МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий
(наименование института, филиала)

Кафедра информатики
(наименование кафедры)

КУРСОВАЯ РАБОТА

По дисциплине **Основы машинного обучения**
(наименование дисциплины)

Тема курсовой работы **Реализация алгоритмов решающих деревьев**

Студент группы БФБО-05-22 Нахамкин Константин Дмитриевич
(учебная группа, фамилия, имя, отчество студента) *(подпись студента)*

Руководитель курсовой работы зав. каф., к.ф.-м.н., доцент, Шмелева А.Г.
(должность, звание, ученая степень, ФИО) *(подпись руководителя)*

Рецензент (при наличии) – –
(должность, звание, учёная степень) *(подпись рецензента)*

Работа представлена к защите «___» _____ 2023 г.

Допущен к защите «___» _____ 2023 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий
(наименование института, филиала)

Кафедра информатики
(наименование кафедры)

Утверждаю
заведующий кафедрой информатики
_____ Шмелева А.Г.
Подпись *ФИО*
«____» _____ 20___ г.

ЗАДАНИЕ
на выполнение курсовой работы по дисциплине
«Основы машинного обучения»

Студент Нахамкин Константин Дмитриевич Группа БФБО-05-22

Тема «Реализация алгоритмов решающих деревьев»

Исходные данные: **Анонимные данные клиентов банка**

Перечень вопросов, подлежащих разработке, и обязательного графического материала: изучение библиотек Pandas, Matplotlib, Sklearn языка программирования Python, методы построения решающих деревьев, решение прикладной задачи анализа датасета, представление результата в виде значений и графиков.

Срок предоставления к защите курсового проекта (работы): до «___» _____ 20__ г.

Задание на курсовой проект (работу) выдал _____ (_____)
«01» марта 2023 г. *Подпись руководителя* *Ф.И.О. руководителя*

Задание на курсовой проект (работу) получил _____ (_____)
Подпись обучающегося *Ф.И.О. исполнителя*.

Оглавление

Введение

Дисциплина «Основы машинного обучения» направлена на получение навыков создания прикладных программ и реализации алгоритмов на высокоуровневом объектно-ориентированном языке программирования Python. Язык программирования Python является одним из наиболее популярных и востребованных языков, применяемых для решения прикладных задач анализа данных как в крупных компаниях, так и стартапах. Популярность данного языка обеспечивает наличие достаточно большого количества модулей и библиотек, а также бесплатной среды разработки [1].

Деревья решений – популярный и универсальный инструмент интеллектуального анализа данных и предсказательной аналитики, использующийся для решения широкого класса задач. Они представляют из себя графическое представление процесса принятия решений, которое моделирует результаты и предсказывает будущие события. Деревья решений состоят из узлов, представляющих принятие решения, листьев, представляющих окончательные результаты, и веток (ребер графа), соединяющих их друг с другом [2][3]. Схематическое изображение представлено на рисунке 1.1.

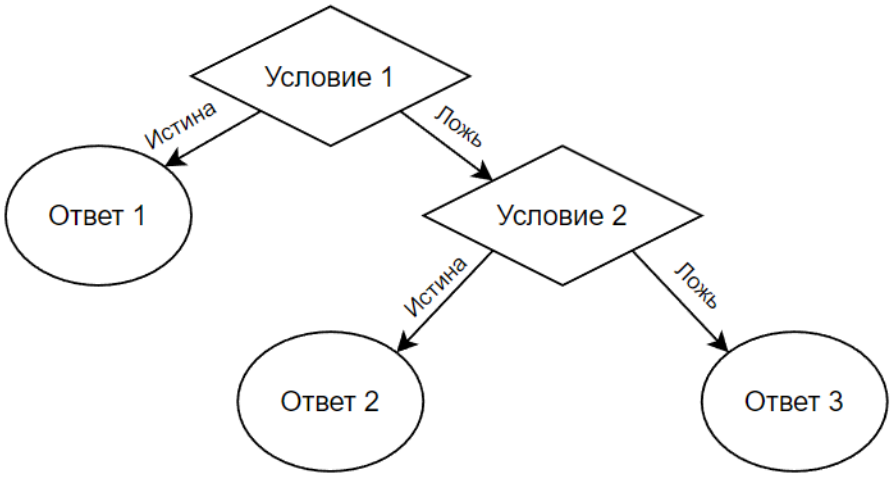

Рисунок 1.1 – Пример дерева решений

Деревья решений широко используются в различных областях, включая науку о данных, машинное обучение и искусственный интеллект. Они часто используются при анализе решений для определения наилучшего курса действий с учетом набора возможных результатов и связанных с ними вероятностей.

Деревья решений также используются в задачах классификации и регрессии для классификации данных или прогнозирования конкретного результата на основе набора входных переменных.

Построение дерева решений включает в себя процесс, называемый рекурсивным разделением, когда данные разбиваются на более мелкие подмножества на основе наиболее значимого признака. Этот признак выбирается на основе его способности разбивать данные на подмножества, максимально схожие с точки зрения переменной результата. Есть множество алгоритмов выбора признака для следующего ветвления (ID3, C4.5, CART), но в основном все они основаны на какой-либо формуле понижения энтропии.

Целью данной работы является подробное ознакомление с решающими деревьями и алгоритмами их построения, а также их программная реализация на большом датасете.

1. Теоретические основы алгоритмов реализации решающих деревьев

1.1. Основные определения и формулы метода решающих деревьев

1.1.1 Дерево решений и алгоритмы его построения

Дерево — это связный ациклический граф. Связность означает наличие маршрута между любой парой вершин, ацикличность — отсутствие циклов. Отсюда, в частности, следует, что число рёбер в дереве на единицу меньше числа вершин, а между любыми парами вершин имеется один и только один путь [4].

Дерево принятия решений (в данной работе будет использоваться именно дерево классификации) — средство поддержки принятия решений, использующееся в машинном обучении, анализе данных и статистике. Структура дерева представляет собой листья и ветки. На рёбрах(ветках) дерева решения записаны признаки, от которых зависит целевая функция, в листьях записаны значения целевой функции, а в остальных вершинах — признаки, по которым различаются случаи. Чтобы классифицировать новый случай, надо спуститься по дереву от корня до одного из листов и выдать соответствующее значение [5].

Очень сложно сразу же взять и определить критерии классификации элементов. Чтобы это сделать, используют различные алгоритмы. Например:

- Алгоритм ID3, относительно простой и быстрый алгоритм.
- Алгоритм C4.5, улучшенная версия ID3.
- Алгоритм CART, более универсальный и сложный [6].

Все эти алгоритмы имеют свои особенности, но в основе имеют один и тот же алгоритм, который показан на блок-схеме (рисунок 1.2).

Выбор алгоритма не всегда прост и основывается на различных критериях: поставленная задача (классификации или регрессии), характеристики желаемого дерева (например глубина и ширина), данные (их размер, наличие в них пропусков, их тип), время, требуемое на построение дерева и т. д. Но для небольших проектов зачастую используется алгоритм, встроенный в используемый инструмент

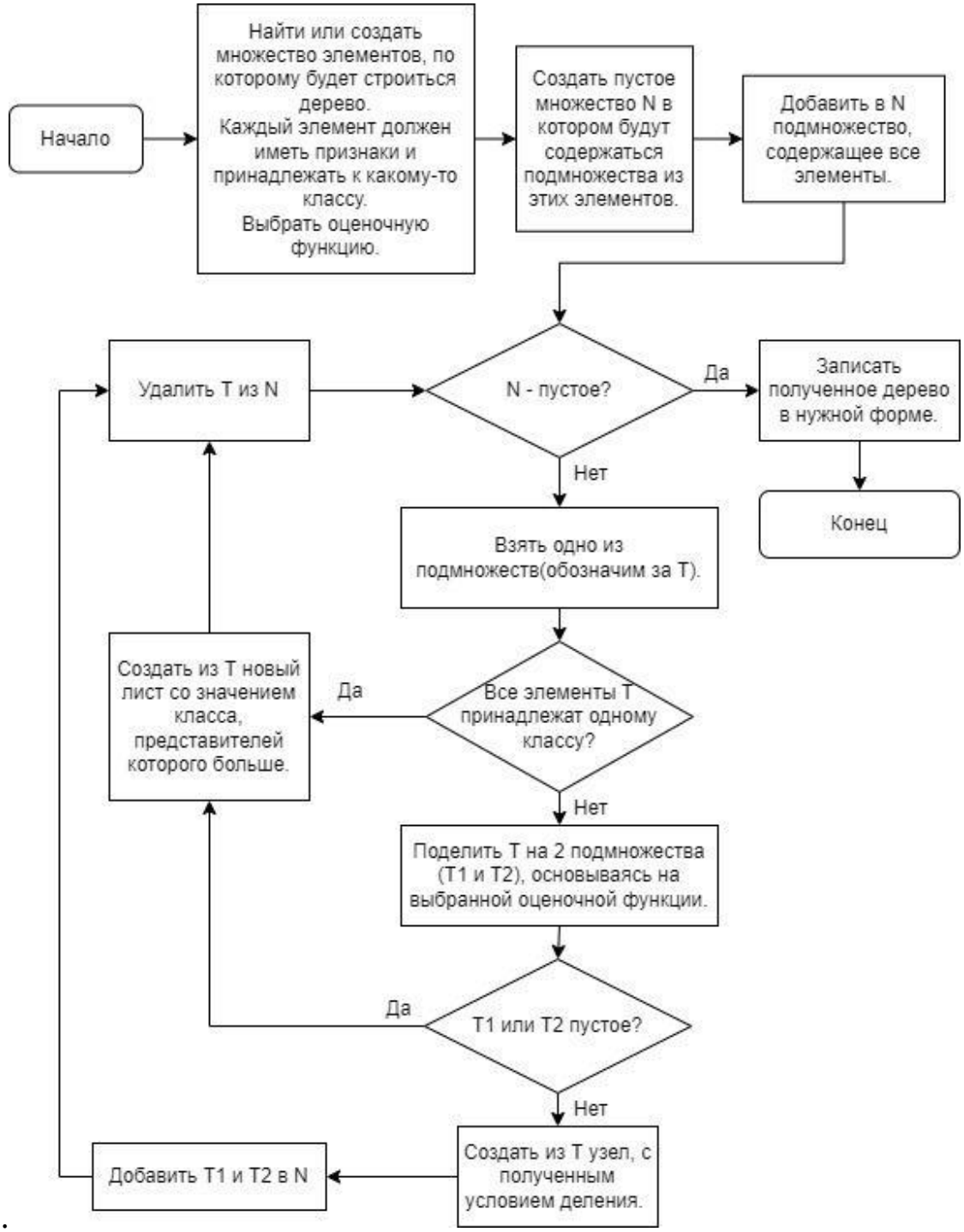

Рисунок 1.2

Для программного решения задачи будут использоваться методы класса DecisionTreeClassifier [7] библиотеки sklearn, в котором для построения дерева применяется модифицированная версия алгоритма CART (Classification And Regression Tree) [8]. Рассмотрим его основные особенности:

- Использование индекса Джини в качестве оценочной функции
- Проведение обрезки готовых деревьев (англ. “Pruning”) — процесс удаления маловажных вершин, для предотвращения переобучения.

- Создание исключительно бинарных деревьев
- Возможность работы с порядковыми и номинальными переменными
- Возможность построения деревьев как классификации, так и регрессии

Обратим внимание на самые важные из них.

1.1.2 Индекс Джини

Индекс Джини для одного множества элементов T вычисляется следующим образом [6]:

$$Gini(T) = 1 - \sum_{i=1}^{n} p_i^2,$$

где n — это количество классов в T, p_i — частота появления класса i в T. Если множество T мощностью N разбить на две части T_l и T_r с числом элементов в каждом L и R соответственно, тогда показатель качества разбиения можно найти как:

$$Gini_{split} = \frac{L}{N} \cdot Gini(T_l) + \frac{R}{N} \cdot Gini(T_r).$$

Наилучшим считается то разбиение, для которого $Gini_{split}(T)$ минимально. Обозначим l_i и r_i — число экземпляров i-го класса в левом/правом потомке. Тогда качество разбиения оценивается по следующей формуле:

$$Gini_{split} = \frac{L}{N}\left(1 - \sum_{i=1}^{n} \frac{l_i^2}{L^2}\right) + \frac{R}{N}\left(1 - \sum_{i=1}^{n} \frac{r_i^2}{R^2}\right) \to \min \quad .$$

Для упрощения вычислений формулу можно упростить:

$$Gini_{split} = N\left(L\left(1 - \frac{1}{L^2}\sum_{i=1}^{n} l_i^2\right) + R\left(1 - \frac{1}{R^2}\sum_{i=1}^{n} r_i^2\right)\right) \to \min \quad .$$

Так как при минимизации константы не играют роли, можно сократить N:

$$Gini_{split} = L - \frac{1}{L}\sum_{i=1}^{n} {l^2}_i + R - \frac{1}{R}\sum_{i=1}^{n} {r_i}^2 \to min,$$

$$Gini_{split} = N - \left(\frac{1}{L}\sum_{i=1}^{n} l_i^2 + \frac{1}{R}\sum_{i=1}^{n} r_i^2\right) \to \min \quad .$$

Или же:

$$G_{split} = \frac{1}{L}\sum_{i=1}^{n} l_i^2 + \frac{1}{R}\sum_{i=1}^{n} r_i^2 \to \max \quad .$$

Таким образом, лучшим будет разбиение с максимальным G_{split} или минимальным $Gini_{split}$.

1.1.3 Обрезка дерева (Pruning)

Алгоритмы обрезки делятся на два основных типа [9]: работающие во время и после построения дерева. Алгоритмы, использующиеся во время

построения, обычно ограничивают какой-либо параметр, не давая дереву расти. Например, максимальную глубину или минимальное количество элементов в листе. Алгоритмы, применяющиеся на уже готовом дереве, как правило дают оценку каждому узлу и удаляют все, не соответствующие заданным критериям.

Здесь будет описан алгоритм “Minimal Cost-Complexity Pruning” [10], который используется для оптимизации деревьев решения в библиотеке sklearn.

Для каждого дерева T задается критерий стоимости-сложности (cost-complexity measure), зависящий от параметра α (по-умолчанию $\alpha = 0$).

$$R_\alpha(T) = R(T) + \alpha|T|,$$

где $R(T)$ — взвешенный индекс Джини (может использоваться и другая формула, например ошибка классификации), $|T|$— количество листьев в дереве. Из этой формулы можно вывести критерий стоимости-сложности отдельного узла t:

$$R_\alpha(t) = R(t) + \alpha.$$

Если T_t — дерево, полученное сразу после деления узла t, то, при $\alpha = 0$, $R_\alpha(T_t) < R_\alpha(t)$, но при определенном значении α они будут равны. Обозначим это значение как α_{eff} Его можно найти, приравняв предыдущие уравнения:

$$\alpha_{eff} = \frac{R(t)-R(T_t)}{|T|-1}.$$

Так как алгоритм CART создает только бинарные деревья, то

$$\alpha_{eff} = R(t) - R(T_t).$$

Это и есть оценочный параметр в данном алгоритме. Все узлы, α_{eff}которых меньше параметра ccp_alpha, который задается при создании дерева, будут удалены.

Этот метод обрезки деревьев является относительно время затратным, но в то же время эффективным.

1.2. Подробное решение модельной задачи

Рассмотрим алгоритм построения дерева решений на небольшом датасете. Он был сгенерирован при помощи чат-бота ChatGPT [11]. В нем содержится 10 элементов, у каждого из которых есть 3 параметра. Первые два из них числовые,

а третий — категориальный. Каждый элемент также принадлежит к одному из двух классов, указанном в столбце Label (Рисунок 1.3).

Feature 1	Feature 2	Feature 3	Label
0.3	3.2	Category A	Yes
0.6	-5.5	Category B	No
0.8	7.1	Category A	Yes
0.1	-9.8	Category C	Yes
0.5	1.3	Category B	No
0.2	4.6	Category C	Yes
0.9	-2.4	Category A	No
0.4	8.9	Category B	No
0.7	-7.2	Category C	No
0.6	0.1	Category A	Yes

Рисунок 1.3

Для начала можно найти индекс Джини начального состояния T_0 (корневой вершины), чтобы убедиться, что с ростом дерева он уменьшится.

$$Gini(T_0) = 1 - \left(\left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^2\right) = 1 - 0{,}5 = 0{,}5\ .$$

Наше дерево на данном этапе выглядит как на рисунке 1.4

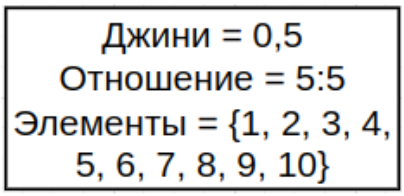

Рисунок 1.4

Методом перебора находим условие, минимизирующее $Gini_{split}$. В данном случае это $Feature\,1 \leq 0{,}35$. Элементы с индексами 1, 4, 6 уйдут в левую вершину, а остальные в правую.

$$Gini_{split} = \frac{3}{10} * \left(1 - \frac{9}{9}\right) + \frac{7}{10} * \left(1 - \frac{4}{49} - \frac{25}{49}\right) = \frac{3}{10} * 0 + \frac{7}{10} * 0{,}408 = 0{,}285\ .$$

В ходе вычислений так же были получены индексы Джини новых вершин: 0 и 0,285 для левой и правой соответственно. Так как в левой вершине содержатся представители только одного класса, она становится листом со значением “Yes”, а последующие деления будут касаться только правого поддерева.

После этого деления дерево имеет вид как на рисунке 1.5

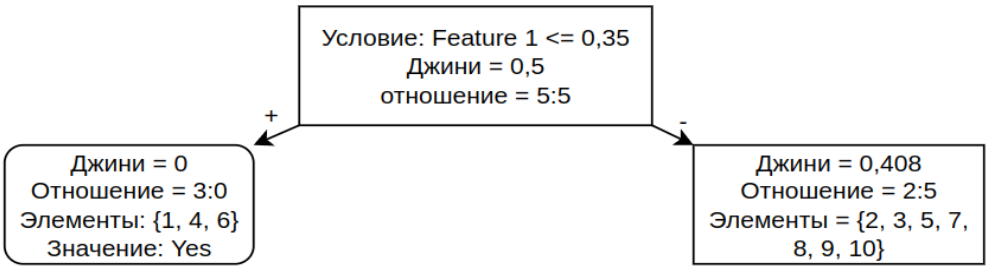

Рисунок 1.5

Следующим условием будет $Feature\ 3\ \neq\ Category\ A$.

$$Gini_{split} = \frac{4}{7} * \left(1 - \frac{16}{16}\right) + \frac{3}{7} * \left(1 - \frac{4}{9} - \frac{1}{9}\right) = \frac{4}{7} * 0 + \frac{3}{7} * 0{,}444 = 0{,}19$$

В ходе деления так же образуется новый лист и узел (рисунок 1.6).

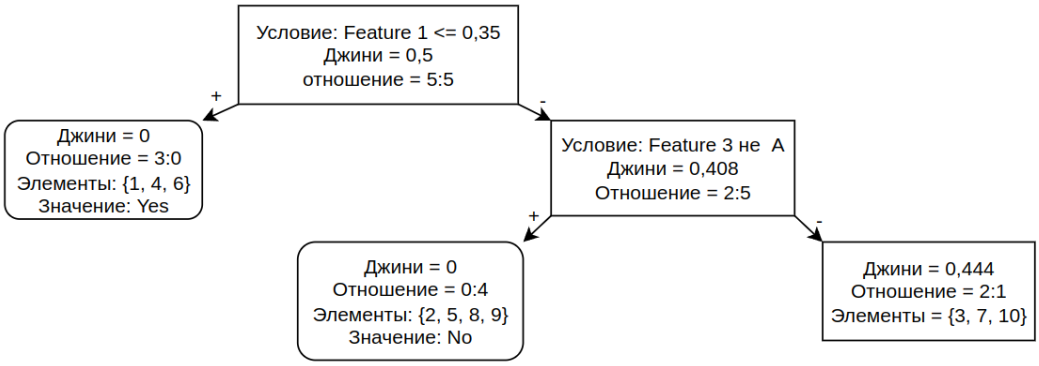

Рисунок 1.6

Последним условием деления будет $Feature\ 1\ \leq\ 0{,}85$. Оно делит узел на два, в каждом из которых индекс Джини равен 0.

$$Gini_{split} = \frac{2}{3} * \left(1 - \frac{4}{4}\right) + \frac{1}{3} * \left(1 - \frac{1}{1}\right) = 0$$

Тогда окончательный вид дерева будет как на рисунке 1.7.

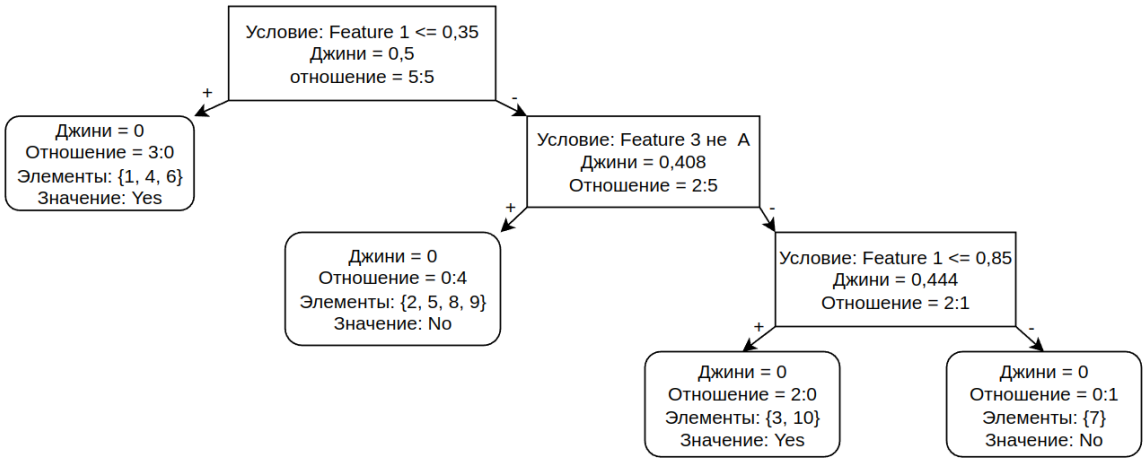

Рисунок 1.7

Итоговым значением индекса Джини для дерева будет сумма произведений индексов Джини каждого листа на отношение количества элементов в нем к общему количеству элементов в дереве. Так как в каждом листе элементы принадлежат одному и тому же классу, это значение будет равно 0, что меньше начального $Gini(T_0) = 0{,}5$.

Теперь рассчитаем значение α_{eff} для каждого узла снизу вверх и посмотрим какой вид будет принимать дерево при различных ccp_alpha(рисунок 1.8).

1) $\alpha_{eff} = \frac{3}{10} * 0{,}444 - 0 = 0{,}13$
2) $\alpha_{eff} = \frac{7}{10} * 0{,}408 - \frac{3}{10} * 0{,}444 = 0{,}15$,
3) $\alpha_{eff} = \frac{10}{10} * 0{,}5 - \frac{7}{10} * 0{,}408 = 0{,}21$.

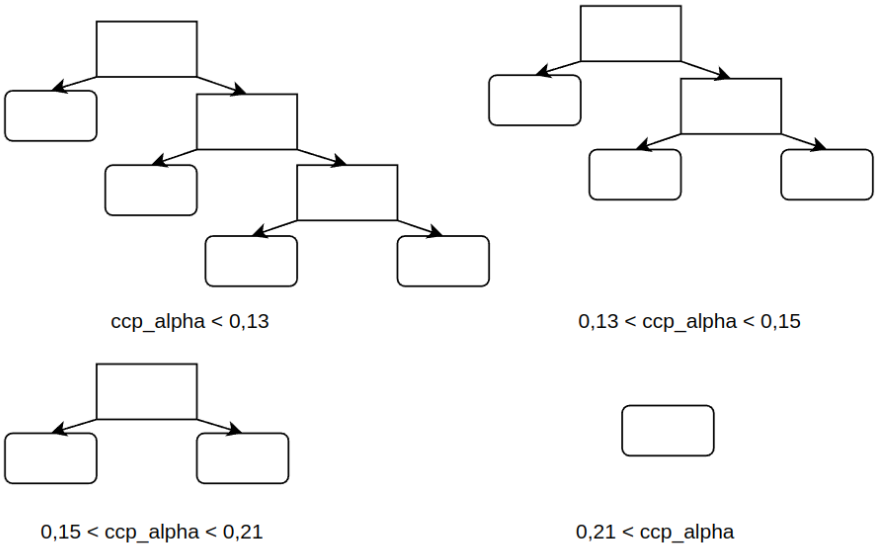

Рисунок 1.8

Если бы этот датасет был разделен на обучающую и тестовую выборки, то можно было бы найти оптимальное значение ccp_alpha, подставляя его из каждого интервала и оценивая ошибку на тестовых значениях. Этот процесс будет представлен в программном решении задачи, так как он не представляет большой пользы для маленьких выборок и выглядит более понятно на больших.

2. Программная реализация решающих деревьев

2.1 Описание датасета и постановка задачи

Используемый датасет представляет собой зашифрованные данные клиентов банка [12]. В них входят идентификационный номер(id), 11 различных числовых параметров (fea_1 - fea_11) и принадлежность к одной из двух групп риска(label). 1 означает высокий риск, 0 означает низкий риск. Данные хранятся как .csv файл. Исходный код программы и ссылка на него указаны в приложении.

В задачу входит анализ данных и их подготовка к использованию для построения деревьев классификации, построение нескольких деревьев различными методами для сравнения их эффективности и анализ полученных результатов.

2.2 Анализ и подготовка данных

Преобразуем данные из .csv файла в структуру DataFrame из библиотеки pandas и посмотрим на их описание (рисунок 2.1).

	label	id	fea_1	fea_2	fea_3	fea_4	fea_5	fea_6	fea_7	fe
count	1125.000000	1.125000e+03	1125.000000	976.000000	1125.000000	1.125000e+03	1125.000000	1125.000000	1125.000000	1125.000
mean	0.200000	5.783677e+07	5.482667	1283.911373	2.333333	1.208836e+05	1.928889	10.872000	4.832889	100.802
std	0.400178	1.817150e+06	1.383338	51.764022	0.878773	8.844523e+04	0.257125	2.676437	2.971182	11.988
min	0.000000	5.498235e+07	1.000000	1116.500000	1.000000	1.500000e+04	1.000000	3.000000	-1.000000	64.000
25%	0.000000	5.499050e+07	4.000000	1244.000000	1.000000	7.200000e+04	2.000000	8.000000	5.000000	90.000
50%	0.000000	5.898975e+07	5.000000	1281.500000	3.000000	1.020000e+05	2.000000	11.000000	5.000000	105.000
75%	0.000000	5.899799e+07	7.000000	1314.500000	3.000000	1.390000e+05	2.000000	11.000000	5.000000	111.000
max	1.000000	5.900624e+07	7.000000	1481.000000	3.000000	1.200000e+06	2.000000	16.000000	10.000000	115.000

Рисунок 2.1 (некоторые столбцы скрыты, но в них не содержится критически важная информация)

Из первой строки видно, что некоторые данные отсутствуют. Получим их полный список (рисунок 2.2). Не хватает 149 значений в столбце fea_2. Так как все значения в нем образуют нормальное распределение, и существенные выбросы отсутствуют (Рисунок 2.3), можно заполнить пропуски средним арифметическим.

```
label       0
id          0
fea_1       0
fea_2     149
fea_3       0
fea_4       0
fea_5       0
fea_6       0
fea_7       0
fea_8       0
fea_9       0
fea_10      0
fea_11      0
dtype: int64
```

Рисунок 2.2

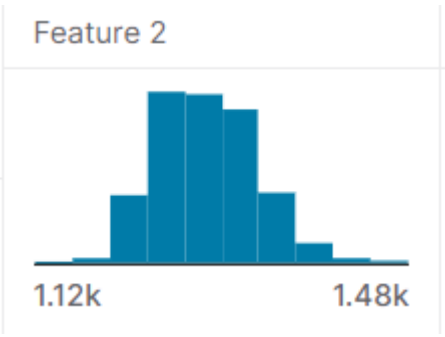

Рисунок 2.3

Также посмотрим на долю представителей из каждой группы риска (рисунок 2.4).

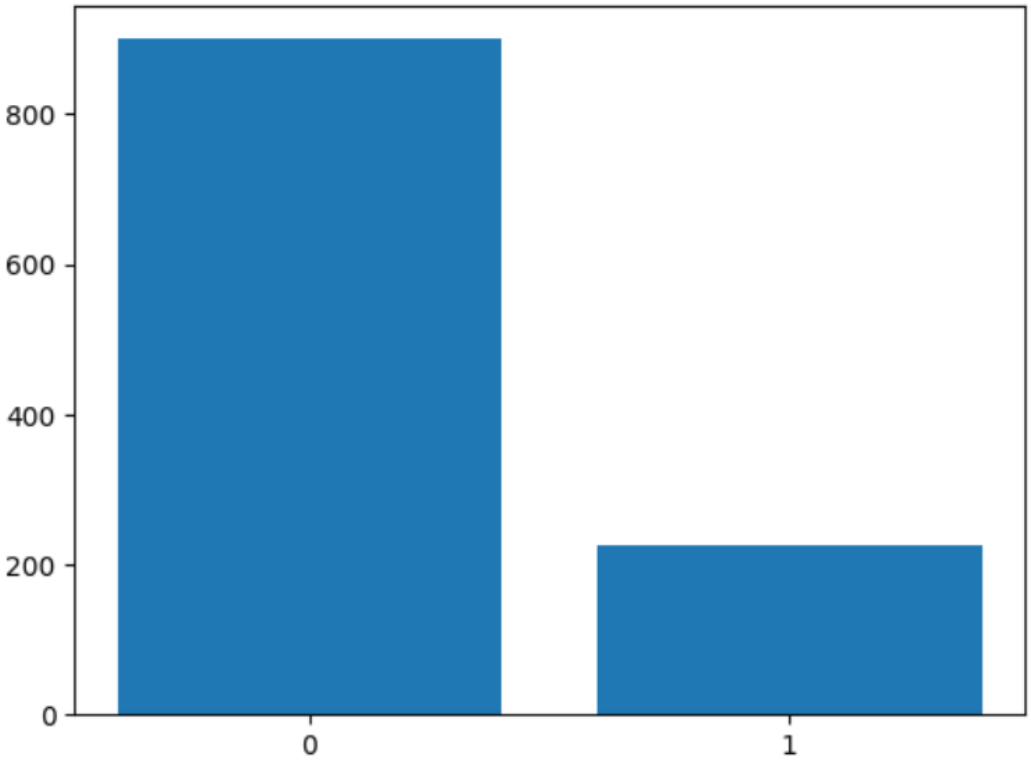

Рисунок 2.4

Из-за такого сильного неравенства модель плохо научится распознавать людей с высокой группой риска, что плохо скажется на результатах теста.

Чтобы решить эту проблему, можно использовать один из алгоритмов передискретизации (англ. oversampling). Один из них называется ADASYN. Он разбивает значения на области методом k ближайших соседей и добавляет в них представителей меньшинства, пока соотношение обоих представителей не будет

равным. Новые элементы получаются, как правило, взятием средних значений двух элементов и добавлением шума [13]. Для оценки эффективности этого метода построим также классификатор, не используя сгенерированные данные.

После небольшой подготовки можно перейти к созданию решающих деревьев классификации.

2.3 Построение деревьев

2.3.1 Дерево по начальным данным

Разобьем дата фрейм на параметрические значения и ключевые (x и y), а затем сделаем разбиение на обучающую и тестовую выборки и построим дерево (рисунок 2.5). При использовании каждой функции, подразумевающей генерацию случайных чисел, в нее будет передаваться параметр random_state равный 14 для возможности воспроизведения полученного результата.

```
# doing a train/test split
x1_train, x1_test, y1_train, y1_test = train_test_split(x, y, test_size=0.20, random_state=14)
# building a tree
clf1 = DecisionTreeClassifier(random_state=14)
clf1.fit(x1_train, y1_train)
plot_tree(clf1)
plt.show()
```

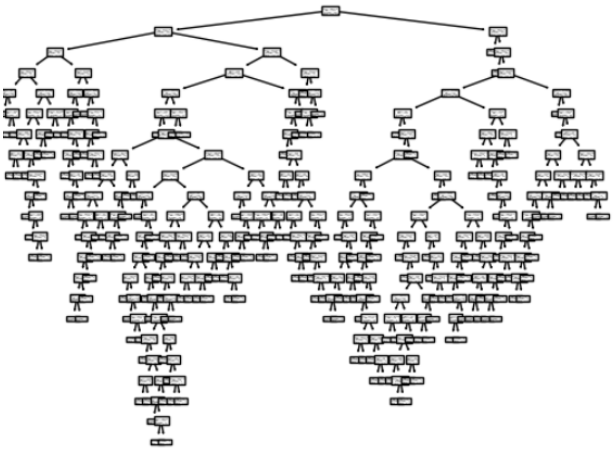

Рисунок 2.5

Посмотрим на отчет о тестировании и матрицу ошибок, а также на глубину дерева и количество узлов (рисунок 2.6).

```
print(f"tree depth: {clf1.get_depth()}, \namount of nodes: {clf1.get_n_leaves()}")
```

```
tree depth: 21,
amount of nodes: 190
```

```
print(classification_report(y1_test, clf1.predict(x1_test)))
```

	precision	recall	f1-score	support
0	0.81	0.79	0.80	178
1	0.26	0.28	0.27	47
accuracy			0.68	225
macro avg	0.53	0.53	0.53	225
weighted avg	0.69	0.68	0.69	225

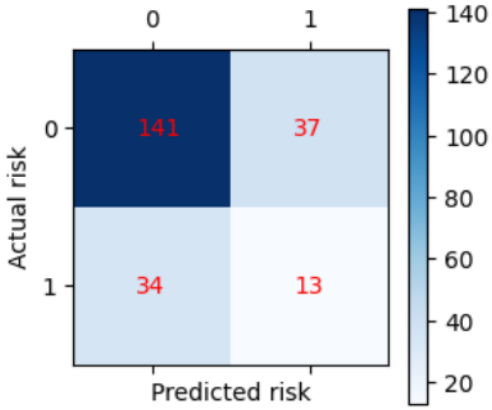

Рисунок 2.6

На данном рисунке самыми важными являются 3 числа: accuracy и f1-score каждого из классов. Все они принимают значение от 0 до 1, чем больше, тем лучше работает модель. Критерий accuracy находится как отношение верно классифицированных объектов к их общему количеству: $\frac{141+13}{141+37+34+13} = 0.68$. Критерий f1-score для каждого класса отдельный. Он отражает точность классификации определенной группы. Его можно найти по формуле $\frac{2TP}{2TP+FP+FN}$, где TP — верно классифицированные представители этой группы, FN — неверно классифицированные представители этой группы, FP — представители других групп, отнесенные к этой. Для класса 0 f1-score равен $\frac{2 \cdot 141}{2 \cdot 141+37+34} = 0.8$, а для 1 — $\frac{2 \cdot 13}{2 \cdot 13+37+34} = 0.27$.

Как и предполагалось, общая оценка классификации неплоха, но количество ошибок при опознавании клиентов с категорией риска 1 значительно больше приемлемых значений.

2.3.2 Дерево по обработанным данными

Прогоним датасет через алгоритм ADASYN и построим новое дерево по этим данным (рисунок 2.7).

```
# training a decision tree model

clf2 = DecisionTreeClassifier(random_state=14)
clf2.fit(x2_train, y2_train)
plot_tree(clf2)
plt.show()
```

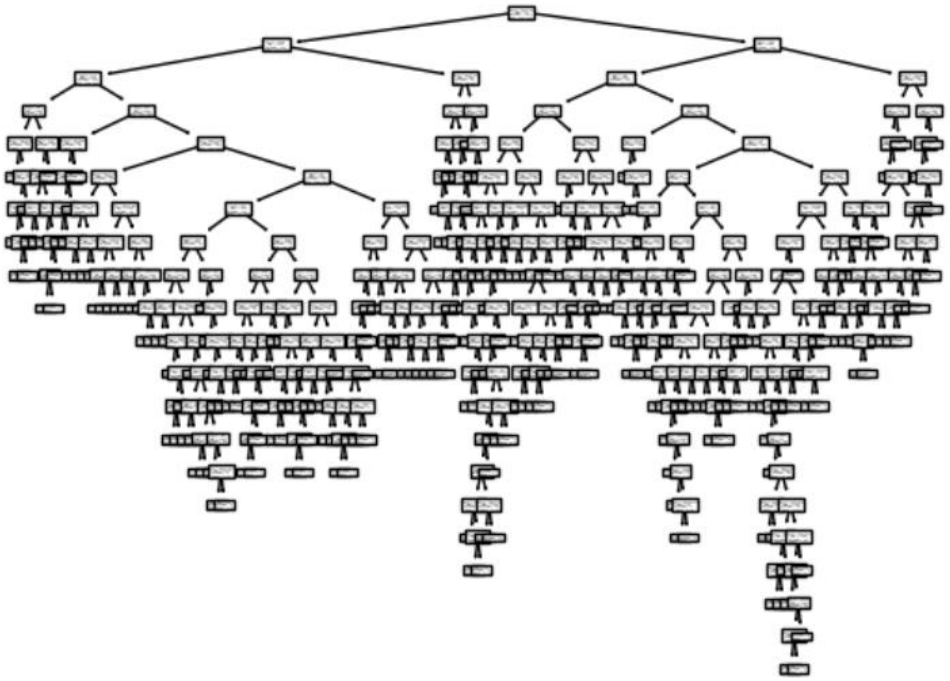

Рисунок 2.7

Оценим новое дерево (рисунок 2.8).

```
print(f"tree depth: {clf2.get_depth()}\namount of nodes: {clf2.get_n_leaves()}")
```

```
tree depth: 20
amount of nodes: 254
```

```
print(classification_report(y2_test, clf2.predict(x2_test)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.72      0.76       180
           1       0.74      0.81      0.78       178

    accuracy                           0.77       358
   macro avg       0.77      0.77      0.77       358
weighted avg       0.77      0.77      0.77       358
```

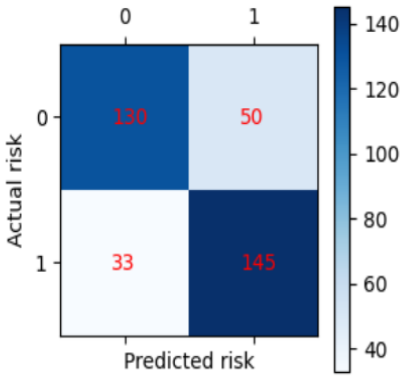

Рисунок 2.8

Несмотря на то, что f1-score на группе 0 упал, результат получился хорошим, так как на группе 1 он сильно возрос. Вместе с этим заметно увеличился и размер дерева, с 190 до 254 узлов. Чтобы это компенсировать, а также предотвратить переобучение, стоит произвести обрезку дерева.

2.3.3 Обрезка полученного дерева

Для начала необходимо рассчитать эффективные альфа для всех узлов. Это можно сделать при помощи метода классификатора cost_complexity_pruning_path. Из него также можно получить соответствующие значения неточности и отобразить их зависимости на графике (рисунок 2.9).

```
# finding efficient alpha values

path = clf2.cost_complexity_pruning_path(x2_train, y2_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
ccp_alphas[:5], impurities[:5], len(ccp_alphas)
```

```
(array([0.        , 0.00052411, 0.00052411, 0.00055905, 0.00058234]),
 array([0.        , 0.00104822, 0.00209644, 0.00321454, 0.00437922]),
 128)
```

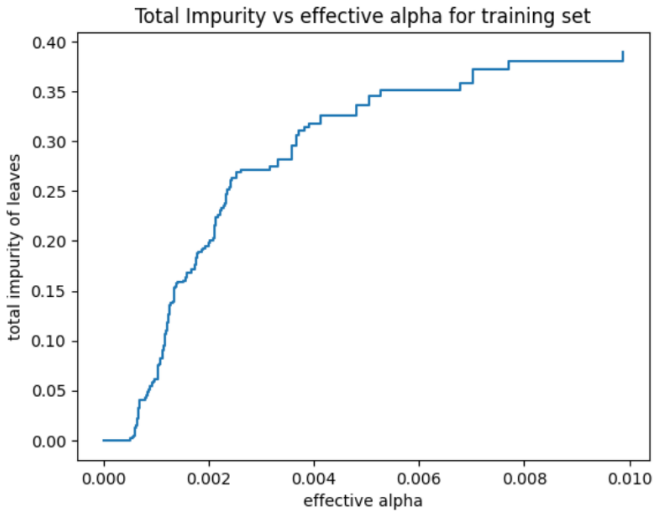

Рисунок 2.9

Посмотрим, как изменяется оценка классификации обучающей и тестовой выборок с изменением альфа (рисунок 2.10).

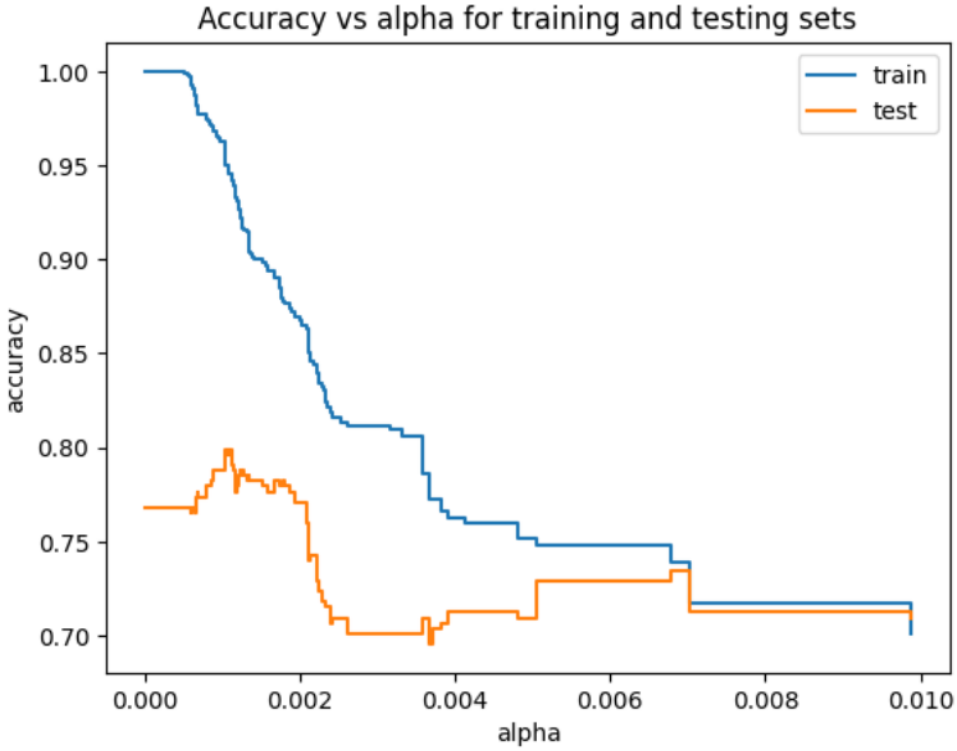

Рисунок 2.10

Легко заметить, что оптимальное значение примерно равно 0.001, так как при нем возрастает оценка тестирования, а оценка на тренировочных данных еще не успевает упасть.

Это значение возможно получить при помощи класса GridSearchCV библиотеки sklearn (рисунок 2.11). Он находит оптимальное значение методом перебора, что занимает значительное время.

```
# using GridSearchCV to look for the best value of ccp_alpha
alpha_grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=14),
                                 scoring=make_scorer(accuracy_score),
                                 param_grid=ParameterGrid({'ccp_alpha': [[a] for a in ccp_alphas]}))
```

```
alpha_grid_search.fit(x2_train, y2_train)
```

```
# the best value
alpha_grid_search.best_params_['ccp_alpha']
```

```
0.0010482180293501049
```

Рисунок 2.11

Теперь построим дерево на тех же данных, но с заданным параметром ccp_alpha (рисунок 2.12), и посмотрим на оценку тестирования (рисунок 2.13).

```
# building pruned tree
clf3 = DecisionTreeClassifier(random_state=14, ccp_alpha=alpha_grid_search.best_params_['ccp_alpha'])
clf3.fit(x2_train, y2_train)
plot_tree(clf3)
plt.show()
```

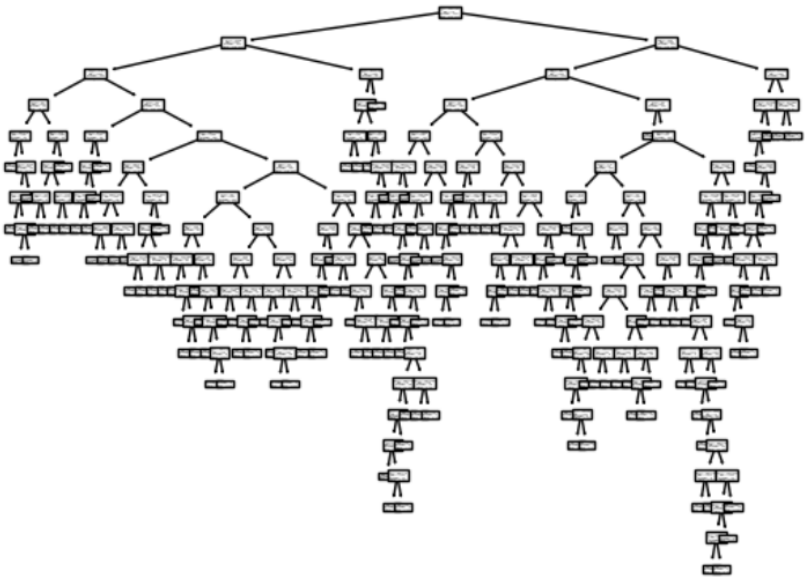

Рисунок 2.12

```
print(f"tree depth: {clf3.get_depth()}\namount of nodes: {clf3.get_n_leaves()}")
```

```
tree depth: 18
amount of nodes: 153
```

```
print(classification_report(y2_test, clf3.predict(x2_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.80       180
           1       0.79      0.81      0.80       178

    accuracy                           0.80       358
   macro avg       0.80      0.80      0.80       358
weighted avg       0.80      0.80      0.80       358
```

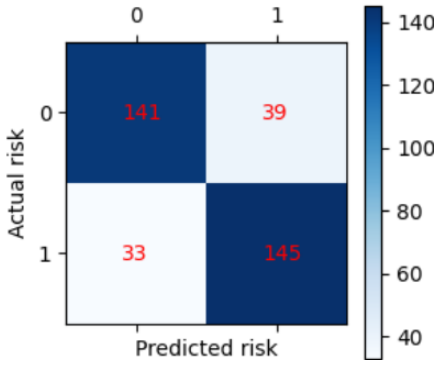

Рисунок 2.13

Конечное дерево получилось меньше необрезанного, а также все его показатели точности возросли.

Таким образом, в ходе работы были получены 3 классификатора, построение которых потребовало различного количества времени и знаний, но в итоге показало соответствующие результаты.

Заключение

Деревья решений являются ценным и широко используемым инструментом для решения различных задач. Они предлагают несколько преимуществ, включая возможность обработки как категорийных, так и числовых данных, интерпретируемость и т. д. Деревья решений могут быть построены по алгоритму CART с использованием рекурсивного разделения, когда данные разбиваются на все более мелкие подмножества на основе индекса Джини. Однако у деревьев решений есть некоторые минусы, например склонность к переобучению данных, что может привести к плохому обобщению и неточным прогнозам. Чтобы устранить это ограничение, были разработаны различные методы, один из которых — это подрезка крайних вершин.

В целом, понимание принципов построения дерева решений и методов устранения их ограничений может позволить людям эффективно использовать этот инструмент и принимать более обоснованные решения. Деревья решений можно использовать для решения самых разных проблем, включая анализ решений, классификацию и задачи регрессии. Поскольку область машинного обучения и науки о данных продолжает развиваться, деревья решений останутся важным и универсальным инструментом для решения сложных задач.

Список использованных источников

1) Учебное пособие РТУ МИРЭА https://online-edu.mirea.ru/mod/resource/view.php?id=444019 (дата обращения 30.04.23)

2) Учебное пособие РТУ МИРЭА https://online-edu.mirea.ru/mod/resource/view.php?id=441333 (дата обращения 30.04.23)

3) Учебное пособие РТУ МИРЭА https://online-edu.mirea.ru/mod/resource/view.php?id=444026 (дата обращения 30.04.23)

4) Википедия: Дерево (теория графов) https://ru.wikipedia.org/?curid=56079&oldid=127098718 (дата обращения 30.04.23)

5) Википедия: Дерево решений https://ru.wikipedia.org/?curid=633393&oldid=123848527 (дата обращения 30.04.23)

6) Википедия: CART (алгоритм) https://ru.wikipedia.org/?curid=4544102&oldid=122100468 (дата обращения 30.04.23)

7) Документация библиотеки scikitlearn: DecisionTreeClassifier https://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (дата обращения 30.04.23)

8) Документация библиотеки scikitlearn: алгоритмы деревьев https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart (дата обращения 30.04.23)

9) Открытый источник Medium: Build Better Decision Trees With Pruning https://towardsdatascience.com/build-better-decision-trees-with-pruning-8f467e73b107 (дата обращения 30.04.23)

10) Открытый источник Medium: Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning https://towardsdatascience.com/decision-

trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c (дата обращения 30.04.23)

11) Чат-бот ChatGPT https://chat.openai.com/

12) Kaggle: Credit Risk Classification Dataset https://www.kaggle.com/datasets/praveengovi/credit-risk-classification-dataset (дата обращения 30.04.23)

13) Открытый источник Medium: Fixing Imbalanced Datasets: An Introduction to ADASYN (with code!) https://medium.com/@ruinian/an-introduction-to-adasyn-with-code-1383a5ece7aa (дата обращения 30.04.23)

Приложение

Исходный код можно так же найти по ссылке https://github.com/ksnkh/decision-trees-coursework/blob/main/decision%20tree%20classifier.ipynb

```
# installing libraries
! pip install pandas
! pip install matplotlib
! pip install imbalanced-learn
# importing all the nessesary tools
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, ParameterGrid
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix, make_scorer, accuracy_score
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import ADASYN
def plot_cmatrix(test, pred):
    cmat = confusion_matrix(test, pred)
    fig = plt.figure(figsize=(3, 3))
    plt.matshow(cmat, cmap=plt.cm.Blues, fignum=1)
    plt.yticks(range(2))
    plt.xticks(range(2))
    plt.colorbar()
    plt.xlabel('Predicted risk')
    plt.ylabel('Actual risk')
    for i in range(2):
        for j in range(2):
            plt.text(i - 0.1, j+0.05, str(cmat[j, i]), color='red')
```

```
# converting csv file to pandas dataframe
customer_data = pd.read_csv("./data/customer_data.csv")
customer_data.describe()
# amount of Nan in each column
print(customer_data.isnull().sum())
# filling out all Nans in "fea_2" with its mean value
customer_data['fea_2'].fillna(customer_data['fea_2'].mean(), inplace=True)
# complete dataset
# key element is 'label'. it divides all customers into low(0) and high(1) credit risk groups
customer_data.describe()
x, y = customer_data.iloc[:, 2:], customer_data.iloc[:, 0]
# doing a train/test split
x1_train, x1_test, y1_train, y1_test = train_test_split(x, y, test_size=0.20, random_state=14)
# building a tree
clf1 = DecisionTreeClassifier(random_state=14)
clf1.fit(x1_train, y1_train)
plot_tree(clf1)
plt.show()
print(f"tree depth: {clf1.get_depth()}, \namount of nodes: {clf1.get_n_leaves()}")
print(classification_report(y1_test, clf1.predict(x1_test)))
print(f'test score: {clf1.score(x1_train, y1_train)}')
plot_cmatrix(y1_train, clf1.predict(x1_train))
print(f'test score: {clf1.score(x1_test, y1_test)}')
plot_cmatrix(y1_test, clf1.predict(x1_test))
print(f'test score: {clf1.score(x, y)}')
plot_cmatrix(y, clf1.predict(x))
# amount of risk group 1 and 0 are very different
```

```
distribution = pd.DataFrame({'Risk group': customer_data["label"].value_counts().index, 'Count': customer_data["label"].value_counts().values})
plt.bar(distribution['Risk group'], distribution['Count'], )
plt.xticks(distribution.iloc[:, 0])
plt.show()
# using ADASYN
adasyn = ADASYN(random_state=14)
x2, y2 = adasyn.fit_resample(x, y)
# new distribution is almost even
distribution = pd.DataFrame({'Risk group': y2[:].value_counts().index,
                             'Count': y2[:].value_counts().values
                            })
plt.bar(distribution['Risk group'], distribution['Count'], )
plt.xticks(distribution.iloc[:, 0])
plt.show()
# doing a  train/test split on new dataset
x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size=0.20, stratify=y2, random_state=14)
x2_train.describe()
# training a decision tree model
clf2 = DecisionTreeClassifier(random_state=14)
clf2.fit(x2_train, y2_train)
plot_tree(clf2)
plt.show()
print(f"tree depth: {clf2.get_depth()}\namount of nodes: {clf2.get_n_leaves()}")
print(classification_report(y2_test, clf2.predict(x2_test)))
print(f'train score: {clf2.score(x2_train, y2_train)}')
plot_cmatrix(y2_train, clf2.predict(x2_train))
print(f'test score: {clf2.score(x2_test, y2_test)}')
plot_cmatrix(y2_test, clf2.predict(x2_test))
```

```
print(f'original database score: {clf2.score(x, y)}')
plot_cmatrix(y, clf2.predict(x))
# finding efficient alpha values
path = clf2.cost_complexity_pruning_path(x2_train, y2_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
ccp_alphas[:5], impurities[:5], len(ccp_alphas)
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
# building trees with different ccp_alpha values, which we have gotten before
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=14, ccp_alpha=ccp_alpha)
    clf.fit(x2_train, y2_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
# getting rid of the last value because there is no split
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]
# checking out dependency of some parameters from ccp_alpha
node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alphas, node_counts, drawstyle="steps-post")
ax[0].set_xlabel("alpha")
```

```
        ax[0].set_ylabel("number of nodes")
        ax[0].set_title("Number of nodes vs alpha")
        ax[1].plot(ccp_alphas, depth, drawstyle="steps-post")
        ax[1].set_xlabel("alpha")
        ax[1].set_ylabel("depth of tree")
        ax[1].set_title("Depth vs alpha")
        fig.tight_layout()
        #comparing train and test scores with different ccp_alpha
        train_scores = [clf.score(x2_train, y2_train) for clf in clfs]
        test_scores = [clf.score(x2_test, y2_test) for clf in clfs]
        fig, ax = plt.subplots()
        ax.set_xlabel("alpha")
        ax.set_ylabel("accuracy")
        ax.set_title("Accuracy vs alpha for training and testing sets")
        ax.plot(ccp_alphas, train_scores, label="train", drawstyle="steps-post")
        ax.plot(ccp_alphas, test_scores, label="test", drawstyle="steps-post")
        ax.legend()
        plt.show()
        # using GridSearchCV to look for the best value of ccp_alpha
        alpha_grid_search =
GridSearchCV(estimator=DecisionTreeClassifier(random_state=14),
scoring=make_scorer(accuracy_score), param_grid=ParameterGrid({'ccp_alpha': [[a]
for a in ccp_alphas]}))
        alpha_grid_search.fit(x2_train, y2_train)
        # the best value
        alpha_grid_search.best_params_['ccp_alpha']
        # building pruned tree
        clf3 = DecisionTreeClassifier(random_state=14,
ccp_alpha=alpha_grid_search.best_params_['ccp_alpha'])
        clf3.fit(x2_train, y2_train)
        plot_tree(clf3)
```

```
plt.show()
print(f"tree depth: {clf3.get_depth()}\namount of nodes: {clf3.get_n_leaves()}")
print(classification_report(y2_test, clf3.predict(x2_test)))
print(f'train score: {clf3.score(x2_train, y2_train)}')
plot_cmatrix(y2_train, clf3.predict(x2_train))
print(f'test score: {clf3.score(x2_test, y2_test)}')
plot_cmatrix(y2_test, clf3.predict(x2_test))
print(f'original database score: {clf3.score(x, y)}')
plot_cmatrix(y, clf3.predict(x))
```