Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Базовая кафедра «Интеллектуальные системы управления»

УТВЕРЖДАЮ

Заведующий кафедрой

_____ Ю.Ю. Якунин

подпись

«_____» _____20 ___г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Применение метода анализа текстовой информации для задачи классификации научных статей

09.04.04 Программная инженерия

09.04.04.02 Технологии индустриального производства программного обеспечения интеллектуальных систем управления

Научный руководитель	_____ подпись, дата	доцент, канд. техн. наук	А.В. Хныкин
Выпускник	_____ подпись, дата		Н.О. Уханов
Рецензент	_____ подпись, дата	доцент, канд. техн. наук	А.С. Амузаде
Нормоконтролер	_____ подпись, дата		Н.Б. Позолотина

Красноярск 2021

РЕФЕРАТ

Выпускная квалификационная работа по теме «Применение метода анализа текстовой информации для задачи классификации научных статей» содержит 73 страницы текстового документа, 9 рисунков, 17 таблиц, 54 формулы, 19 использованных источников, 1 приложение.

КАТЕГОРИЗАЦИЯ ТЕКСТОВ, КЛАССИФИКАЦИЯ ТЕКСТОВ, КЛАССИФИКАЦИЯ НАУЧНЫЙ СТАТЕЙ, ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ, ПЕРЕКРЕСТНАЯ ЭНТРОПИЯ, КРОСС-ЭНТРОПИЯ, TF-IDF, МАШИННОЕ ОБУЧЕНИЕ, ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА, ЛЕММАТИЗАЦИЯ, МЕШОК СЛОВ, PYTHON.

Цель: разработать программу, реализующую классификатор научных статей с совместным применением модели логистической регрессии, статистической меры TF-IDF и модели мешка слов.

Задачи:

- Проанализировать алгоритмы машинного обучения, применимые для решения задачи классификации научных статей;
- Разработать классификатор научных статей;
- Разработать архитектуру программы;
- Реализовать классификатор научных статей;
- Проверить влияние разных типов токенов на точность предсказаний разработанного классификатора и сравнить с результатами работы других популярных алгоритмов.

В результате проделанной работы был разработан классификатор научных статей на основе совместного применения модели логистической регрессии, статистической меры TF-IDF и модели мешка слов. В рамках оценки эффективности реализованного классификатора произведено сравнение точности с другими алгоритмами машинного обучения при использовании разных типов токенов.

СОДЕРЖАНИЕ

Введение.....5
1 Обзор моделей машинного обучения, применяемых для классификации текстов. 8
1.1 Общая задача классификации.....8
1.2 Метод *k*-ближайших соседей.....8
1.3 Линейная регрессия.....9
1.4 Метод опорных векторов.....11
1.4.1 Алгоритм с жестким зазором.....12
1.4.2 Алгоритм с мягким зазором.....15
1.4.3 Ядра.....16
1.5 Дерево решений.....17
1.5.1 Алгоритм CART.....17
1.5.2 Алгоритмы ID3, C4.5, C 5.0.....20
1.6 Алгоритм Random Forest.....21
1.6.1 Бутстрэп-агрегирование.....21
1.6.2 Метод случайных подпространств.....23
1.7 Алгоритм AdaBoost.....23
1.8 Перцептрон.....25
2 Подготовка эксперимента.....28
2.1 Подготовка матрицы признаков.....28
2.2 Алгоритм обучения модели логистической регрессии.....33
2.3 Программирование модуля.....35
2.3.1 Программирование токенизаторов.....35
2.3.2 Программирование модуля TFIDF.....37
2.3.3 Программирование алгоритма обучения модели логистической регрессии.....40
3 Анализ полученных результатов экспериментов.....42
3.1 Оценка точности модели логистической регрессии.....43

3.2 Проверка влияния типов токенов на эффективность других алгоритмов машинного обучения....44
3.2.1 Алгоритм *k*-ближайших соседей....44
3.2.2 Метод опорных векторов....46
3.2.3 Дерево решений....48
3.2.4 Алгоритм Random Forest....50
3.2.5 Алгоритм AdaBoost....54
3.2.6 Многослойный персептрон....55
Заключение....57
Список использованных источников....58
Приложение А....60

ВВЕДЕНИЕ

Классификация научных статей – актуальная задача для широкого круга специалистов и ученых, занимающихся написанием и публикацией своих трудов в научных журналах. Например, при написании научной статьи (или тезисов) встает задача присвоения труду определенного кода УДК. Для решения этой задачи в настоящее время активно используется сервис «teacode» [1], он значительно упрощает решение проблемы определения УДК, однако все же требует немалых усилий в поиске и не исключает большой вероятности ошибочного выбора. Еще одним примером задачи классификации научной статьи является задача правильного соотнесения научной статьи к одной из специальностей ВАК, зафиксированных в соответствующих паспортах.

Задача классификации документов возникает не только при написании научных статей. Библиотеки, электронные библиотечные системы должны постоянно поддерживать систематизацию библиотечного фонда в актуальном состоянии. Это позволяет наиболее быстро и эффективно осуществлять поиск необходимой информации по всему библиотечному фонду. Задача классификации документов также встречается в лингвистике, исторической науке, юриспруденции, биологии, медицине, литературе и других науках и видах хозяйственной деятельности.

Задача классификации документов – одна из задач информационного поиска, заключающаяся в отнесении документа к одной из нескольких категорий на основании содержания документа. Решить задачу классификации текстов можно тремя разными способами:

- вручную, с помощью рубрикатора;
- автоматически, с помощью созданного набора решающих правил;
- автоматически, с применением методов машинного обучения;

Классификация ручным способом – достаточно трудоемкая операция и не подходит для случаев, когда необходимо классифицировать большой объем документов с максимально возможной скоростью.

Способ, связанный с созданием решающих правил исключает главные недостатки классификации вручную. С его помощью возможно автоматически классифицировать большой объем документов с минимальными временными затратами, но данный способ имеет ряд недостатков:

- правила необходимо писать вручную;
- правила необходимо поддерживать в актуальном состоянии;
- при создании правил существует вероятность ошибки;
- правила могут иметь большую и сложную структуру.

Третий предложенный способ наиболее современный. Он подразумевает, что набор правил или критерии принятия решений вычисляются автоматически из обучающего набора данных. Главным недостатком данного способа является необходимость создания обучающего набора данных. Необходимо собрать набор документов, в котором будут представлены все классы и с помощью одного из предыдущих способов присвоить каждому из документов соответствующий класс. В дальнейшем для поиска новых признаков и поддержания моделей в актуальном состоянии, можно обучать новые модели на расширенных наборах данных, тексты в которых уже прошли классификацию с помощью методов машинного обучения.

Целью данной работы является разработка программной реализации научных статей с совместным использованием модели логистической регрессии и модели мешка слов, а также сравнение результатов эффективности разработанного классификатора с другими популярными алгоритмами машинного обучения при использовании разных типов токенов.

В связи с этим возникают следующие задачи:

- Изучить алгоритмы машинного обучения, применимые для решения задачи классификации научных статей;

- Разработать классификатор научных статей;
- Разработать архитектуру программы;
- Реализовать классификатор научных статей;
- Проверить влияние разных типов токенов на точность разработанного классификатора и сравнить с результатами других популярных алгоритмов.

Научная новизна:

- Предложен метод классификации научных статей при помощи совместного использования модели логистической регрессии, статистической меры TF-IDF и модели мешка слов, с использованием разных типов токенов.
- Исследовано влияние разных типов токенов на точность предсказаний популярных алгоритмов машинного обучения с целью выявления типов токенов, по которым точность предсказаний для каждой модели будет наибольшей.

Практическая значимость заключается в предложенном методе классификации научных статей, который можно использовать как самостоятельный классификатор для решения задачи классификации научных статей как при подготовке к публикации научной статьи в журнале, так и при автоматизированной классификации больших неразмеченных корпусов текстов.

1 Обзор моделей машинного обучения, применяемых для классификации текстов

1.1 Общая задача классификации

Пусть X – множество признаков объектов, Y – множество номеров (меток) классов. Существует неизвестная целевая зависимость – отображение $X \to Y$, значения которой известны только на объектах конечной обучающей выборки. Требуется построить алгоритм, аппроксимирующий целевую функцию на всем множестве X.

1.2 Метод *k*-ближайших соседей

Метод *k*-ближайших соседей – это непараметрический метод классификации, который впервые был представлен Эвелин Фикс и Джозефом Ходжесом в 1951 году. При использовании алгоритма для классификации, классифицируемый объект относится к тому классу, который наиболее распространен среди его ближайших соседей. Вкладам соседей можно присвоить веса, чтобы наиболее близкие соседи вносили больший вклад в среднее значение, чем удаленные. Наиболее часто в качестве веса используется значение, пропорциональное обратной величине расстояния от точки классифицируемого объекта до контрольной точки.

Алгоритм может быть применен к выборкам с большим количеством признаков, для этого необходимо определить функцию расстояния, с помощью которой алгоритм будет искать ближайших соседей классифицируемого объекта в многомерном пространстве. Примерами таких функций расстояния могут быть функции Евклидова расстояния [2], расстояние городских кварталов, расстояние Хэмминга.

Выбор параметра k зависит от природы данных. Более высокие значения уменьшают влияние шума на классификацию, но при этом размывают границы между классами.

Признаки объектов могут иметь разный диапазон представленных значений, а значит значения дистанций могут сильно зависеть от признаков с большими диапазонами значений. Поэтому значения признаков обычно подлежат нормализации.

1.3 Линейная регрессия

Линейная регрессия – это регрессионная модель зависимости одной переменной от другой или набора других переменных с линейной функцией зависимости, которая используется в статистике. Метод позволяет предсказывать значения зависимой переменной по значениям независимой переменной или набора независимых переменных.

Регрессионную модель можно определить следующим уравнением (1):

$$y = f(x, b) + \epsilon, E(\epsilon) = 0, \tag{1}$$

где b – параметры модели;

ε – случайная ошибка модели.

Модель называется линейной регрессией, если функция $f(x,b)$ является функцией линейного предсказания и имеет вид (2):

$$f(x, b) = b_0 + b_1 x_1 + b_2 x_2 + ... + b_k x_k, \tag{2}$$

где b_j – коэффициенты регрессии (веса);

x_j – регрессоры (факторы модели, признаки);

k – количество факторов модели [3].

Параметры модели b_j показывают относительное влияние на зависимую переменную по одному из факторов модели. Поскольку модель линейная, скорость изменения зависимой переменной по одному из факторов модели будет постоянной.

$$\forall j\, b_j = \frac{\partial f}{\partial x_j} = \text{const} \tag{3}$$

Коэффициент регрессии b_0 – это значение функции линейного предсказания при нулевом значении всех факторов. Можно считать, что коэффициент b_0 – это параметр при факторе равном 1, а значит функцию линейной регрессии можно записать в виде (4):

$$f(x, b) = b_1 x_1 + b_2 x_2 + \ldots + b_k x_k = \sum_{j=1}^{k} b_j x_j = x^T b, \tag{4}$$

где $x^T=(x_1,x_2,\ldots,x_3)$ – вектор регрессоров;

$b=(b_1,b_2,\ldots,b_3)^{\mathrm{T}}$ – вектор-столбец коэффициентов регрессии.

В данном представлении модели, первый фактор x_1 может быть равен единице, если модель с константой, либо он может являться обычным фактором, если модель без константы.

Коэффициенты для модели линейной регрессии часто подбираются при помощи метода наименьших квадратов, но они могут быть подобраны и другими способами, например, методом наименьших модулей или минимизируя функцию потерь.

1.4 Метод опорных векторов

Метод опорных векторов был разработан в компании «AT&T Bell Laboratories» Владимиром Н. Вапником и Алексеем Я. Червоненкисом в 1963 году. Метод прогнозирования основан на теории Вапника-Червоненкиса, которая пытается объяснить процесс обучения со статистической точки зрения.

Предположим, что имеется набор обучающих примеров, каждый из которых помечен как принадлежащий к одной из двух категорий. Обучающий алгоритм строит модель, которая присваивает новым примерам ту или иную категорию. Метод опорных векторов сопоставляет обучающие примеры с точками в пространстве, с целью найти гиперплоскость максимально удаленную от ближайшей точки данных с обеих сторон (гиперплоскость с максимальным запасом). Затем новые примеры отображаются в том же пространстве и предсказывается их принадлежность к классу в зависимости от того, с какой стороны гиперплоскости они будут расположены.

Каждый объект данных принадлежит одному из двух классов y, имеет количество признаков p и представляется как вектор в p-мерном пространстве. Точки имеют вид:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\}, \tag{5}$$

где y_i принимает значение 1 или –1, в зависимости от того, какому классу принадлежит точка x_i. Вектор x_i размерности p нормализуют значениями [0,1] или [–1,1]. Если значения не будут нормализованы, то точка с большими отклонениями от средних значений координат точек может сильно повлиять на классификатор.

Для того, чтобы обучить модель, необходимо построить разделяющую гиперплоскость в p–1 мерном пространстве, которая будет разделять точки принадлежащие разным классам. Гиперплоскость имеет вид (6):

$$w^T \cdot X - b = 0, \tag{6}$$

где w – вектор нормали к гиперплоскости.

Параметр $\frac{b}{\|w\|}$ определяет смещение гиперплоскости от начала координат вдоль вектора нормали w.

1.4.1 Алгоритм с жестким зазором

Если обучающая выборка линейно разделима, то возможно выбрать две параллельные гиперплоскости, которые разделяют точки разных классов и находятся на максимальном расстоянии друг от друга. Область между двумя гиперплоскостями называется «полем», а гиперплоскость, которая находится посередине между ними называется гиперплоскостью с максимальным запасом (рисунок 1). Эти гиперплоскости можно описать уравнениями (7, 8):

$$w^T \cdot X - b = 1, \tag{7}$$

$$w^T \cdot X - b = -1 \tag{8}$$

Расстояние между этими двумя гиперплоскостями равно $\frac{2}{\|w\|}$ [4]. Для того, чтобы найти максимально возможное расстояние между плоскостями, необходимо минимизировать $\|w\|$.

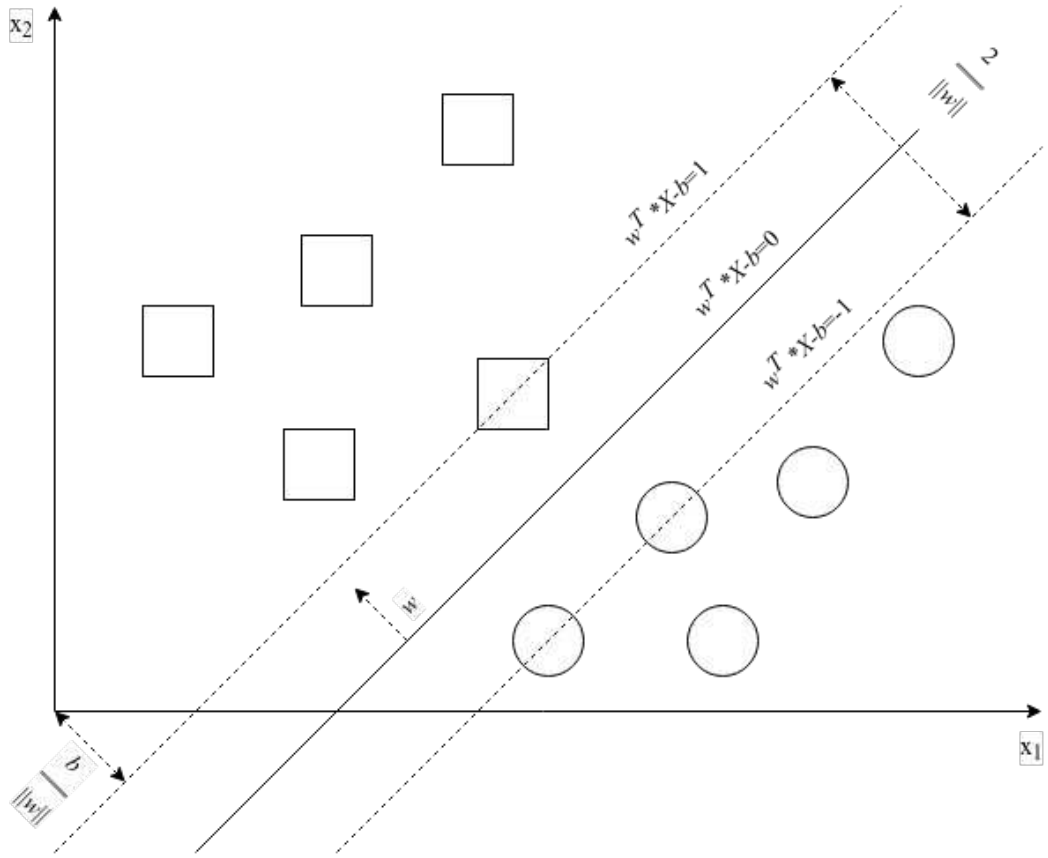

Рисунок 1 – Гиперплоскость с максимальным запасом

Также необходимо убедиться, что ни одна из точек не находится в поле:

$$\begin{cases} w^T \cdot x_i - b \geq 1, \, y_i = 1 \\ w^T \cdot x_i - b \leq -1, \, y_i = -1 \end{cases} \tag{9}$$

Или можно записать так:

$$y_i \left(w^T \cdot x_i - b \right) \geq 1, \, 1 \leq i \leq n \tag{10}$$

Задача построения оптимальной разделяющей гиперплоскости сводится к минимизации $||w||$, при условии (10)

$$\begin{cases} \|w\|^2 \to \min \\ y_i\left(w^T \cdot x_i - b\right) \geq 1,\ 1 \leq i \leq n \end{cases} \tag{11}$$

Необходимо найти вектор множителей лагранжа (двойственных переменных), $\lambda \in R^m$ чтобы для функции Лагранжа (12) выполнялись условия:

$$L(x) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \tag{12}$$

- стационарности: $\min L(x) = L(\hat{x})$;
- дополняющей нежесткости: $\lambda_i g_i(\hat{x}) = 0,\ 1 \leq i \leq m$;
- неотрицательности: $\lambda_i \geq 0,\ 1 \leq i \leq n$.

$$\begin{cases} L(w, b; \lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i \left(y_i\left((w \cdot x_i) - b\right) - 1\right) \to \min_{w,b} \max_{\lambda} \\ \lambda_i \geq 0,\ 1 \leq i \leq n \end{cases} \tag{13}$$

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{n} \lambda_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \left(x_i \cdot x_j\right) \to \min_{\lambda} \\ \lambda_i \geq 0,\ 1 \leq i \leq n \\ \sum_{i=1}^{n} \lambda_i y_i = 0 \end{cases} \tag{14}$$

После того, как вектор двойственных переменных найден, можно найти w и b по формулам:

$$\begin{cases} w=\sum_{i=1}^{n} \lambda_i y_i x_i \\ b=w\cdot x_i - y_i, \lambda_i > 0 \end{cases} \tag{15}$$

Классификатор можно определить формулой (16):

$$a(x)=\operatorname{sign}\left(\sum_{i=1}^{n} \lambda_i y_i x_i \cdot x - b, \lambda_i \neq 0\right) \tag{16}$$

Суммируются только опорные векторы, для которых $\lambda_i \neq 0$.

1.4.2 Алгоритм с мягким зазором

Если выборка линейно неразделима, то для того, чтобы алгоритм мог построить гиперплоскость, введем набор дополнительных переменных $\xi_i \geq 0$, характеризующих величину ошибки на объектах x_i, $1 \leq i \leq n$.

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \rightarrow \min_{w,b,\xi_i} \\ y_i(w\cdot x_i - b) \geq 1-\xi_i, 1 \leq i \leq n \\ \xi_i \geq 0, 1 \leq i \leq n \end{cases} \tag{17}$$

Коэффициент C – параметр настройки метода, который позволяет регулировать отношение между максимизацией ширины поля и минимизацией суммарной ошибки.

$$y_i\left(w^T \cdot x_i - b\right) \geq 1, 1 \leq i \leq n \tag{18}$$

$$\begin{cases} -\sum_{i=1}^{n} \xi_i \left(\lambda_i + \eta_i - C\right) \to \min_{w,b,\eta} \max_{\lambda,\eta} \\ \xi_i \geq 0, \lambda_i \geq 0, \eta_i \geq 0, 1 \leq i \leq n \end{cases} \tag{19}$$

$$\begin{cases} \lambda_i = 0, 1 \leq i \leq n \\ y_i \left(w \cdot x_i - b\right) \geq 1 - \xi_i \end{cases} \tag{20}$$

$$\begin{cases} \eta_i = 0, 1 \leq i \leq n \\ \xi_i = 0 \end{cases} \tag{21}$$

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{n} \lambda_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \left(x_i \cdot x_j\right) \to \min_{\lambda} \\ 0 \leq \lambda_i \leq C, 1 \leq i \leq n \\ \sum_{i=1}^{n} \lambda_i y_i = 0 \end{cases} \tag{22}$$

1.4.3 Ядра

В 1992 году Владимир Вапник, Изабель Гийон и Бернхард Босер предложили способ создания нелинейного классификатора, в основе которого лежит переход от скалярных произведений к произвольным ядрам, позволяющий строить нелинейные разделители. Итоговый алгоритм отличается только тем, что все скалярные произведения в нем заменяются нелинейной функцией ядра. Поскольку размерность полученного пространства может быть больше, чем размерность исходного, преобразование, сопоставляющее скалярные произведения, будет нелинейным, а значит функция, соответствующая в исходном пространстве оптимальной разделяющей гиперплоскости, будет также нелинейной.

1.5 Дерево решений

Дерево принятие решений (дерево классификации, регрессионное дерево) – средство поддержки принятия решений, которое использует древовидную модель решений и их возможных последствий. Древовидные модели широко используются в исследовании операций, анализе решений, статистике, машинном обучении и интеллектуальном анализе данных.

Дерево решений имеет структуру, похожую на блок-схему, в которой внутренние узлы – это признаки, по которым различают случаи, ветви – значения этих признаков, а листья – значения целевой функции. Пути от корня дерева к листьям представляют собой правила классификации. Для того, чтобы классифицировать объект, необходимо спуститься по дереву до листа и получить значение целевой функции, которое и будет предсказанием. Древовидные модели, в которых целевая переменная может принимать дискретный набор значений, называются деревьями классификации. В деревьях классификации листья представляют метки (названия, идентификаторы) классов, ветви – конъюнкции (логическое умножение) признаков, которые ведут к метке класса, соответствующей классифицируемому объекту, а каждый узел соответствует одному из входных признаков классифицируемого объекта.

1.5.1 Алгоритм CART

Алгоритм CART был разработан Лео Брейманом, Джеромом Фридманом, Чарлзом Стоуном и Ричардом Олшеном в 1974–1984 годах. Данный алгоритм предназначен для построения бинарного дерева решений.

Алгоритм построения дерева подразумевает, что на каждом шаге построения, правило, формируемое в новом узле, делит заданное множество примеров на две части. Правая часть, в которой правило выполняется и левая часть, в которой

правило не выполняется. В данном методе перебираются все возможные варианты ветвления для каждого узла и выбирается та предикторная переменная, при которой оценочная функция дает наилучший показатель.

В качестве оценочной функции используется критерий Джини. Это мера того, как часто случайно выбранный из множества элемент был бы неправильно помечен, если бы он был случайно помечен в соответствии с распределением меток в подмножестве. Критерий Джини для набора элементов T с J классами определяется следующей формулой (23):

$$\mathrm{Gini}(T)=\sum_{i=1}^{J}\left(p_i\sum_{k\neq i}p_k\right)=\sum_{i=1}^{J}p_i(1-p_i)=\sum_{i=1}^{J}\left(p_i-p_i^2\right)=\sum_{i=1}^{J}p_i-\sum_{i=1}^{J}p_i^2= \tag{23}$$

$$=1-\sum_{i=1}^{J}p_i^2,$$

где J – классы объектов в наборе;

p_i – относительная частота класса i в наборе.

Критерий принимает нулевое значение в том случае, когда все случаи в узле попадают в одну целевую категорию.

Предположим, что набор T был разбит на две части T_1 и T_2 с количеством примеров N_1 и N_2 соответственно. Показатель качества разбиения можно вычислить по следующей формуле (24):

$$\mathrm{Gini}_{\mathrm{split}}(T)=\frac{N_1}{N}\cdot\mathrm{Gini}(T_1)+\frac{N_2}{N}\cdot\mathrm{Gini}(T_2) \tag{24}$$

Наилучшим разбиением будет то, для которого показатель качества разбиения будет наименьшим.

$$\mathrm{Gini}_{\mathrm{split}}=\frac{N_l}{N}\cdot\left(1-\sum_{i=1}^{J}\left(\frac{y_{li}}{N_l}\right)^2\right)+\frac{N_r}{N}\cdot\left(1-\sum_{i=1}^{J}\left(\frac{y_{ri}}{N_r}\right)^2\right)\rightarrow\min \tag{25}$$

$$\mathrm{Gini}_{\mathrm{split}}=N-\left(\frac{1}{N_l}\cdot\sum_{i=1}^{J}y_{li}^2+\frac{1}{N_r}\cdot\sum_{i=1}^{J}y_{ri}^2\right)\rightarrow\min, \tag{26}$$

где N_l и N_r – число примеров в левом и правом потомке;

y_{li} и y_{ri} – количество экземпляров i-го класса в потомке;

J – классы объектов в наборе.

При построении дерева решений по алгоритму CART, ищется такой вариант ветвления, показатель качества разбиения которого будет наименьшим по сравнению с другими вариантами.

После построения дерева решений, возникает необходимость уменьшения его размера. За счет этого можно упростить дерево решений, повысить точность классификации и избежать переобучения модели. Алгоритм CART использует алгоритм обрезки деревьев с минимальными затратами и сложностью, основная идея которого заключается в получении последовательности уменьшающихся деревьев, каждое из которых получается путем удаления поддерева из предыдущего дерева и заменяя его листовым узлом со значением. Удаляемое поддерево определяется следующим образом:

$$\frac{\mathrm{err}\left(\mathrm{prune}\left(T,t\right),S\right)-\mathrm{err}\left(T,S\right)}{\left|\mathrm{leaves}\left(T\right)\right|-\left|\mathrm{leaves}\left(\mathrm{prune}\left(T,t\right)\right)\right|}, \tag{27}$$

где функция prune*(T, t)* определяет дерево, полученное обрезкой поддеревьев t из дерева T;

функция err*(T, S)* определяет частоту ошибок дерева T по набору данных S.

После получения последовательности уменьшающихся деревьев, производится перекрестная проверка каждого дерева и выбирается наилучшее.

1.5.2 Алгоритмы ID3, C4.5, C 5.0

Алгоритмы генерации деревьев ID3, C4.5, C5.0 были разработаны Джоном Квинланом. В качестве оценочной функции они используют расстояние Кульбака-Лейблера (информационный выигрыш), которое является несимметричной мерой удаленности друг от друга двух вероятностных распределений, определенных на общем пространстве элементарных событий. Информационный выигрыш определяется следующей формулой (28):

$$\mathrm{IG}(T,a)=H(T)-H(T|a)=-\sum_{i=1}^{J} p_i \log_2 p_i - \\ -\sum_{a} p(a)\sum_{i=1}^{J} -\Pr(i|a)\log_2 \Pr(i|a) \rightarrow \max, \quad (28)$$

где *H(T)* – энтропия узла родителя;

H(T|a) – взвешенная сумма энтропии узлов наследников.

Информационный выигрыш используется для принятия решения, какой признак использовать для расщепления на каждом шаге построения дерева.

Алгоритм C4.5 является улучшенной версией алгоритма ID3. Алгоритм поддерживает дискретные и непрерывные атрибуты, отсутствующие значения атрибутов, атрибуты с разной стоимостью и обрезку деревьев.

Алгоритм C5.0 является улучшением алгоритма C4.5. Преимущества алгоритма заключаются в его быстродействии, эффективном использовании памяти, меньшем размере деревьев решений, более высокой точности, возможности взвешивания различных атрибутов и уменьшении шума.

1.6 Алгоритм Random Forest

Алгоритм Random decision forests был впервые предложен Тин Кам Хо в 1995 году и использовал метод случайных подпространств. Лео Брейман и Адель Картер разработали расширение алгоритма [5], которое сочетало в себе идею бутстрэп-агрегирования (бэггинга) и метод случайных подпространств.

Большие деревья решений имеют тенденцию к переобучению, что плохо сказывается на обобщающей способности и точности предсказаний модели. Случайный лес – это способ усреднения нескольких глубоких деревьев решений, обученных на разных частях одного и того же обучающего набора данных, с целью уменьшения дисперсии.

1.6.1 Бутстрэп-агрегирование

Предположим, что имеется набор T с n количеством обучающих примеров, в котором $X=x_1,x_2,\ldots,x_3$ – векторы признаков обучающих примеров, $Y=y_1,y_2,\ldots,y_3$ – известные метки классов.

$$T=\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\ldots,(x_n,y_n)\} \tag{29}$$

При помощи семплирования из набора T генерируется B наборов с повторениями, каждый из которых имеет k обучающих примеров. Если $k=n$, то в среднем $(1-1/n)^n\approx 36{,}8\%$ обучающих примеров из исходного набора не попадут в новый набор. Затем для каждого нового набора $b=1,\ldots,B$ обучается дерево решений, но без процедуры отсечения ветвей.

Классификация объектов происходит путем голосования: каждое дерево из ансамбля деревьев делает предсказание и выбирается тот класс, за который проголосовало наибольшее количество деревьев.

Оценка модели происходит путем вычисления ошибки out-of-bag: для каждого обучающего примера подбирается набор деревьев решений, в обучении которых данный пример не использовался. Каждое дерево решений из этого набора делает предсказание и выбирается класс, за который проголосовало наибольшее количество деревьев. Ошибка вычисляется как отношение количества правильных предсказаний к количеству сделанных предсказаний.

1.6.2 Метод случайных подпространств

Поскольку при построении решающих деревьев выбор признаков зависит от результата вычислений целевой функции, предпочтение будет отдаваться признакам, значение целевой функции которых будет оптимальным, что приведет к корреляции обученных деревьев решений.

Чтобы избежать этого, применяется метод случайных подпространств. В процессе обучения признаки выбираются случайным образом, с заменой для каждого обучаемого дерева решений. Это приводит к тому, что обучаемые деревья не привязываются к признакам, которые кажутся наиболее описательными в обучающем наборе, но не могут быть столь же предсказуемыми для примеров, не входящих в этот набор. Поэтому метод случайных подпространств будет хорошим выбором для задач, в которых количество признаков значительно превышает количество обучающих примеров.

1.7 Алгоритм AdaBoost

Алгоритм ансамблевого обучения AdaBoost был предложен Йоавом Фройндом и Робертом Шапире в 1996 году. Данный алгоритм можно применять в сочетании с разными алгоритмами классификации для улучшения их эффективности. Основная идея алгоритма состоит в том, что сильный классификатор получается путем объединения слабых классификаторов в ансамбль. В отличие от алгоритма бустинга (усиления), алгоритм AdaBoost является адаптивным к слабому обучению. Каждый новый ансамбль классификаторов строится по обучающим примерам, которые были неверно классифицированы предыдущими ансамблями.

Рассмотрим алгоритм построения бинарного классификатора. Предположим, что имеется набор обучающих примеров, который имеет вид (30):

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)\}, \tag{30}$$

где $y_i \in Y = \{-1,1\}$ – класс обучающего примера;

$x_i \in X$ – вектор признаков.

Начальный вектор весов $W_1=(\omega_{1,1},\dots,\omega_{n,1})$ обучающих примеров инициализируется значениями по формуле (31):

$$\omega_{i,1} = \frac{1}{i}, i = 1, \dots, n \tag{31}$$

Для каждого t=1,…,T, находим в ансамбле классификатор h_t, взвешенная ошибка классификации которого будет наименьшая:

$$\epsilon_j = \sum_{i=1}^{n} \omega_{i,t} \left[y_i \neq h_j(x_i) \right] \tag{32}$$

$$h_t = \arg\min \epsilon_j, h_j \in H \tag{33}$$

Пересчитываем коэффициент взвешенного голосования α_t для алгоритма классификации h_t (34):

$$\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} \tag{34}$$

Затем обновляем веса объектов и нормализуем их таким образом, чтобы W_{t+1} являлось распределением вероятностей (35):

$$\omega_{i,t+1} = \omega_{i,t} \cdot \exp\left(-\alpha_t \cdot y_i \cdot h_t(x_i)\right), i = 1, \dots, n \tag{35}$$

$$\omega_{i,t+1}=\frac{\omega_{i,t+1}}{\sum_{j=1}^{n}\omega_{j,t+1}},i=1,\dots,n \tag{36}$$

После обновления весов строим результирующий классификатор (37):

$$H(x)=\mathrm{sign}\left(\sum_{i=1}^{T}\alpha_i\cdot h_i(x)\right) \tag{37}$$

Повторяем процесс обучения, пока для каждого классификатора $h_t \in H$ наименьшая взвешенная ошибка $\varepsilon_t \leq 0{,}5$.

1.8 Перцептрон

Перцептрон – упрощенная математическая модель биологического нейрона, предложенная Френком Розенблатом в 1957 году.

Элементарный перцептрон состоит из трех типов элементов: *S*-элементы, *A*-элементы и *R*-элементы. *S*-элементы – слой сенсоров (рецепторов), которые активируются (возбуждаются) и выдают сигнал +1, если входной сигнал превышает некоторый порог θ, в остальных случаях они выдают сигнал 0. *A*-элементы – ассоциативные элементы. Это логические решающие элементы, которые активируются и выдают сигнал +1, когда алгебраическая сумма его выходных сигналов превышает пороговую величину θ, в остальных случаях они выдают сигнал 0. *R*-элемент – сумматор. Если сумма входных сигналов сумматора положительная, то элемент выдает сигнал +1, а если отрицательная, то –1. В случае, если сумма входных сигналов равна 0, выход считается либо равным нулю, либо неопределенным. Пороговую функцию *R*-элемента можно определить следующей формулой (38):

$$f(x)=sign\left(\sum_{i=1}^{n} w_i x_i-\theta\right), \tag{38}$$

где ω_i – весовой коэффициент i признака;

x_i – значение i признака;

n – количество входов персептрона.

Обучение перцептрона состоит в изменении весовых коэффициентов ω_i связей между A-элементами и R-элементом [6]. Сначала все веса инициализируются небольшим значением, например 0. Для каждого примера j рассчитывается значение пороговой функции и делается предсказание (39):

$$y_j(t)=f\left[\omega(t)\cdot x_j\right]=f\left[\omega_0(t)x_{j,0}+\omega_1(t)x_{j,1}+...+\omega_n(t)x_{j,n}\right] \tag{39}$$

Затем в соответствии с предсказанием обновляются веса:

$$\omega_i(t+1)=\omega_0(t)+r\cdot\left(d_j-y_j(t)\right)x_{j,i}, \tag{40}$$

где t – время;

r – скорость обучения;

d_j – желаемое значение пороговой функции;

$y_j(t)$ – вычисленное значение пороговой функции.

Процесс обучения может повторяться до тех пор, пока ошибка итерации не станет меньше необходимого порога, или не будет завершено определенное количество итераций. Значение ошибки итерации можно рассчитать по формуле:

$$\frac{1}{s}\sum_{j=1}^{s}\left|d_j - y_j(t)\right|, \tag{41}$$

где s – количество обучающих примеров.

После обучения перцептрон готов работать в режиме распознавания (или обобщения) [7]. В случае задачи двоичной классификации, значение пороговой функции будет являться положительным или отрицательным предсказанием принадлежности объекта к определенному классу.

2 Подготовка эксперимента

2.1 Подготовка матрицы признаков

Для подготовки матрицы признаков использовалась модель мешка слов в совокупности со статистической мерой TF-IDF.

Рассмотрим подробнее этап подготовки матрицы признаков. Мешок слов может быть представлен в виде матрицы, строки которой соответствуют отдельным документам или текстам, а столбцы токену. Значением ячейки на пересечении будет количество вхождений слова в документ. При подготовке мешка слов текст при помощи лексера (токенизатора, сегментатора) разбивается на токены, которыми могут быть слова, числа и знаки препинания, а также другие лексемы или наборы символов, которые может распознать лексер. Для каждого токена считается количество вхождений в документ и на основании этих данных создается матрица мешка слов. Строки матрицы соответствуют документам, а столбцы – токенам. Токены и документы в матрице не должны повторяться.

В качестве токенов можно использовать токены, состоящие из лемм или символов. Токены состоящие из разного количества лемм или символов содержат в себе разную информацию и объем словарей до и после прореживания по частоте тоже будет сильно отличаться. Это может значительно повлиять на точность разных алгоритмов, т.к. многие из них используют разные способы подсчета информации и по разному оптимизированы к векторам признаков большой размерности. Рассмотрим пример токенизации предложения 1 (таблица 2) несколькими разными способами (таблица 1):

Таблица 1 – Предложения, токенизированные разными способами

Токены	Тип токена	Количество токенов
1 "алексей", "иванович", "завести", "машина", "и", "поехать", "в", "город"	1 лемма	8
2 “алексей иванович”, “иванович завести”, “завести машина”, “машина и”, “и поехать”, “поехать в”, “в город”	2 леммы	7
3 “алексей иванович завести”, “иванович завести машина”, “завести машина и”, “машина и поехать”, “и поехать в”, “поехать в город”	3 леммы	6
4 “алексей иванович завести машина”, “иванович завести машина и”, “завести машина и поехать”, “машина и поехать в”, “и поехать в город”	4 леммы	5
5 "але", "лек", "екс", "ксе", "сей", "ейи", "йив", "ива", "ван", "ано", "нов", "ови", "вич", "ичз", "чза", "зав", "аве", "вел", "елм", "лма", "маш", "аши", "шин", "ину", "нуи", "уип", "ипо", "пое", "оех", "еха", "хал", "алв", "лвг", "вго", "гор", "оро", "род"	3 символа	37
6 "алек", "лекс", "ексе", "ксей", "сейи", "ейив", "йива", "иван", "вано", "анов", "нови", "ович", "вичз", "ичза", "чзав", "заве", "авел", "велм", "елма", "лмаш", "маши", "ашин", "шину", "инуи", "нуип", "уипо", "ипое", "поех", "оеха", "ехал", "халв", "алвг", "лвго", "вгор", "горо", "ород"	4 символа	36
7 "алекс", "лексе", "ексей", "ксейи", "сейив", "ейива", "йиван", "ивано", "ванов", "анови", "нович", "овичз", "вичза", "ичзав", "чзаве", "завел", "авелм", "велма", "елмаш", "лмаши", "машин", "ашину", "шинуи", "инуип", "нуипо", "уипое", "ипоех", "поеха", "оехал", "ехалв", "халвг", "алвго", "лвгор", "вгоро", "город"	5 символов	35

Рассмотрим пример построения мешка слов из нескольких предложений в таблице 2:

Таблица 2 – Предложения подлежащие токенизации

Предложение	Длина предложения
1 Алексей Иванович завел машину и поехал в город.	8
2 Мотор завелся с первого раза.	5

Окончание таблицы 2

Предложение	Длина предложения
3 Город Москва - столица России.	4
4 Поезд едет до станции “Московская”.	5
5 Ломоносов шел пешком до Москвы.	5

Допустим, что токенизатор должен разбить все предложения на токены, которые будут представлять из себя словарные формы слов. Тогда для каждого предложения мы получим списки лемм, входящих в предложение (таблица 3).

Таблица 3 – Токенизированные предложения

Токенизированное предложение	Длина предложения
1 "алексей", "иванович", "завести", "машина", "и", "поехать", "в", "город"	8
2 "мотор", "завестись", "с", "первый", "раз"	5
3 "город", "москва", "столица", "россия"	4
4 "поезд", "ехать", "до", "станция", "московский"	5
5 "ломоносов", "идти", "пешком", "до", "москва"	5

Построим матрицу вхождений токенов в предложения:

Таблица 4 – Матрица вхождений токенов в документы

Лемма	предложение 1	предложение 2	предложение 3	предложение 4	предложение 5
1 алексей	1	0	0	0	0

Окончание таблицы 4

Лемма	предложение 1	предложение 2	предложение 3	предложение 4	предложение 5
2 иванович	1	0	0	0	0
...	...	...	...	...	...
...	...	...	...	...	...
8 город	1	0	1	0	0
9 мотор	0	1	0	0	0
...	...	...	...	...	...
14 москва	0	0	1	0	1
15 столица	0	0	1	0	0
...	...	...	...	...	...
24 пешком	0	0	0	0	1

Следующим шагом является расчет статистической меры TF-IDF для каждой пары токена и документа. Чтобы узнать частоту термина, необходимо вычислить отношение количества вхождений токена в документ к общему количеству слов в документе [8]:

$$tf(t,d)=\frac{n_t}{\sum_k n_k}, \tag{42}$$

где n_t – число вхождений токена t в документ.

Для того чтобы вычислить обратную частоту документа, необходимо вычислить отношение общего количества документов в коллекции к количеству документов, в которых встречается токен [8]:

$$idf(t, D) = \log\left(\frac{|D|}{|\{d_i \in D | t \in d_i\}|}\right), \tag{43}$$

где $|D|$ число документов в коллекции;

$|\{d_i \in D | t \in d_i\}$ число документов из коллекции D, в которых встречается токен t (когда $n_t \neq 0$).

Далее перемножаем значение TF и IDF и строим матрицу признаков, где строки соответствуют отдельным документам или текстам, а столбцы – токену. Выбор основания логарифма не имеет значения, поскольку вес каждого слова меняется на постоянный множитель, что не влияет на соотношение множителей. В ячейке на пересечении будет значение TF-IDF для соответствующего токена в соответствующем документе [8]:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{44}$$

Таблица 5 – Матрица TF-IDF

Лемма	Предложение 1	Предложение 2	Предложение 3	Предложение 4	Предложение 5
1 алексей	0,625	0	0	0	0
2 иванович	0,625	0	0	0	0
...	...	...	...	...	...
8 город	0,3125	0	0,625	0	0

Окончание таблицы 5

Лемма	Предложение 1	Предложение 2	Предложение 3	Предложение 4	Предложение 5
9 мотор	0	1	0	0	0
...	...	...	...	...	...
14 москва	0	0	0,625	0	0,5
15 столица	0	0	1,25	0	0
...	...	...	...	...	...
24 пешком	0	0	0	0	1

Затем необходимо нормализовать все значения матрицы от 0 до 1. Полученная матрица будет подаваться на вход модели в качестве матрицы признаков.

2.2 Алгоритм обучения модели логистической регрессии

Рассмотрим процесс обучения модели логистической регрессии. Поскольку данные представляют собой оценки важности терминов в контексте документов, предположим, что зависимость между входными данными и классами будет линейная. Модель можно определить следующей формулой [9, 10]:

$$\hat{y}(x)=\sigma(\omega^T x+b), \tag{45}$$

$$\sigma(x)=\frac{1}{1+e^{-x}}, \tag{46}$$

где $\hat{y}(x)$ – выход модели;

$x \in R^d$ – вектор признаков, дополненный фиктивным единичным признаком;

$\omega \in R^d$ – вектор весов;

b – bias.

Перед обучением выборку с примерами необходимо поделить на обучающую и тестовую. Это необходимо для того, чтобы после обучения модели на обучающей выборке, можно было проверить качество модели на тестовой выборке. Обучение модели происходит итерационно, одна итерация называется эпохой. За одну эпоху модель получает партию из некоторого фиксированного количества обучающих примеров, называемую батчем. Для каждого обучающего примера модель делает предсказание. Для неверных предсказаний вычисляется среднее значение функции потерь (бинарная кросс-энтропия) [11]:

$$BCE(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \rightarrow min, \quad (47)$$

где y – класс обучающего примера;

$\hat{y}$ – вычисленное значение функции логистической регрессии.

Для того чтобы понять, куда нужно сместить вес, чтобы минимизировать ошибку для конкретного обучающего примера, необходимо взять производную от функции потерь относительно этого веса.

Вычислим средние значения градиентов для каждого веса и используем их для подстройки весов. После завершения эпохи обучения модели, модель делает предсказания для каждого примера из тестовой выборки. Для неправильных предсказаний вычисляется среднее значение функции потерь. В случае если значение функции потерь уменьшилось по сравнению со значением, рассчитанным на предыдущей эпохе, необходимо сохранить копию модели, так как дальнейшее обучение не гарантирует улучшение модели. Обучение модели останавливается в том случае, когда значение функции потерь не уменьшалось в течение некоторого фиксированного количества эпох, устанавливаемого в начале процесса обучения.

Данная модель способна предсказывать принадлежность к двум классам. Для того чтобы модель могла предсказывать принадлежность к более чем двум классам, вектор весов надо заменить матрицей весов, а в качестве функции потерь необходимо использовать кросс-энтропию (перекрестную энтропию) [12]:

$$\mathrm{CE}(\hat{y}, y) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i) \tag{48}$$

Вместо логистической функции необходимо применить функцию Softmax [13]:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{49}$$

2.3 Программирование модуля

2.3.1 Программирование токенизаторов

В рамках данной работы было разработано три типа токенизаторов:

1. Токенизатор разбивающий текст на токены из 1 леммы;
2. Токенизатор разбивающий текст на токены из *n* лемм;
3. Токенизатор разбирающий текст на токены из *n* символов.

Во время разработки токенизаторов были использованы следующие модули из «pip» менеджера пакетов «Python» [14]: модуль «re» предоставляет операции сопоставления регулярных выражений, модуль «abc» предоставляет структуру для определения абстрактных базовых классов, модуль «natasha» [15] позволяет решить базовые задачи обработки естественного языка для русского языка, такие как токенизация, сегментация, лемматизация, нормализация фраз, синтаксический анализ, тегирование, извлечение фактов.

Для реализации токенизаторов был определен базовый абстрактный класс (рисунок 2), в котором определены сигнатуры методов «tokenize()» и «tokenizer_type()». Метод «Tokenizer_type()» возвращает тип токенизатора, а «tokenize()» токенизирует текст. Приватные методы «__get_tokens_…()» каждого класса реализуют процесс токенизации и вызываются из публичного метода «tokenize()».

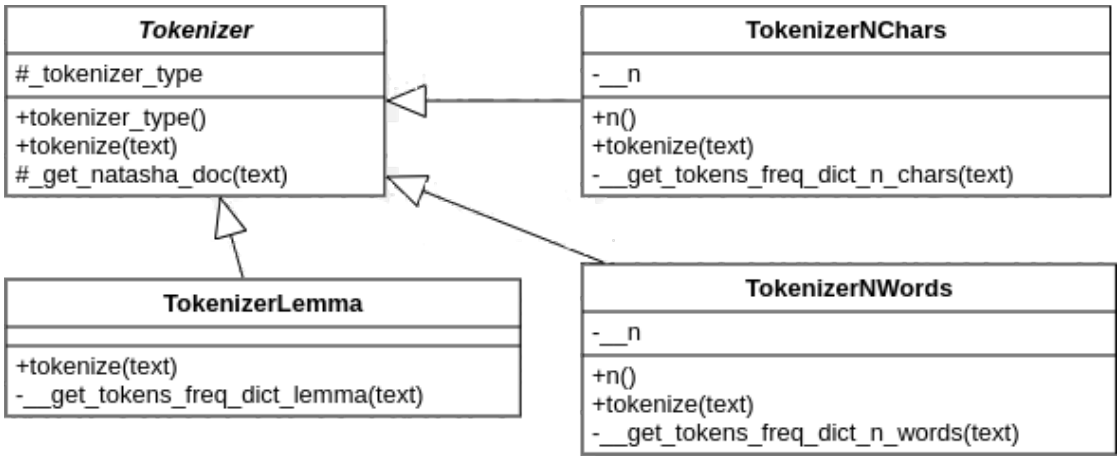

Рисунок 2 – Диаграмма классов токенизаторов

Процесс токенизации начинается с разбиения текста на токены с помощью модуля «natasha». Токенами могут быть русские слова, слова иностранных языков, цифры, знаки препинания, наборы из символов, отделенные пробелами или знаками препинания с обеих сторон. Затем с помощью регулярного выражения отбираются только токены, которые являются словами русского языка. В случае, если токены будут использовать леммы слов, все токены лемматизируются. Далее в зависимости от типа токенов, токенизатор проходит по имеющемуся отобранному списку токенов и формирует токены необходимого типа. Если сформированный токен имеется в словаре, то количество увеличивается на 1, иначе токен добавляется в словарь со значением 1. Результатом работы токенизатора является отсортированный частотный словарь токенов, в котором подсчитана частота встречаемости токена в тексте.

```
{'услуга': 62, 'в': 38, 'учет': 36, 'и': 36, 'на': 23, 'от': 18, 'с': 17, 'дт': 17, 'кт': 17, 'руб'
{'в': 59, 'и': 31, 'вирусный': 20, 'на': 19, 'рынок': 17, 'интернет-маркетинг': 15, 'который': 15,
{'в': 50, 'рф': 34, 'суд': 33, 'по': 26, 'организация': 25, 'ликвидация': 24, 'с': 24, 'о': 21, 'ве
{'научный': 94, 'в': 85, 'труд': 84, 'рынок': 69, 'и': 68, 'работник': 68, 'с': 62, 'на': 38, 'наук
{'в': 84, 'и': 83, 'собственность': 78, 'на': 32, 'экономический': 27, 'с': 26, 'трансформация': 24
{'право': 70, 'и': 46, 'в': 39, 'неимущественный': 30, 'на': 27, 'гражданин': 26, 'личный': 24, 'то
{'и': 55, 'в': 39, 'цифровой': 37, 'экономика': 27, 'развитие': 17, 'технология': 14, 'на': 12, 'уз
{'и': 47, 'в': 45, 'персонал': 37, 'подбор': 30, 'компания': 29, 'кандидат': 22, 'который': 19, 'со
{'и': 156, 'в': 137, 'с': 69, 'север': 62, 'о': 57, 'россия': 56, 'область': 56, 'коренной': 55, 'н
{'и': 77, 'в': 74, 'бизнес': 41, 'социальный': 34, 'с': 23, 'на': 20, 'развитие': 19, 'общество': 1
press close to exit
```

Рисунок 3 – Результат работы токенизатора «TokenizerLemma»

```
{'усл': 64, 'слу': 64, 'луг': 62, 'ост': 56, 'чет': 54, 'уче': 50, 'нос': 42, 'ени': 36, 'ств': 35,
{'ани': 51, 'ого': 51, 'ова': 49, 'льн': 48, 'ени': 46, 'нте': 45, 'про': 42, 'ети': 39, 'тер': 38,
{'ени': 109, 'ого': 80, 'аци': 57, 'ния': 54, 'опр': 50, 'про': 48, 'ани': 45, 'ног': 43, 'ции': 43,
{'нау': 127, 'ост': 113, 'ова': 104, 'ауч': 103, 'учн': 101, 'ани': 100, 'ния': 95, 'уда': 94, 'тру'
{'ств': 165, 'енн': 158, 'ост': 132, 'тве': 124, 'нно': 121, 'вен': 118, 'ени': 111, 'нос': 108, 'ст
{'пра': 83, 'рав': 80, 'ств': 76, 'ест': 70, 'енн': 58, 'вен': 48, 'уще': 47, 'щес': 46, 'нны': 44,
{'ров': 55, 'кон': 45, 'ени': 45, 'ном': 40, 'оми': 40, 'оно': 39, 'циф': 38, 'ифр': 38, 'фро': 38,
{'ост': 67, 'ани': 64, 'ени': 52, 'ния': 50, 'льн': 49, 'нал': 47, 'пер': 46, 'она': 45, 'под': 40,
{'ого': 216, 'ско': 200, 'енн': 187, 'льн': 187, 'ени': 168, 'нны': 163, 'ных': 149, 'ств': 147, 'аг
{'ств': 114, 'тве': 91, 'льн': 89, 'ост': 80, 'енн': 77, 'ени': 66, 'аль': 61, 'нно': 59, 'ьно': 55,
press close to exit
```

Рисунок 4 – Результат работы токенизатора «TokenizerNChars»

2.3.2 Программирование модуля TFIDF

Для расчета статистической меры TFIDF в разработаном модуле «tfidf» определены 3 класса. Класс «Text» предоставляет функциональность, связанную с хранением токенизированных текстов в программе. Он имеет свойства: название текста, текст, название класса, частотный словарь токенов, количество токенов, тип токенов.

Класс «TextsCoprus» хранит список токенизированных текстов и рассчитывает статистические данные для всего корпуса текстов, такие как: количество вхождений токена в корпус, количество текстов, в которых встречается токен, количество уникальных токенов в корпусе, количество текстов в корпусе,

тип токенов и вспомогательные словари соответствий идентификаторов и токенов, с помощью которых класс «StatCorpus» ориентируется в корпусе текстов.

Класс «StatCorpus» предоставляет функциональность, связанную с расчетом матриц признаков по имеющемуся токенизированному корпусу текстов. Данный класс поддерживает прореживание словаря по частоте встречаемости токенов, максимальный размер словаря, соотношение разделения выборки на обучающую и валидационную, может рассчитать веса классов текстов в общей выборке, матрицу вхождений токена в текст (рисунок 5), матрицу количества вхождений токена в текст (рисунок 6), матрицу TF, матрицу IDF, матрицу TF-IDF (рисунок 7), нормализовать матрицы значениями от 0 до 1. Также с помощью этого класса можно случайным образом сформировать обучающую и валидационную выборку.

```
  (2283, 11920) 1.0
  (2283, 13983) 1.0
  (2283, 14278) 1.0
  (2283, 14665) 1.0
  (2283, 16858) 1.0
  (2283, 17017) 1.0
  (2283, 17467) 1.0
  (2283, 17781) 1.0
  (2283, 18396) 1.0
  (2283, 18775) 1.0
  (2283, 18804) 1.0
  (2283, 19893) 1.0
press close to exit
```

Рисунок 5 – Результат расчета матрицы вхождений

```
  (2283, 13962) 1.0
  (2283, 14458) 4.0
  (2283, 14775) 1.0
  (2283, 15857) 1.0
  (2283, 15984) 8.0
  (2283, 16330) 2.0
  (2283, 17842) 1.0
  (2283, 18166) 1.0
  (2283, 18811) 1.0
  (2283, 19736) 1.0
  (2283, 19897) 1.0
  (2283, 20325) 1.0
press close to exit
```

Рисунок 6 – Результат расчета матрицы количества вхождений

```
  (2283, 13260) 0.28606853
  (2283, 13594) 1.1442741
  (2283, 14368) 0.28606853
  (2283, 15605) 0.32693547
  (2283, 15933) 1.9071234
  (2283, 16136) 2.2885482
  (2283, 16451) 0.38142473
  (2283, 16851) 0.76284945
  (2283, 17767) 0.38142473
  (2283, 17986) 0.4577097
  (2283, 19328) 0.4577097
  (2283, 19609) 0.9154194
press close to exit
```

Рисунок 7 – Результат расчета матрицы TF-IDF

Поскольку результирующие матрицы имеют очень большой размер, а большая часть матриц заполнена значениями 0, для хранения матриц в памяти компьютера используются разреженные матрицы на основе словаря ключей, реализованные в модуле «scipy.sparse» [16].

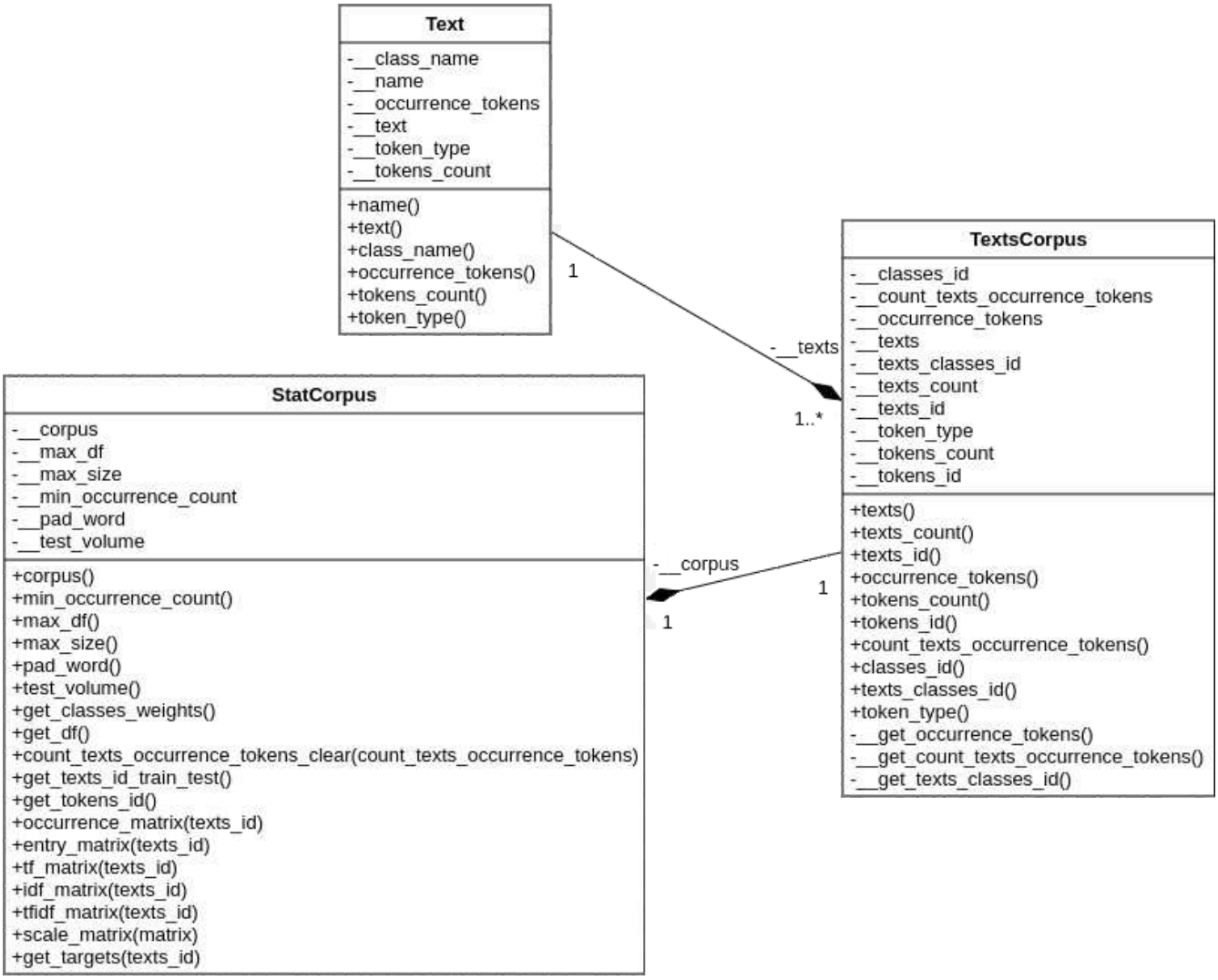

Рисунок 8 – Диаграмма классов модуля tfidf

2.3.3 Программирование алгоритма обучения модели логистической регрессии

При разработке алгоритма обучения модели логистической регрессии «nn.Linear» использовались модули «torch» [17] и «numpy» [18]. На первом этапе модель переводится в режим обучения. Для оптимизации модели использовался метод адаптивной оценки моментов «torch.optim.Adam», но предусмотрена возможность использовать другой оптимизатор. Регулятор скорости обучения «logictic_regression.lr_scheduler» уменьшает шаг на плато, с целью попытаться

найти локальный минимум при оптимизации функции потерь. На каждой итерации обученная модель делает предсказание для обучающих признаков. На основании предсказаний, вычисляются значения функции потерь и значения градиентов для каждого обучающего примера. Затем оптимизатор делает шаг в сторону противоположную значениям градиентов.

На следующем этапе модель переводится в режим проверки. Модель повторяет те же шаги, что и на этапе обучения, но оптимизатор не делает градиентный шаг. Если модель улучшила среднее значение функции потерь, то новая модель сохраняется как лучшая модель. Затем начинается новая эпоха и процесс обучения и проверки повторяется. Если модель не улучшила среднее значение функции потерь в течении установленного количества эпох, процесс обучения прекращается.

```
Среднее значение функции потерь на валидации 0.5124317519366741

Эпоха 17
Эпоха: 71 итераций, 0.48 сек
Среднее значение функции потерь на обучении 0.01096718481534885
Среднее значение функции потерь на валидации 0.5118942372500896

Эпоха 18
Эпоха: 71 итераций, 0.48 сек
Среднее значение функции потерь на обучении 0.010371709333008362
Среднее значение функции потерь на валидации 0.5130483619868755
Модель не улучшилась за последние 10 эпох, прекращаем обучение
press close to exit
```

Рисунок 9 – Результат обучения модели логистической регрессии

3 Анализ полученных результатов экспериментов

Для оценки точности классификатора был собран корпус из 2825 текстов, опубликованных в научных журналах. Всем текстам присвоен код паспорта научной специальности ВАК, корпус содержит тексты, относящиеся к 7 наукам (таблица 6): биологические науки (03.00.00), технические науки (05.00.00), экономические науки (08.00.00), филологические науки (10.00.00), медицинские науки (14.00.00), психологические науки (19.00.00), социологические науки (22.00.00).

Таблица 6 – Распределение классов

Класс	1	2	3	4	5	6	7
Количество текстов	615	1143	243	219	170	280	155

При подготовке к экспериментам были составлены словари с токенами, состоящими из 1 слова, 2 слов, 3 слов, 4 слов, 3 символов, 4 символов и 5 символов.

Таблица 7 – Количество токенов в корпусе по типам

Тип токена	Количество токенов до прореживания словаря	Количество *n* токенов после прореживания словаря	$n' = \sqrt{n}$ количество признаков (округлено до ближайшего целого)	$n' = \log_2 n$ количество признаков (округлено до ближайшего целого)
1 лемма	114304	20362	143	14
2 леммы	1267086	69421	263	16
3 леммы	2409446	25198	159	15
4 леммы	2677090	7379	86	13
3 символа	35912	18070	134	14

Окончание таблицы 7

Тип токена	Количество токенов до прореживания словаря	Количество *n* токенов после прореживания словаря	$n' = \sqrt{n}$ количество признаков (округлено до ближайшего целого)	$n' = \log_2 n$ количество признаков (округлено до ближайшего целого)
4 символа	264916	109481	331	17
5 символов	1080037	294957	543	18

3.1 Оценка точности модели логистической регрессии

Для оценки точности классификатора было проведено 7 экспериментов (таблица 8), в которых проверялась точность модели с разными типами токенов.

Таблица 8 – Результаты экспериментов с видами токенов

Тип токена	Количество уникальных токенов	Количество уникальных токенов после прореживания словаря	Доля правильных предсказаний лучшей модели	Среднее значение функции потерь при обучении	Среднее значение функции потерь при валидации
1 лемма	114304	20362	0.866	0,24047	0,40865
2 леммы	1267086	69421	0.846	0,97052	0,48109
3 леммы	2409446	25198	0.842	0,05909	0,40843
4 леммы	2677090	7379	0.814	0,12485	0,59561
3 символа	35912	18070	0.830	0,19970	0,52240
4 символа	264916	109481	0.814	0,91202	0,57146
5 символов	1080037	294957	0.842	0,79328	0,39698

Модели логистической регрессии показали хорошую эффективность при решении задачи классификации научных статей. Точность модели слабо зависит от выбранного типа токена, но при использовании других моделей, правильный выбор токенизатора может значительно снизить количество признаков и повысить точность классификатора.

3.2 Проверка влияния типов токенов на эффективность других алгоритмов машинного обучения

3.2.1 Алгоритм *k*-ближайших соседей

Модели использующие алгоритм *k*-ближайших соседей инициализировались с использованием трех видов метрик, с помощью которых измерялось расстояние между точками: евклидово расстояние, расстояние минковского и расстояние городских кварталов (таблицы А.1–А.6).

Евклидово расстояние – расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора:

$$d(p,q)=\sqrt{\sum_{k=1}^{n}(p_k-q_k)^2} \tag{50}$$

Расстояние городских кварталов – сумма длин проекций отрезка между точками на оси координат.

$$d_1(p,q)=\sum_{i=1}^{n}|p_i-q_i| \tag{51}$$

Расстояние Минковского – параметрическая метрика в евклидовом пространстве, которую можно рассматривать как обобщение евклидова расстояния и расстояния городских кварталов.

$$\rho(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{52}$$

Для расчета вклада ближайших соседних точек в результат предсказания использовались два вида весовых функций: однородные веса и веса обратно пропорциональные расстоянию. Были проведены расчеты для *k*-ближайших соседей, при *k*=(5,10,…,45,50,60,…,90,100). В таблице 9 приведены результаты предсказаний некоторых классификаторов:

Таблица 9 – Результаты обучения моделей *k*-ближайших соседей

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
k=5 euclidean uniform	0,402	0,362	0,642	0,524	0,417	0,118	0,268
k=10 euclidean uniform	0,228	0,126	0,709	0,394	0,283	0,146	0,197
k=20 euclidean uniform	0,126	0,016	0,63	0,472	0,213	0,122	0,118
k=5 euclidean distance	0,469	0,15	0,579	0,425	0,244	0,217	0,421
k=10 euclidean distance	0,331	0,264	0,543	0,528	0,441	0,102	0,303
k=20 euclidean distance	0,071	0,087	0,732	0,417	0,252	0,118	0,228
k=5 manhattan uniform	0,354	0,425	0,409	0,524	0,15	0,157	0,209
k=10 manhattan uniform	0,291	0,421	0,417	0,433	0,26	0,213	0,169
k=20 manhattan uniform	0,5	0,398	0,343	0,441	0,331	0,181	0,083
k=5 manhattan distance	0,386	0,437	0,394	0,508	0,272	0,165	0,157
k=10 manhattan distance	0,469	0,437	0,374	0,382	0,299	0,157	0,138
k=20 manhattan distance	0,571	0,374	0,417	0,48	0,386	0,134	0,169

Окончание таблицы 9

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
k=5 minkowski uniform	0,339	0,209	0,728	0,508	0,213	0,126	0,307
k=10 minkowski uniform	0,193	0,169	0,61	0,52	0,177	0,102	0,087
k=20 minkowski uniform	0,173	0,079	0,701	0,409	0,087	0,177	0,122
k=5 minkowski distance	0,394	0,142	0,535	0,461	0,445	0,165	0,346
k=10 minkowski distance	0,189	0,154	0,72	0,425	0,217	0,098	0,26
k=20 minkowski distance	0,228	0,087	0,535	0,421	0,126	0,11	0,181

Из 630 обученных классификаторов (Приложение А), лучшую точность имеют классификаторы, матрица признаков которых была построена по токенам, состоящим из 3 лемм, а расстояние между точками измерялось при помощи евклидовой метрики и метрики Минковского. Использование весов обратно пропорциональных расстоянию от классифицируемой точки до ближайшей соседней позволяет в ряде случаев (k=15,…,25) повысить точность классификаторов с евклидовой метрикой на 11–23%.

Классификаторы измеряющие расстояние с помощью расстояния городских кварталов имеют большую точность на токенах из 1 и 2 лемм, однако точность лучших экземпляров таких моделей не превышает 60%.

При исследовании влияние параметра k на точность модели, было выявлено, что при (k=5,…,45) возможно обучить классификатор, точность которого будет от 50% до 80%, но при больших значениях параметра, точность классификаторов снижается.

3.2.2 Метод опорных векторов

Модели основанные на методе опорных векторов (SVM) могут строить нелинейные гиперплоскости, если заменить все скалярные произведения

нелинейными функциями ядра. При обучении моделей проверялось влияние выбора разных функций ядра на точность предсказаний. Были проверены 4 вида функций ядра: радиально базисная функция, линейная функция, полиномиальное ядро и сигмоидальное ядро. В качестве решающей стратегии в мультиклассовом режиме используется стратегия «один против одного».

Также проверим, зависит ли точность классификации текстов при использовании метрики TF-IDF от размерности вектора признаков. Гипотеза проверялась с помощью разных типов токенов, общее количество которых зависит от количества найденных часто повторяющихся элементов текста.

Таблица 10 – Результаты обучения моделей основанных на методе опорных векторов «libsvm»

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
SVC (rbf)	0,496	0,598	0,378	0,429	0,445	0,476	0,398
SVC (linear)	0,083	0,039	0,075	0,197	0,035	0,126	0,047
SVC (polynomial)	0,138	0,193	0,512	0,37	0,157	0,138	0,13
SVC (sigmoid)	0,783	0,846	0,791	0,756	0,744	0,776	0,85

При тестировании использовались классификаторы, реализованные в программной библиотеке «libsvm». Модели с сигмоидальным ядром имеют большую точность (74–85%) по сравнению с моделями использующими другие ядерные функции (3–59%). Модели с сигмоидальным ядром, матрицы признаков которых были построены на токенах из 2 лемм и 5 символов достигли точности прогнозирования 84–85%.

Для задачи классификации текстов наиболее подходят модели с сигмоидальным ядром, т.к. построенная с помощью сигмоидального ядра

гиперплоскость более точно разделяет обучающие примеры разных классов в *n*-мерном пространстве.

При использовании метрики TF-IDF для классификации текстов, зависимости между размерностью векторов признаков и точностью прогнозов не обнаружено.

Протестируем линейные классификаторы, реализованные в программной библиотеке «liblinear». Основные отличия от реализации «libsvm» – лучшее масштабирование для большого количества выборок, реализация мультиклассового режима «один против всех».

Таблица 11 – Результаты обучения моделей основанных на методе опорных векторов «liblinear»

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
LinearSVC (squared hinge)	0,811	0,831	0,78	0,748	0,602	0,776	0,835
LinearSVC (hinge)	0,764	0,819	0,764	0,693	0,713	0,756	0,815
LinearSVC (crammer_singer)	0,752	0,827	0,756	0,681	0,642	0,732	0,78

Наибольшую точность показали классификаторы обученные на матрицах признаков, которые были построены по токенам из 2 лемм, а наименьшую точность – классификаторы обученные на матрицах признаков, построенных по токенам из 3 символов.

3.2.3 Дерево решений

При обучении деревьев решений проверялась точность деревьев, обученные по алгоритмам 4.5 и CART использующие в качестве критериев информационный выигрыш и критерий Джини соответственно. Каждый алгоритм проверялся на

двух стратегиях выбора лучшего разбиения: лучшее значение критерия и случайное разбиение. Также проверялось влияние гиперпараметра максимальной глубины дерева решений (таблицы А.7–А.8).

Таблица 12 – Результаты обучения моделей деревьев решений

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
gini best 5	0,913	0,563	0,622	0,882	0,85	0,972	0,992
gini best 10	0,945	0,984	0,992	0,992	0,854	0,953	0,953
gini best 15	0,902	0,996	0,988	0,996	0,819	0,976	0,972
gini best 30	0,917	1	0,992	0,992	0,886	0,969	0,972
gini best 50	0,913	0,996	0,996	1	0,89	0,972	0,941
gini random 5	0,921	0,98	0,98	0,992	0,874	0,98	0,957
gini random 10	0,949	0,992	0,984	0,996	0,862	0,969	0,988
gini random 15	0,917	1	0,992	0,992	0,886	0,969	0,972
gini random 30	0,89	0,996	0,988	0,984	0,807	0,961	0,961
gini random 50	0,921	0,972	0,913	0,988	0,831	0,961	0,957
entropy best 5	0,929	0,984	0,972	0,988	0,87	0,953	0,965
entropy best 10	0,933	0,988	0,996	1	0,839	0,984	0,972
entropy best 15	0,929	0,992	0,984	0,984	0,902	0,957	0,976
entropy best 30	0,937	0,992	0,984	0,996	0,929	0,961	0,969
entropy best 50	0,933	0,996	0,996	0,984	0,925	0,965	0,921
entropy random 5	0,913	0,996	0,996	1	0,89	0,972	0,941
entropy random 10	0,693	0,378	0,555	0,906	0,598	0,909	0,976

Окончание таблицы 12

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
entropy random 15	0,787	0,976	0,996	0,984	0,705	0,945	0,929
entropy random 30	0,89	0,988	0,98	0,976	0,811	0,933	0,874
entropy random 50	0,878	0,98	0,984	0,984	0,764	0,953	0,906

Деревья решений показывают лучшие результаты на токенах, которые сохраняют информацию о расположении слов относительно друг друга, но при обучении таких деревьев решений, им требуется большая глубина по сравнению с деревьями решений, обученными на токенах, не сохраняющих информацию о взаимном расположении слов.

3.2.4 Алгоритм Random Forest

При обучении моделей Random Forest проверялось влияние выбора критерия, по которому алгоритм принимает решение о выборе признака при формировании правила в узле. Также проверялось влияние выбора максимального количества деревьев, максимальной глубины и максимального количества признаков на точность обучаемых моделей (таблицы А.9–А.12). При n – общем количестве признаков, для определения максимального количества признаков, участвующих в классификации, было протестировано два возможных варианта:

$$n' = \sqrt{n} \tag{53}$$

$$n' = \log_2 n \tag{54}$$

Таблица 13 – Результаты обучения моделей Random Forest с критерием Джини

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=5 max_depth=5 max_features=sqrt)	0,602	0,732	0,197	0,791	0,589	0,669	0,646
Random Forest (gini n_trees=5 max_depth=20 max_features=sqrt)	0,75	0,858	0,453	0,976	0,624	0,715	0,732
Random Forest (gini n_trees=10 max_depth=5 max_features=sqrt)	0,717	0,679	0,242	0,823	0,683	0,797	0,756
Random Forest (gini n_trees=10 max_depth=20 max_features=sqrt)	0,764	0,892	0,663	0,974	0,744	0,795	0,846
Random Forest (gini n_trees=15 max_depth=5 max_features=sqrt)	0,787	0,846	0,226	0,965	0,764	0,909	0,854
Random Forest (gini n_trees=15 max_depth=20 max_features=sqrt)	0,825	0,915	0,713	0,986	0,738	0,876	0,925
Random Forest (gini n_trees=20 max_depth=5 max_features=sqrt)	0,787	0,768	0,512	0,951	0,872	0,886	0,909
Random Forest (gini n_trees=20 max_depth=20 max_features=sqrt)	0,864	0,955	0,634	0,992	0,77	0,904	0,937
Random Forest (gini n_trees=5 max_depth=5 max_features=log2)	0,396	0,281	0,789	0,27	0,392	0,36	0,315
Random Forest (gini n_trees=5 max_depth=20 max_features=log2)	0,614	0,411	0,781	0,616	0,443	0,457	0,433
Random Forest (gini n_trees=10 max_depth=5 max_features=log2)	0,555	0,49	0,884	0,488	0,5	0,482	0,441
Random Forest (gini n_trees=10 max_depth=20 max_features=log2)	0,632	0,547	0,756	0,839	0,449	0,52	0,618
Random Forest (gini n_trees=15 max_depth=5 max_features=log2)	0,551	0,37	0,933	0,636	0,549	0,533	0,406
Random Forest (gini n_trees=15 max_depth=20 max_features=log2)	0,724	0,61	0,846	0,896	0,543	0,555	0,642
Random Forest (gini n_trees=20 max_depth=5 max_features=log2)	0,616	0,465	0,986	0,675	0,573	0,539	0,661

Окончание таблицы 13

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=20 max_depth=20 max_features=log2)	0,699	0,624	0,636	0,9	0,516	0,577	0,555

Всего было проведено 448 обучений и замеров точности моделей Random Forest с разными параметрами (таблицы А.9–А.12). При использовании критерия Джини, если количество признаков участвующих в классификации определяется по формуле (53), то наибольшая точность достигается при использовании матрицы признаков, рассчитанной по токенам, состоящим из 4 лемм. Немного уступают в точности модели, обученные по матрицам признаков, рассчитанных по токенам, состоящим из 2 лемм. Худшие результаты показали модели, обученные на матрицах признаков построенных по токенам, состоящим из 3 лемм.

При использовании критерия Джини, если количество признаков определяется по формуле (54), то результаты точности прогнозов моделей сильно отличаются от результатов моделей, в которых количество признаков определялось по формуле (53). Наиболее эффективными оказались модели, обученные на матрицах признаков, которые были построены по токенам, состоящим из 3 лемм. При увеличении глубины деревьев с 15 до 20 узлов, замечено резкое снижение точности классификации моделей с разным количеством деревьев в ансамбле (таблица 14).

Таблица 14 – Пример снижения точности при увеличении глубины деревьев для моделей с параметрами gini sqrt 4_chars

Глубина	Количество деревьев			
	5	10	15	20
5	0,789	0,884	0,933	0,986
10	0,864	0,896	0,959	0,967

Окончание таблицы 14

Глубина	Количество деревьев			
	5	10	15	20
15	0,846	0,959	0,978	0,99
20	0,781	0,756	0,846	0,636

Таблица 15 – Результаты обучения моделей Random Forest с информационным выигрышем

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=5 max_depth=5 max_features=sqrt)	0,638	0,447	0,22	0,754	0,596	0,657	0,693
Random Forest (entropy n_trees=5 max_depth=20 max_features=sqrt)	0,701	0,783	0,581	0,955	0,622	0,675	0,76
Random Forest (entropy n_trees=10 max_depth=5 max_features=sqrt)	0,76	0,76	0,315	0,947	0,785	0,854	0,76
Random Forest (entropy n_trees=10 max_depth=20 max_features=sqrt)	0,795	0,902	0,728	0,988	0,711	0,841	0,854
Random Forest (entropy n_trees=15 max_depth=5 max_features=sqrt)	0,722	0,862	0,518	0,984	0,685	0,833	0,854
Random Forest (entropy n_trees=15 max_depth=20 max_features=sqrt)	0,787	0,921	0,616	0,994	0,752	0,801	0,815
Random Forest (entropy n_trees=20 max_depth=5 max_features=sqrt)	0,805	0,846	0,488	0,982	0,811	0,911	0,819
Random Forest (entropy n_trees=20 max_depth=20 max_features=sqrt)	0,846	0,943	0,701	0,992	0,774	0,868	0,85
Random Forest (entropy n_trees=5 max_depth=5 max_features=log2)	0,421	0,315	0,766	0,171	0,368	0,392	0,26
Random Forest (entropy n_trees=5 max_depth=20 max_features=log2)	0,52	0,474	0,728	0,524	0,415	0,433	0,453
Random Forest (entropy n_trees=10 max_depth=5 max_features=log2)	0,569	0,348	0,86	0,541	0,478	0,404	0,445

Окончание таблицы 15

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=10 max_depth=20 max_features=log2)	0,608	0,583	0,77	0,805	0,47	0,52	0,543
Random Forest (entropy n_trees=15 max_depth=5 max_features=log2)	0,638	0,364	0,943	0,561	0,569	0,488	0,598
Random Forest (entropy n_trees=15 max_depth=20 max_features=log2)	0,685	0,577	0,945	0,73	0,5	0,518	0,618
Random Forest (entropy n_trees=20 max_depth=5 max_features=log2)	0,583	0,469	0,972	0,624	0,614	0,553	0,575
Random Forest (entropy n_trees=20 max_depth=20 max_features=log2)	0,648	0,612	0,701	0,906	0,53	0,539	0,622

При проведении аналогичных расчетов с использованием в качестве критерия информационный выигрыш результаты воспроизвелись. Использование разных критериев при неизменности остальных параметров моделей показало, что значительные отличия в точности предсказаний отсутствуют.

В целом увеличение количества и глубины деревьев в ансамбле положительно влияет на точность предсказаний, но при чрезмерном увеличении глубины деревьев, точность некоторых моделей может значительно снизиться.

3.2.5 Алгоритм AdaBoost

Во время проверки эффективности алгоритма AdaBoost для классификации научных статей с использованием статистической метрики TF-IDF проверялись два вида алгоритмов обучения на семи типах токенов. Алгоритм SAMME.R использует вероятности классов при обучении модели, а алгоритм SAMME использует только результаты прогнозов.

Таблица 16 – Результаты обучения моделей AdaBoost

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
AdaBoost (algorithm=SAMME)	0,928	0,987	0,992	0,991	0,898	0,922	0,961
AdaBoost (algorithm=SAMME.R)	0,635	0,776	0,872	0,864	0,742	0,265	0,612

Алгоритм SAMME показал более высокую эффективность, по сравнению с алгоритмом SAMME.R при решении задачи классификации научных статей. Модели, использующие оба алгоритма имеют наибольшую точность при использовании матриц признаков, которые построены по токенам, сохраняющим информацию о взаимном расположении слов.

3.2.6 Многослойный персептрон

При обучении моделей многослойных перцептронов исследовалось влияние выбранной функции активации и оптимизатора на точность классификации. Из функций активации проверялись: линейный выпрямитель (ReLU), безоперационная активация (identity), гиперболический тангенс (tanh), логистическая функция (logistic). В качестве оптимизаторов применялись алгоритмы: стохастический градиентный спуск (SGD), алгоритм Бройдена-Флетчера-Гольдфарба-Шанно с ограниченной памятью (LBFGS) и метод адаптивной оценки моментов (Adam).

Таблица 17 – Результаты обучения моделей многослойных перцептронов

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
relu adam	0,815	0,862	0,803	0,772	0,776	0,815	0,78
identity adam	0,823	0,866	0,799	0,768	0,776	0,807	0,78
logistic adam	0,811	0,85	0,78	0,776	0,764	0,791	0,756

Окончание таблицы 17

Алгоритмы	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
tanh adam	0,823	0,87	0,803	0,768	0,783	0,811	0,78
relu sgd	0,362	0,386	0,378	0,425	0,433	0,421	0,346
identity sgd	0,362	0,386	0,378	0,425	0,433	0,421	0,346
logistic sgd	0,362	0,386	0,378	0,425	0,433	0,421	0,346
tanh sgd	0,362	0,386	0,378	0,425	0,433	0,421	0,346
relu lbfgs	0,787	0,854	0,732	0,713	0,756	0,776	-
identity lbfgs	0,791	0,843	0,783	0,717	0,736	0,78	-
logistic lbfgs	0,362	0,386	0,378	0,425	0,433	0,421	0,425
tanh lbfgs	0,783	0,823	0,764	0,72	0,764	0,803	-

Лучшие результаты (82–87%) показали модели, обученные на матрицах признаков, построенных на токенах из 2 слов и оптимизированные с помощью метода адаптивной оценки моментов и алгоритма LBFGS (кроме логистической функции активации). На других типах токенов данные модели показали точность 70–80%.

Модели обученные методом стохастического градиентного спуска со всеми видами функций активации и модель обученная алгоритмом LBFGS с логистической функцией активации показали низкую эффективность при решении задачи классификации текстов со всеми типами токенов.

ЗАКЛЮЧЕНИЕ

В работе предложен метод классификации научных статей, основанный на совместном использовании модели логистической регрессии, статистической меры TF-IDF и модели мешка слов. Эффективность предложенного метода подтверждена экспериментальными исследованиям на тестовых выборках обучающих примеров, которые не участвовали при обучении моделей.

Исследовано влияние разных типов токенов на точность предсказаний реализованного классификатора и точность предсказаний некоторых популярных алгоритмов машинного обучения с разными значениями гиперпараметров. В ходе анализа результатов сравнения было выявлено, что разные модели показывают разную точность прогнозирования при использовании матриц признаков, рассчитанных на основе разных типов токенов. В некоторых случаях можно улучшить точность моделей при использовании определенного типа токенов путем подбора оптимальных значений гиперпараметров перед началом обучения финальной модели.

Результаты сравнения эффективности популярных алгоритмов машинного обучения при использовании разных типов токенов для решения задачи классификации научных статей можно применить при дальнейших исследованиях в области разработки результирующего классификатора на основе ансамбля классификаторов, использующих метод бустинга и разные типы токенов при расчете матриц признаков, аналогично методу случайных подпространств, но с использованием разных типов токенов вместо использования случайных выборок признаков.

В процессе работы была написана и опубликована статья на тему «Разработка алгоритма поиска аномалий в метеосводках для повышения качества прогнозов» в журнале ВАК «Вестник компьютерных и информационных технологий» [19].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Справочник кодов УДК и ВАК [Электронный ресурс]. – «TeaCode.com» Справочники, автоматически сгенерированные в системе Libretto из объектных моделей спецификаций. – Режим доступа: https://teacode.com/ (дата обращения 24.05.2021).

2 S. Madeh Piryonesi Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems / S. Madeh Piryonesi, Tamer E. El-Diraby // Journal of Transportation Engineering, Part B: Pavements. – 2020. – № 146 (2).

3 Демиденко, Е. З. Линейная и нелинейная регрессия // Финансы и статистика – 1981. – С. 302.

4 Воронцов, К. В. Лекции по методу опорных векторов: лекционный материал . / К. В. Воронцов – 2007.

5 Breiman, L. Random Forests // Machine Learning – 2001. – № 45 (1) – С. 5– 32.

6 Фомин, С. В. Математические проблемы в биологии / С. В. Фомин, Беркинблит М. Б. – 1973.

7 Розенблатт, Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга – Москва: «Мир», 1965. – С. 158–163.

8 Осипова, Ю. А. Применение кластерного анализа методом *k*-средних для классификации текстов научной направленности / Ю. А. Осипова, Д. Н. Лавров // Математические структуры и моделирование. – 2017. – № 3 (43). – С. 108–121.

9 Таскин, А. С. Линейная регрессия с кластеризацией по признаку на данных с действительными величинами / А. С. Таскин, Е. М. Миркес // Вестник Сибирского государственного аэрокосмического университета имени академика М. Ф. Решетнева. – 2012. – № 3 (43). – С. 71–76.

10 Жамалова, Г. Б. Информационное моделирование с применением искусственных нейронных сетей // Academic Research in Educational Sciences. – 2020. – № 1 (3). – С. 730–742.

11 Козлова, Ю. А. Метод идентификации пользователей по рукописному кодовому слову / Ю. А. Козлова, К. А. Майков // Новые информационные технологии в автоматизированных системах. – 2019. – № 22. – С. 19–20.

12 Бабичев, С. А. Оптимизация процесса предобработки информации в системах кластеризации высокоразмерных данных // Радіоелектроніка, інформатика, управління [Радиоэлектроника, информатика, управление]. – 2014. – № 2. – С. 135–142.

13 Харламов, А. А. Нейросетевые подходы к классификации текстов на основе морфологического анализа / А. А. Харламов, Ле Мань Ха // Труды МФТИ. – 2017. – № 9 (2). – С. 143–150.

14 Язык программирования «Python» [Электронный ресурс]. – «Python». – Режим доступа: https://www.python.org/ (дата обращения 24.05.2021).

15 Github репозиторий программного модуля «Natasha» [Электронный ресурс]. – Интернет сайт «GitHub». – Режим доступа: https://github.com/natasha/natasha/ (дата обращения 24.05.2021).

16 Программный модуль «SciPy» [Электронный ресурс]. – Интернет сайт «SciPy.org». – Режим доступа: https://www.scipy.org/ (дата обращения 24.05.2021).

17 Программный модуль «PyTorch» [Электронный ресурс]. – Интернет сайт «PyTorch». – Режим доступа: https://pytorch.org/ (дата обращения 24.05.2021).

18 Программный модуль «NumPy» [Электронный ресурс]. – Интернет сайт «NumPy». – Режим доступа: https://numpy.org/ (дата обращения 24.05.2021).

19 Ященков, К. Г., Дымко, К. С., Уханов, Н. О., Хныкин, А. В. Разработка алгоритма поиска аномалий в метеосводках для повышения качества прогнозов // Вестник компьютерных и информационных технологий. – 2021. – № 5 (203). – С. 45–52.

ПРИЛОЖЕНИЕ А

Результаты экспериментов с другими популярными алгоритмами машинного обучения

Таблица А.1 – Полные результаты обучения моделей k-ближайших соседей с настройками euclidean uniform

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 euclidean uniform)	0,402	0,362	0,642	0,524	0,417	0,118	0,268
KNeighbours (k=10 euclidean uniform)	0,228	0,126	0,709	0,394	0,283	0,146	0,197
KNeighbours (k=15 euclidean uniform)	0,213	0,154	0,563	0,469	0,425	0,165	0,228
KNeighbours (k=20 euclidean uniform)	0,126	0,016	0,63	0,472	0,213	0,122	0,118
KNeighbours (k=25 euclidean uniform)	0,134	0,063	0,512	0,453	0,268	0,13	0,146
KNeighbours (k=30 euclidean uniform)	0,106	0,075	0,543	0,445	0,272	0,146	0,185
KNeighbours (k=35 euclidean uniform)	0,142	0,067	0,406	0,402	0,118	0,197	0,134
KNeighbours (k=40 euclidean uniform)	0,161	0,063	0,689	0,48	0,232	0,217	0,154
KNeighbours (k=45 euclidean uniform)	0,142	0,055	0,661	0,457	0,106	0,165	0,118
KNeighbours (k=50 euclidean uniform)	0,098	0,142	0,579	0,421	0,236	0,256	0,114
KNeighbours (k=60 euclidean uniform)	0,146	0,142	0,516	0,386	0,189	0,146	0,087
KNeighbours (k=70 euclidean uniform)	0,13	0,102	0,567	0,421	0,201	0,142	0,087
KNeighbours (k=80 euclidean uniform)	0,079	0,094	0,382	0,39	0,307	0,126	0,118
KNeighbours (k=90 euclidean uniform)	0,098	0,102	0,409	0,398	0,094	0,197	0,094
KNeighbours (k=100 euclidean uniform)	0,126	0,122	0,449	0,425	0,185	0,193	0,106

Продолжение приложения А

Таблица А.2 – Полные результаты обучения моделей k-ближайших соседей с настройками euclidean distance

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 euclidean distance)	0,469	0,15	0,579	0,425	0,244	0,217	0,421
KNeighbours (k=10 euclidean distance)	0,331	0,264	0,543	0,528	0,441	0,102	0,303
KNeighbours (k=15 euclidean distance)	0,189	0,059	0,732	0,476	0,28	0,087	0,138
KNeighbours (k=20 euclidean distance)	0,071	0,087	0,732	0,417	0,252	0,118	0,228
KNeighbours (k=25 euclidean distance)	0,173	0,106	0,748	0,465	0,26	0,142	0,118
KNeighbours (k=30 euclidean distance)	0,181	0,059	0,303	0,472	0,24	0,157	0,157
KNeighbours (k=35 euclidean distance)	0,146	0,047	0,717	0,339	0,169	0,26	0,146
KNeighbours (k=40 euclidean distance)	0,157	0,173	0,657	0,429	0,224	0,11	0,098
KNeighbours (k=45 euclidean distance)	0,224	0,079	0,634	0,437	0,169	0,126	0,13
KNeighbours (k=50 euclidean distance)	0,197	0,106	0,484	0,421	0,217	0,134	0,157
KNeighbours (k=60 euclidean distance)	0,138	0,051	0,598	0,382	0,177	0,181	0,13
KNeighbours (k=70 euclidean distance)	0,106	0,142	0,512	0,37	0,244	0,197	0,094
KNeighbours (k=80 euclidean distance)	0,138	0,067	0,472	0,386	0,173	0,193	0,13
KNeighbours (k=90 euclidean distance)	0,142	0,094	0,465	0,425	0,15	0,236	0,134
KNeighbours (k=100 euclidean distance)	0,098	0,098	0,409	0,406	0,193	0,154	0,098

Продолжение приложения А

Таблица А.3 – Полные результаты обучения моделей k-ближайших соседей с настройками manhattan uniform

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 manhattan uniform)	0,354	0,425	0,409	0,524	0,15	0,157	0,209
KNeighbours (k=10 manhattan uniform)	0,291	0,421	0,417	0,433	0,26	0,213	0,169
KNeighbours (k=15 manhattan uniform)	0,555	0,449	0,402	0,437	0,311	0,193	0,146
KNeighbours (k=20 manhattan uniform)	0,5	0,398	0,343	0,441	0,331	0,181	0,083
KNeighbours (k=25 manhattan uniform)	0,528	0,441	0,366	0,429	0,22	0,169	0,142
KNeighbours (k=30 manhattan uniform)	0,472	0,449	0,374	0,386	0,291	0,138	0,122
KNeighbours (k=35 manhattan uniform)	0,394	0,417	0,417	0,39	0,24	0,165	0,087
KNeighbours (k=40 manhattan uniform)	0,339	0,37	0,402	0,402	0,335	0,134	0,098
KNeighbours (k=45 manhattan uniform)	0,484	0,378	0,339	0,374	0,484	0,205	0,118
KNeighbours (k=50 manhattan uniform)	0,311	0,48	0,413	0,35	0,504	0,197	0,102
KNeighbours (k=60 manhattan uniform)	0,268	0,425	0,394	0,386	0,465	0,138	0,134
KNeighbours (k=70 manhattan uniform)	0,335	0,409	0,386	0,343	0,528	0,15	0,11
KNeighbours (k=80 manhattan uniform)	0,193	0,433	0,335	0,39	0,508	0,094	0,106
KNeighbours (k=90 manhattan uniform)	0,157	0,437	0,394	0,378	0,594	0,146	0,098
KNeighbours (k=100 manhattan uniform)	0,189	0,461	0,37	0,346	0,547	0,213	0,134

Продолжение приложения А

Таблица А.4 – Полные результаты обучения моделей k-ближайших соседей с настройками manhattan distance

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 manhattan distance)	0,386	0,437	0,394	0,508	0,272	0,165	0,157
KNeighbours (k=10 manhattan distance)	0,469	0,437	0,374	0,382	0,299	0,157	0,138
KNeighbours (k=15 manhattan distance)	0,567	0,374	0,465	0,413	0,311	0,106	0,185
KNeighbours (k=20 manhattan distance)	0,571	0,374	0,417	0,48	0,386	0,134	0,169
KNeighbours (k=25 manhattan distance)	0,5	0,433	0,398	0,402	0,228	0,15	0,087
KNeighbours (k=30 manhattan distance)	0,268	0,402	0,35	0,366	0,394	0,161	0,102
KNeighbours (k=35 manhattan distance)	0,543	0,504	0,406	0,421	0,429	0,217	0,142
KNeighbours (k=40 manhattan distance)	0,512	0,386	0,398	0,339	0,382	0,122	0,13
KNeighbours (k=45 manhattan distance)	0,441	0,453	0,374	0,402	0,48	0,138	0,114
KNeighbours (k=50 manhattan distance)	0,177	0,421	0,386	0,413	0,425	0,272	0,114
KNeighbours (k=60 manhattan distance)	0,374	0,398	0,354	0,409	0,547	0,213	0,102
KNeighbours (k=70 manhattan distance)	0,287	0,39	0,362	0,409	0,508	0,417	0,11
KNeighbours (k=80 manhattan distance)	0,181	0,472	0,406	0,366	0,52	0,307	0,094
KNeighbours (k=90 manhattan distance)	0,224	0,394	0,39	0,39	0,528	0,224	0,102
KNeighbours (k=100 manhattan distance)	0,138	0,421	0,35	0,37	0,5	0,209	0,122

Продолжение приложения А

Таблица А.5 – Полные результаты обучения моделей k-ближайших соседей с настройками minkowski uniform

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 minkowski uniform)	0,339	0,209	0,728	0,508	0,213	0,126	0,307
KNeighbours (k=10 minkowski uniform)	0,193	0,169	0,61	0,52	0,177	0,102	0,087
KNeighbours (k=15 minkowski uniform)	0,106	0,244	0,618	0,535	0,205	0,173	0,169
KNeighbours (k=20 minkowski uniform)	0,173	0,079	0,701	0,409	0,087	0,177	0,122
KNeighbours (k=25 minkowski uniform)	0,142	0,087	0,689	0,488	0,177	0,154	0,154
KNeighbours (k=30 minkowski uniform)	0,201	0,079	0,587	0,354	0,138	0,154	0,142
KNeighbours (k=35 minkowski uniform)	0,189	0,094	0,598	0,461	0,181	0,173	0,173
KNeighbours (k=40 minkowski uniform)	0,15	0,114	0,799	0,449	0,217	0,134	0,13
KNeighbours (k=45 minkowski uniform)	0,142	0,075	0,724	0,386	0,283	0,181	0,106
KNeighbours (k=50 minkowski uniform)	0,173	0,114	0,398	0,425	0,244	0,205	0,173
KNeighbours (k=60 minkowski uniform)	0,177	0,13	0,528	0,472	0,181	0,106	0,138
KNeighbours (k=70 minkowski uniform)	0,102	0,13	0,461	0,354	0,209	0,146	0,13
KNeighbours (k=80 minkowski uniform)	0,106	0,138	0,441	0,378	0,189	0,161	0,122
KNeighbours (k=90 minkowski uniform)	0,138	0,154	0,465	0,406	0,083	0,138	0,11
KNeighbours (k=100 minkowski uniform)	0,087	0,118	0,382	0,453	0,15	0,169	0,091

Продолжение приложения А

Таблица А.6 – Полные результаты обучения моделей k-ближайших соседей с настройками minkowski distance

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
KNeighbours (k=5 minkowski distance)	0,394	0,142	0,535	0,461	0,445	0,165	0,346
KNeighbours (k=10 minkowski distance)	0,189	0,154	0,72	0,425	0,217	0,098	0,26
KNeighbours (k=15 minkowski distance)	0,209	0,24	0,65	0,476	0,213	0,185	0,098
KNeighbours (k=20 minkowski distance)	0,228	0,087	0,535	0,421	0,126	0,11	0,181
KNeighbours (k=25 minkowski distance)	0,138	0,059	0,736	0,437	0,11	0,138	0,154
KNeighbours (k=30 minkowski distance)	0,173	0,067	0,626	0,433	0,122	0,193	0,094
KNeighbours (k=35 minkowski distance)	0,24	0,106	0,531	0,457	0,213	0,181	0,157
KNeighbours (k=40 minkowski distance)	0,197	0,079	0,709	0,48	0,169	0,138	0,161
KNeighbours (k=45 minkowski distance)	0,205	0,083	0,531	0,476	0,201	0,094	0,161
KNeighbours (k=50 minkowski distance)	0,209	0,079	0,583	0,465	0,197	0,118	0,13
KNeighbours (k=60 minkowski distance)	0,075	0,071	0,547	0,402	0,197	0,138	0,106
KNeighbours (k=70 minkowski distance)	0,094	0,098	0,374	0,417	0,224	0,236	0,15
KNeighbours (k=80 minkowski distance)	0,185	0,165	0,413	0,445	0,157	0,209	0,106
KNeighbours (k=90 minkowski distance)	0,185	0,15	0,453	0,346	0,157	0,11	0,114
KNeighbours (k=100 minkowski distance)	0,142	0,094	0,437	0,441	0,165	0,169	0,106

Продолжение приложения А

Таблица А.7 – Полные результаты обучения моделей деревьев решений с настройкой gini

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Decision Tree (max depth=5 gini best)	0,913	0,563	0,622	0,882	0,85	0,972	0,992
Decision Tree (max depth=10 gini best)	0,945	0,984	0,992	0,992	0,854	0,953	0,953
Decision Tree (max depth=15 gini best)	0,902	0,996	0,988	0,996	0,819	0,976	0,972
Decision Tree (max depth=20 gini best)	0,921	0,98	0,98	0,992	0,874	0,98	0,957
Decision Tree (max depth=25 gini best)	0,949	0,992	0,984	0,996	0,862	0,969	0,988
Decision Tree (max depth=30 gini best)	0,917	1	0,992	0,992	0,886	0,969	0,972
Decision Tree (max depth=35 gini best)	0,929	0,984	0,972	0,988	0,87	0,953	0,965
Decision Tree (max depth=40 gini best)	0,933	0,988	0,996	1	0,839	0,984	0,972
Decision Tree (max depth=45 gini best)	0,929	0,992	0,984	0,984	0,902	0,957	0,976
Decision Tree (max depth=50 gini best)	0,913	0,996	0,996	1	0,89	0,972	0,941
Decision Tree (max depth=5 gini random)	0,693	0,378	0,555	0,906	0,598	0,909	0,976
Decision Tree (max depth=10 gini random)	0,787	0,976	0,996	0,984	0,705	0,945	0,929
Decision Tree (max depth=15 gini random)	0,902	0,98	1	0,996	0,795	0,906	0,969
Decision Tree (max depth=20 gini random)	0,902	0,972	0,996	0,992	0,736	0,882	0,976
Decision Tree (max depth=25 gini random)	0,882	0,988	0,992	0,98	0,78	0,957	0,98
Decision Tree (max depth=30 gini random)	0,89	0,996	0,988	0,984	0,807	0,961	0,961
Decision Tree (max depth=35 gini random)	0,874	0,965	0,925	0,992	0,827	0,945	0,98
Decision Tree (max depth=40 gini random)	0,854	0,988	0,988	0,992	0,815	0,886	0,965
Decision Tree (max depth=45 gini random)	0,87	0,984	0,988	0,988	0,717	0,917	0,988
Decision Tree (max depth=50 gini random)	0,921	0,972	0,913	0,988	0,831	0,961	0,957

Продолжение приложения А

Таблица А.8 – Полные результаты обучения моделей деревьев решений с настройкой entropy

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Decision Tree (max depth=5 entropy best)	0,752	0,992	0,992	1	0,913	0,961	0,988
Decision Tree (max depth=10 entropy best)	0,933	0,988	0,992	1	0,898	0,972	0,941
Decision Tree (max depth=15 entropy best)	0,945	0,98	0,992	0,996	0,906	0,98	0,972
Decision Tree (max depth=20 entropy best)	0,953	0,996	1	1	0,909	0,965	0,953
Decision Tree (max depth=25 entropy best)	0,913	0,992	0,988	0,996	0,925	0,969	0,98
Decision Tree (max depth=30 entropy best)	0,937	0,992	0,984	0,996	0,929	0,961	0,969
Decision Tree (max depth=35 entropy best)	0,925	0,972	0,996	0,992	0,909	0,988	0,969
Decision Tree (max depth=40 entropy best)	0,941	0,996	0,98	0,996	0,925	0,957	0,972
Decision Tree (max depth=45 entropy best)	0,913	1	0,992	0,988	0,906	0,976	0,988
Decision Tree (max depth=50 entropy best)	0,933	0,996	0,996	0,984	0,925	0,965	0,921
Decision Tree (max depth=5 entropy random)	0,819	0,709	0,748	0,85	0,539	0,909	0,976
Decision Tree (max depth=10 entropy random)	0,87	0,984	0,988	0,992	0,783	0,929	0,961
Decision Tree (max depth=15 entropy random)	0,89	0,98	0,98	0,988	0,776	0,917	0,976
Decision Tree (max depth=20 entropy random)	0,909	0,996	0,976	0,992	0,732	0,913	0,902
Decision Tree (max depth=25 entropy random)	0,945	0,965	0,988	0,98	0,803	0,976	0,988
Decision Tree (max depth=30 entropy random)	0,89	0,988	0,98	0,976	0,811	0,933	0,874
Decision Tree (max depth=35 entropy random)	0,909	0,988	0,984	0,996	0,803	0,929	0,984
Decision Tree (max depth=40 entropy random)	0,913	0,996	0,988	1	0,843	0,941	0,972
Decision Tree (max depth=45 entropy random)	0,921	0,972	0,992	0,992	0,791	0,921	0,976
Decision Tree (max depth=50 entropy random)	0,878	0,98	0,984	0,984	0,764	0,953	0,906

Продолжение приложения А

Таблица А.9 – Полные результаты обучения моделей Random Forest с настройками gini sqrt

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=5 max_depth=5 max_features=sqrt)	0,6	0,73	0,2	0,79	0,59	0,67	0,65
Random Forest (gini n_trees=5 max_depth=10 max_features=sqrt)	0,75	0,83	0,38	0,95	0,56	0,72	0,75
Random Forest (gini n_trees=5 max_depth=15 max_features=sqrt)	0,74	0,78	0,37	0,97	0,59	0,76	0,79
Random Forest (gini n_trees=5 max_depth=20 max_features=sqrt)	0,75	0,86	0,45	0,98	0,62	0,72	0,73
Random Forest (gini n_trees=10 max_depth=5 max_features=sqrt)	0,72	0,68	0,24	0,82	0,68	0,8	0,76
Random Forest (gini n_trees=10 max_depth=10 max_features=sqrt)	0,83	0,86	0,61	0,99	0,74	0,85	0,84
Random Forest (gini n_trees=10 max_depth=15 max_features=sqrt)	0,8	0,79	0,68	0,97	0,7	0,78	0,85
Random Forest (gini n_trees=10 max_depth=20 max_features=sqrt)	0,76	0,89	0,66	0,97	0,74	0,8	0,85
Random Forest (gini n_trees=15 max_depth=5 max_features=sqrt)	0,79	0,85	0,23	0,97	0,76	0,91	0,85
Random Forest (gini n_trees=15 max_depth=10 max_features=sqrt)	0,83	0,9	0,54	0,98	0,77	0,88	0,96
Random Forest (gini n_trees=15 max_depth=15 max_features=sqrt)	0,81	0,95	0,55	0,99	0,74	0,85	0,89
Random Forest (gini n_trees=15 max_depth=20 max_features=sqrt)	0,83	0,92	0,71	0,99	0,74	0,88	0,93
Random Forest (gini n_trees=20 max_depth=5 max_features=sqrt)	0,79	0,77	0,51	0,95	0,87	0,89	0,91

Продолжение приложения А

Окончание таблицы А.9

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=20 max_depth=10 max_features=sqrt)	0,85	0,91	0,69	0,99	0,81	0,89	0,88
Random Forest (gini n_trees=20 max_depth=15 max_features=sqrt)	0,85	0,94	0,68	0,99	0,67	0,91	0,91
Random Forest (gini n_trees=20 max_depth=20 max_features=sqrt)	0,86	0,96	0,63	0,99	0,77	0,9	0,94

Таблица А.10 – Полные результаты обучения моделей Random Forest с настройками gini log2

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=5 max_depth=5 max_features=log2)	0,4	0,28	0,79	0,27	0,39	0,36	0,32
Random Forest (gini n_trees=5 max_depth=10 max_features=log2)	0,44	0,4	0,86	0,67	0,42	0,36	0,4
Random Forest (gini n_trees=5 max_depth=15 max_features=log2)	0,56	0,47	0,85	0,59	0,48	0,43	0,49
Random Forest (gini n_trees=5 max_depth=20 max_features=log2)	0,61	0,41	0,78	0,62	0,44	0,46	0,43
Random Forest (gini n_trees=10 max_depth=5 max_features=log2)	0,56	0,49	0,88	0,49	0,5	0,48	0,44
Random Forest (gini n_trees=10 max_depth=10 max_features=log2)	0,62	0,48	0,9	0,67	0,5	0,55	0,45
Random Forest (gini n_trees=10 max_depth=15 max_features=log2)	0,59	0,49	0,96	0,72	0,51	0,54	0,5
Random Forest (gini n_trees=10 max_depth=20 max_features=log2)	0,63	0,55	0,76	0,84	0,45	0,52	0,62

Продолжение приложения А

Окончание таблицы А.10

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (gini n_trees=15 max_depth=5 max_features=log2)	0,55	0,37	0,93	0,64	0,55	0,53	0,41
Random Forest (gini n_trees=15 max_depth=10 max_features=log2)	0,64	0,58	0,96	0,79	0,55	0,62	0,59
Random Forest (gini n_trees=15 max_depth=15 max_features=log2)	0,71	0,55	0,98	0,85	0,53	0,6	0,63
Random Forest (gini n_trees=15 max_depth=20 max_features=log2)	0,72	0,61	0,85	0,9	0,54	0,56	0,64
Random Forest (gini n_trees=20 max_depth=5 max_features=log2)	0,62	0,47	0,99	0,68	0,57	0,54	0,66
Random Forest (gini n_trees=20 max_depth=10 max_features=log2)	0,7	0,62	0,97	0,64	0,64	0,6	0,54
Random Forest (gini n_trees=20 max_depth=15 max_features=log2)	0,76	0,65	0,99	0,85	0,56	0,6	0,67
Random Forest (gini n_trees=20 max_depth=20 max_features=log2)	0,7	0,62	0,64	0,9	0,52	0,58	0,56

Таблица А.11 – Полные результаты обучения моделей Random Forest с настройками entropy sqrt

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=5 max_depth=5 max_features=sqrt)	0,64	0,45	0,22	0,75	0,6	0,66	0,69
Random Forest (entropy n_trees=5 max_depth=10 max_features=sqrt)	0,69	0,75	0,48	0,93	0,7	0,75	0,74
Random Forest (entropy n_trees=5 max_depth=15 max_features=sqrt)	0,67	0,78	0,4	0,96	0,58	0,75	0,71

Продолжение приложения А

Окончание таблицы А.11

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=5 max_depth=20 max_features=sqrt)	0,7	0,78	0,58	0,96	0,62	0,68	0,76
Random Forest (entropy n_trees=10 max_depth=5 max_features=sqrt)	0,76	0,76	0,32	0,95	0,79	0,85	0,76
Random Forest (entropy n_trees=10 max_depth=10 max_features=sqrt)	0,8	0,81	0,49	0,97	0,7	0,85	0,91
Random Forest (entropy n_trees=10 max_depth=15 max_features=sqrt)	0,83	0,83	0,5	0,99	0,77	0,83	0,87
Random Forest (entropy n_trees=10 max_depth=20 max_features=sqrt)	0,8	0,9	0,73	0,99	0,71	0,84	0,85
Random Forest (entropy n_trees=15 max_depth=5 max_features=sqrt)	0,72	0,86	0,52	0,98	0,69	0,83	0,85
Random Forest (entropy n_trees=15 max_depth=10 max_features=sqrt)	0,78	0,85	0,57	0,99	0,79	0,89	0,89
Random Forest (entropy n_trees=15 max_depth=15 max_features=sqrt)	0,86	0,89	0,7	0,99	0,75	0,85	0,93
Random Forest (entropy n_trees=15 max_depth=20 max_features=sqrt)	0,79	0,92	0,62	0,99	0,75	0,8	0,82
Random Forest (entropy n_trees=20 max_depth=5 max_features=sqrt)	0,81	0,85	0,49	0,98	0,81	0,91	0,82
Random Forest (entropy n_trees=20 max_depth=10 max_features=sqrt)	0,92	0,88	0,67	0,99	0,8	0,89	0,93
Random Forest (entropy n_trees=20 max_depth=15 max_features=sqrt)	0,83	0,9	0,58	0,99	0,78	0,9	0,88
Random Forest (entropy n_trees=20 max_depth=20 max_features=sqrt)	0,85	0,94	0,7	0,99	0,77	0,87	0,85

Продолжение приложения А

Таблица А.12 – Полные результаты обучения моделей Random Forest с настройками entropy log2

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=5 max_depth=5 max_features=log2)	0,42	0,32	0,77	0,17	0,37	0,39	0,26
Random Forest (entropy n_trees=5 max_depth=10 max_features=log2)	0,57	0,58	0,79	0,61	0,42	0,43	0,37
Random Forest (entropy n_trees=5 max_depth=15 max_features=log2)	0,59	0,39	0,73	0,48	0,43	0,43	0,47
Random Forest (entropy n_trees=5 max_depth=20 max_features=log2)	0,52	0,47	0,73	0,52	0,42	0,43	0,45
Random Forest (entropy n_trees=10 max_depth=5 max_features=log2)	0,57	0,35	0,86	0,54	0,48	0,4	0,45
Random Forest (entropy n_trees=10 max_depth=10 max_features=log2)	0,66	0,5	0,97	0,82	0,46	0,55	0,56
Random Forest (entropy n_trees=10 max_depth=15 max_features=log2)	0,58	0,47	0,95	0,68	0,52	0,53	0,55
Random Forest (entropy n_trees=10 max_depth=20 max_features=log2)	0,61	0,58	0,77	0,81	0,47	0,52	0,54
Random Forest (entropy n_trees=15 max_depth=5 max_features=log2)	0,64	0,36	0,94	0,56	0,57	0,49	0,6
Random Forest (entropy n_trees=15 max_depth=10 max_features=log2)	0,62	0,48	0,97	0,62	0,58	0,61	0,49
Random Forest (entropy n_trees=15 max_depth=15 max_features=log2)	0,62	0,55	0,98	0,71	0,52	0,54	0,57
Random Forest (entropy n_trees=15 max_depth=20 max_features=log2)	0,69	0,58	0,95	0,73	0,5	0,52	0,62
Random Forest (entropy n_trees=20 max_depth=5 max_features=log2)	0,58	0,47	0,97	0,62	0,61	0,55	0,58

Конец приложения А

Окончание таблицы А.12

Алгоритм	1 лемма	2 леммы	3 леммы	4 леммы	3 символа	4 символа	5 символов
Random Forest (entropy n_trees=20 max_depth=10 max_features=log2)	0,68	0,58	0,99	0,8	0,59	0,62	0,57
Random Forest (entropy n_trees=20 max_depth=15 max_features=log2)	0,69	0,65	0,98	0,91	0,56	0,6	0,65
Random Forest (entropy n_trees=20 max_depth=20 max_features=log2)	0,65	0,61	0,7	0,91	0,53	0,54	0,62

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Базовая кафедра «Интеллектуальные системы управления»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Ю.Ю. Якунин
подпись
«21» 06 2021 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Применение метода анализа текстовой информации для задачи классификации научных статей

09.04.04 Программная инженерия

09.04.04.02 Технология индустриального производства программного обеспечения интеллектуальных систем управления

Научный руководитель _____ 21.06.21 доцент, канд. техн. наук А.В. Хныкин
подпись, дата

Выпускник _____ 21.06.21 Н.О. Уханов
подпись, дата

Рецензент _____ 21.06.21 доцент, канд. техн. наук А.С. Амузаде
подпись, дата

Нормоконтролер _____ 21.06.21 Н.Б. Позолотина
подпись, дата

Красноярск 2021