

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФИЛИАЛ ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»
В Г. СМОЛЕНСКЕ**

Кафедра «Вычислительная техника»

Направление **09.04.01 «Информатика и вычислительная техника»**
магистерская программа «Информационное и программное обеспечение
автоматизированных систем»

КУРСОВАЯ РАБОТА
по курсу «Интеллектуальный анализ данных и знаний»

студента 1 курса группы ВМ-22(маг.) _____ Старостенкова А.А.
(подпись) (фамилия, инициалы)

на тему: «Реализация алгоритма градиентного бустинга деревьев
решений Фридмана»

Преподаватель:

доцент Зернов М.М.
(должность) (подпись) (расшифровка подписи)

Защита проекта состоялась «__» _____ 20__ г.

Оценка за проект _____
(неудовлетворительно, удовлетворительно, хорошо, отлично)

Смоленск 2023

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студента

Старостенкова А.А.
(фамилия, инициалы)

Тема работы: Реализация алгоритма градиентного бустинга деревьев решений Фридмана

Содержание задания

В соответствии с выбранной темой необходимо выполнить следующие этапы.

1. Дать характеристику кругу задач, решаемого с помощью градиентного бустинга деревьев решений.
2. Описать способы реализации алгоритма градиентного бустинга деревьев решений.
3. Охарактеризовать разновидности и усовершенствования базовых методов и моделей алгоритма градиентного бустинга деревьев решений.
4. Сформировать тестовый пример и с помощью него, сделать оценки реализуемого алгоритма.
5. Реализовать выбранный вариант рассматриваемого алгоритма.
6. Тестирование алгоритма и сравнение его с другими реализациями.

Студент:

(подпись)

Старостенков А.А.
(инициалы, фамилия)

Руководитель проекта:

(подпись)

доцент Зернов М.М.
(инициалы, фамилия)

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ	2
1. ХАРАКТЕРИСТИКА КРУГА ЗАДАЧ, РЕШАЕМОГО С ПОМОЩЬЮ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА	4
2. ВАРИАНТЫ РЕАЛИЗАЦИИ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА.....	7
3. ВАРИАНТЫ УСОВЕРШЕНСТВОВАНИЙ БАЗОВЫХ АЛГОРИТМОВ	14
4. РЕАЛИЗАЦИЯ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА.....	16
5. ОЦЕНКА АЛГОРИТМА	19
ЗАКЛЮЧЕНИЕ	21
СПИСОК ЛИТЕРАТУРЫ.....	22
ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ.....	23

1. ХАРАКТЕРИСТИКА КРУГА ЗАДАЧ, РЕШАЕМОГО С ПОМОЩЬЮ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА

Алгоритм градиентного бустинга деревьев решений (Gradient Boosting on Decision Trees, GBDT), разработанный Фридманом, представляет собой мощный метод машинного обучения, который используется для решения разнообразных задач.

В ходе обучения случайного леса каждый базовый алгоритм строится независимо от остальных. Бустинг, в свою очередь, воплощает идею последовательного построения линейной комбинации алгоритмов. Каждый следующий алгоритм старается уменьшить ошибку текущего ансамбля.

Бустинг, использующий деревья решений в качестве базовых алгоритмов, называется градиентным бустингом над решающими деревьями. Он отлично работает на выборках с «табличными», неоднородными данными. Примером таких данных может служить описание пользователя Яндекса через его возраст, пол, среднее число поисковых запросов в день, число заказов такси и так далее. Такой бустинг способен эффективно находить нелинейные зависимости в данных различной природы. Этим свойством обладают все алгоритмы, использующие деревья решений, однако именно GBDT обычно выигрывает в подавляющем большинстве задач. Благодаря этому он широко применяется во многих конкурсах по машинному обучению и задачах из индустрии (поисковом ранжировании, рекомендательных системах, таргетировании рекламы, предсказании погоды, пункта назначения такси и многих других).

Не так хорошо бустинг проявляет себя на однородных данных: текстах, изображениях, звуке, видео. В таких задачах нейросетевые подходы почти всегда демонстрируют лучшее качество [7].

Основные области применения алгоритма:

1. Классификация и регрессия. Градиентный бустинг деревьев решений может быть применен как для задач классификации, так и для задач регрессии. В классификации алгоритм помогает разделять объекты на различные классы на основе входных признаков, в то время как в регрессии он используется для предсказания числовых значений.

2. Ранжирование. Градиентный бустинг также может применяться для задач ранжирования, например, в поисковых системах, где необходимо определить порядок отображения результатов поиска.

3. Детекция аномалий. Алгоритм может быть использован для выявления аномалий в данных, таких как мошеннические транзакции в банковском секторе или нештатные события в производственных процессах.

4. Работа с текстом и изображениями. Градиентный бустинг может применяться в задачах обработки естественного языка, анализа тональности текста, классификации изображений и даже в задачах, связанных с генетическими данными.

5. Соревнования по анализу данных. Алгоритм градиентного бустинга деревьев решений широко применяется в соревнованиях по анализу данных на платформах, таких как Kaggle, и демонстрирует высокую эффективность.

6. Работа с большими данными. Градиентный бустинг способен обрабатывать большие объемы данных и автоматически выбирать наиболее информативные признаки для улучшения качества прогнозов.

7. Мета-обучение и стекинг. Градиентный бустинг может использоваться как компонент в мета-обучении и стекинге, что позволяет улучшить качество прогнозов за счет комбинирования разных моделей.

8. Прогнозирование временных рядов. Алгоритм может быть применен для задач прогнозирования временных рядов, таких как продажи, финансовые показатели и т. д.

Основными преимуществами градиентного бустинга деревьев решений являются высокая точность прогнозов, способность работать с разнородными

данными и автоматический отбор признаков. Однако он также требует тщательной настройки гиперпараметров и может быть склонен к переобучению на малых выборках данных

2. ВАРИАНТЫ РЕАЛИЗАЦИИ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА

Существуют различные варианты реализации алгоритма градиентного бустинга деревьев решений.

1. Случайный лес (Random Forest).

Случайный лес – это ансамбль машинного обучения, основанный на деревьях решений. Он создает множество решающих деревьев во время обучения и комбинирует их прогнозы для получения более устойчивых и точных результатов.

Основные характеристики случайного леса.

- Бутстрэп выборка. для каждого дерева создается подвыборка из обучающего набора данных с повторением (бутстрэп выборка). Это позволяет разнообразить обучающие данные для каждого дерева.
- Случайные подпространства признаков. при построении каждого узла дерева выбирается случайное подмножество признаков для разделения данных. Это способствует снижению корреляции между деревьями и улучшению обобщающей способности модели.
- Голосование большинства (или усреднение). Прогнозы каждого дерева объединяются путем голосования большинства (для классификации) или усреднения (для регрессии) для получения итогового результата.

Случайный лес обладает высокой устойчивостью к переобучению, хорошей способностью обобщения и высокой производительностью. Этот алгоритм часто используется в задачах классификации, регрессии и выбора наиболее важных признаков.

2. AdaBoost (Adaptive Boosting)

AdaBoost – это алгоритм адаптивного бустинга, который использует взвешивание обучающих примеров, чтобы сконцентрироваться на тех, которые трудно классифицировать. Он создает слабые классификаторы (часто деревья решений) и комбинирует их для получения сильного классификатора.

Основные характеристики AdaBoost.

- Взвешивание обучающих примеров. Примеры, которые были неправильно классифицированы предыдущими слабыми классификаторами, получают больший вес на следующей итерации. Это позволяет алгоритму сфокусироваться на трудно классифицируемых примерах.
- Комбинация слабых классификаторов. AdaBoost создает ансамбль из слабых классификаторов и взвешивает их прогнозы в зависимости от их точности.
- Итеративность. Алгоритм работает итеративно, добавляя новые слабые классификаторы на каждой итерации и обновляя веса обучающих примеров.

AdaBoost также имеет хорошую способность обобщения и хорошо работает на разнообразных задачах классификации. Он может быть уязвим к выбросам в данных, поэтому важно проводить предварительную обработку данных.

3. XGBoost

XGBoost (Extreme Gradient Boosting) — это оптимизированная библиотека для градиентного бустинга, которая предоставляет высокую производительность и эффективность. Основными особенностями XGBoost являются использование регуляризации для предотвращения переобучения, поддержка распределенных вычислений и возможность работы с различными типами данных. XGBoost доступен для разных языков программирования, включая Python, R, Java и другие.

4. LightGBM

LightGBM — это еще одна библиотека для градиентного бустинга, разработанная Microsoft. Она известна своей высокой скоростью работы и эффективностью. LightGBM использует алгоритм градиентного спуска и оптимизацию гистограмм для построения деревьев, что делает его быстрее по

сравнению с некоторыми другими библиотеками. Он также поддерживает категориальные признаки и работу с большими данными.

5. CatBoost

CatBoost (Categorical Boosting) — это библиотека, разработанная Яндексом, специально оптимизированная для работы с категориальными признаками. Она автоматически обрабатывает категориальные данные, не требуя их предварительного кодирования, что делает ее очень удобной для задач, где категориальные признаки важны. CatBoost также обладает встроенной поддержкой распределенных вычислений.

6. Градиентный бустинг с решающими деревьями

Градиентный бустинг с решающими деревьями (Gradient Boosting with Decision Trees) представляет собой основной вариант градиентного бустинга. В этом методе каждое дерево обучается с учетом остатков (градиента) предыдущего дерева. Это позволяет модели постепенно улучшать свои прогнозы, минимизируя ошибку. Наиболее популярной библиотекой для реализации этого варианта является Scikit-Learn (Python).

Градиентный бустинг с решающими деревьями представляет собой мощный алгоритм машинного обучения, который сочетает в себе два ключевых компонента: градиентный бустинг и решающие деревья. Этот метод широко применяется в задачах классификации и регрессии, благодаря своей способности создавать сильные ансамбли моделей.

Основные характеристики Градиентного бустинга с решающими деревьями.

1. Итеративное обучение. Алгоритм работает итеративно и последовательно создает решающие деревья. На каждой итерации строится новое дерево, и оно «учится» исправлять ошибки предыдущих деревьев.

2. Градиентный спуск. Основной идеей является использование градиентного спуска для нахождения направления наибольшего убывания функции потерь. Градиент вычисляется на основе остатков между текущими прогнозами и истинными значениями целевой переменной.

3. Слабые ученики. Каждое решающее дерево, создаваемое в процессе обучения, обычно является слабым учеником, то есть деревом с ограниченной глубиной и низкой мощностью. Это снижает риск переобучения.

4. Ансамбль деревьев. Прогнозы всех созданных деревьев комбинируются в конечный прогноз. Это делается путем суммирования или усреднения результатов всех деревьев, что позволяет улучшить обобщающую способность модели.

5. Регуляризация. Для предотвращения переобучения, Градиентный бустинг может использовать регуляризацию, такую как ограничение глубины деревьев или введение коэффициентов для управления вкладом каждого дерева в итоговый прогноз.

6. Подбор параметров. Оптимальные параметры, такие как скорость обучения (learning rate), количество деревьев и их глубина, часто подбираются с использованием кросс-валидации.

В данной курсовой работе будет представлена реализация градиентного бустинга для задачи распознавания пола по акустическим свойствам голоса. Основные особенности реализации градиентного бустинга.

1. Выбор базовой модели. в данной реализации в качестве базовой модели используются решающие деревья (Decision Trees), которые способны аппроксимировать нелинейные зависимости между акустическими признаками и полом говорящего.

2. Инициализация списка для базовых моделей. мы начинаем с инициализации пустого списка для хранения базовых моделей, которые будут создаваться на каждой итерации.

3. Инициализация списка для весов моделей. также инициализируется пустой список для хранения весов базовых моделей, которые будут определять их вклад в итоговый прогноз.

4. Инициализация предсказаний. создаются пустые массивы для хранения предсказаний на обучающей и тестовой выборках.

5. Цикл по числу деревьев (`n_estimators`). далее следует цикл, в котором будут создаваться и обучаться базовые модели (решающие деревья). В каждой итерации.

- Создается и обучается базовая модель (`DecisionTree`) с ограниченной глубиной (`max_depth`).
- Вычисляются ошибки (разница между реальными метками и предсказаниями) на обучающей выборке.
- Вычисляется вес базовой модели как произведение `learning_rate` на среднюю ошибку.
- Обновляются предсказания для обучающей и тестовой выборок, учитывая вес базовой модели.
- Базовая модель и её вес добавляются в соответствующие списки.

6. Финальные предсказания. после завершения цикла, вычисляются финальные предсказания модели, принимая знак от суммы предсказаний на тестовой выборке.

Таким образом, данная реализация градиентного бустинга с решающими деревьями обучает ансамбль базовых моделей (решающих деревьев) с учетом градиента ошибки и весовых коэффициентов. Полученные предсказания комбинируются для формирования окончательного прогноза для задачи распознавания пола по голосу.

Для оценки реализации работы программы будет проведено сравнение результатов с работой библиотечной реализации случайного леса, `AdaBoost`, `GradientBoostingClassifier`.

Логистическая регрессия – это алгоритм машинного обучения, который широко используется для задач классификации с использованием линейного дискриминанта Фишера, включая фильтрацию спама, значением функции является вероятность того, что данное исходное значение принадлежит к определенному классу.

Для оценки классификации будут использованы следующие метрики.

Accuracy (точность модели) – метрика, которая измеряет общую долю правильно классифицированных образцов (включая истинно положительные и истинно отрицательные результаты) относительно всех образцов.

Формула для вычисления accuracy.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TN (True Negative) – количество верно классифицированных отрицательных результатов.

FN (False Negative) – количество неверно классифицированных отрицательных результатов.

TP (True Positive) – количество верно классифицированных положительных результатов.

FP (False Positive) – количество неверно классифицированных положительных результатов.

Precision (точность) – метрика, которая показывает долю правильно классифицированных положительных результатов.

Формула для вычисления precision.

$$precision = \frac{TP}{TP + FP}$$

Recall (полнота) – метрика, которая измеряет, насколько хорошо модель обнаруживает все положительные результаты.

Формула для вычисления recall.

$$recall = \frac{TP}{TP + FN}$$

F1-мера (F1-score) – гармоническое среднее между precision и recall и представляет собой общую метрику, которая учитывает и точность, и полноту. Она представляет собой баланс между точностью и полнотой и позволяет оценить производительность модели на основе обеих метрик.

Формула для вычисления F1-меры.

$$F1 - score = \frac{2 * precision * recall}{precision + recall}$$

F1-мера близка к 1, если и точность, и полнота высоки. Она является более информативной метрикой, чем точность или полнота в отдельности, когда необходимо учесть их взаимосвязь [7].

7. ВАРИАНТЫ УСОВЕРШЕНСТВОВАНИЙ БАЗОВЫХ АЛГОРИТМОВ

Градиентный бустинг с решающими деревьями можно усовершенствовать с помощью различных техник и стратегий. Вот некоторые варианты усовершенствований базовых алгоритмов.

1. Использование разных функций потерь (Loss Functions). Основной функцией потерь в градиентном бустинге с решающими деревьями обычно является среднеквадратичная ошибка (MSE) для задач регрессии и логистическая функция потерь для задач классификации. Однако, выбор подходящей функции потерь может зависеть от конкретной задачи и данных. Например, Huber loss может быть более устойчив к выбросам в данных, а квантильная регрессия позволяет моделировать квантили распределения целевой переменной.

2. Настройка параметров базовых деревьев. Варьирование параметров базовых деревьев, таких как глубина деревьев (max_depth), минимальное количество объектов в листе (min_samples_leaf) и другие, может существенно повлиять на производительность модели. Эксперименты с разными значениями параметров и поиск оптимальных комбинаций могут улучшить качество бустинга.

3. Использование регуляризации. Для предотвращения переобучения можно применять различные методы регуляризации, такие как уменьшение шага обучения (learning rate), увеличение количества деревьев (n_estimators), и использование ограничений на глубину деревьев (max_depth). Также можно применять L1 и L2 регуляризацию.

4. Сэмплирование данных. Можно использовать различные методы сэмплирования данных для улучшения обобщающей способности модели. Например, бутстрап-сэмплирование и случайное сэмплирование объектов (bagging) могут снизить дисперсию модели, а также позволить обнаруживать разные паттерны в данных.

5. Использование разных базовых алгоритмов. Градиентный бустинг можно комбинировать с разными базовыми алгоритмами, такими как линейные модели, SVM, или нейронные сети. Это позволяет модели смотреть на данные с разных точек зрения и улучшить обобщающую способность.

6. Раннее прекращение обучения (Early Stopping). Для предотвращения переобучения можно использовать раннее прекращение обучения, когда производительность на валидационной выборке перестает улучшаться после определенного количества итераций.

7. Оптимизация гипер-параметров. Процесс поиска оптимальных гипер-параметров, таких как `learning rate`, `n_estimators`, `max_depth` и другие, с использованием методов оптимизации, таких как кросс-валидация, может значительно повысить эффективность градиентного бустинга.

8. РЕАЛИЗАЦИЯ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА

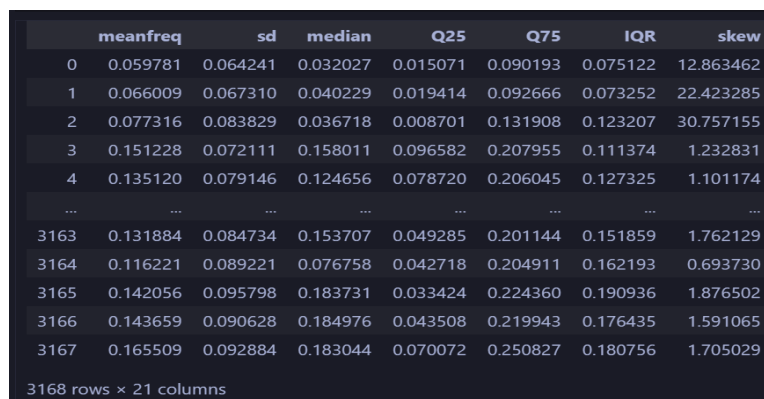
В качестве алгоритма для реализации был выбран градиентный бустинг решающих деревьев.

Цель. классификация пола по голосу (мужской, женский).

Реализация написана на языке Python в среде разработки Jupyter Notebook.

Данные для проверки реализованного алгоритма были взяты в формате csv с сайта Kaggle. Набор данных состоит из 3168 записанных голосовых сэмплов, собранных от мужчин и женщин.

На рисунке 1 представлен первичный вид данных, для удобства работы необходимо их обработать. вставить метки класса в числовом варианте и переименовать столбцы.



	meanfreq	sd	median	Q25	Q75	IQR	skew
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174
...
3163	0.131884	0.084734	0.153707	0.049285	0.201144	0.151859	1.762129
3164	0.116221	0.089221	0.076758	0.042718	0.204911	0.162193	0.693730
3165	0.142056	0.095798	0.183731	0.033424	0.224360	0.190936	1.876502
3166	0.143659	0.090628	0.184976	0.043508	0.219943	0.176435	1.591065
3167	0.165509	0.092884	0.183044	0.070072	0.250827	0.180756	1.705029

3168 rows x 21 columns

Рисунок 1 –Первичный вид данных

За обучение ансамбля базовых моделей с использованием градиентного бустинга отвечает функция `gradient_boosting(X_train, y_train, X_test, n_estimators, learning_rate, max_depth)`. Функция `gradient_boosting` реализует алгоритм градиентного бустинга для задачи бинарной классификации. Она строит ансамбль из `n_estimators` базовых моделей (деревьев решений) с использованием градиентного спуска. На каждой итерации, функция обучает новое дерево решений на ошибках предыдущих предсказаний, взвешивает предсказания базовых моделей с помощью `learning_rate`, и возвращает финальные предсказания ансамбля для тестовых данных.



Рисунок 2 – Структура программы

Также, был написан свой класс `DecisionTree`, который реализует алгоритм построения дерева решений для задачи бинарной классификации. В конструкторе класса можно задать параметры, такие как максимальная глубина дерева (`max_depth`), минимальное количество образцов для разделения узла (`min_samples_split`), и минимальное количество образцов в листовом узле (`min samples leaf`).

Метод `fit` обучает дерево на обучающих данных, а метод `predict` используется для предсказания меток классов на новых данных. Для внутренней работы, класс также содержит методы `_grow_tree` для построения дерева, `_best_split` для выбора наилучшего разделения, и `_most_common_label` для определения наиболее часто встречающегося класса в узле.

Класс `DecisionTreeNode` представляет узел дерева и содержит информацию о разделении (по какому признаку и порогу), детях узла (левый и правый), и значении узла (класс в листовом узле).

Функция `print_tree` используется для визуализации структуры дерева, выводя информацию о признаках и порогах разделения на каждом уровне дерева.

Исходный датасет был разделен на обучающую и тестовую выборку в соотношении 80/20.

```
Accuracy: 0.94086389148264
Точность (Precision): 0.9838187702265372
Полнота (Recall): 0.9020771513353115
F1-мера: 0.9411764705882352

model_weights:
[0.04921073401736385, 0.046901674714023456, 0.04470096077420602, 0.04260350843164363, 0.04060447245985104, 0.04060447245985104, 0.04060447245985104, 0.04060447245985104, 0.04060447245985104, 0.04060447245985104]

Structure of Decision Tree 1:
Feature 12 <= 0.1394832907893745
  Feature 5 <= 0.07127338175740824
    Class: 0
  Class: 1
  Feature 12 <= 0.1476299329104555
    Class: 0
  Class: 0

Structure of Decision Tree 2:
Feature 12 <= 0.1394832907893745
  Feature 5 <= 0.07127338175740824
    Class: 0
  Class: 1
  Feature 12 <= 0.1476299329104555
    ...
    Class: 1
  Feature 12 <= 0.1476299329104555
    Class: 0
```

Рисунок 3 – Результат программы

На основе полученных результатов проводится оценка эффективности по следующим показателям: точность модели (accuracy), точность (precision), полнота (recall) и F1-мера.

9. ОЦЕНКА АЛГОРИТМА

Оценка реализованного алгоритма будет проведена посредством сравнения с библиотечными реализациями (Scikit-learn). AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier по следующим метрикам. точность модели (accuracy), точность (precision), полнота (recall) и F1-мера.

Для проверки было создано два тестовых набора. один из исходного набора данных путем разделения на обучающую и тестовую выборку (тестовый набор 1); второй набор был собран из датасета kaggle (тестовый набор 2).

Оценка результатов работы алгоритмов представлены в таблице 1.

1. Таблица 1 – Оценка работы алгоритмов

	Реализованный алгоритм	AdaBoost	Gradient Boosting	Random Forest Classifier
Тестовая выборка 1	Точность модели. 0.96	Точность модели. 0.97	Точность модели. 0.97	Точность модели. 0.98
	Точность. 0.98	Точность. 0.96	Точность. 0.96	Точность. 0.98
	Полнота. 0.94	Полнота. 0.97	Полнота. 0.97	Полнота. 0.97
	F1-мера. 0.96	F1-мера. 0.96	F1-мера. 0.96	F1-мера. 0.97
Тестовая выборка 2	Точность модели. 0.94	Точность модели. 0.95	Точность модели. 0.94	Точность модели. 0.98
	Точность. 0.98	Точность. 0.96	Точность. 0.96	Точность. 0.99
	Полнота. 0.90	Полнота. 0.0.94	Полнота. 0.92	Полнота. 0.96
	F1-мера. 0.94	F1-мера. 0.95	F1-мера. 0.94	F1-мера. 0.97

По результатам оценки работы алгоритмов на тестовой выборке 1 было получено, что лучшая общая точность модели у случайного леса (0.98), у всех остальных 0.96-0.97. Доля правильно классифицированных положительных результатов наибольшая у реализованного алгоритма и случайного леса (0,98), у остальных 0.96. Наилучшую оценку полноты получили все библиотечные алгоритмы (0.97), наихудшую реализованный алгоритм (0.94). Это означает,

что модель ошиблась в определении пола. Общая оценка работы алгоритмов (F1-мера) примерно одинакова у всех. она больше 0.9, это значит, что все модели достаточно хорошо классифицируют сообщения.

По результатам оценки реализованного алгоритма на тестовом наборе 1 можно сделать вывод о том, что алгоритм градиентного бустинга хорошо справляется с задачей классификации, полученные высокие оценки точности модели, точности и F1-меры, почти эквивалентная относительно остальных алгоритмов полнота означает, что модель отлично определяет пол по голосу.

Результаты оценки работы всех алгоритмов несущественно изменились при тестировании на 2 наборе. Первое место по результатам оценивания получил алгоритм случайного леса, второе AdaBoost, третье GradientBoostingClassifier (scikit-learn), худший результат получил реализованный алгоритм. Стоит отметить, что значения метрик не сильно разнятся и в целом, можно сказать, эквивалентны.

На основе вышесказанного можно сделать вывод о том, что реализованный алгоритм градиентного бустинга деревьев решений достаточно хорошо справляется с классификацией пола по голосу и на основе тестовых наборов практически не уступает остальным алгоритмам.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был изучен алгоритм градиентного бустинга с решающими деревьями, который является мощным инструментом в машинном обучении. Был разработан и реализован код для данного алгоритма, а также проведено его тестирование на задаче распознавания пола по голосу с использованием набора данных, состоящего из 3168 записанных голосовых сэмплов от мужчин и женщин.

В процессе работы над алгоритмом были определены основные параметры, такие как количество базовых моделей (`n_estimators`), скорость обучения (`learning_rate`) и глубина деревьев (`max_depth`), которые влияют на производительность алгоритма. Эти параметры были настроены с целью достижения наилучших результатов на тестовых данных.

Полученные результаты были оценены с использованием метрик, таких как точность, полнота и F1-мера, что позволило сделать вывод о эффективности реализованного алгоритма. Сравнение результатов с другими методами машинного обучения также подтвердило высокую производительность градиентного бустинга с решающими деревьями.

Таким образом, данная работа позволила изучить и применить градиентный бустинг с решающими деревьями для решения задачи классификации на практике. Полученные результаты подтверждают, что этот алгоритм является мощным инструментом машинного обучения и может быть успешно применен для решения разнообразных задач.

СПИСОК ЛИТЕРАТУРЫ

1. Чيو, К. Машинное обучение и безопасность. руководство / К. Чيو, Д. Фримэн; перевод с английского А. В. Снастина. — Москва. ДМК Пресс, 2020. — 388 с.
2. Proglibs [Электронный ресурс] — Режим доступа. <https://proglib.io/p/izuchaem-naivnyy-bayesovskiy-algoritm-klassifikacii-dlya-mashinnogo-obucheniya-2021-11-12>
3. Пальмов, С. В. Системы и методы искусственного интеллекта . учебное пособие / С. В. Пальмов. — Самара. ПГУТИ, 2020. — 191 с.
4. Храмов, А. Г. Методы и алгоритмы интеллектуального анализа данных. учебное пособие / А. Г. Храмов. — Самара. Самарский университет, 2019. — 176 с.
5. Шолле, Ф. Глубокое обучение с R и Keras / Ф. Шолле; перевод с английского В. С. Яценкова. — Москва. ДМК Пресс, 2023. — 646 с.
6. Школа анализа данных. Учебник по машинному обучению. <https://academy.yandex.ru/handbook/ml>
7. Towards data science [Электронный ресурс] — Режим доступа. <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9>
8. Neurohive.io. Градиентный бустинг — просто о сложном. <https://neurohive.io/ru/osnovy-data-science/gradientyj-busting/>
9. Kaggle datasets. Gender Recognition by Voice. <https://www.kaggle.com/datasets/primaryobjects/voicegender>
10. Хабр. Открытый курс машинного обучения. Тема 10. Градиентный бустинг. <https://habr.com/ru/companies/ods/articles/327250/>
11. How to explain gradient boosting. <https://explained.ai/gradient-boosting/index.html>
12. Ансамблевые алгоритмы Spark ML. градиентный бустинг. <https://spark-school.ru/blogs/gradient-boosting-ml/>

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score
from sklearn import preprocessing

le = preprocessing.LabelEncoder()

# Загрузка данных
data = pd.read_csv('voice.csv')

data

# категориальные метки в числовые
data['label'] = le.fit_transform(data['label'])

# Разделение данных на признаки (X) и метки
(y)
X = data.drop(columns=['label'])
y = data['label']

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

X_train = X_train.values # DataFrame в numpy
array
X_test = X_test.values

data

class DecisionTree().
    def __init__(self, max_depth=None,
min_samples_split=2, min_samples_leaf=1).
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y).
        self.X = X
        self.y = y
        self.n_classes_ = len(np.unique(y))
        self.n_samples, self.n_features = X.shape
        self.tree_ = self._grow_tree(X, y, depth=0)

    def predict(self, X).
        return np.array([self._predict(inputs) for
inputs in X])

    def _predict(self, inputs).
        node = self.tree_
        while not node.is_leaf.
            if inputs[node.feature_index] <
node.threshold.
                node = node.left_child
            else.
                node = node.right_child
        return node.value

    def _grow_tree(self, X, y, depth=0).
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        if (n_labels == 1) or (n_samples <
self.min_samples_split) or \
            (self.max_depth is not None and depth >=
self.max_depth).
            leaf_value =
            self._most_common_label(y)
            return
            DecisionTreeNode(value=leaf_value)
```

```

        feature_indices = np.arange(n_features)
        feature_index, threshold = self._best_split(X, y, feature_indices)

        if feature_index is None:
            leaf_value = self._most_common_label(y)
            return DecisionTreeNode(value=leaf_value)

        left_indices = X[:, feature_index] < threshold
        right_indices = ~left_indices
        left_child = self._grow_tree(X[left_indices], y[left_indices],
                                      depth + 1)
        right_child = self._grow_tree(X[right_indices], y[right_indices],
                                      depth + 1)
        return DecisionTreeNode(feature_index=feature_index,
                                threshold=threshold,
                                left_child=left_child,
                                right_child=right_child)

    def _best_split(self, X, y, feature_indices):
        m, n = X.shape
        if m <= 1:
            return None, None

        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_idx, best_thr = None, None

        for idx in feature_indices:
            thresholds, classes = zip(*sorted(zip(X[:, idx],
                                                  num_left = [0] * self.n_classes_
                                                  num_right = num_parent.copy()
                                                  for i in range(1, m).
                                                  c = classes[i - 1]
                                                  num_left[c] += 1
                                                  num_right[c] -= 1
                                                  gini_left = 1.0 - sum(
                                                  (num_left[x] / i) ** 2 for x in
                                                  range(self.n_classes_))
                                                  gini_right = 1.0 - sum(
                                                  (num_right[x] / (m - i)) ** 2 for x in
                                                  range(self.n_classes_))
                                                  )
                                                  gini = (i * gini_left + (m - i) *
                                                  gini_right) / m
                                                  if thresholds[i] == thresholds[i - 1]:
                                                      continue
                                                  if gini < best_gini:
                                                      best_gini = gini
                                                      best_idx = idx
                                                      best_thr = (thresholds[i] +
                                                                  thresholds[i - 1]) / 2
                                                  return best_idx, best_thr

    def _most_common_label(self, y):
        return np.bincount(y).argmax()

class DecisionTreeNode:
    def __init__(self, feature_index=None, threshold=None, left_child=None, right_child=None, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left_child = left_child

```



```

self.right_child = right_child
self.value = value

@property
def is_leaf(self):
    return self.value is not None

def print_tree(node, depth=0):
    indent = " " * depth
    if node.is_leaf:
        print(indent + f"Class. {node.value}")
    else:
        print(indent + f"Feature
{node.feature_index} <= {node.threshold}")
        print_tree(node.left_child, depth + 1)
        print_tree(node.right_child, depth + 1)

def gradient_boosting(X_train, y_train, X_test,
y_test, n_estimators, learning_rate, max_depth):
    # Инициализируем список для хранения
    базовых моделей
    base_models = []

    # Инициализируем список для хранения
    весов базовых моделей
    model_weights = []

    # Инициализируем предсказания для
    обучающей и тестовой выборок
    train_predictions = np.zeros(len(X_train))
    test_predictions = np.zeros(len(X_test))

    for i in range(n_estimators):
        # Создаем и обучаем базовую модель
        (DecisionTree

base_model
=
DecisionTree(max_depth=max_depth)
base_model.fit(X_train, y_train)

# Вычисляем ошибку (разницу между
реальными метками и предсказаниями)
errors = y_train - train_predictions

# Вычисляем вес базовой модели как
learning_rate умноженное на ошибку
model_weight = learning_rate *
errors.mean()

# Обновляем предсказания для
обучающей и тестовой выборок
train_predictions += model_weight *
base_model.predict(X_train)
test_predictions += model_weight *
base_model.predict(X_test)

# Добавляем базовую модель и ее вес в
списки
base_models.append(base_model)
model_weights.append(model_weight)

# Вычисляем финальные предсказания
модели
final_predictions = np.sign(test_predictions)

return base_models, model_weights,
final_predictions

n_estimators = 10
learning_rate = 0.1
max_depth = 2

```

```
base_models, model_weights, final_predictions
= gradient_boosting(X_train, y_train, X_test,
y_test, n_estimators, learning_rate, max_depth)
```

```
# вычисляем точность (accuracy) модели
accuracy      =      accuracy_score(y_test,
final_predictions)
```

```
# вычисляем точность (precision)
precision      =      precision_score(y_test,
final_predictions)
```

```
# вычисляем полноту (recall)
recall = recall_score(y_test, final_predictions)
```

```
# вычисляем F1-меру
f1 = f1_score(y_test, final_predictions)
```

```
print("Accuracy.", accuracy)
print("Точность (Precision).", precision)
print("Полнота (Recall).", recall)
print("F1-мера.", f1)
```

```
print("\nmodel_weights.\n", model_weights)
```

```
for i,m in enumerate(base_models).
    print("\n")
    print(f"Structure of Decision Tree {i + 1 }.")
    print_tree(m.tree_)
```