ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)» (МАИ)

Направление подготовки: 27.03.05 «Инноватика»

**Лабораторная работа №2**

по дисциплине «Спецглавы высшей математики»

Выполнила:

Студент гр. М3О-234Б-22

Корешников Н.А.

Преподаватель:

Александрова С.С.

Москва 2023

Оглавление

[Цель работы: 3](#_Toc153650529)

[Ссылка на гитхаб: 3](#_Toc153650530)

[Шаг 1. Откройте файл с данными и изучите общую информацию 3](#_Toc153650531)

[Шаг 2. Подготовьте данные 4](#_Toc153650532)

[Шаг 3. Проведите исследовательский анализ данных 7](#_Toc153650533)

[Шаг 4. Составьте портрет пользователя каждого региона 15](#_Toc153650534)

[Вывод: 19](#_Toc153650535)

# **Цель работы**: изучить принципы работы и применение деревьев решений в машинном обучении, а также овладеть базовыми навыками их реализации.

# Начало формы

# Ссылка на гитхаб: https://github.com/Ling205/math\_d8/tree/main/lab2

# Назначение и основные принципы работы деревьев решений

Дерево решений — тип контролируемого машинного обучения, который в основном используется в задачах классификации. Дерево решений само по себе — это в основном жадное, нисходящее, рекурсивное разбиение. «Жадное», потому что на каждом шагу выбирается лучшее разбиение. «Сверху вниз» — потому что мы начинаем с корневого узла, который содержит все записи, а затем делается разбиение.

Основные назначения деревьев решений:

Область применения деревья решений в настоящее время широка, но все задачи, решаемые этим аппаратом, могут быть объединены в следующие три класса:

* Описание данных: Деревья решений позволяют хранить информацию о данных в компактной форме, вместо них мы можем хранить дерево решений, которое содержит точное описание объектов.
* Классификация: Деревья решений отлично справляются с задачами классификации, т.е. отнесения объектов к одному из заранее известных классов. Целевая переменная должна иметь дискретные значения.
* Регрессия: Если целевая переменная имеет непрерывные значения, деревья решений позволяют установить зависимость целевой переменной от независимых(входных) переменных. Например, к этому классу относятся задачи численного прогнозирования (предсказания значений целевой переменной).

Деревья решений также могут иметь недостатки, такие как переобучение, особенно при большой глубине дерева, что может привести к низкой обобщающей способности модели. В таких случаях обычно используются методы регуляризации, такие как отсечение дерева или использование ансамблевых методов, например, случайного.

Основные принципы работы деревьев решений:

В ходе построения дерева решений нужно решить несколько основных проблем, с каждой из которых связан соответствующий шаг процесса обучения:

1. Выбор атрибута разбиения:

- Теоретико-информационный критерий использует энтропию для измерения неоднородности классов в узле.

- Статистический подход использует индекс Джини для оценки неоднородности классов в узле.

- Выбор атрибута осуществляется на основе максимизации прироста информации или минимизации индекса Джини.

2. Критерий остановки:

- Ранняя остановка предлагает прекратить построение дерева при достижении определенного критерия, например, определенной точности классификации.

- Отсечение ветвей предлагает построить полное дерево и затем упростить его для достижения баланса между сложностью и точностью.

3. Отсечение ветвей:

- Включает построение полного дерева, измерение его точности и отсечение ветвей для улучшения обобщающей способности.

4. Извлечение правил:

- Позволяет преобразовать сложное дерево в набор правил для лучшей интерпретации.

- Каждый путь от корня к листу представляет собой правило.

5. Оценка точности:

- Включает измерение точности построенного дерева на тестовых данных.

# Реализация кода из статьи 1

**1. Импортируем библиотеки.**

**import** numpy **as** np

**import** pandas **as** pd

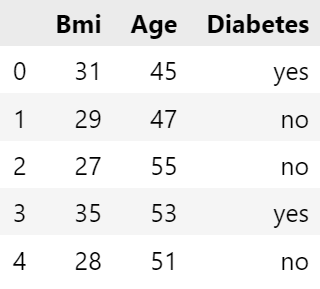
**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**2. Загрузим данные.**

df**=**pd**.**read\_csv("Diabetes1.csv")

df**.**head()



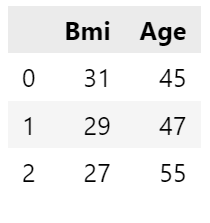
**3. Разделим переменные на x и y.**

x**=**df**.**iloc[:,:2]

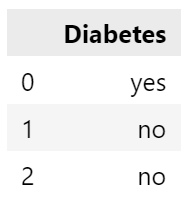
y**=**df**.**iloc[:,2:]

In [4]:

x**.**head(3)



y**.**head(3)



**4. Построим модель с помощью sklearn.**

**from** sklearn **import** tree

model**=**tree**.**DecisionTreeClassifier(criterion**=**"entropy")

model**.**fit(x,y)

Вывод:

DecisionTreeClassifier(criterion='entropy')

**5. Оценка модели**

model**.**score(x,y)

Вывод: 1.0

**6. Прогнозирование с помощью модели**

model**.**predict([[29,47]])

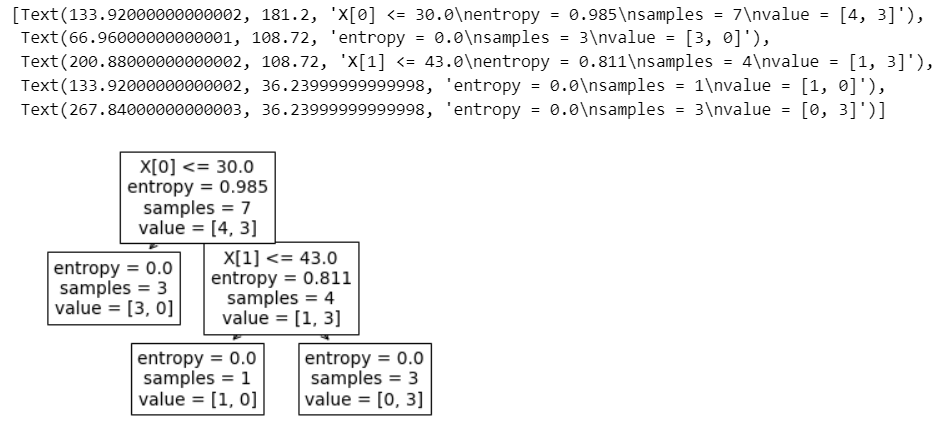
Вывод: array(['no'], dtype=object)

model**.**predict([[45,47]])

Вывод: array(['yes'], dtype=object)

**7. Визуализация модели:**

tree**.**plot\_tree(model)



# Реализация кода из статьи 2 с пояснением

Реализация дерева решений без библиотек.

Два основных класса:

1. *Узел* — отображает отдельный узел дерева.

2. *DecisionTree* — для реализации алгоритма

 Класс *Node. Вспомогательный класс, реализующий один узел дерева.*

|  |  |
| --- | --- |
|  | class Node: |
|  | def \_\_init\_\_(self, feature=None, threshold=None, data\_left=None, data\_right=None, gain=None, value=None): |
|  | self.feature = feature |
|  | self.threshold = threshold |
|  | self.data\_left = data\_left |
|  | self.data\_right = data\_right |
|  | self.gain = gain |
|  | self.value = value |

Вот список всех методов, которые мы будем реализовывать в следующем классе *DecisionTree* :

**\_\_init\_\_()** — функция-конструктор. Это инициализирует значения наших гиперпараметров *min\_samples\_split* и *max\_depth.* Первый используется для определения количества выборок, которое требуется для дальнейшего разделения узла, а последний определяет глубину. дерева решений, которое мы строим. Эти гиперпараметры действуют как условия нарушения рекурсивной функции построения дерева.

**\_entropy(s)***–*определяет нечистоту конкретного вектора.

**\_information\_gain(parent, left\_child, right\_child)***—* чтобы найти прирост информации при разделении узла на дочерние узлы.

**\_best\_split(X, y)***—*самая важная функция; тот, который определяет наилучшие параметры разделения. Он использует входные функции *X* и целевую переменную *y* для поиска оптимальных значений.

**\_build(X, y, depth)** — основной метод. Он строит дерево с рекурсивными вызовами узлов разделения до тех пор, пока не будут выполнены условия останова, как мы описали в гиперпараметрах.

**fit(X, y)** — используется для вызова метода *\_build()* для сохранения обновленного дерева в конструкторе после каждой итерации.

**\_predict(x)** — для прогнозирования тестового набора данных путем обхода всего дерева и преобразования входных данных в выходные.

**predict(X)** — при наличии матрицы входных признаков *X* эта функция применяет *\_predict()* для каждой записи в этой матрице.

**Код методов представлен ниже в классе:**

 Класс *DecisionTree* — для реализации алгоритма

class DecisionTree:

def \_\_init\_\_(self, min\_samples\_split=2, max\_depth=5):

self.min\_samples\_split = min\_samples\_split

self.max\_depth = max\_depth

self.root = None

@staticmethod

def \_entropy(s):

counts = np.bincount(np.array(s, dtype=np.int64))

percentages = counts / len(s)

entropy = 0

for pct in percentages:

if pct > 0:

entropy += pct \* np.log2(pct)

return -entropy

def \_information\_gain(self, parent, left\_child, right\_child):

num\_left = len(left\_child) / len(parent)

num\_right = len(right\_child) / len(parent)

# One-liner which implements the previously discussed formula

return self.\_entropy(parent) - (num\_left \* self.\_entropy(left\_child) + num\_right \* self.\_entropy(right\_child))

def \_best\_split(self, X, y)

best\_split = {}

best\_info\_gain = -1

n\_rows, n\_cols = X.shape

for f\_idx in range(n\_cols):

X\_curr = X[:, f\_idx]

for threshold in np.unique(X\_curr):

df = np.concatenate((X, y.reshape(1, -1).T), axis=1)

df\_left = np.array([row for row in df if row[f\_idx] <= threshold])

df\_right = np.array([row for row in df if row[f\_idx] > threshold])

if len(df\_left) > 0 and len(df\_right) > 0:

y = df[:, -1]

y\_left = df\_left[:, -1]

y\_right = df\_right[:, -1]

gain = self.\_information\_gain(y, y\_left, y\_right)

if gain > best\_info\_gain:

best\_split = {

'feature\_index': f\_idx,

'threshold': threshold,

'df\_left': df\_left,

'df\_right': df\_right,

'gain': gain

}

best\_info\_gain = gain

return best\_split

def \_build(self, X, y, depth=0):

n\_rows, n\_cols = X.shape

if n\_rows >= self.min\_samples\_split and depth <= self.max\_depth:

# Get the best split

best = self.\_best\_split(X, y)

if best['gain'] > 0:

left = self.\_build(

X=best['df\_left'][:, :-1],

y=best['df\_left'][:, -1],

depth=depth + 1

)

right = self.\_build(

X=best['df\_right'][:, :-1],

y=best['df\_right'][:, -1],

depth=depth + 1

)

return Node(

feature=best['feature\_index'],

threshold=best['threshold'],

data\_left=left,

data\_right=right,

gain=best['gain']

)

return Node(

value=Counter(y).most\_common(1)[0][0]

)

def fit(self, X, y):

self.root = self.\_build(X, y)

def \_predict(self, x, tree):

if tree.value != None:

return tree.value

feature\_value = x[tree.feature]

if feature\_value <= tree.threshold:

return self.\_predict(x=x, tree=tree.data\_left)

if feature\_value > tree.threshold:

return self.\_predict(x=x, tree=tree.data\_right)

def predict(self, X):

return [self.\_predict(x, self.root) for x in X]

**Оценка модели.**

Импорт библиотек.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

df = pd.read\_csv('car.csv', header=None)

df.head()

Создание имен столбцов.

colnames = ['Buying-Price', 'Maintenance-Price', 'Doors', 'Capacity', 'Luggage-Boot', 'Safety', 'Class']

df.columns = colnames

df.head()

Выделение целевого класса.

from sklearn.preprocessing import OneHotEncoder

OneHotEncoder().fit\_transform(df)

X = df.drop(['Class'], axis=1)

y = df['Class']

Разделение данных на наборы данных для обучения и тестирования.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 42)

Следующее тренировачная часть. Создаем экземпляр класса, ранее написанный, а затем вывод метода fit(). После чего используем метод predict() для вывода прогнозов модели.

model = DecisionTree()

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

preds = model.predict(X\_test)

**Сравнение с Scikit-Learn**

from sklearn.tree import DecisionTreeClassifier

sk\_model = DecisionTreeClassifier()

sk\_model.fit(X\_train, y\_train)

sk\_preds = sk\_model.predict(X\_test)

accuracy\_score(y\_test, sk\_preds)

Сравнивая результаты обоих классов, получаем оценку в 94%.

Вывод:В ходе лабораторной работы мы сделали классификатор дерева решений с нуля. От загрузки набора данных до проверки производительности нашей модели каждая часть разработки модели было подробно написано. В итоге мы добились довольно хорошей точности нашего классификатора.