# **Решающие деревья**

# **Решающее дерево для классификации**

Для того, чтобы разобраться, как деревья строятся, посмотрим на набор данных из двух признаков (чтобы была возможность визуализации):

# Импорт необходимых модулей

import matplotlib

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Настройки для визуализации

# Если используется темная тема - лучше текст сделать белым

TEXT\_COLOR = 'black'

matplotlib.rcParams['figure.figsize'] = (15, 10)

matplotlib.rcParams['text.color'] = 'black'

matplotlib.rcParams['font.size'] = 14

matplotlib.rcParams['axes.labelcolor'] = TEXT\_COLOR

matplotlib.rcParams['xtick.color'] = TEXT\_COLOR

matplotlib.rcParams['ytick.color'] = TEXT\_COLOR

# Зафиксируем состояние случайных чисел

RANDOM\_STATE = 0

np.random.seed(RANDOM\_STATE)

from sklearn.datasets import make\_classification

X\_data, y\_data = make\_classification(

n\_samples=10,

n\_features=2,

n\_redundant=0,

n\_informative=1,

n\_clusters\_per\_class=1,

random\_state=RANDOM\_STATE

)

pnts\_scatter = plt.scatter(X\_data[:, 0], X\_data[:, 1], marker='o', c=y\_data, s=50, edgecolor='k', )

plt.xlabel('$x\_1$')

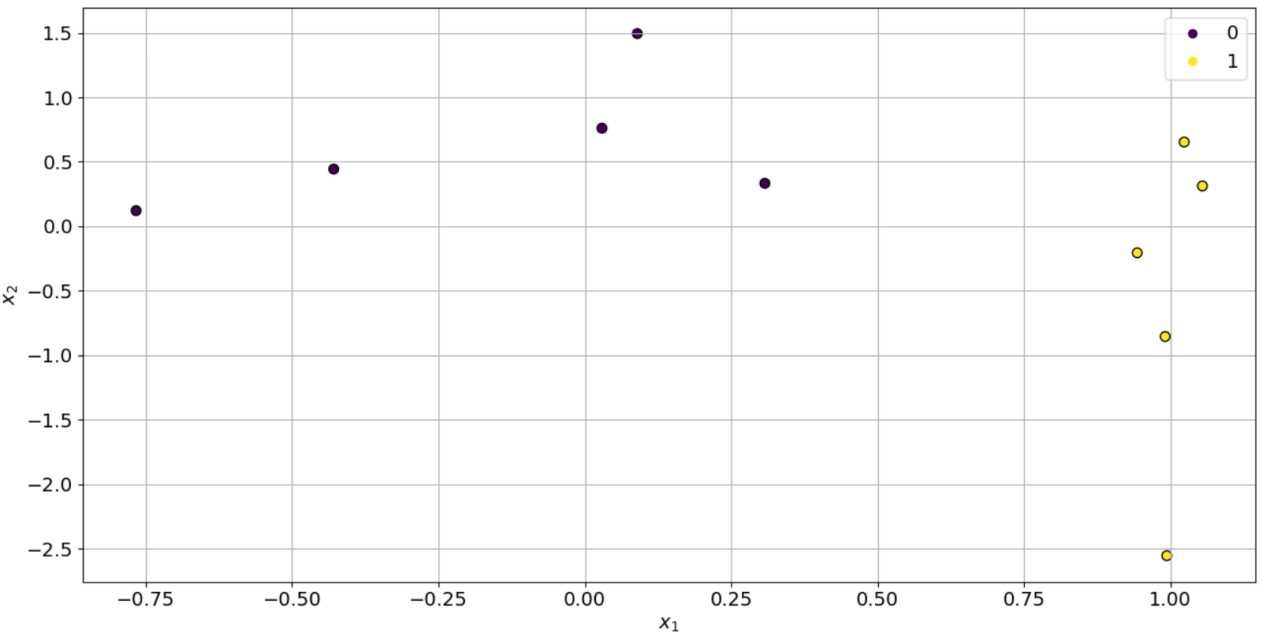
plt.ylabel('$x\_2$')

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

Результат



**Примеси Джини (Gini impurity)**

Давайте напишем реализацию этой функции:

# TODO - напишите реализацию функции вычисления Джини

def gini\_impurity(y):

if (len(y)==0):

return 0

p0=len(y[y==1])/len(y)

p1=len(y[y==0])/len(y)

gini=p0\*(1-p0)+p1\*(1-p1)

return gini

# TEST

y1 = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

y2 = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

y3 = np.array([0, 0, 1, 1, 1, 1, 1, 1, 1, 1])

y4 = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

assert gini\_impurity(y1) == 0.5

assert gini\_impurity(y2) == 0

assert gini\_impurity(y3) == 0.32

assert gini\_impurity(y4) == 0

assert gini\_impurity(np.array([])) == 0

Для чего нам нужен этот показатель? Суть решающего дерева заключается в том, что каждым узлом производится раздел пространства на части. То есть, если мы говорим, что узел разделяет по признаку IMG_256 с порогом 0.5, то все пространство правее линии IMG_257 становится классом 1, а все левее этой линии - классом 0. Для проверки напишем первый вариант функции предсказания и построим визуализацию решений модели.

# TODO

def predict\_v1(X):

# Напишите реализацию функции предсказания

# решающего дерева с одним узлом

# разделение по признаку (x1) с порогом 0.5

# \*Не забывайте о размерности данных X

y\_pred=np.zeros\_like(X[:,0])

y\_pred[X[:,0]>0.5]=1

return y\_pred

# TEST

X = np.array([

[1, 1],

[2, 1],

[0, 1],

])

assert np.all(predict\_v1(X) == np.array([1, 1, 0]))

Теперь вернемся к нашим данным и посмотрим, как работает предсказание разделением по единственному признаку:

X = X\_data

y\_true = y\_data

x1\_vals = np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 100)

x2\_vals = np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 100)

xx, yy = np.meshgrid(x1\_vals, x2\_vals)

space\_X = np.c\_[xx.ravel(), yy.ravel()]

y\_pred = predict\_v1(space\_X)

y\_pred = y\_pred.reshape(xx.shape)

plt.contourf(xx, yy, y\_pred)

pnts\_scatter = plt.scatter(X[:, 0], X[:, 1], c=y\_true, s=50, edgecolor='k')

plt.xlabel("$x\_1$")

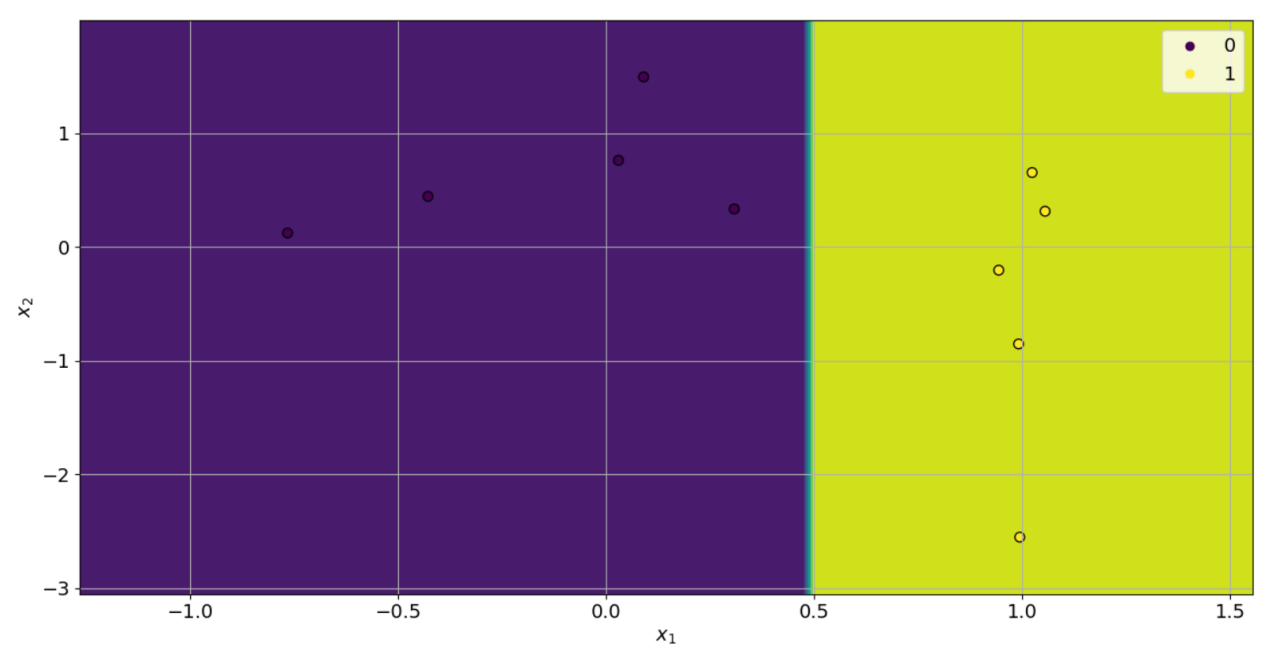
plt.ylabel("$x\_2$")

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1', '2'])

plt.show()

Результат



Для начала, мы же не знаем лучшее разделение из данныхнам надо получить его автоматически. Попробуем три разных порога для разделения данных по признаку x1 (который стоит в колонке 0) и посчитаем примеси Джини каждой части после разделения:

for threshold in thresholds:

print(f'\tSplit by {threshold}')

split\_mask = X[:, feature\_index] > threshold

y\_true\_left = y\_true[split\_mask]

y\_true\_right = y\_true[~split\_mask]

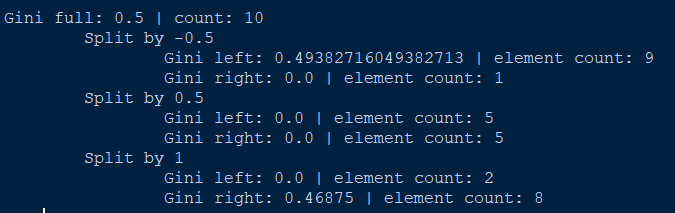
gini\_left = gini\_impurity(y\_true\_left)

gini\_right = gini\_impurity(y\_true\_right)

print(f'\t\tGini left: {gini\_left} | element count: {len(y\_true\_left)}')

print(f'\t\tGini right: {gini\_right} | element count: {len(y\_true\_right)}')

Результат



Давайте сделаем расчет в нашем случае:

thresholds = [-0.5, 0.5, 1]

feature\_index = 0

X = X\_data

y\_true = y\_data

gini\_full = gini\_impurity(y\_true)

print(f'Gini full: {gini\_full}')

for threshold in thresholds:

print(f'\tSplit by {threshold}')

split\_mask = X[:, feature\_index] > threshold

y\_true\_left = y\_true[split\_mask]

y\_true\_right = y\_true[~split\_mask]

gini\_left = gini\_impurity(y\_true\_left)

gini\_right = gini\_impurity(y\_true\_right)

print(f'\t\tGini left: {gini\_left}')

print(f'\t\tGini right: {gini\_right}')

weight\_left = len(y\_true\_left)/len(y\_true)

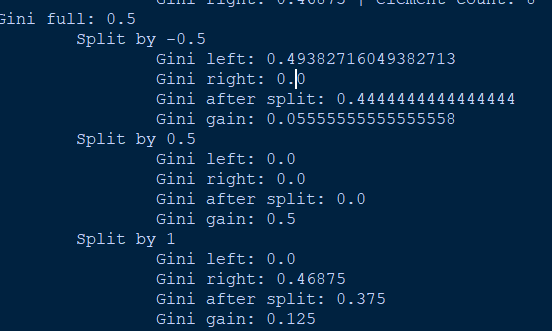
weight\_right = len(y\_true\_right)/len(y\_true)

weighted\_gini = weight\_left \* gini\_left + weight\_right \* gini\_right

print(f'\t\tGini after split: {weighted\_gini}')

print(f'\t\tGini gain: {gini\_full-weighted\_gini}')

Результат



**Выбор лучшего разделения (сплита)**

А теперь, реализуйте его в качестве функции:

def get\_best\_split(X, y\_true):

best\_gini\_gain = 0

best\_gini\_impurity = 0

best\_feature\_idx = 0

best\_threshold = 0

gini\_full = gini\_impurity(y\_true)

# TODO - дополните реализацию функции получения наилучшего разделения

for feat in range(X.shape[1]):

for dat in range(X.shape[0]):

split\_mask = X[:, feat] > X[dat,feat]

y\_true\_left = y\_true[split\_mask]

y\_true\_right = y\_true[~split\_mask]

gini\_left = gini\_impurity(y\_true\_left)

gini\_right = gini\_impurity(y\_true\_right)

weight\_left = len(y\_true\_left)/len(y\_true)

weight\_right = len(y\_true\_right)/len(y\_true)

weighted\_gini = weight\_left \* gini\_left + weight\_right \* gini\_right

gini\_gain=gini\_full-weighted\_gini

if (gini\_gain>best\_gini\_gain):

best\_gini\_gain=gini\_gain

best\_threshold=X[dat,feat]

best\_feature\_idx=feat

best\_gini\_impurity=weighted\_gini

return best\_gini\_impurity, best\_feature\_idx, best\_threshold

# TEST

X = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1)

y = np.array([1, 1, 1, 1, 0, 0, 0, 1])

best\_gini, best\_feature\_idx, best\_threshold = get\_best\_split(X, y)

assert np.isclose(best\_gini, 0.1875)

assert np.isclose(best\_threshold, 4)

assert best\_feature\_idx == 0

Проверим наши данные:

best\_gini, best\_feature\_idx, best\_threshold = get\_best\_split(X\_data, y\_data)

print(

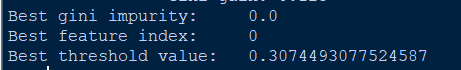
f"Best gini impurity:\t{best\_gini}",

f"\nBest feature index:\t{best\_feature\_idx}",

f"\nBest threshold value:\t{best\_threshold}"

)

Результат



Теперь самое время реализовать второй вариант функции предсказания, которая будет производить предсказание на разделения по признаку и порогу, заданными через аргументы:

# TODO

def predict\_v2(X, feature\_index, threshold):

# Напишите реализацию функции предсказания

# решающего дерева с одним узлом

# разделение по признаку (x1) с порогом 0.5

# \*Не забывайте о размерности данных X

y\_pred=np.zeros\_like(X[:,feature\_index])

y\_pred[X[:,feature\_index]>threshold]=1

return y\_pred

# TEST

X = X\_data

y\_true = y\_data

y\_pred = predict\_v2(X, best\_feature\_idx, best\_threshold)

assert np.all(y\_true == y\_pred)

X = X\_data

y\_true = y\_data

x1\_vals = np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 100)

x2\_vals = np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 100)

xx, yy = np.meshgrid(x1\_vals, x2\_vals)

space\_X = np.c\_[xx.ravel(), yy.ravel()]

y\_pred = predict\_v2(space\_X, best\_feature\_idx, best\_threshold)

y\_pred = y\_pred.reshape(xx.shape)

plt.contourf(xx, yy, y\_pred)

pnts\_scatter = plt.scatter(X[:, 0], X[:, 1], c=y\_true, s=50, edgecolor='k')

plt.xlabel("$x\_1$")

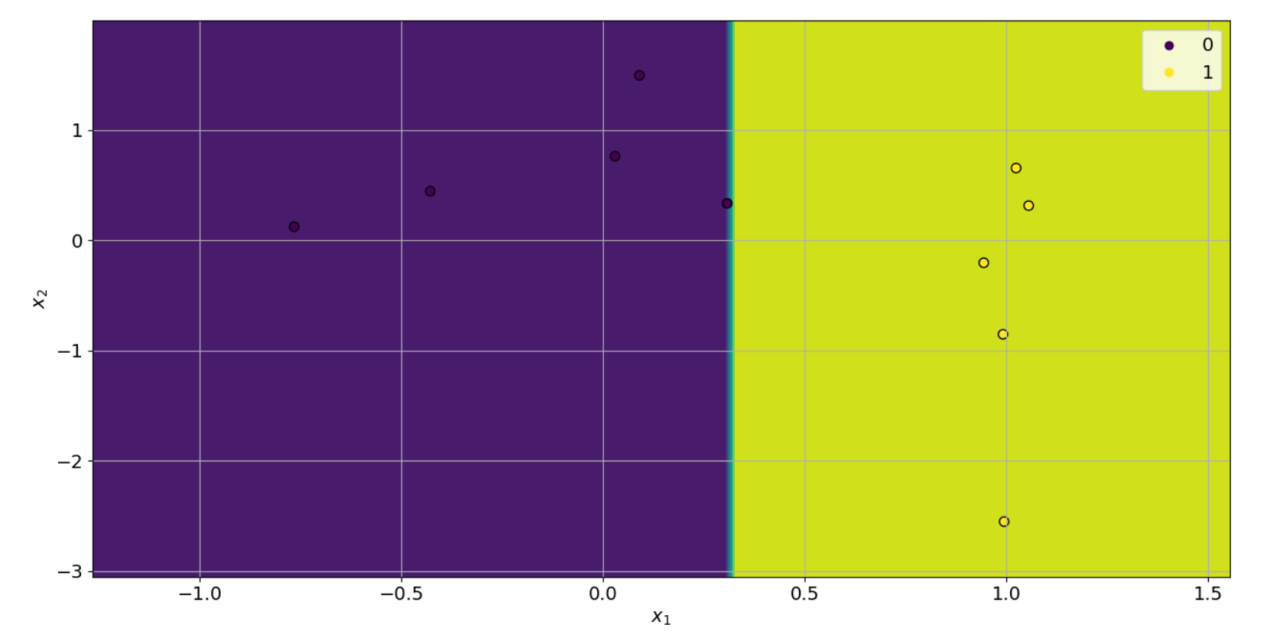
plt.ylabel("$x\_2$")

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1', '2'])

plt.show()

Результат



# **Представление структуры дерева**

X\_data, y\_data = make\_classification(

n\_samples=100,

n\_features=2,

n\_redundant=0,

n\_informative=2,

n\_clusters\_per\_class=2,

random\_state=3

)

pnts\_scatter = plt.scatter(X\_data[:, 0], X\_data[:, 1], marker='o', c=y\_data, s=50, edgecolor='k', )

plt.xlabel('$x\_1$')

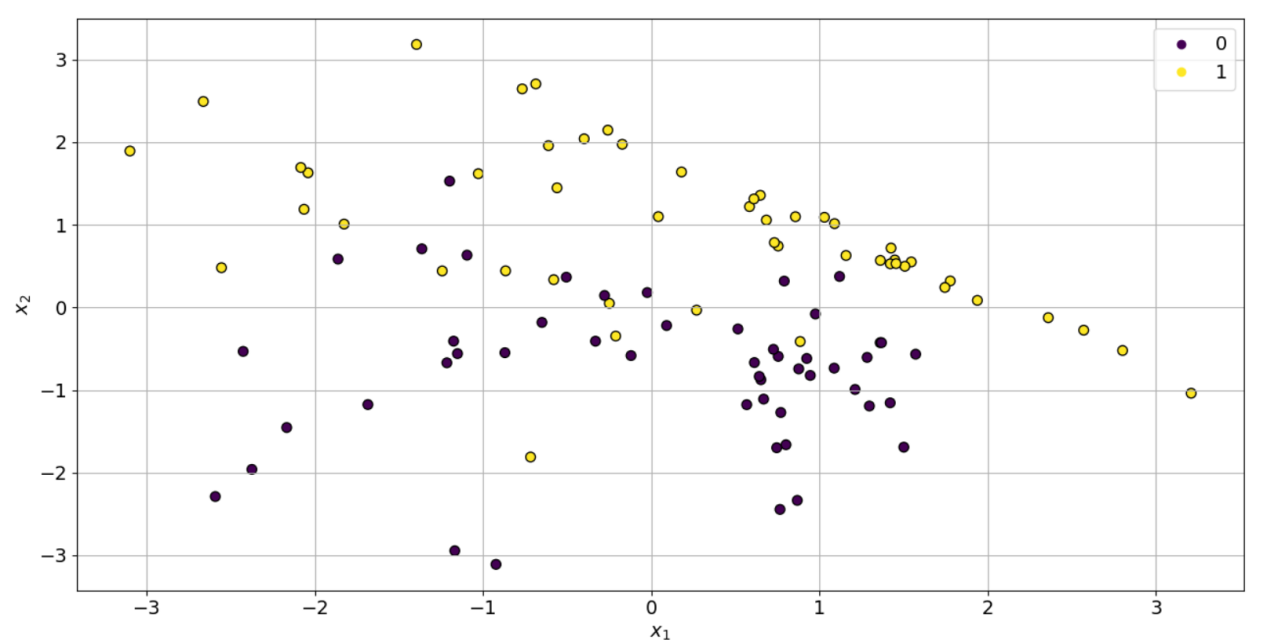
plt.ylabel('$x\_2$')

plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1'])

plt.show()

Результат



Метод получения информации о глубине .get\_max\_depth() просто вернет глубину листа, на котором он находится.

class DecisionLeaf:

def \_\_init\_\_(self, depth):

''' Конструктор класса

Аргументы

---------

depth: int

глубина листа, на котором он располагается

'''

self.predict\_class = None

self.depth = depth

def predict(self, X):

''' Функция предсказания листа

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных

Возвращает

----------

predict: ndarray [n\_samples]

вектор предсказаний, заполненный значениями

класса листа

'''

# TODO - напишите функцию предсказания

y\_pred=np.full(X.shape[0],self.predict\_class)

return y\_pred

def fit(self, X, y):

''' Метод находит в данных класс с наибольшим количеством записей

и присваивает его листу как наиболее вероятно

предсказываемый класс

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных для обучения

y : ndarray [n\_samples]

вектор истинных значений классов

'''

# TODO - напишите функцию обучения

# Выбираем из y наиболее часто встречающееся значение

# и присваиваем self.predict\_class

# это и будет предсказываемый класс листа

class1=np.unique(y)

for i in class1:

if(len([y==self.predict\_class])<len(y[y==i])):

self.predict\_class=i

def get\_max\_depth(self):

''' Получение информации о максимальной глубине

Возвращает

----------

depth: int

глубина листа

'''

# TODO - напишите функцию возврата глубины, на которой находится лист

return self.depth

def print(self):

''' Вывод информации о листе '''

print(f'{self.depth\*" "}> Class {self.predict\_class}')

# TEST

leaf = DecisionLeaf(1)

assert leaf.get\_max\_depth() == 1

X = np.array([1, 1, 1, 3]).reshape(-1, 1)

y = np.array([0, 1, 1, 2])

leaf.fit(X, y)

y\_pred = leaf.predict(X)

y\_true = np.array([1, 1, 1, 1])

assert np.all(y\_pred == y\_true)

assert np.all(y\_pred.shape == y\_true.shape)

class DecisionNode:

def \_\_init\_\_(self, depth, depth\_limit, min\_samples\_split):

''' Конструктор класса

Аргументы

---------

depth: int

глубина узла, на которой он располагается

depth\_limit: int

максимальная глубина дерева

min\_samples\_split: int

минимальное количество записей для создания узла

'''

# Глубина, на которой узел находится

self.depth = depth

# Максимальная глубина

self.depth\_limit = depth\_limit

# Минимальное количество записей после сплита, чтобы создать узел

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

# Индекс признака, по которому узел делает разделение

self.feature\_index = None

# Порог для разделения

self.threshold = None

# Аттрибуты для веток (правая ~ true, левая ~ false)

self.true\_elem = None

self.false\_elem = None

def \_create\_new\_element(self, X, y):

''' Метод создания нового элемента

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных для обучения

y : ndarray [n\_samples]

вектор истинных значений классов

'''

# Если в разметке остались уникальные классы - создаем лист

if len(set(y)) == 1:

return DecisionLeaf(self.depth+1)

# TODO - допишите ограничения

# на минимальное количество записей в данных

# и ограничение глубины

if (len(y)<=self.min\_samples\_split):

return DecisionLeaf(self.depth+1)

if (self.depth>=self.depth\_limit-1):

return DecisionLeaf(self.depth+1)

# Если так и не вернули лист - то возвращаем узел

# У него увеличиваем глубину на 1 и пробрасываем инфу об ограничениях

return DecisionNode(

self.depth+1,

self.depth\_limit,

self.min\_samples\_split

)

def predict(self, X):

''' Функция предсказания узла

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных

Возвращает

----------

predict: ndarray [n\_samples]

вектор предсказаний

'''

# TODO - напишите реализацию метода предсказания

# Получите маску разделения

mask = X[:, self.feature\_index] > self.threshold

right\_X = X[mask]

left\_X=X[~mask]

# Вот формируем вектор предсказания

prediction = np.ndarray(X.shape[0], dtype=int)

# Вот заполняем предсказания одной ветви

prediction[~mask] = self.false\_elem.predict(left\_X)

prediction[mask] = self.true\_elem.predict(right\_X)

# Сделайте заполнения для второй ветви

print(prediction[mask])

return prediction

def get\_best\_split(self, X, y\_true):

best\_gini\_gain = 0

best\_gini\_impurity = 0

best\_feature\_idx = 0

best\_threshold = 0

gini\_full = gini\_impurity(y\_true)

# TODO - дополните реализацию функции получения наилучшего разделения

for fear in range(X.shape[1]):

for dat in range(X.shape[0]):

split\_mask = X[:, fear] > X[dat,fear]

y\_true\_left = y\_true[split\_mask]

y\_true\_right = y\_true[~split\_mask]

gini\_left = gini\_impurity(y\_true\_left)

gini\_right = gini\_impurity(y\_true\_right)

weight\_left = len(y\_true\_left)/len(y\_true)

weight\_right = len(y\_true\_right)/len(y\_true)

weighted\_gini = weight\_left \* gini\_left + weight\_right \* gini\_right

gini\_gain=gini\_full-weighted\_gini

if (gini\_gain>best\_gini\_gain):

best\_gini\_gain=gini\_gain

best\_threshold=X[dat,fear]

best\_feature\_idx=fear

best\_gini\_impurity=weighted\_gini

return best\_feature\_idx, best\_threshold

def fit(self, X, y):

''' Метод обучения узла

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных для обучения

y : ndarray [n\_samples]

вектор истинных значений классов

'''

# TODO - напишите реализацию метода обучения

# Получите лучший сплит

# Сохраните параметры сплита в self.feature\_index и self.threshold

self.feature\_index, self.threshold = self.get\_best\_split(X, y)

# Вот здесь мы создаем маску для деления

mask = X[:, self.feature\_index] > self.threshold

right\_X = X[mask]

right\_y = y[mask]

self.true\_elem = self.\_create\_new\_element(right\_X, right\_y)

self.true\_elem.fit(right\_X, right\_y)

# Вам нужно сделать аналогичные действия для другой ветки

left\_X = X[~mask]

left\_y = y[~mask]

self.false\_elem = self.\_create\_new\_element(left\_X, left\_y)

self.false\_elem.fit(left\_X, left\_y)

def get\_max\_depth(self):

''' Получение информации о максимальной глубине

Возвращает

----------

depth: int

глубина листа

'''

# Берем максимум от максимальной глубины по веткам

return max([

self.true\_elem.get\_max\_depth(),

self.false\_elem.get\_max\_depth()

])

def print(self):

''' Вывод информации об узле '''

print(f'{self.depth\*" "}| {self.feature\_index} > {self.threshold}')

self.true\_elem.print()

self.false\_elem.print()

# TEST

# Допустим, что узел на уровне 1 и максимальная глубина = 2

# Ограничение на минимальное количество уберем

node = DecisionNode(1, 2, 0)

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([0, 0, 1, 1, 0])

node.fit(X, y)

y\_pred = node.predict(X)

y\_true = np.array([0, 0, 1, 1, 1])

assert node.get\_max\_depth() == 2

assert np.all(y\_pred == y\_true)

assert np.all(y\_pred.shape == y\_true.shape)

Результат



class DecisionTree:

def \_\_init\_\_(self, depth\_limit, min\_samples\_split):

''' Конструктор класса

Аргументы

---------

depth\_limit: int

максимальная глубина дерева

min\_samples\_split: int

минимальное количество записей для создания узла

'''

self.root = DecisionNode(0, depth\_limit, min\_samples\_split)

def predict(self, X):

''' Функция предсказания узла

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных

Возвращает

----------

predict: ndarray [n\_samples]

вектор предсказаний

'''

return self.root.predict(X)

def fit(self, X, y):

''' Функция обучения

Аргументы

---------

X : ndarray [n\_samples, n\_features]

матрица данных

'''

self.root.fit(X, y)

def get\_depth(self):

''' Получение информации о глубине дерева

Возвращает

----------

depth: int

глубина листа

'''

return self.root.get\_max\_depth()

def print(self):

''' Вывод информации о дереве '''

self.root.print()

# TEST

X = X\_data

y\_true = y\_data

# Снимем ограничения дерева

# Не ограничиваем глубину и минимальное кол-во записей для узла

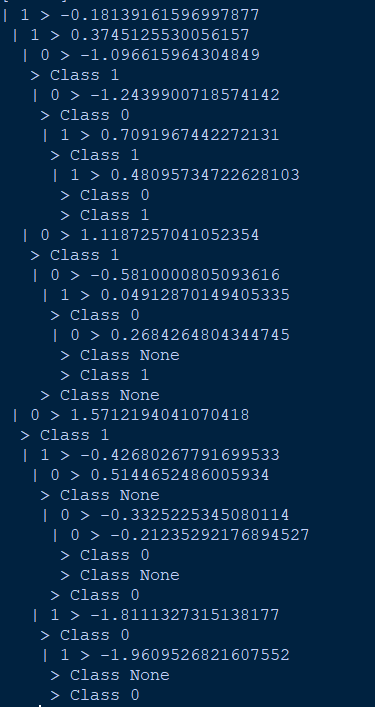
tree = DecisionTree(100, 0)

tree.fit(X,y\_true)

assert tree.get\_depth() == 6

tree.print()

Результат



Если дерево обучилось, тест на соответствующую глубину пройден - можно взглянуть на пространство принятия решений:

Если prediction = np.ndarray(X.shape[0],dtype=int),то



Поэтому prediction = np.ndarray(X.shape[0])

def plot\_tree\_decision\_space(X, y\_true, tree):

x1\_vals = np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 300)

x2\_vals = np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 300)

xx, yy = np.meshgrid(x1\_vals, x2\_vals)

space\_X = np.c\_[xx.ravel(), yy.ravel()]

y\_pred = tree.predict(space\_X)

y\_pred = y\_pred.reshape(xx.shape)

plt.contourf(xx, yy, y\_pred)

pnts\_scatter = plt.scatter(X[:, 0], X[:, 1], c=y\_true, s=50, edgecolor='k')

plt.xlabel("$x\_1$")

plt.ylabel("$x\_2$")

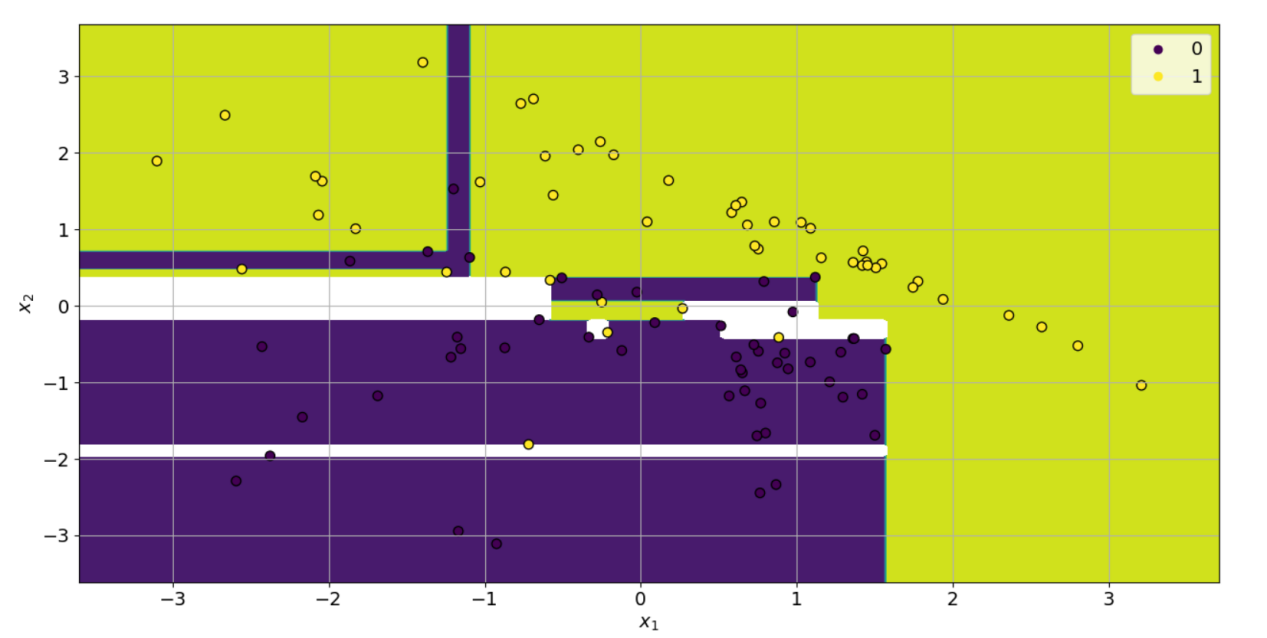
plt.grid(True)

plt.legend(handles=pnts\_scatter.legend\_elements()[0], labels=['0', '1', '2'])

plt.show()

plot\_tree\_decision\_space(X, y\_true, tree)

Результат

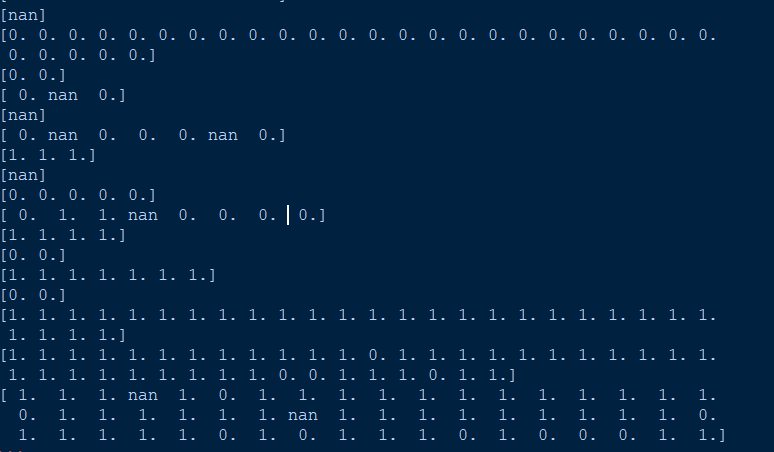


Обратите внимание, как нелинейно произошло разделение пространства! Давайте для простой проверки посмотрим, сколько элементов не соответсвует вектору истинных значений:

y\_pred = tree.predict(X)

(y\_pred != y\_true).sum()

Результат



# Вопросики!

1.Чем отличаются корень, узлы и листья у дерева? Почему внезапно такая ассоциация с деревьями? Корень - это самый первый узел. В узлы пр ходят данные разделяются в соответсвии с заданным порогом. То есть узлы проверяют значения, а листья это конкретне конечные классы, которые присваиваются этим значениям.

2.Зачем нужен порог дереву? Порог нужен для разделения данных и присвоения им класса.

3.Когда в дереве много примесей - это хорошо, плохо или ещё как-то? Примеси необходимо минимизировать. При это достижения их минимума в узле является основанием для выбора порога.

4.Чем опасна рекурсия и как держать её в узде? При рекурсии возможно зацикливание. Необходимо четко задавать условие выхода.

5.Когда дерево может точно переобучиться? Дерево точно переобучится если его не ограничить. Можно ограничивать по глубине, по наличию данных или уникальных классов.

6.Что лучше одно супер-дерево или много нормальных таких деревьев? Одно нормальное дерево может дать результат хуже чем одно супер, но усли использовать несколько таких, и их данные усреднить, но много нормальных деревьев будет лучше. Плюс ликвидируется вероятность переобучиться.

7.Как связан бэггинг и ансамблирование? Бэггинг создает из одной нашей выборки N случайных того же размера, состоящих из тех же элементов. Соответственно деревьев тоже становится N. То есть мы работает с ансамблем из N деревьев.

8.Что такое бутстрэпинг? Метод бутстрэпинга - это метод выборки, при котором мы берем случайную запись из данных и заносим ее в новую выборку, но не исключаем из исходных данных, то есть она может попасть в эту же выборку снова или в другие.

9.(Вопрос на расширение сознания - гляньте в инет) Что такое показатель энтропии и как его можно использовать при построении деревьев? Энтропия - это степень однородности данных. Или насколько мы уверены в принятии решении. Она должна стремиться к нулю. То есть на ее основе можно искать порог переключения.

10.Почему такое название "обучение с учителем"? Где этот учитель прячется? Обучение с учителем - это способ машинного обучения, при котором веса целевой функции меняются в соответствии со степеью близости ответа модели и реальных значений. Собственно реальные значения (обучающая выборка) и есть учитель.

11.Можно ли переобучить модель линейной регрессии? В линейной регрессии есть явления overfit, когда модель сложнее, чем нужно, что и есть по сути переобучение.