Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования «Рыбинский государственный

авиационный технический университет имени П. А. Соловьева»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ УПРАВЛЕНИЯ

Математического и программного обеспечения электронных

вычислительных средств

ОТЧЁТ

по дисциплине:

«Методы и алгоритмы анализа данных»

на тему:

«Решающие деревья»

Выполнил: студент группы ИВМ-24 Морозов А. А.

Руководитель: ассистент Вязниковцев Д. А.

Рыбинск 2024

Содержание

[Цель работы 3](#_Toc183623536)

[1 Первичная обработка данных 4](#_Toc183623537)

[2 Критерий ошибки 7](#_Toc183623538)

[3 Решающее дерево 12](#_Toc183623539)

[Вывод 15](#_Toc183623540)

# Цель работы

Целями данной лабораторной работы являются:

- изучение теории о решающих деревьях;

- первичная обработка данных;

- построение решающих деревьев.

# 1 Первичная обработка данных

В начале необходимо загрузить датасет о физических упражнениях и их эффективности *linnerud* из библиотеки *sklearn*:

def preparation():

linnerud = load\_linnerud()

print(linnerud.DESCR)

print(f"Ключи датасета: {linnerud.keys()}")

print(f"D.keys() -> a set-like object providing a view on D's keys")

print(f"Признаки датасета: {linnerud.feature\_names, linnerud.target\_names}")

# X: pd.DataFrame = linnerud.data

X = pd.DataFrame(data=linnerud.data, columns=linnerud.feature\_names)

print(X)

print(f"Последние 5 значений X:\n{X[-5:]}")

print(f"Размер X: {len(X)}")

y = linnerud.target

print(f"Последние 5 значений y:\n{y[-5:]}")

print(f"Размер y: {len(y)}")

return linnerud, X, y

После выполнения кода были созданы *dataframe* и *ndarray* и в консоли отобразилась некоторая базовая информация о датасете.

Выведем график распределения целевой переменной (рисунок 1):

def raspredelenie(y):

plt.title('... distribution')

plt.xlabel('...')

plt.ylabel('# samples')

plt.hist(y, bins=20)

plt.show()

Изображение выглядит как текст, снимок экрана, диаграмма, График

Автоматически созданное описание

Рисунок 1.1 – График распределения целевой переменной

Теперь разобьём данные на тестовую и обучающую выборки в соотношении 1 к 4:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=13)

В данных выборках у *X* и *y* по 20 записей в каждом. Каждая запись содержит 3 числа.

Теперь можно приступать к построению решающего дерева.

# 2 Критерий ошибки

Для построения оптимального решающего дерева необходимо исходить из того, что критерий ошибки должен стремится к 0:

|  |  |
| --- | --- |
|  | (2.1) |

Для расчёта критерия ошибки воспользуемся кодом:

def H(R, y):

return y[R.index].var(ddof=0)

def split\_node(R\_m, feature, t):

left = R\_m[R\_m[feature] <= t]

right = R\_m[R\_m[feature] > t]

return left, right

def q\_error(R\_m, feature, t, y):

left, right = split\_node(R\_m, feature, t)

return (len(left) / len(R\_m) \* H(left,y)+len(right)/len(R\_m)\* H(right, y))

С помощью данных функций получим критерий ошибки для какого-либо признака. Используя его, можно найти оптимальное разбиение вершины по заданному признаку:

def zadanie\_2\_3(X\_train, y):

feature = 'Chins'

feature\_values = np.unique(X\_train[feature])

print(feature\_values, len(feature\_values))

Q\_array = list(map(lambda x: q\_error(X\_train, feature, x, y), feature\_values))

print(Q\_array)

nan\_value = feature\_values[np.where(np.isnan(Q\_array))]

plt.figure(figsize=(10, 6))

plt.plot(feature\_values, Q\_array, marker='o', linestyle='-')

plt.xlabel('Порог')

plt.ylabel('Значение ошибки')

plt.title(f'Feature {feature}')

plt.grid(True)

plt.show()

return feature, nan\_value

После выполнения кода получим график критерия ошибки в зависимости от порога признака (рисунок 2).

Изображение выглядит как линия, диаграмма, График, текст

Автоматически созданное описание

Рисунок 2.1 – График критерия ошибки признака *Chins*

Теперь найдём признак с наименьшим значением ошибки:

def zadanie\_2\_4(X\_train):

results = []

for f in X\_train.columns:

t, Q\_array = get\_optimal\_split(X\_train, f, y)

# print(t, Q\_array, Q\_array[np.argmin(Q\_array)])

results.append((f, t, Q\_array[np.argmin(Q\_array)]))

results = sorted(results, key=lambda x: x[2])

print(f"Результаты 2.4: {results}")

results\_df = pd.DataFrame(results, columns=['feature', 'optimal t', 'min Q error'])

optimal\_feature, optimal\_t, optimal\_error = results[0]

print(results\_df)

\_, optimal\_Q\_array = get\_optimal\_split(X\_train, optimal\_feature, y)

plt.figure(figsize=(10, 6))

# print(X\_train[optimal\_feature], optimal\_Q\_array)

# print(np.unique(X\_train[optimal\_feature]))

# print(nan\_value)

plt.plot(np.delete(np.unique(X\_train[optimal\_feature]),

np.where(np.unique(X\_train[optimal\_feature]) == nan\_value)[0][0]), optimal\_Q\_array, marker='o',

linestyle='-')

plt.xlabel('Порог')

plt.ylabel('Значение ошибки')

plt.title(f'Feature {feature}')

plt.grid(True)

plt.show()

return optimal\_feature, optimal\_t, optimal\_error, X\_train

В результате выполнения кода выше в консоли выведутся строчки изображённые на рисунке 3.

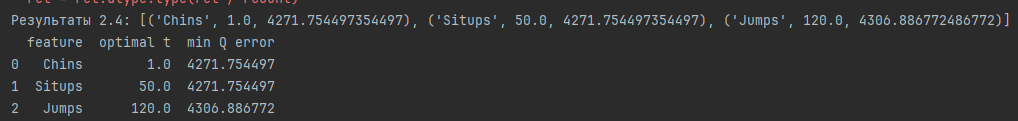


Рисунок 2.2 – Вывод в консоль

Из вывод в консоль следует то, что признак *Chins* со значением разбиения 1 имеет наименьшую ошибку. График критерия ошибки для признака *Chins* изображён на рисунке 2.

Изобразим разбиение визуально (рисунок 4). Для этого построим диаграмму рассеяния целевой переменной в зависимости от значения найденного признака. Далее изобразим вертикальную линию, соответствующую порогу разбиения.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рисунок 2.3 – График разбиения

# 3 Решающее дерево

Создадим решающее дерево и выведем (рисунок 5) его с помощью кода:

def zadanie\_3\_1(X\_train, X\_test, y\_train, y\_test):

from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(max\_depth=3, random\_state=13)

dt.fit(X\_train, y\_train)

from sklearn.tree import plot\_tree

plot\_tree(dt, feature\_names=X.columns, filled=True, rounded=True)

plt.show()

return dt

Изображение выглядит как текст, рукописный текст, Шрифт, линия

Автоматически созданное описание

Рисунок 3.1 – Решающее дерево

При выполнении кода с рисунка 6 будут выводиться графики (рисунок 7) с зависимостью значения ошибки от количества слоёв дерева и минимального количества примеров в листе.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 3.2 – Код для оценки качества обучения

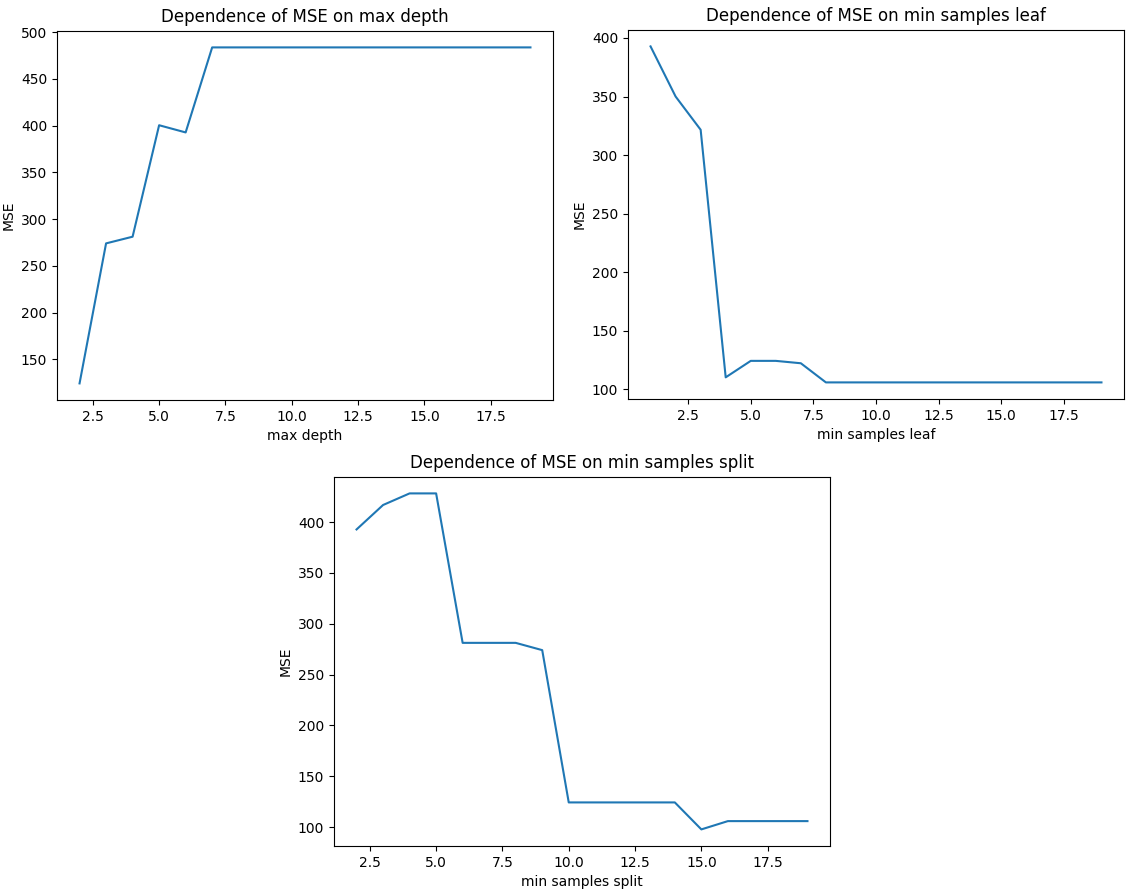


Рисунок 3.3 – Графики оценки качества обучения

# Вывод

Таким образом, построение решающего дерева – задача нахождения минимального критерия ошибки и оптимизации множества других параметров.

Решающие деревья являются мощным инструментом для решения задач классификации и регрессии. Однако для повышения их эффективности в реальных задачах рекомендуется использовать ансамблевые методы, такие как случайные леса (*Random Forest*) или градиентный бустинг (*Gradient Boosting*), которые уменьшают недостатки одиночных деревьев.

Эта работа продемонстрировала важность грамотной настройки гиперпараметров и визуального анализа модели для достижения высокого качества предсказаний.