

**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	21
ЗАКЛЮЧЕНИЕ	23
СПИСОК ЛИТЕРАТУРЫ.....	24
ПРИЛОЖЕНИЕ А. Код классов CustomNode и CustomDecisionTree.....	25
ПРИЛОЖЕНИЕ Б. Код класса CustomGBDT	28
ПРИЛОЖЕНИЕ В. Код программы в Jupyter Notebook.....	29

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:

Инициализация параметров:

Задаются основные параметры алгоритма, такие как количество базовых моделей (деревьев) `n_estimators`, скорость обучения (`learning_rate`) и максимальная глубина каждого дерева (`max_depth`).

Инициализация композиции предсказаний:

Создается начальный вектор предсказаний, инициализированный нулями. Этот вектор будет постепенно корректироваться на каждой итерации.

Цикл по числу базовых моделей (деревьев):

Алгоритм выполняет итерации в течение `n_estimators` раз, создавая и обучая новое дерево на каждой итерации.

Вычисление градиента:

На каждой итерации вычисляется градиент ошибки, который представляет собой разницу между истинными метками классов и текущими предсказаниями.

Обучение базовой модели:

Создается новая базовая модель, в данном случае - решающее дерево с ограниченной глубиной `max_depth`.

Дерево обучается на обучающей выборке с использованием вычисленного градиента вместо истинных меток классов. Это позволяет модели фокусироваться на ошибках предыдущих моделей.

Вычисление предсказаний базовой модели:

Новое дерево используется для вычисления предсказаний на обучающей выборке.

Обновление композиции с учетом скорости обучения:

Предсказания базовой модели умножаются на скорость обучения (`learning_rate`) и добавляются к текущей композиции предсказаний. Это позволяет каждой базовой модели внести свой вклад в итоговый прогноз с учетом заданной скорости обучения.

Добавление базовой модели в список:

Обученное дерево (базовая модель) добавляется в список моделей (`models`), чтобы оно могло использоваться для предсказаний на следующих итерациях.

Финальные предсказания:

После завершения цикла по всем базовым моделям, итоговые предсказания модели вычисляются как сумма предсказаний всех базовых моделей. Если значение превышает порог 0.5, то объект классифицируется как класс 1, в противном случае как класс 0.

Таким образом, данная реализация градиентного бустинга с решающими деревьями обучает ансамбль базовых моделей (решающих деревьев) с учетом градиента ошибки. 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.

Для оценки классификации будут использованы следующие метрики.

Ассурасу (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].

3. 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.

4. РЕАЛИЗАЦИЯ АЛГОРИТМА ГРАДИЕНТНОГО БУСТИНГА ДЕРЕВЬЕВ РЕШЕНИЙ ФРИДМАНА

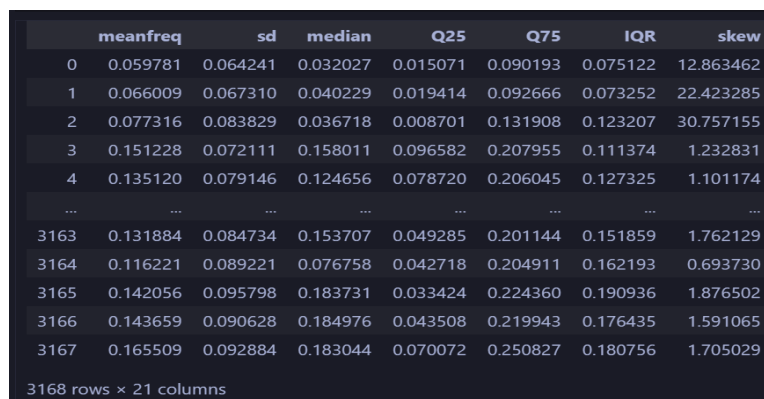
В качестве алгоритма для реализации был выбран градиентный бустинг решающих деревьев.

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 класс CustomGBDT. Класс CustomGBDT реализует алгоритм градиентного бустинга для задачи бинарной классификации.

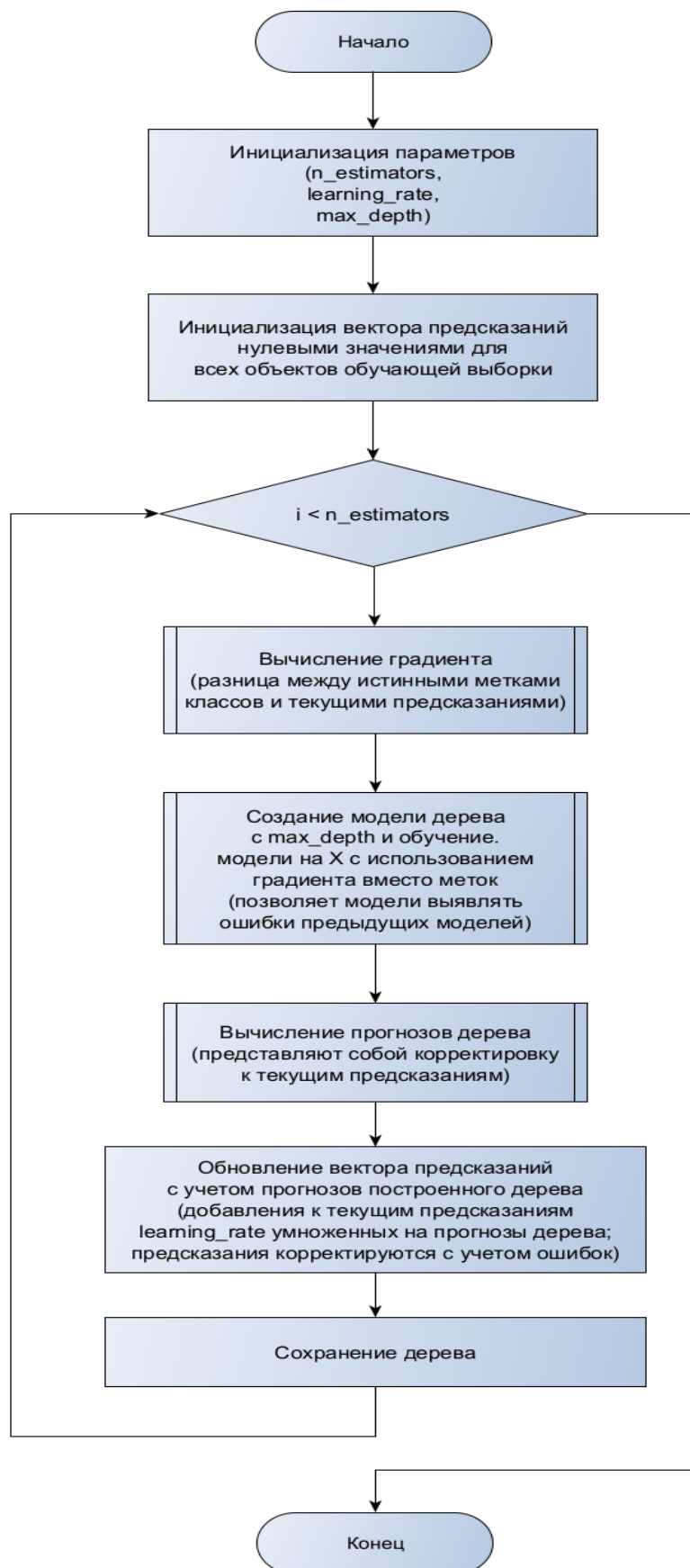


Рисунок 2 – Алгоритм обучения GBDT

Также, был написан свой класс CustomDecisionTree, 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).

Класс CustomNode определяет структуру узла дерева. Узел может быть решающим (decision node) или листовым (leaf node). У решающего узла есть feature (признак), threshold (пороговое значение), left и right потомки (узлы) и value (значение, если узел листовой).

Таблица 1 – Методы класса CustomDecisionTree

№	Название	Описание
1)	_is_finished	Проверяет критерии останова по глубине дерева, количеству классов (n_class_labels), и минимальному количеству образцов для разделения узла. Если один из этих критериев выполняется, возвращает True, иначе False
2)	_unique_counts	Рассчитывает количество уникальных значений в массиве y и возвращает словарь, где ключи - уникальные значения, а значения - их количество в массиве.
3)	_entropy	Рассчитывает энтропию для массива меток y. Энтропия используется для измерения неопределенности. Чем больше неопределенность, тем выше энтропия. Вычисление происходит на основе количества классов и их относительных частот.

Таблица 1 – Продолжение

4)	<code>_create_split</code>	Разделяет индексы образцов на две части: одни, у которых значение признака X меньше или равно пороговому значению <code>thresh</code> , и другие - больше порогового значения. Возвращает индексы обеих частей.
5)	<code>_information_gain</code>	Рассчитывает информационный выигрыш при разделении по признаку X и пороговому значению <code>thresh</code> . Вычисляется как разница между энтропией родительского узла и взвешенной суммой энтропий дочерних узлов (слева и справа после разделения).
6)	<code>_best_split</code>	Выбирает наилучшее разделение (признак и порог) с наибольшим информационным выигрышем. Проходит по всем признакам и порогам, вычисляя информационный выигрыш, и выбирает наилучшее разделение.
7)	<code>_build_tree</code>	Рекурсивно строит дерево. Если выполняются критерии остановки, создается листовой узел с наиболее часто встречающимся классом. В противном случае выбирается наилучшее разделение, и узлу присваиваются соответствующие значения.
8)	<code>_traverse_tree</code>	Рекурсивно проходит по дереву для принятия решения. Начиная с корневого узла, он следует соответствующей ветви, пока не достигнет листового узла, и возвращает значение листового узла.

Таблица 1 – Продолжение

9)	fit	Обучает дерево на обучающих данных X и метках y. Если данные не являются массивами NumPy, они преобразуются в таковые. Затем вызывается <code>_build_tree</code> для построения дерева
10)	predict	Применяет обученное дерево для предсказания меток на новых данных X. Если данные не являются массивами NumPy, они преобразуются. Для каждого образца вызывается <code>_traverse_tree</code> для принятия решения.
11)	print_tree_structure	Выводит структуру дерева

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

```

Tree= 15
Tree= 16
Tree= 17
Tree= 18
Tree= 19
Tree= 20
#####

Средняя точность (Accuracy) по 5 экспериментам: 0.97
Средняя точность (Precision) по 5 экспериментам: 0.96
Средняя полнота (Recall) по 5 экспериментам: 0.98
Средняя F1-мера по 5 экспериментам: 0.97

```

Рисунок 3 – Результат программы

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

5. 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. 0.98	Точность модели. 0.97	Точность модели. 0.98	Точность модели. 0.97
	Точность. 0.98	Точность. 0.97	Точность. 0.98	Точность. 0.98
	Полнота. 0.98	Полнота. 0.97	Полнота. 0.98	Полнота. 0.97
	F1-мера. 0.98	F1-мера. 0.97	F1-мера. 0.98	F1-мера. 0.97
Тестовая выборка 2	Точность модели. 0.97	Точность модели. 0.96	Точность модели. 0.97	Точность модели. 0.98
	Точность. 0.96	Точность. 0.94	Точность. 0.96	Точность. 0.98
	Полнота. 0.98	Полнота. 0.97	Полнота. 0.98	Полнота. 0.98
	F1-мера. 0.97	F1-мера. 0.95	F1-мера. 0.97	F1-мера. 0.98

По результатам оценки работы алгоритмов на тестовых выборках можно заметить, что результаты не сильно разнятся. В целом все примерно одинаково отработали. Единственное, что стоит отметить, так это то, что реализованный алгоритм дольше всех выполнялся (более 40 минут) по сравнению с готовыми библиотеками (~0.4 – 2.1 сек).

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. Единственным минусом является время расчетов.

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 точность предсказаний и вычислений.

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 классов CustomNode и CustomDecisionTree

```
import numpy as np

# Класс узла ДР
class CustomNode:
    def __init__(self, feature=None,
threshold=None, left=None, right=None,
value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

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

## Класс ДР
class CustomDecisionTree():
    # критерии остановки: max_depth,
min_samples_split, root_node
    def __init__(self, max_depth=100,
min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.root = None

    def _is_finished(self, depth):
        # вычисляем проверки
        limit1 = depth >= self.max_depth
        limit2 = self.n_class_labels == 1
        limit3 = self.n_samples <
self.min_samples_split
        if(limit1 or limit2 or limit3):
            return True
        return False

    def _unique_counts(self, y):
        unique_values, counts = np.unique(y,
return_counts=True)
        # используем функцию zip, чтобы
объединить массив unique_values и массив
counts в кортежи,
        # где каждый кортеж содержит
уникальное значение и количество его
повторений
        # dict() используется для создания
словаря из кортежей
        return dict(zip(unique_values, counts))

    def _entropy(self, y):
        n = len(y)
        unique_counts = self._unique_counts(y)
        entropy = 0.0
        for count in unique_counts.values():
            p = count / n
            entropy -= p * np.log2(p)
        return entropy

    def _create_split(self, X, thresh):
        left_idx = np.argwhere(X <=
thresh).flatten()
        right_idx = np.argwhere(X >
thresh).flatten()
        return left_idx, right_idx

    def _information_gain(self, X, y, thresh):
        parent_loss = self._entropy(y)
        left_idx, right_idx = self._create_split(X,
thresh)
        n, n_left, n_right = len(y), len(left_idx),
len(right_idx)

        if (n_left == 0 or n_right == 0):
```

```

        return 0

        child_loss = (n_left / n) *
self._entropy(y[left_idx]) + (n_right / n) *
self._entropy(y[right_idx])
        return parent_loss - child_loss

def _best_split(self, X, y, features):
    split = {"score": -1, "feat": None, "thresh":
None}

    for feat in features:
        X_feat = X[:, feat]
        thresholds = np.unique(X_feat)
        for thresh in thresholds:
            score = self._information_gain(X_feat,
y, thresh)

            if (score > split["score"]):
                split["score"] = score
                split["feat"] = feat
                split["thresh"] = thresh

    return split["feat"], split["thresh"]

def _build_tree(self, X, y, depth=0):
    self.n_samples, self.n_features = X.shape
    self.n_class_labels = len(np.unique(y))

    # критерий останковки
    if (self._is_finished(depth)):
        most_common_Label =
max(self._unique_counts(y),
key=self._unique_counts(y).get)
        return
CustomNode(value=most_common_Label)

    # лучшее разделение

```

```

        rnd_feats =
np.random.choice(self.n_features,
self.n_features, replace=False)
        best_feat, best_thresh = self._best_split(X,
y, rnd_feats)

        # рекурсивное получение потомков
(узлов)
        left_idx, right_idx = self._create_split(X[:,
best_feat], best_thresh)
        left_child = self._build_tree(X[left_idx, :],
y[left_idx], depth + 1)
        right_child = self._build_tree(X[right_idx,
:], y[right_idx], depth + 1)
        return CustomNode(best_feat, best_thresh,
left_child, right_child)

def _traverse_tree(self, x, node):
    if (node.is_leaf()):
        return node.value
    if (x[node.feature] <= node.threshold):
        return self._traverse_tree(x, node.left)
    return self._traverse_tree(x, node.right)

def fit(self, X, y):
    if not isinstance(X, np.ndarray):
        X = X.to_numpy()
    if not isinstance(y, np.ndarray):
        y = y.to_numpy()
    self.root = self._build_tree(X, y)

def predict(self, X):
    if not isinstance(X, np.ndarray):
        X = X.to_numpy()
    predictions = [self._traverse_tree(x,
self.root) for x in X]
    return np.array(predictions)

```

```

# структура дерева
def print_tree_structure(self, node=None,
depth=0):
    if node is None:
        node = self.root

    indent = " " * depth
    if node.is_leaf():
        print(indent + f"Leaf Node: Class
{node.value}")
    else:
        print(indent + f"Decision Node: Feature
{node.feature}, Threshold {node.threshold}")
        print(indent + " Left Branch:")
        self.print_tree_structure(node.left, depth
+ 1)
        print(indent + " Right Branch:")
        self.print_tree_structure(node.right,
depth + 1)

```

ПРИЛОЖЕНИЕ Б. Код класса CustomGBDT

```
import numpy as np
from CustomDecisionTree import CustomDecisionTree

class CustomGBDT:
    def __init__(self, n_estimators=100,
learning_rate=0.1, max_depth=3):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.models = [] # храним базовые модели
(деревья)

    def fit(self, X, y):
        # инициализируем композицию
предсказаний нулевым вектором
        predictions = np.zeros(len(y))
        # print("predictions", predictions)
        idx = 0
        for _ in range(self.n_estimators):
            # вычисляем градиент
            gradient = y - predictions
            # print("gradient")
            # print(gradient)
            idx += 1
            print("Tree=",idx)

            # обучаем дерево на градиенте

        tree = CustomDecisionTree(max_depth=self.max_depth)
        tree.fit(X, gradient) # градиент вместо
меток

        # вычисляем прогнозы базовой модели
        tree_predictions = tree.predict(X)

        # обновляем композицию с учетом
learning_rate
        predictions += self.learning_rate *
tree_predictions

        # добавляем дерево в список
        self.models.append(tree)

    def predict(self, X):
        # для прогноза суммируем прогнозы всех
деревьев
        predictions = np.zeros(len(X))
        for model in self.models:
            tree_predictions = model.predict(X)
            predictions += self.learning_rate *
tree_predictions

        # преобразуем предсказания в бинарные
метки классов (0 и 1)
        return np.where(predictions >= 0.5, 1, 0)
```

ПРИЛОЖЕНИЕ В. Код программы в Jupyter Notebook

```
from CustomGBDT import CustomGBDT

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

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

# Разделяем данные на признаки (X) и целевую переменную (y)
X = data.iloc[:, :-1] # Все столбцы, кроме последнего
y = data['label']      # Последний столбец

# Преобразуем метки классов в числовой формат (male -> 0, female -> 1)
y = y.map({'male': 0, 'female': 1})

num_experiments = 5
n_e = [16, 17, 18, 19, 20]
l_r = [0.1, 0.1, 0.1, 0.1, 0.1]
m_d = [3, 3, 3, 4, 5]

# хранения результатов метрик для подсчёта среднего
accuracy_results = []
precision_results = []
recall_results = []
f1_results = []

def evaluate(X, y, n_e=20, l_r=0.1, m_d=5):

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

    clf = CustomGBDT(n_estimators=n_e, learning_rate=l_r, max_depth=m_d)
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    # оценка модели
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return accuracy, precision, recall, f1

idx = 0
for i in range(num_experiments):
    idx += 1
    print("experiment #", idx)
    print("n_e", n_e[i])
    print("l_r", l_r[i])
    print("m_d", m_d[i])
    accuracy, precision, recall, f1 = evaluate(X = X, y = y, n_e=n_e[i], l_r=l_r[i], m_d=m_d[i])

    accuracy_results.append(accuracy)
    precision_results.append(precision)
    recall_results.append(recall)
    f1_results.append(f1)

print("#####\n")

# вычисление средних значений
```

```

average_accuracy = sum(accuracy_results) /
num_experiments
average_precision = sum(precision_results) /
num_experiments
average_recall = sum(recall_results) /
num_experiments
average_f1 = sum(f1_results) / num_experiments

```

```

# средние значения
print(f'Средняя точность (Accuracy) по
{num_experiments} экспериментам:
{average_accuracy:.2f}')
print(f'Средняя точность (Precision) по
{num_experiments} экспериментам:
{average_precision:.2f}')
print(f'Средняя полнота (Recall) по
{num_experiments} экспериментам:
{average_recall:.2f}')
print(f'Средняя F1-мера по {num_experiments}
экспериментам: {average_f1:.2f}')

```