim.models import Word2Vec

# Загружаем данные из Excel файла

file\_path = r'C:\Users\darap\Desktop\literature21.xlsx'

df = pd.read\_excel(file\_path)

# Очистка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip(): # Проверяем, является ли текст строкой и не пустой

# Приводим текст к нижнему регистру

text = text.lower()

# Убираем все неалфавитные символы

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return '' # Если строка пуста или NaN, возвращаем пустую строку

# Применяем очистку ко всем строкам

df['Processed\_Synopsis'] = df['synopsis'].apply(preprocess\_text)

# Убираем строки с пропущенными значениями

df = df.dropna(subset=['synopsis', 'genre']) # Удаляем строки с пропущенными значениями

# Подготовка данных для классификации

X = df['Processed\_Synopsis']

y = df['genre']

# Разделение на обучающую и тестовую выборки

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

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2)) # Используем CountVectorizer

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=4000, min\_df=10, max\_df=0.7, ngram\_range=(1, 3)) # Используем TfidfVectorizer

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели word2vec

model\_w2v = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы с использованием word2vec

def get\_sentence\_vector(sentence, model\_w2v):

valid\_words = [word for word in sentence.split() if word in model\_w2v.wv]

if valid\_words:

vectors = np.array([model\_w2v.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(model\_w2v.vector\_size)

# Преобразуем все предложения в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, model\_w2v) for sentence in X\_test])

# Кодируем целевые метки

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Обучение модели случайного леса и оценка для каждого метода векторизации ---

# Модель случайного леса

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Для BOW

rf\_model.fit(X\_train\_bow, y\_train\_encoded)

y\_pred\_bow = rf\_model.predict(X\_test\_bow)

y\_pred\_genres\_bow = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_genres\_bow))

# Для TF-IDF

rf\_model.fit(X\_train\_tfidf, y\_train\_encoded)

y\_pred\_tfidf = rf\_model.predict(X\_test\_tfidf)

y\_pred\_genres\_tfidf = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_genres\_tfidf))

# Для Word2Vec

rf\_model.fit(X\_train\_w2v, y\_train\_encoded)

y\_pred\_w2v = rf\_model.predict(X\_test\_w2v)

y\_pred\_genres\_w2v = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_genres\_w2v))

**Публицистическая литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier # Импортируем модель случайного леса

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Пути к файлам

file\_paths = [

r'C:\Users\darap\Desktop\публицистика\education\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\entertainment\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\sports\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\business\_data.xlsx',

r'C:\Users\darap\Desktop\публицистика\technology.xlsx'

]

# Загрузка данных из всех файлов в один DataFrame

dataframes = []

for file\_path in file\_paths:

df = pd.read\_excel(file\_path)

dataframes.append(df)

df = pd.concat(dataframes, ignore\_index=True)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text) # Удаляем все символы, кроме букв и пробелов

return text

else:

return ''

# Очистка данных

df['Processed\_Content'] = df['content'].apply(preprocess\_text)

df = df.dropna(subset=['Processed\_Content', 'category'])

# Подготовка данных

X = df['Processed\_Content']

y = df['category']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=3500, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=100, window=5, min\_count=1, workers=4)

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

return vectors.mean(axis=0)

else:

return np.zeros(w2v\_model.vector\_size)

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели случайного леса для каждого метода векторизации ---

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42) # Модель случайного леса

# --- Векторизация BOW ---

rf\_model.fit(X\_train\_bow, y\_train\_encoded)

y\_pred\_bow = rf\_model.predict(X\_test\_bow)

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW:")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

rf\_model.fit(X\_train\_tfidf, y\_train\_encoded)

y\_pred\_tfidf = rf\_model.predict(X\_test\_tfidf)

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF:")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

rf\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = rf\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec:")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))

**Научная литература:**

import pandas as pd

import re

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier # Импортируем модель случайного леса

from sklearn.metrics import classification\_report

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

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from gensim.models import Word2Vec

# Загрузка данных

file\_path = r'C:\Users\darap\Desktop\science.xlsx'

df = pd.read\_excel(file\_path)

# Функция для маппинга подкатегорий на основные категории

def map\_to\_primary\_category(subcategory):

if subcategory.startswith("cs."):

return "Computer Science"

elif subcategory.startswith("econ."):

return "Economics"

elif subcategory.startswith("eess."):

return "Electrical Engineering and Systems Science"

elif subcategory.startswith("math."):

return "Mathematics"

elif subcategory.startswith("astro-ph.") or subcategory.startswith("cond-mat"):

return "Physics"

elif subcategory.startswith("q-bio."):

return "Quantitative Biology"

elif subcategory.startswith("q-fin."):

return "Quantitative Finance"

elif subcategory.startswith("stat."):

return "Statistics"

else:

return "Physics"

# Применение маппинга

df['Primary Category Mapped'] = df['Primary Category'].apply(map\_to\_primary\_category)

# Предварительная обработка текста

def preprocess\_text(text):

if isinstance(text, str) and text.strip():

text = text.lower()

text = re.sub(r'[^a-z\s]', '', text)

return text

else:

return ''

# Очистка данных

df['Processed\_Summary'] = df['Summary'].apply(preprocess\_text)

# Проверка и удаление пропущенных значений

df = df.dropna(subset=['Processed\_Summary', 'Primary Category Mapped'])

# Подготовка данных

X = df['Processed\_Summary']

y = df['Primary Category Mapped']

# Разделение на обучающую и тестовую выборки

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

# Кодирование целевых меток

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# --- Метод 1: Векторизация с использованием "мешка слов" (BOW) ---

vectorizer\_bow = CountVectorizer(max\_features=1000, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 2: Векторизация с использованием TF-IDF ---

vectorizer\_tfidf = TfidfVectorizer(max\_features=1000, min\_df=10, max\_df=0.7, ngram\_range=(1, 2))

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

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

# --- Метод 3: Векторизация с использованием Word2Vec ---

# Обучение модели Word2Vec

w2v\_model = Word2Vec(sentences=[text.split() for text in X\_train], vector\_size=50, window=5, min\_count=1, workers=4)

# Функция для преобразования предложений в векторы

def get\_sentence\_vector(sentence, w2v\_model):

valid\_words = [word for word in sentence.split() if word in w2v\_model.wv]

if valid\_words:

vectors = np.array([w2v\_model.wv[word] for word in valid\_words])

sentence\_vector = vectors.mean(axis=0)

return sentence\_vector

else:

return np.zeros(w2v\_model.vector\_size)

# Преобразование всех предложений в векторы

X\_train\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_train])

X\_test\_w2v = np.array([get\_sentence\_vector(sentence, w2v\_model) for sentence in X\_test])

# Масштабирование данных для Word2Vec

scaler = MinMaxScaler()

X\_train\_w2v\_scaled = scaler.fit\_transform(X\_train\_w2v)

X\_test\_w2v\_scaled = scaler.transform(X\_test\_w2v)

# --- Обучение и оценка модели случайного леса для каждого метода векторизации ---

# Модель случайного леса

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# --- Векторизация BOW ---

rf\_model.fit(X\_train\_bow, y\_train\_encoded)

y\_pred\_bow = rf\_model.predict(X\_test\_bow)

y\_pred\_bow\_labels = label\_encoder.inverse\_transform(y\_pred\_bow)

print("Результаты для метода BOW (случайный лес):")

print(classification\_report(y\_test, y\_pred\_bow\_labels))

# --- Векторизация TF-IDF ---

rf\_model.fit(X\_train\_tfidf, y\_train\_encoded)

y\_pred\_tfidf = rf\_model.predict(X\_test\_tfidf)

y\_pred\_tfidf\_labels = label\_encoder.inverse\_transform(y\_pred\_tfidf)

print("Результаты для метода TF-IDF (случайный лес):")

print(classification\_report(y\_test, y\_pred\_tfidf\_labels))

# --- Векторизация Word2Vec ---

rf\_model.fit(X\_train\_w2v\_scaled, y\_train\_encoded)

y\_pred\_w2v = rf\_model.predict(X\_test\_w2v\_scaled)

y\_pred\_w2v\_labels = label\_encoder.inverse\_transform(y\_pred\_w2v)

print("Результаты для метода Word2Vec (случайный лес):")

print(classification\_report(y\_test, y\_pred\_w2v\_labels))