Алгоритмы анализа данных

Урок 4. Алгоритм построения дерева решений

1. В коде из методички реализуйте один или несколько из критериев останова (количество листьев, количество используемых признаков, глубина дерева и т.д.).
2. Для задачи классификации обучить дерево решений с использованием критериев разбиения Джини и Энтропия. Сравнить качество классификации, сделать выводы.
3. [**опция**] Реализуйте дерево для задачи регрессии. Возьмите за основу дерево, реализованное в методичке, заменив механизм предсказания в листе на взятие среднего значения по выборке, и критерий Джини на дисперсию значений.

Практическое задание

Реализация из методички

В [1]:
```
import matplotlib.pyplot as plt
import random
import math

from matplotlib.colors import ListedColormap
from sklearn import datasets

import numpy as np
```

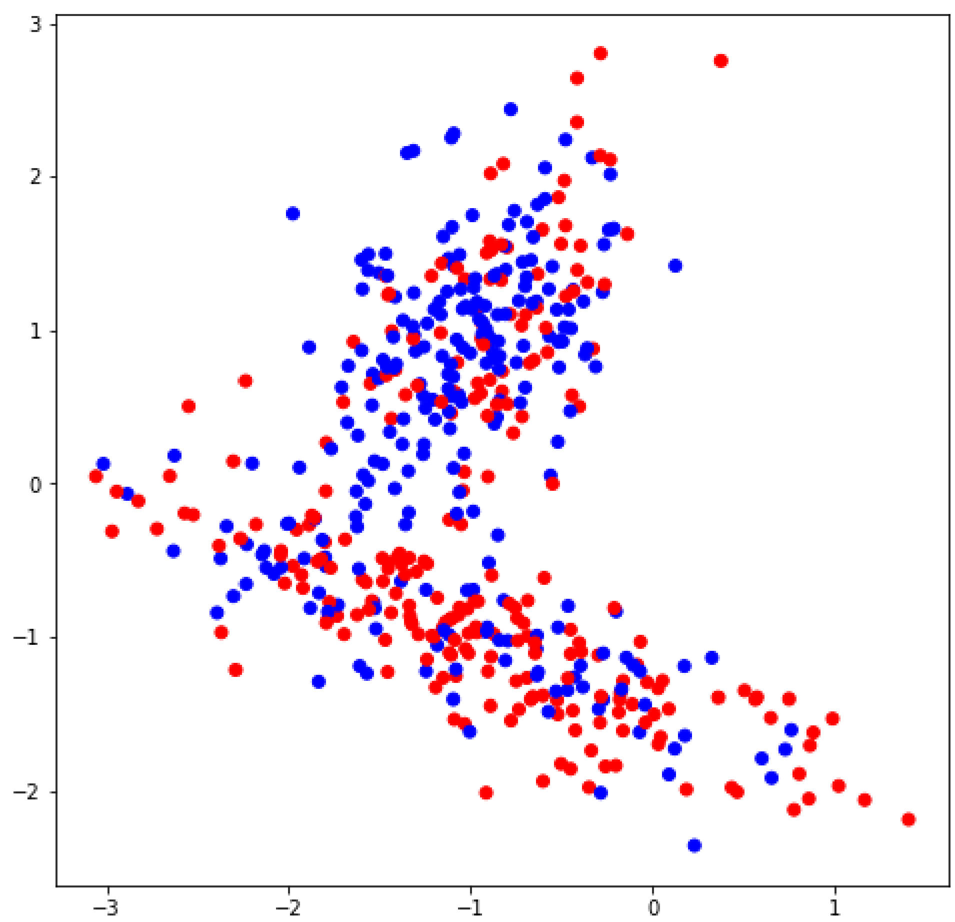
В [2]:
```
# сгенерируем данные
classification_data, classification_labels = datasets.make_classification(
    n_samples=500,
    n_features=2,
    n_informative=2,
    n_classes=2,
    n_redundant=0,
    n_clusters_per_class=1,
    flip_y=0.75,
    random_state=5
)
```

В [3]:
```
# визуализируем сгенерированные данные

colors = ListedColormap(['red', 'blue'])
light_colors = ListedColormap(['lightcoral', 'lightblue'])

plt.figure(figsize=(8,8))
plt.scatter(list(map(lambda x: x[0], classification_data)), list(map(lambda x: x[1], classification_data)),
              c=classification_labels, cmap=colors)
```

Out[3]: `<matplotlib.collections.PathCollection at 0xd26440d3a0>`



В [4]:
```
# Реализуем класс узла

class Node:

    def __init__(self, index, t, true_branch, false_branch):
        self.index = index  # индекс признака, по которому ведется сравнение с порогом в этом узле
        self.t = t  # значение порога
        self.true_branch = true_branch  # поддерево, удовлетворяющее условию в узле
        self.false_branch = false_branch  # поддерево, не удовлетворяющее условию в узле
```

В [5]:
```
# И класс терминального узла (листа)

class Leaf:

    def __init__(self, data, labels):
        self.data = data
        self.labels = labels
        self.prediction = self.predict()

    def predict(self):
        # подсчет количества объектов разных классов
        classes = {}  # сформируем словарь "класс: количество объектов"
        for label in self.labels:
            if label not in classes:
                classes[label] = 0
            classes[label] += 1
        #  найдем класс, количество объектов которого будет максимальным в этом листе и вернем его
        prediction = max(classes, key=classes.get)
        return prediction
```

За функционал качества при работе с деревом решений принимается функционал вида

$$Q(X_m, j, t) = H(X_m) - \frac{|X_l|}{|X_m|}H(X_l) - \frac{|X_r|}{|X_m|}H(X_r),$$

где X_m - множество объектов, попавших в вершину на данном шаге, X_l и X_r - множества, попадающие в левое и правое поддерево, соответственно, после разбиения. $H(X)$ - *критерий информативности*. $\frac{|X_l|}{|X_m|} \equiv p$ - доля выбоки, ушедшая в левое поддерево:

$$Q(X_m, j, t) = H(X_m) - p \cdot H(X_l) - (1 - p) \cdot H(X_r),$$

Критерии информативности

Критерий Джини или **индекс Джини** выглядит следующим образом:

$$H(X) = \sum_{k=1}^{K} p_k(1 - p_k) = \sum_{k=1}^{K}(p_k - p_k^2) = \sum_{k=1}^{K} p_k - \sum_{k=1}^{K} p_k^2 = 1 - \sum_{k=1}^{K} p_k^2,$$

где K - количество классов в наборе данных X.

В [6]:
```
# Расчет критерия Джини
# labels - Y_m
# data - X_m

def gini(labels):
    #  подсчет количества объектов разных поддеревьев
    classes = {}
    for label in labels:
        # значение отсутствует в classes, добавляем его
        if label not in classes:
            classes[label] = 0
        classes[label] += 1

    # labels = [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1]
    # classes = {0: 7, 1: 5}
    # gini(labels) = 0.48611111111111094

    #  расчет критерия
    impurity = 1  # примесь
    for label in classes:
        p = classes[label] / len(labels)  # p=X_l/X_m
        impurity -= p ** 2

    return impurity
```

Энтропийный критерий (**энтропиия Шеннона**)

$$H(X) = -\sum_{k=1}^{K} p_k \log_2 p_k.$$

Минимум энтропии также достигается когда все объекты относятся к одному класссу, а максимум - при равномерном распределении. Стоит отметить, что в формуле полагается, что $0 \cdot \log_2 0 = 0$.

В [7]:
```
# См. реализацию во 2-м задании
```

В [8]:
```
# Расчет качества

def quality(left_labels, right_labels, current_gini):

    # доля выбоки, ушедшая в левое поддерево
    p = float(left_labels.shape[0]) / (left_labels.shape[0] + right_labels.shape[0])

    return current_gini - p * gini(left_labels) - (1 - p) * gini(right_labels)
```

В [9]:
```
# Нахождение наилучшего разбиения

def find_best_split(data, labels):

    #  обозначим минимальное количество объектов в узле
    min_leaf = 5

    current_gini = gini(labels)

    best_quality = 0
    best_t = None
    best_index = None

    n_features = data.shape[1]

    for index in range(n_features):
        # data - матрица признаков
        # labels это y - вектор значений (classification_labels)
        # index - номер признака
        # t - уникальные значения признака

        # будем проверять только уникальные значения признака, исключая повторения
        t_values = np.unique([row[index] for row in data])

        for t in t_values:
            true_data, false_data, true_labels, false_labels = split(data, labels, index, t)
            #  пропускаем разбиения, в которых в узле остается менее 5 объектов
            if len(true_data) < min_leaf or len(false_data) < min_leaf:
                continue

            current_quality = quality(true_labels, false_labels, current_gini)

            #  выбираем порог, на котором получается максимальный прирост качества
            if current_quality > best_quality:
                best_quality, best_t, best_index = current_quality, t, index

    return best_quality, best_t, best_index
```

В [ ]:
```

```

В [10]:
```
# Разбиение датасета в узле

def split(data, labels, index, t):
    # data - матрица признаков
    # labels это y - вектор значений
    # index - номер признака
    # t - уникальные значения признака

    left = np.where(data[:, index] <= t)
    right = np.where(data[:, index] > t)

    true_data = data[left]
    false_data = data[right]
    true_labels = labels[left]
    false_labels = labels[right]

    return true_data, false_data, true_labels, false_labels
```

В [11]:
```
def classify_object(obj, node):

    #  Останавливаем рекурсию, если достигли листа
    if isinstance(node, Leaf):
        answer = node.prediction
        return answer

    if obj[node.index] <= node.t:
        return classify_object(obj, node.true_branch)
    else:
        return classify_object(obj, node.false_branch)
```

В [12]:
```
def predict(data, tree):

    # Получаем ответы
    classes = []
    for obj in data:
        prediction = classify_object(obj, tree)
        classes.append(prediction)
    return classes
```

В [13]:
```
# Разобьем выборку на обучающую и тестовую

from sklearn import model_selection

train_data, test_data, train_labels, test_labels = model_selection.train_test_split(
    classification_data, classification_labels, test_size = 0.3, random_state = 1
)
```

В [14]:
```
# Введем функцию подсчета точности как доли правильных ответов
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

Практическое задание

1. Задача:

В коде из методички реализуйте один или несколько из критериев останова (количество листьев, количество используемых признаков, глубина дерева и т.д.).

В [15]:
```
# Построение дерева с помощью рекурсивной функции

# Реализуем критериев останова - max глубина дерева
def build_tree(data, labels, depth=0):

    # data - матрица признаков
    # labels это y - вектор значений (classification_labels)
    # index - номер признака
    # t - уникальные значения признака

    global max_depth

    quality, t, index = find_best_split(data, labels)

    #  Базовый случай - прекращаем рекурсию, когда нет прироста в качества
    if quality == 0:
        return Leaf(data, labels)

    #  1 случай - прекращаем рекурсию, когда достигнута максимальная глубина дерева
    if depth < max_depth:

        true_data, false_data, true_labels, false_labels = split(data, labels, index, t)

        # Рекурсивно строим два поддерева
        true_branch = build_tree(true_data, true_labels, depth + 1)
        false_branch = build_tree(false_data, false_labels, depth + 1)

        # Возвращаем класс узла со всеми поддеревьями, то есть целого дерева
        return Node(index, t, true_branch, false_branch)

    return Leaf(data, labels)
```

В [16]:
```
# Напечатаем ход нашего дерева
def print_tree(node, spacing=""):

    # Если лист, то выводим его прогноз
    if isinstance(node, Leaf):
        print(spacing + "Прогноз:", node.prediction)
        return

    # Выведем значение индекса и порога на этом узле
    print(spacing + 'Индекс', str(node.index))
    print(spacing + 'Порог', str(node.t))

    # Рекурсионный вызов функции на положительном поддереве
    print (spacing + '--> True:')
    print_tree(node.true_branch, spacing + "  ")

    # Рекурсионный вызов функции на положительном поддереве
    print (spacing + '--> False:')
    print_tree(node.false_branch, spacing + "  ")
```

В [17]:
```
# Визуализируем дерево на графике
def get_meshgrid(data, step=.05, border=1.2):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))

def plot_tree(my_tree, train_data, train_labels):

    plt.figure(figsize = (16, 7))

    # график обучающей выборки
    plt.subplot(1,2,1)
    xx, yy = get_meshgrid(train_data)
    mesh_predictions = np.array(predict(np.c_[xx.ravel(), yy.ravel()], my_tree)).reshape(xx.shape)
    plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    plt.scatter(train_data[:, 0], train_data[:, 1], c = train_labels, cmap = colors)
    plt.title(f'Train accuracy={train_accuracy:.2f}')

    # график тестовой выборки
    plt.subplot(1,2,2)
    plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    plt.scatter(test_data[:, 0], test_data[:, 1], c = test_labels, cmap = colors)
    plt.title(f'Test accuracy={test_accuracy:.2f}')
```

В [18]:
```
# Задаём максимальную глубину дерева
max_depth = 3

# Построим дерево по обучающей выборке
my_tree = build_tree(train_data, train_labels)
```

В [19]:
```
# Получим ответы для обучающей выборки
train_answers = predict(train_data, my_tree)
```

В [20]:
```
# И получим ответы для тестовой выборки
answers = predict(test_data, my_tree)
```

В [21]:
```
# Точность на обучающей выборке
train_accuracy = accuracy_metric(train_labels, train_answers)
train_accuracy
```

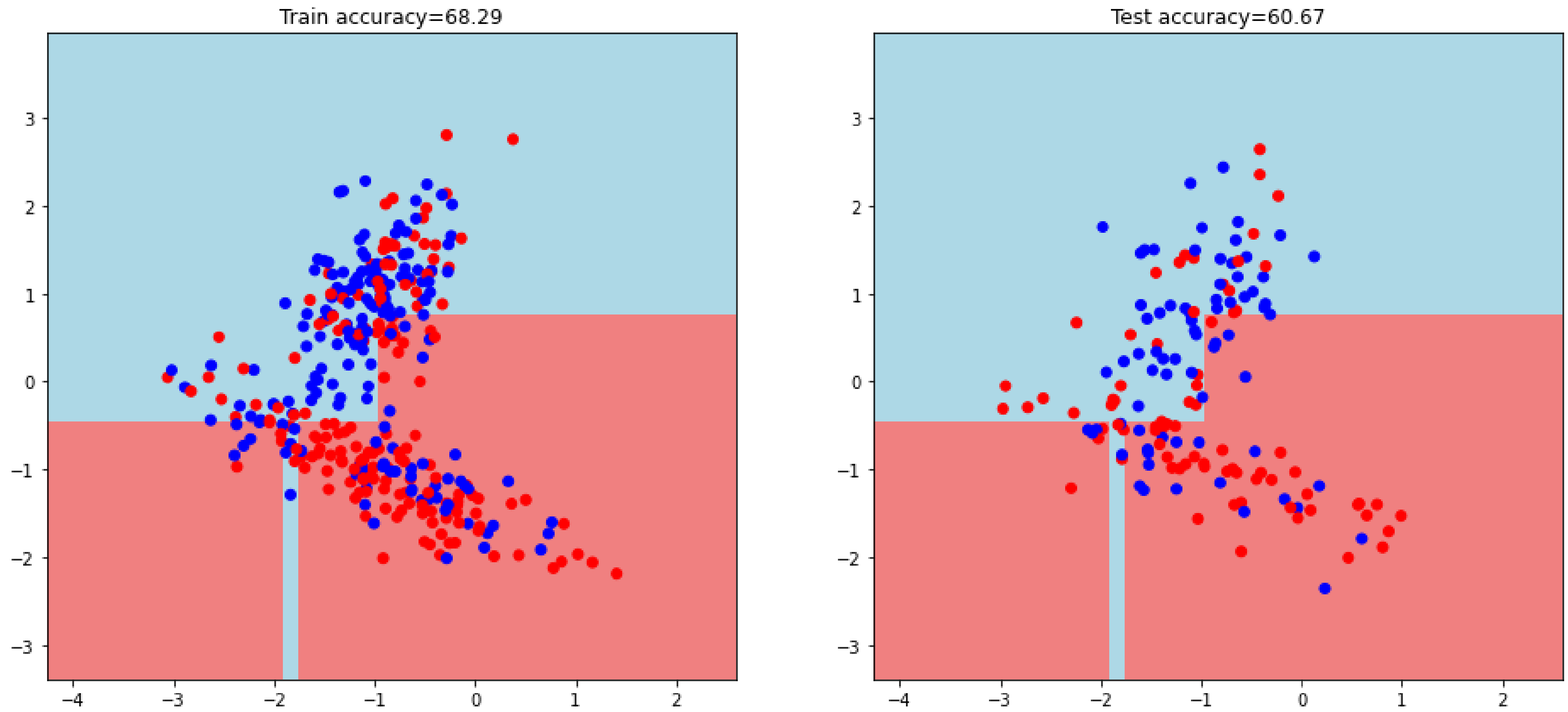
Out[21]: 68.28571428571428

В [22]:
```
# Точность на тестовой выборке
test_accuracy = accuracy_metric(test_labels, answers)
test_accuracy
```

Out[22]: 60.66666666666667

В [23]:
```
# Визуализируем дерево на графике
plot_tree(my_tree, train_data, train_labels)
```

```
<ipython-input-17-21e2eca715e1>:15: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
<ipython-input-17-21e2eca715e1>:21: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
```



В [24]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')

# Напечатаем ход нашего дерева
print_tree(my_tree)
```

```
Максимальную глубину дерева max_depth=3
Индекс 1
Порог -0.46624201061399206
--> True:
  Индекс 0
  Порог -1.79696585158631
  --> True:
    Индекс 0
    Порог -1.9226384124885603
    --> True:
      Прогноз: 0
    --> False:
      Прогноз: 1
  --> False:
    Индекс 0
    Порог -1.1834902220493384
    --> True:
      Прогноз: 0
    --> False:
      Прогноз: 0
--> False:
  Индекс 0
  Порог -0.9820958539320116
  --> True:
    Индекс 0
    Порог -1.6445609784573159
    --> True:
      Прогноз: 1
    --> False:
      Прогноз: 1
  --> False:
    Индекс 1
    Порог 0.7329350213251566
    --> True:
      Прогноз: 0
    --> False:
      Прогноз: 1
```

В [25]:
```
# Задаём максимальную глубину дерева
max_depth = 10

# Построим дерево по обучающей выборке
my_tree = build_tree(train_data, train_labels)
```

В [26]:
```
# Получим ответы для обучающей выборки
train_answers = predict(train_data, my_tree)
```

В [27]:
```
# И получим ответы для тестовой выборки
answers = predict(test_data, my_tree)
```

В [28]:
```
# Точность на обучающей выборке
train_accuracy = accuracy_metric(train_labels, train_answers)
train_accuracy
```

Out[28]: 79.42857142857143

В [29]:
```
# Точность на тестовой выборке
test_accuracy = accuracy_metric(test_labels, answers)
test_accuracy
```

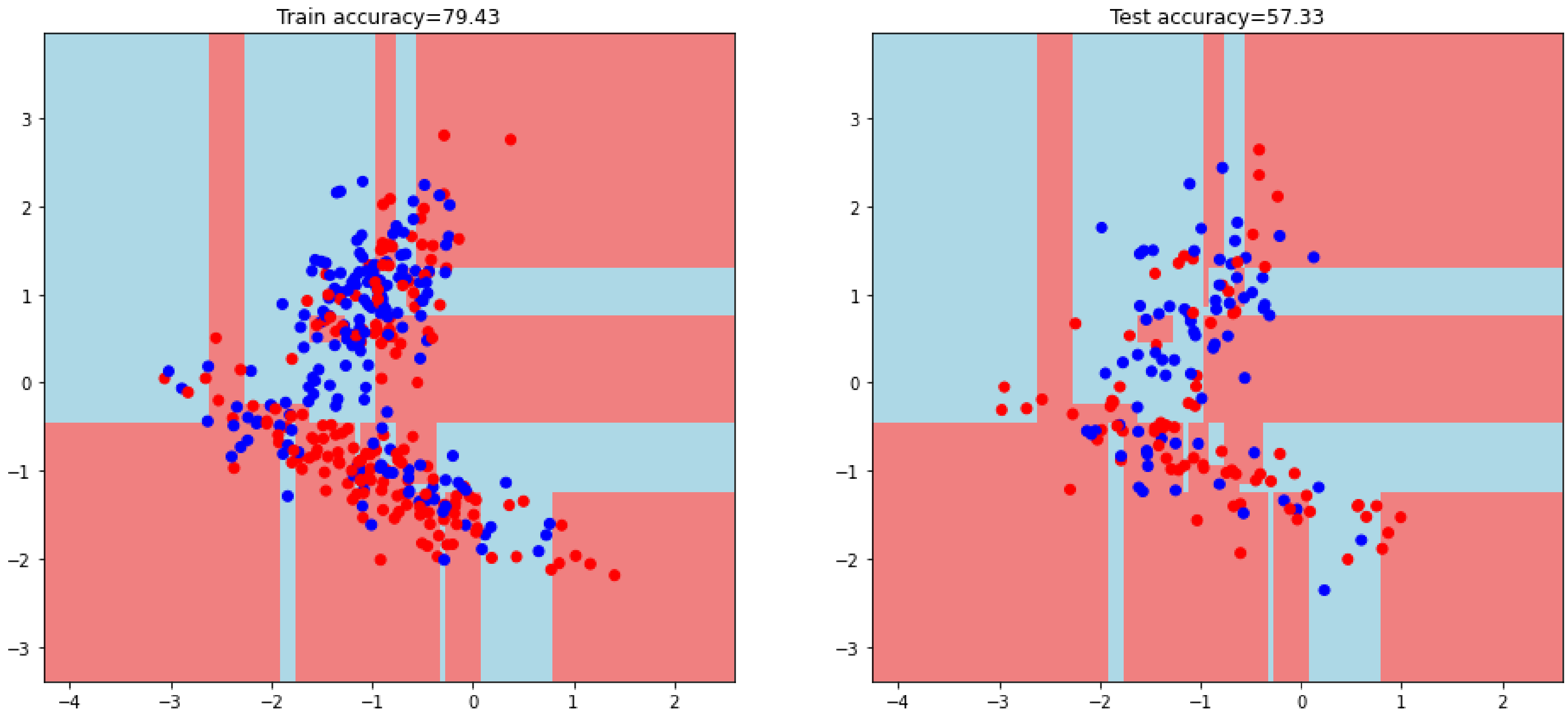
Out[29]: 57.333333333333336

В [30]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')

plot_tree(my_tree, train_data, train_labels)
```

```
Максимальную глубину дерева max_depth=10

<ipython-input-17-21e2eca715e1>:15: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions
as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
<ipython-input-17-21e2eca715e1>:21: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions
as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
```



В [31]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')

# Напечатаем ход нашего дерева
print_tree(my_tree)
```

```
Максимальную глубину дерева max_depth=10
Индекс 1
Порог -0.46624201061399206
--> True:
  Индекс 0
  Порог -1.79696585158631
  --> True:
    Индекс 0
    Порог -1.9226384124885603
    --> True:
      Прогноз: 0
    --> False:
      Прогноз: 1
  --> False:
    Индекс 0
    Порог -1.1834902220493384
    --> True:
      Индекс 0
      Порог -1.6959366052924192
```

2. Задача:

Для задачи классификации обучить дерево решений с использованием критериев разбиения Джини и Энтропия. Сравнить качество классификации, сделать выводы.

Критерии информативности

Энтропийный критерий (энтропиия Шеннона)

$$H(X) = -\sum_{k=1}^{K} p_k \log_2 p_k.$$

Минимум энтропии также достигается когда все объекты относятся к одному класссу, а максимум - при равномерном распределении. Стоит отметить, что в формуле полагается, что

$$0 \cdot \log_2 0 = 0$$

В [32]:

```
# Расчёт энтропии Шеннона - 2-е практическое заднние

def entropy(labels):
    #  подсчет количества объектов разных классов
    classes = {}
    for label in labels:
        if label not in classes:
            classes[label] = 0
        classes[label] += 1

    #  расчет критерия
    impurity = 0
    for label in classes:
        p = classes[label] / len(labels)
        if p==0:
            impurity -= 0  # полагается, что 0·log 0 = 0
        else:
            # impurity -= p*math.log2(p)
            impurity -= p*(np.log(p)/np.log(2))  # используем правило изменения базы логарифмов

    return impurity
```

В [33]:

```
# Расчет качества

def quality_entropy(left_labels, right_labels, current_entropy):

    # доля выбоки, ушедшая в левое поддерево
    p = float(left_labels.shape[0]) / (left_labels.shape[0] + right_labels.shape[0])

    return current_entropy - p * entropy(left_labels) - (1 - p) * entropy(right_labels)
```

В [34]:

```
# Нахождение наилучшего разбиения используя энтропийный критерий Шеннона

def find_best_split_entropy(data, labels):

    #  обозначим минимальное количество объектов в узле
    min_leaf = 5

    current_entropy = entropy(labels)

    best_quality = 0
    best_t = None
    best_index = None

    n_features = data.shape[1]

    for index in range(n_features):
        # будем проверять только уникальные значения признака, исключая повторения
        t_values = np.unique([row[index] for row in data])

        for t in t_values:
            true_data, false_data, true_labels, false_labels = split(data, labels, index, t)
            #  пропускаем разбиения, в которых в узле остается менее 5 объектов
            if len(true_data) < min_leaf or len(false_data) < min_leaf:
                continue

            current_quality = quality_entropy(true_labels, false_labels, current_entropy)

            #  выбираем порог, на котором получается максимальный прирост качества
            if current_quality > best_quality:
                best_quality, best_t, best_index = current_quality, t, index

    return best_quality, best_t, best_index
```

В [35]:
```
# Реализуем критериев останова - max глубина дерева

def build_tree_entropy(data, labels, depth=0):

    # data - матрица признаков
    # labels это y - вектор значений (classification_labels)
    # index - номер признака
    # t - уникальные значения признака

    global max_depth

    quality, t, index = find_best_split_entropy(data, labels)

    #  Базовый случай - прекращаем рекурсию, когда нет прироста в качества
    if quality == 0:
        return Leaf(data, labels)

    #  1 случай - прекращаем рекурсию, когда достигнута максимальная глубина дерева
    if depth < max_depth:

        true_data, false_data, true_labels, false_labels = split(data, labels, index, t)

        # Рекурсивно строим два поддерева
        true_branch = build_tree_entropy(true_data, true_labels, depth + 1)
        false_branch = build_tree_entropy(false_data, false_labels, depth + 1)

        # Возвращаем класс узла со всеми поддеревьями, то есть целого дерева
        return Node(index, t, true_branch, false_branch)

    return Leaf(data, labels)
```

В [36]:
```
# Задаём максимальную глубину дерева
max_depth = 10

# Построим дерево по обучающей выборке
my_tree = build_tree_entropy(train_data, train_labels)
```

В [37]:
```
# Напечатаем ход нашего дерева
# Максимальная глубина дерева max_depth = 3
# print_tree(my_tree)
```

В [38]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')
```

```
Максимальную глубину дерева max_depth=10
```

В [39]:
```
# Получим ответы для обучающей выборки
train_answers = predict(train_data, my_tree)
```

В [40]:
```
# И получим ответы для тестовой выборки
answers = predict(test_data, my_tree)
```

В [41]:
```
# Точность на обучающей выборке
train_accuracy = accuracy_metric(train_labels, train_answers)
train_accuracy
```

Out[41]:
```
80.57142857142857
```

В [42]:
```
# Точность на тестовой выборке
test_accuracy = accuracy_metric(test_labels, answers)
test_accuracy
```

Out[42]:
```
54.0
```

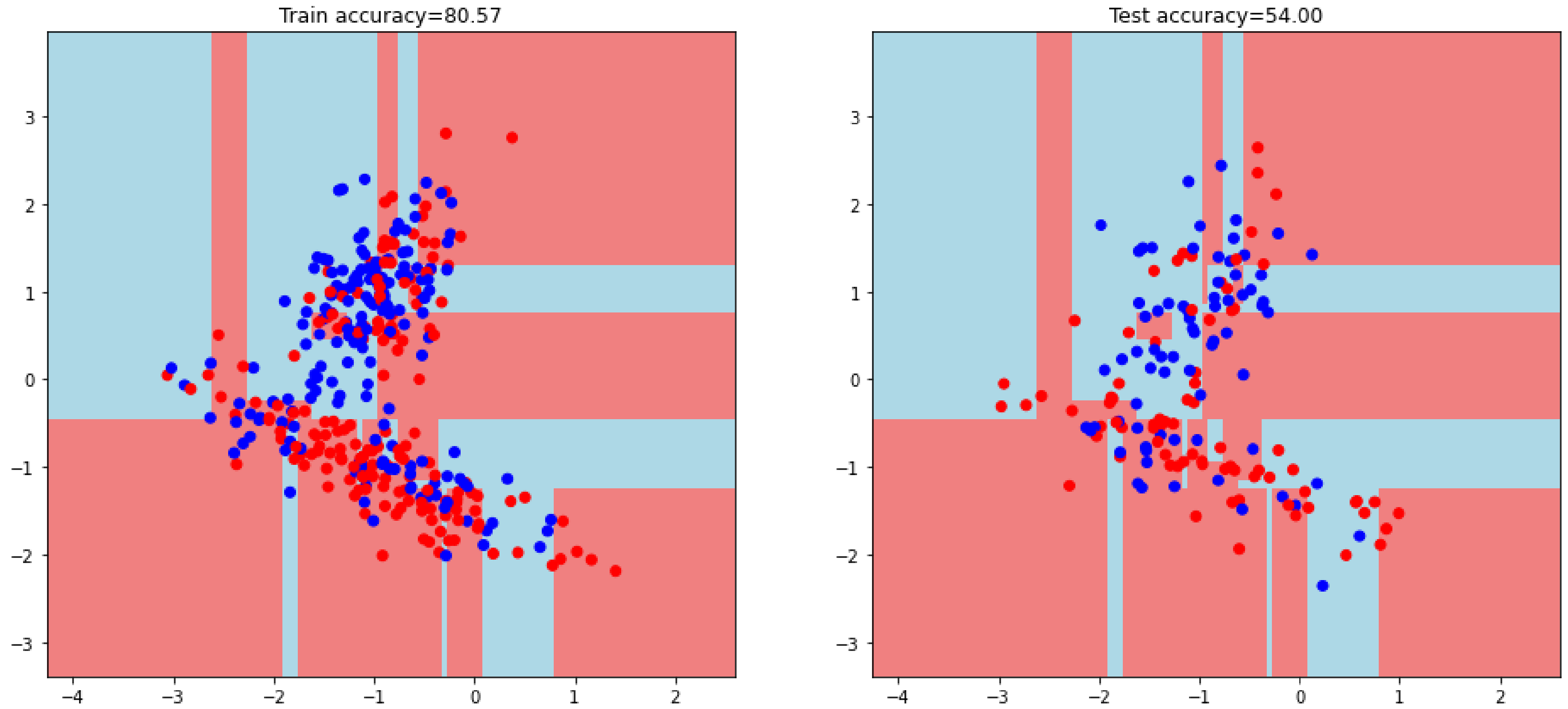
В [43]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')

# Построим дерево по обучающей выборке
my_tree = build_tree(train_data, train_labels)

plot_tree(my_tree, train_data, train_labels)
```

```
Максимальную глубину дерева max_depth=10

<ipython-input-17-21e2eca715e1>:15: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions
as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
<ipython-input-17-21e2eca715e1>:21: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions
as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto',
'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
```



Дерево строит кусочно-постоянную разделяющую гиперплоскость, то есть состоящую из прямых, параллельных осям. Чем глубже дерево, тем сложнее гиперплоскость. Также происходит и в случае регрессии - график зависимости целевого значения восстанавливается кусочно-постоянной функцией.

В [44]:
```
print(f'Максимальную глубину дерева max_depth={max_depth}')
# Напечатаем ход нашего дерева
# print_tree(my_tree)
```

```
Максимальную глубину дерева max_depth=10
```

Максимальную глубину дерева max_depth=10

Критерий Джини (индекс Джини)
Точность на обучающей выборке train_accuracy=79.42857142857143
Точность на тестовой выборке test_accuracy=57.333333333333336

Энтропийный критерий (энтропия Шеннона)
Точность на обучающей выборке train_accuracy=80.57142857142857
Точность на тестовой выборке test_accuracy=54.0

Вывод:

При одинаковых параметрах задачи использование ***энтропийного критерия***, повысило точность на обучающей выборке, но понизило точность на тестовой выборке.

3 [опция]:

Реализуйте дерево для задачи регрессии. Возьмите за основу дерево, реализованное в методичке, заменив механизм предсказания в листе на взятие среднего значения по выборке, и критерий Джини на дисперсию значений.

В [ ]: