Урок 4. Деревья решений

План занятия

Теоретическая часть

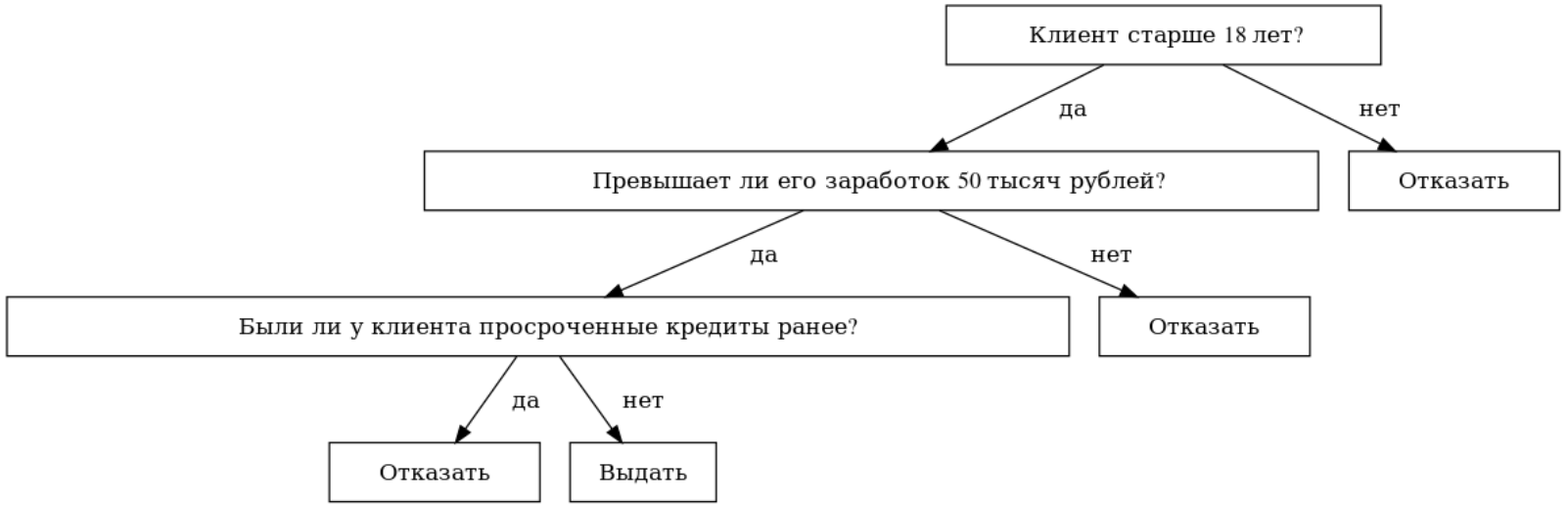
Деревья решений

В этом уроке пойдет речь еще об одном популярном методе машинного обучения - *деревьях решений*. Это семейство алгоритмов значительно отличается от линейных моделей, но применяется также в задачах классификации и регрессии.

Метод основан на известной структуре данных - деревьях, которые по сути представляют собой последовательные инструкции с условиями. Например, в обсуждаемой ранее задаче кредитного скоринга может быть следующий алгоритм принятия решения:

1. Старше ли клиент 18 лет? Если да, то продолжаем, иначе отказываем в кредите.
2. Превышает ли его заработок 50 тысяч рублей? Если да, то продолжаем, иначе отказываем в кредите.
3. Были ли у клиента просроченные кредиты ранее? Если да, отказываем в кредите, иначе выдаем.

В листьях (терминальных узлах) деревьев стоят значения целевой функции (прогноз), а в узлах - условия перехода, определяющие, по какому из ребер идти. Если речь идет о бинарных деревьях (каждый узел производит ветвление на две части), обычно, если условие в узле истинно, то происходит переход по левому ребру, если ложно, то по правому. Изобразим описанный выше алгоритм в виде дерева



В задачах машинного обучения чаще всего в вершинах прописываются максимально простые условия. Обычно это сравнение значения одного из признаков x^j с некоторым заданным порогом t:

$$[x^j \leq t].$$

Если решается задача классификации, конечным прогнозом является класс или распределение вероятностей классов. В случае регрессии прогноз в листе является вещественным числом.

Большим плюсом деревьев является тот факт, что они легко интерпретируемы.

Построение деревьев решений

Деревья обладают и отрицательными качествами - в частности, они очень легко переобучаются. Легко построить дерево, в котором каждый лист будет соответствовать одному объекту обучающей выборки. Оно будет идеально подогнано под обучающую выборку, давать стопроцентный ответ на ней, но при этом не будет восстанавливать оригинальных закономерностей, и качество ответов на новых данных будет неудовлетворительным.

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

from matplotlib.colors import ListedColormap
from sklearn.datasets import make_classification, make_circles
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics import accuracy_score

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

В машинном обучении деревья строятся последовательно от корня к листьям (так называемый "жадный" способ). Вначале выбирается корень и критерий, по которому выборка разбивается на две. Затем то же самое делается для каждого из потомков этого корня и так далее до достаточного уровня ветвления. Задача состоит в выборе способа **разбиения каждого из узлов**, то есть в выборе значения порога, с которым будет сравниваться значение одного из признаков в каждом узле.

Разбиение выбирается с точки зрения некоторого заранее заданного функционала качества $Q(X, j, t)$. Находятся наилучшие значения j и t для создания *предиката* $[x^j < t]$. **Параметры j и t можно выбирать перебором**: признаков конечное число, а из всех возможных значений порога t можно рассматривать только те, при которых получаются различные разбиения на две подвыборки, таким образом, различных значений параметра t будет столько же, сколько различных значений признака x^j в обучающей выборке.

Ввод [2]:
```
df = pd.read_csv('./data/cardio.csv', sep=';')

features = ['age', 'gender', 'height']
target = ['cardio']
df = df.iloc[:5][features + target]
df
```

Out[2]:

	age	gender	height	cardio
0	50	2	168	0
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1
4	47	1	156	0

Ввод [3]:
```
df[(df.gender <= 1)]
```

Out[3]:

	age	gender	height	cardio
1	55	1	156	1
2	51	1	165	1
4	47	1	156	0

Ввод [4]:
```
df[~(df.gender <= 1)]
```

Out[4]:

	age	gender	height	cardio
0	50	2	168	0
3	48	2	169	1

Ввод [5]:
```
df['height'].nunique()
```

Out[5]:
```
4
```

В каждой вершине производится проверка, не выполнилось ли некоторое условие останова (критерии останова рассмотрим далее), и если оно выполнилось, разбиение прекращается, и вершина объявляется листом, и он будет содержать прогноз.

В задаче *классификации* это будет класс, к которому относится большая часть объектов из выборки в листе X_m

$$a_m = \text{argmax}_{y \in Y} \sum_{i \in X_m} [y_i = y]$$

или доля объектов определенного класса k, если требуется предсказать *вероятности классов*

$$a_{mk} = \frac{1}{|X_m|} \sum_{i \in X_m} [y_i = k].$$

Ввод [6]:
```
df
```

Out[6]:

	age	gender	height	cardio
0	50	2	168	0
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1
4	47	1	156	0

Ввод [7]:
```
df[(df.age <= 50.5)]
```

Out[7]:

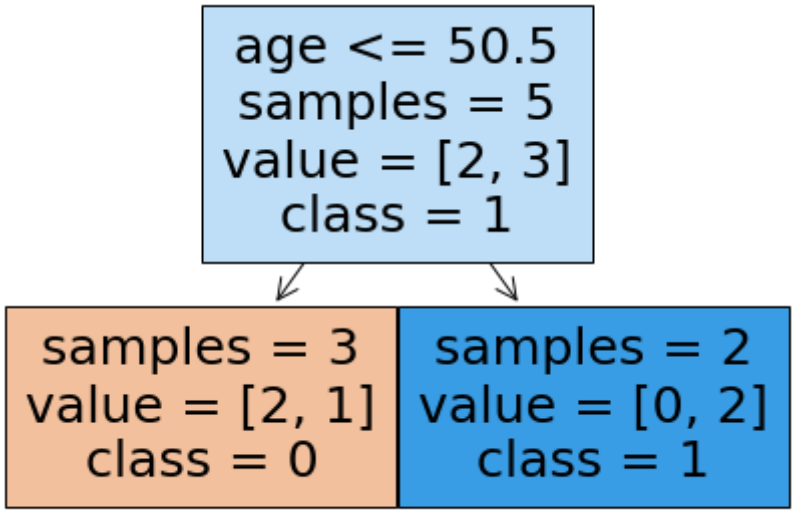
	age	gender	height	cardio
0	50	2	168	0
3	48	2	169	1
4	47	1	156	0

Ввод [8]:
```
df[~(df.age <= 50.5)]
```

Out[8]:

	age	gender	height	cardio
1	55	1	156	1
2	51	1	165	1

Ввод [9]:
```
dt_clsf = DecisionTreeClassifier(random_state=1,
                                 max_depth=1)
dt_clsf.fit(df[features], df[target])

fig, ax = plt.subplots(figsize=(7, 5))
plot_tree(dt_clsf, ax=ax, feature_names=list(features), class_names=['0', '1'], filled=True, impurity=False);
```



В случае *регрессии* можно в качестве ответа давать средний по выборке в листе

$$a_m = \frac{1}{|X_m|} \sum_{i \in X_m} y_i.$$

После построения дерева может проводиться его *стрижка* (pruning) - удаление некоторых вершин согласно некоторому подходу с целью

Ввод [10]:
```
df
```

Out[10]:

	age	gender	height	cardio
0	50	2	168	0
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1
4	47	1	156	0

Ввод [11]:
```
df[(df.cardio <= 0.5)]
```

Out[11]:

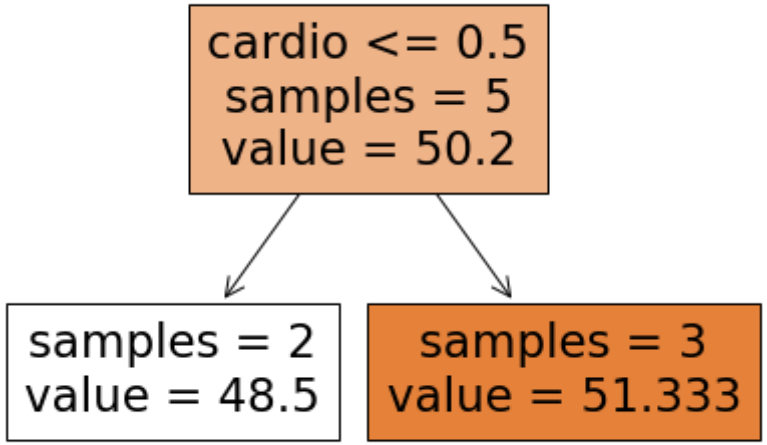
	age	gender	height	cardio
0	50	2	168	0
4	47	1	156	0

Ввод [12]:
```
df[~(df.cardio <= 0.5)]
```

Out[12]:

	age	gender	height	cardio
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1

Ввод [13]:
```
dt_regr = DecisionTreeRegressor(random_state=1,
                                max_depth=1)

features = ['cardio', 'gender', 'height']
target = ['age']

dt_regr.fit(df[features], df[target])

fig, ax = plt.subplots(figsize=(7, 5))
plot_tree(dt_regr, ax=ax, feature_names=list(features), filled=True, impurity=False);
```



За функционал качества при работе с деревом решений принимается функционал вида

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

его еще называют **приростом информации** (information gain).

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*. Он оценивает качество распределения объектов в подмножестве и тем меньше, чем меньше разнообразие ответов в X, соответственно, задача обучения состоит в его минимизации и, соответственно, максимизации $Q(X_m, j, t)$ на данном шаге. Последний, по сути, характеризует прирост качества на данном шаге.

В формуле значения критериев информативности нормируются - домножаются на долю объектов, ушедших в соответствующее подмножество. Например, если у нас множество в узле разбилось на два подмножества размером в 9990 объектов и 10 объектов, но при этом в первом подмножестве все объекты будут принадлежать к одному классу (то есть иметь минимальное значение разброса), а во втором - к разным, то в целом разбиение будет считаться хорошим, так как подавляющее большинство отсортировано правильно.

Критерий информативности

В задаче **классификации** есть несколько способов определить критерий информативности.

Обозначим через p_k долю объектов класса k в выборке X:

$$p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k].$$

p_k будет характеризовать вероятность выдачи класса k.

Энтропийный критерий или *энтропия 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 $0 \log_2 0 = 0$.

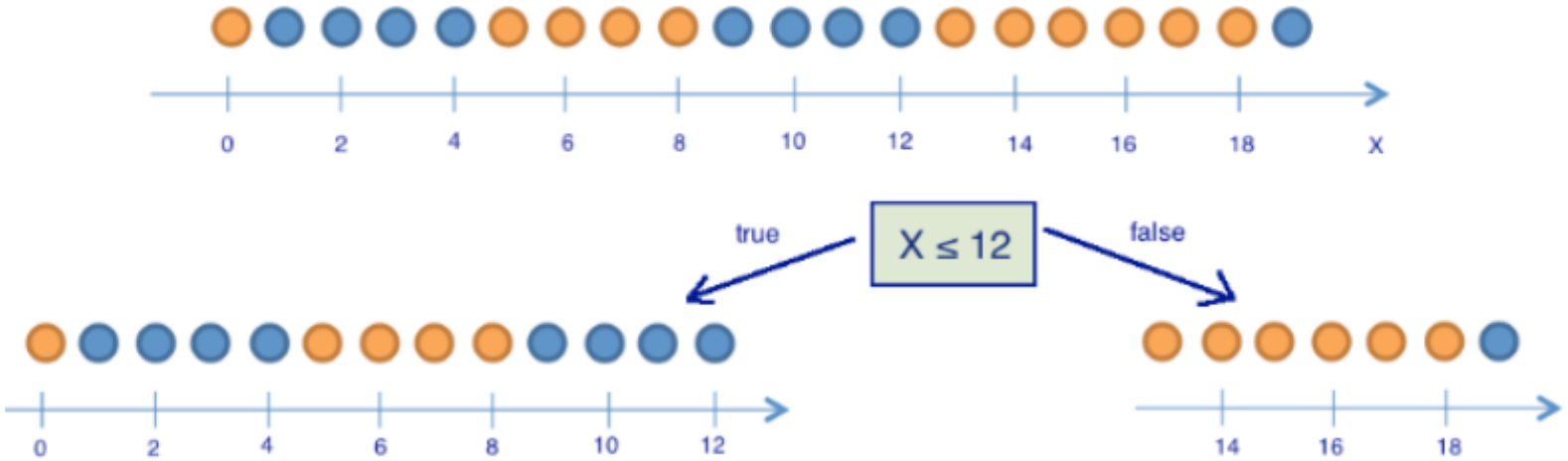
Критерий Джини или *индекс Джини* выглядит следующим образом:

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

где K - количество классов в наборе данных X.

Его минимум достигается когда все объекты в подмножестве относятся к одному классу, а максимум - при равном содержании объектов всех класов. Критерий информативности Джини можно интерпретировать как вероятность ошибки случайного классификатора.

Энтропия и деревья принятия решений (https://habr.com/ru/post/171759/)



Ввод [14]:
```
blue = 9
yellow = 11
total = blue + yellow

p_blue = blue / total
p_yellow = yellow / total
p_blue, p_yellow
```

Out[14]:
```
(0.45, 0.55)
```

Ввод [15]:
```
e0 = - (p_blue * np.log2(p_blue) + p_yellow * np.log2(p_yellow))
e0
```

Out[15]:
```
0.9927744539878083
```

Энтропия левой группы:

Ввод [16]:
```
blue = 8
yellow = 5
total1 = blue + yellow

p_blue = blue / total1
p_yellow = yellow / total1
p_blue, p_yellow
```

Out[16]:
```
(0.6153846153846154, 0.38461538461538464)
```

Ввод [17]:
```
e1 = - (p_blue * np.log2(p_blue) + p_yellow * np.log2(p_yellow))
e1
```

Out[17]:
```
0.9612366047228759
```

Энтропия правой группы:

Ввод [18]:
```
blue = 1
yellow = 6
total2 = blue + yellow

p_blue = blue / total2
p_yellow = yellow / total2
p_blue, p_yellow
```

Out[18]:
```
(0.14285714285714285, 0.8571428571428571)
```

Ввод [19]:
```
e2 = - (p_blue * np.log2(p_blue) + p_yellow * np.log2(p_yellow))
e2
```

Out[19]:
```
0.5916727785823275
```

Ввод [20]:
```
ig = e0 - total1 / total * e1 - total2 / total * e2
ig
```

Out[20]:
```
0.16088518841412436
```

Реализуем критерий информативности Джини

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

Ввод [21]:
```
# Расчет критерия Джини

def gini(labels):
    labels = list(labels)

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

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

    return gini
```

Ввод [22]:
```
def gini(labels):
    labels = list(labels)
    set_labels = set(labels)

    gini = 1
    for label in set_labels:
        p = labels.count(label) / len(labels)
        gini -= p ** 2

    return gini
```

Ввод [23]:
```
# Расчет прироста

def gain(left_labels, right_labels, root_gini):

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

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

Ввод [24]:
```
df
```

Out[24]:

	age	gender	height	cardio
0	50	2	168	0
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1
4	47	1	156	0

Ввод [25]:
```
gini0 = gini(df['cardio'])
gini0
```

Out[25]:
```
0.48
```

Ввод [26]:
```
t = 50.5
```

Ввод [27]:
```
df1 = df[df['age'] <= t]
df2 = df[df['age'] > t]
```

Ввод [28]:
```
df1
```

Out[28]:

	age	gender	height	cardio
0	50	2	168	0
3	48	2	169	1
4	47	1	156	0

Ввод [29]:
```
gini(df1['cardio'])
```

Out[29]:
```
0.4444444444444445
```

Ввод [30]:
```
df2
```

Out[30]:

	age	gender	height	cardio
1	55	1	156	1
2	51	1	165	1

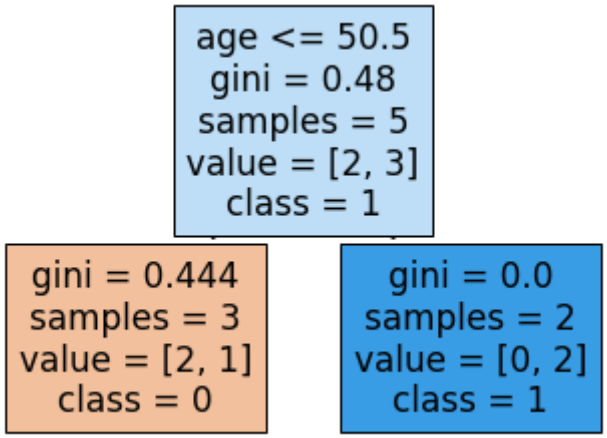
Ввод [31]:
```
gini(df2['cardio'])
```

Out[31]:
```
0.0
```

Ввод [32]:
```
features
```

Out[32]:
```
['cardio', 'gender', 'height']
```

Ввод [33]:
```
features = ['age', 'gender', 'height']
plot_tree(dt_clsf, feature_names=list(features), class_names=['0', '1'], filled=True, impurity=True);
plt.show()
```



Ввод [34]:
```
gain(df1['cardio'], df2['cardio'], gini0)
```

Out[34]:
```
0.21333333333333332
```

В случае **регрессии** разброс будет характеризоваться дисперсией или же *среднеквадратичным отклонением*, поэтому критерий информативности будет записан в виде

$$H(X) = \frac{1}{X}\sum_{i \in X}(y_i - \bar{y}(X))^2,$$

или же *среднеабсолютным отклонением*:

$$H(X) = \frac{1}{X}\sum_{i \in X}(|y_i - \bar{y}(X)|),$$

где $\bar{y}(X)$ - среднее значение ответа в выборке X:

$$\bar{y}(X) = \frac{1}{|X|}\sum_{i \in X} y_i.$$

Реализуем критерий информативности среднеквадратичного отклонения

Ввод [35]:
```
def mse(array):
    mean = array.mean()
    return np.mean((array - mean)**2)

mse(df['age'])
```

Out[35]:
```
7.760000000000001
```

Ввод [36]:
```
df1 = df[df['cardio'] <= 0.5]
```

Ввод [37]:
```
df1
```

Out[37]:

	age	gender	height	cardio
0	50	2	168	0
4	47	1	156	0

Ввод [38]:
```
mse(df1['age'])
```

Out[38]: 2.25

Ввод [39]:
```
df2 = df[df['cardio'] > 0.5]
```

Ввод [40]:
```
mse(df2['age'])
```

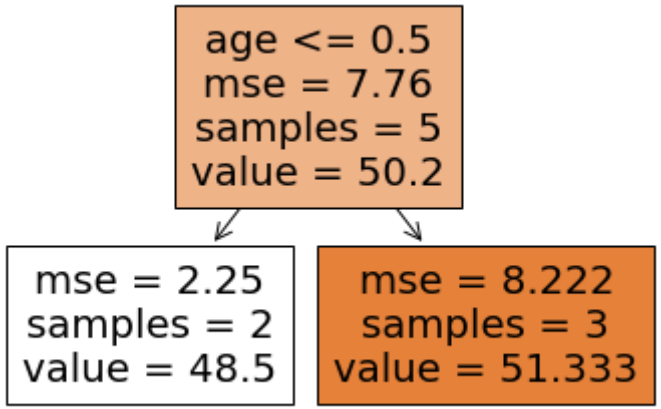
Out[40]: 8.222222222222221

Ввод [41]:
```
df2
```

Out[41]:

	age	gender	height	cardio
1	55	1	156	1
2	51	1	165	1
3	48	2	169	1

Ввод [42]:
```
plot_tree(dt_regr, feature_names=list(features), filled=True, impurity=True);
```



Ввод [43]:
```
gain(df1['age'], df2['age'], mse(df['age']))
```

Out[43]: 7.16

Критерии останова

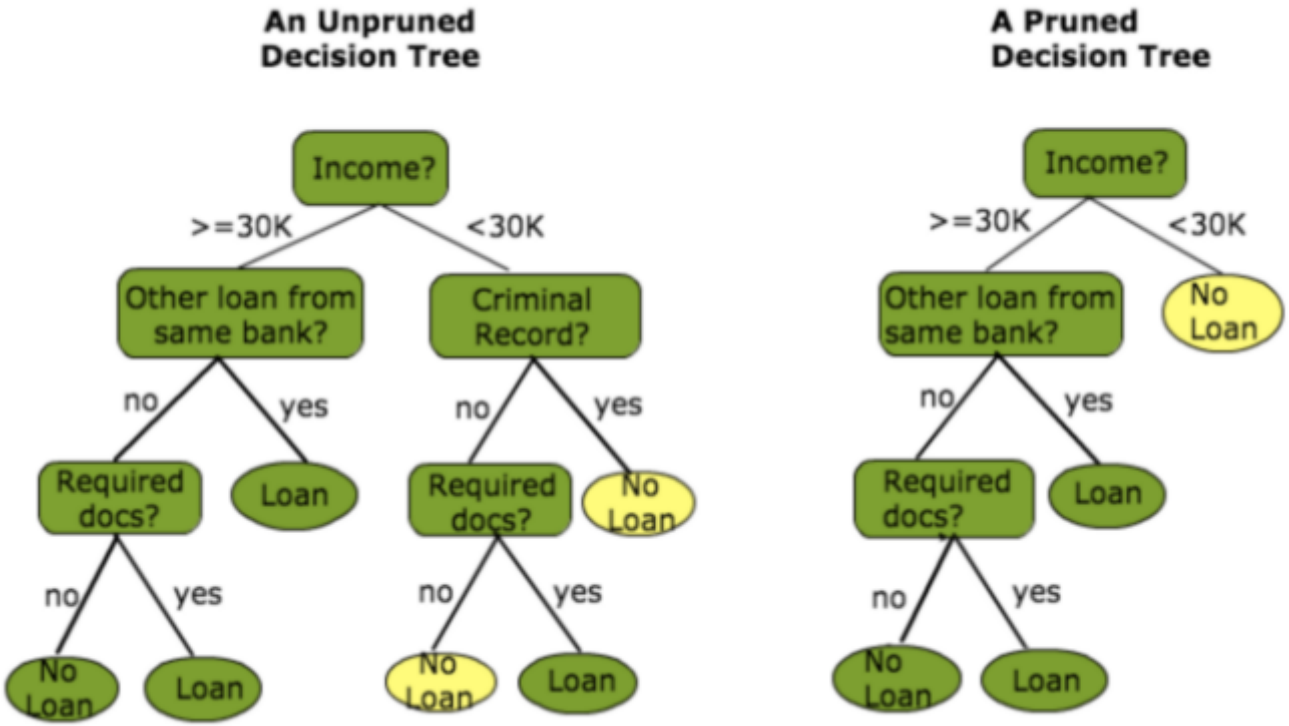
Критерии останова - это критерии, которые показывают, нужно ли остановить процесс построения дерева. Правильный выбор критериев останова роста дерева может существенно повлиять на его качество. Существует большое количество возможных ограничений:

- Ограничение максимальной глубины дерева.
- Ограничение максимального количества листьев.
- Ограничение минимального количества n объектов в листе.
- Останов в случае, когда все объекты в листе относятся к одному классу.

Подбор оптимальных критериев - сложная задача, которая обычно решается методом кросс-валидации.

Стрижка деревьев

В случае применения метода стрижки (обрезки, прунинга) деревьев использовать критерии останова необязательно, и можно строить переобученные деревья, затем снижая их сложность, удаляя листья по некоторому критерию (например, пока улучшается качество на отложенной выборке). Считается, что стрижка работает лучше, чем критерии останова.



Одним из методов стрижки является *cost-complexity pruning*. Допустим, мы построили дерево, обозначенное как T_0. В каждом из листьев находятся объекты одного класса, и значение функционала ошибки $R(T)$ при этом будет минимально на T_0. Для борьбы с переобучением к нему добавляют "штраф" за размер дерева (аналогично регуляризации, рассмотренной нами в предыдущих уроках) и получают новый функционал $R_\alpha(T)$:

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

где $|T|$ - число листьев в дереве, α - некоторый параметр регуляризации. Таким образом если при построении дерева на каком-то этапе построения алгоритма ошибка будет неизменна, а глубина дерева увеличиваться, итоговый функционал, состоящий из их суммы, будет расти.

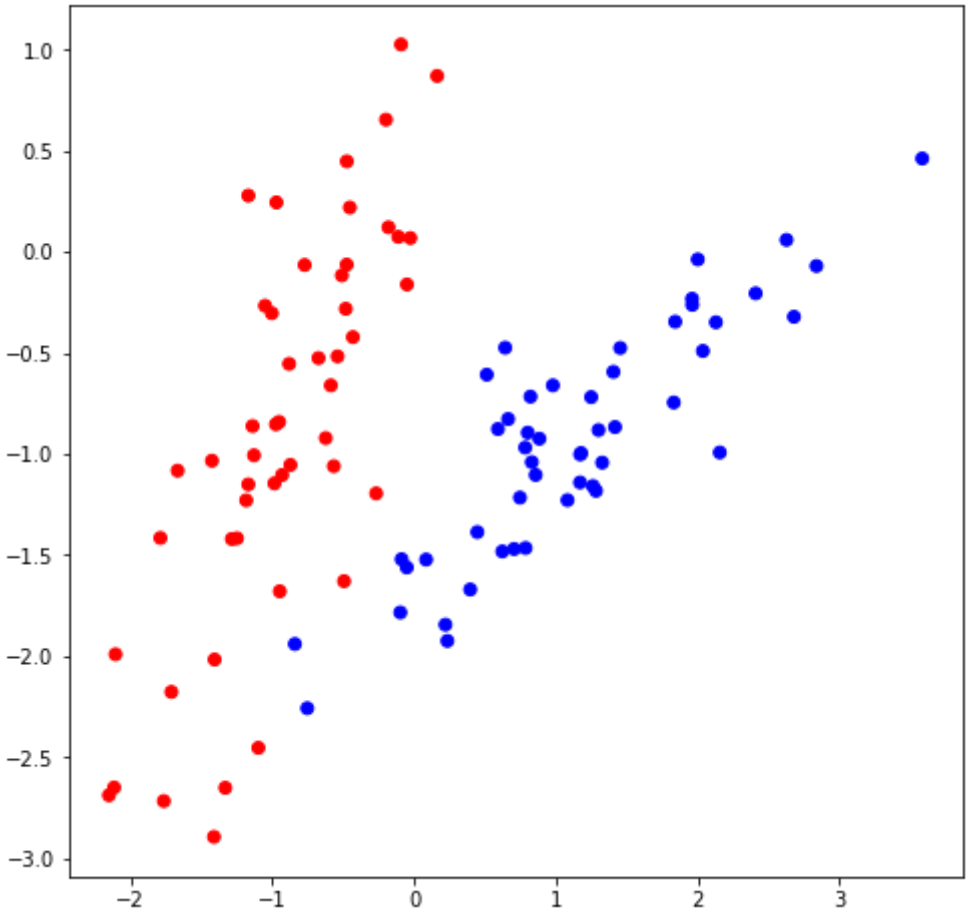
Однако стрижка деревьев обладает существенными минусами. В частности, она является очень трудоемкой процедурой. Например, она может требовать вычисления функционала качества на валидационной выборке на каждом шаге. К тому же, на данный момент одиночные деревья на практике почти не используются, а используются композиции деревьев, и в этом случае стрижка как метод борьбы с переобучением становится еще более сложным подходом. Обычно в такой ситуации достаточно использовать простые критерии останова.

Реализация дерева решений

Реализуем алгоритм алгоритм работы дерева решений своими руками.

Ввод [44]:
```
# сгенерируем данные
classification_data, classification_labels = make_classification(n_features=2, n_informative=2,
                                                                 n_classes=2, n_redundant=0,
                                                                 n_clusters_per_class=1, random_state=5)
# classification_data, classification_labels = make_circles(n_samples=30, random_state=5)
```

Ввод [45]:
```
# 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);
```



Ввод [46]:
```
# 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
```

Ввод [47]:
```
# 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
```

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

Ввод [48]:
```python
# Расчет критерия Джини

def gini(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 = 1
    for label in classes:
        p = classes[label] / len(labels)
        impurity -= p ** 2

    return impurity
```

$$H(X_m) - \frac{|X_l|}{|X_m|} H(X_l) - \frac{|X_r|}{|X_m|} H(X_r),$$

Ввод [49]:
```python
# Расчет прироста

def gain(left_labels, right_labels, root_gini):

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

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

Ввод [50]:

```
# Разбиение датасета в узле

def split(data, labels, column_index, t):

    left = np.where(data[:, column_index] <= t)
    right = np.where(data[:, column_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
```

Ввод [51]:

```python
# Нахождение наилучшего разбиения

def find_best_split(data, labels):

    #  обозначим минимальное количество объектов в узле
    min_samples_leaf = 3

    root_gini = gini(labels)

    best_gain = 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(data[:, index])

        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_samples_leaf or len(false_data) < min_samples_leaf:
                continue

            current_gain = gain(true_labels, false_labels, root_gini)

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

    return best_gain, best_t, best_index
```

Ввод [52]:

```python
import time
# Построение дерева с помощью рекурсивной функции

def build_tree(data, labels):

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

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

    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)

#     print(time.time(), true_branch)
    false_branch = build_tree(false_data, false_labels)

#     print(time.time(), false_branch)

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

Ввод [53]:

```python
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)
```

Ввод [54]:
```
def predict(data, tree):

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

Ввод [55]:
```
# Разобьем выборку на обучающую и тестовую

from sklearn.model_selection import train_test_split

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

Ввод [56]:
```
# Построим дерево по обучающей выборке
my_tree = build_tree(train_data, train_labels)
```

Ввод [57]:

```python
# 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), '<=', str(node.t))

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

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

print_tree(my_tree)
```

```
Индекс 0 <= 0.16261402870113306
--> True:
  Индекс 1 <= -1.5208896621663803
  --> True:
    Индекс 0 <= -0.9478301462477035
    --> True:
      Прогноз: 0
    --> False:
      Индекс 0 <= -0.09712237000978252
      --> True:
        Прогноз: 1
      --> False:
        Прогноз: 1
  --> False:
    Прогноз: 0
--> False:
  Прогноз: 1
```

Ввод [58]:
```
# Получим ответы для обучающей выборки
train_answers = predict(train_data, my_tree)
```

Ввод [59]:
```
# И получим ответы для тестовой выборки
answers = predict(test_data, my_tree)
```

Ввод [60]:
```
# 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
```

Ввод [61]:
```
# Точность на обучающей выборке
train_accuracy = accuracy_metric(train_labels, train_answers)
train_accuracy
```

Out[61]:
```
98.57142857142858
```

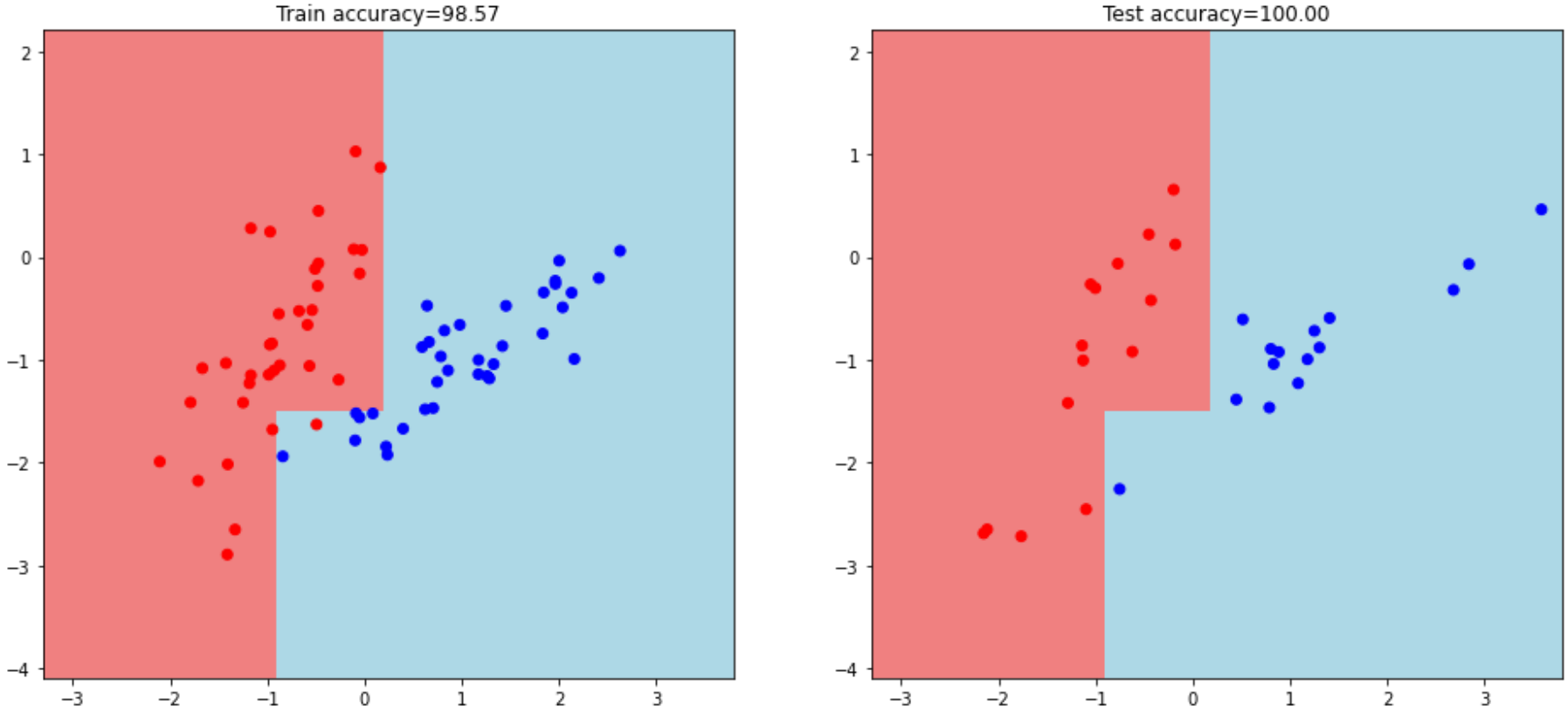
Ввод [62]:
```
# Точность на тестовой выборке
test_accuracy = accuracy_metric(test_labels, answers)
test_accuracy
```

Out[62]:
```
100.0
```

Ввод [63]:

```
# 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 visualize(train_data, test_data):
    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}')
```

Ввод [64]:
```
visualize(train_data, test_data)
```



Как видно, 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.

Пример с датасетом сердечно-сосудистых заболеваний

Ввод [147]:
```
df_full = pd.read_csv('./data/cardio.csv', sep=';')

features = ['age', 'height']
target = ['cardio']

df = df_full[features + target]
print(df.shape)

train_data, test_data, train_labels, test_labels = train_test_split(df[features].values,
                                                                    np.squeeze(df[target].values),
                                                                    test_size=0.3,
                                                                    random_state=1)
```

```
(70000, 3)
```

Ввод [64]:
```
%%time

my_tree = build_tree(train_data, train_labels)
```

```
CPU times: user 20.2 s, sys: 37.7 ms, total: 20.2 s
Wall time: 20.5 s
```

Ввод [65]:
```
print_tree(my_tree)
```

```
Индекс 0 <= 54
--> True:
  Индекс 0 <= 44
  --> True:
    Индекс 0 <= 40
    --> True:
      Индекс 1 <= 159
      --> True:
        Индекс 1 <= 152
        --> True:
          Индекс 0 <= 39
          --> True:
            Индекс 1 <= 151
            --> True:
              Индекс 1 <= 146
              --> True:
                Индекс 1 <= 142
                --> True:
                  Прогноз: 0
```

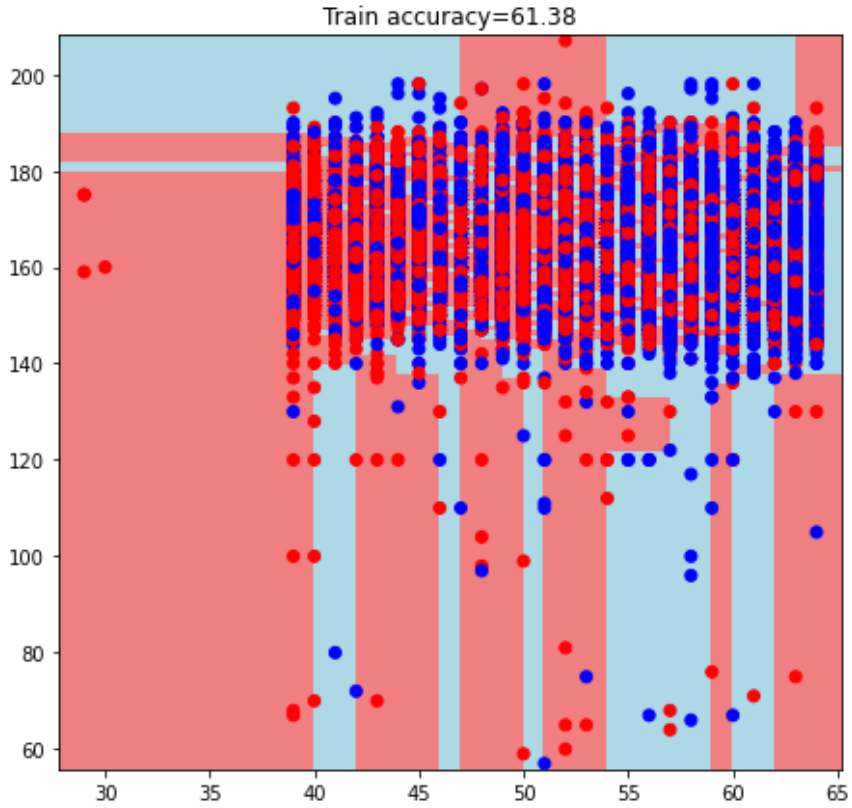
Ввод [66]:
```
train_answers = predict(train_data, my_tree)
test_answers = predict(test_data, my_tree)
```

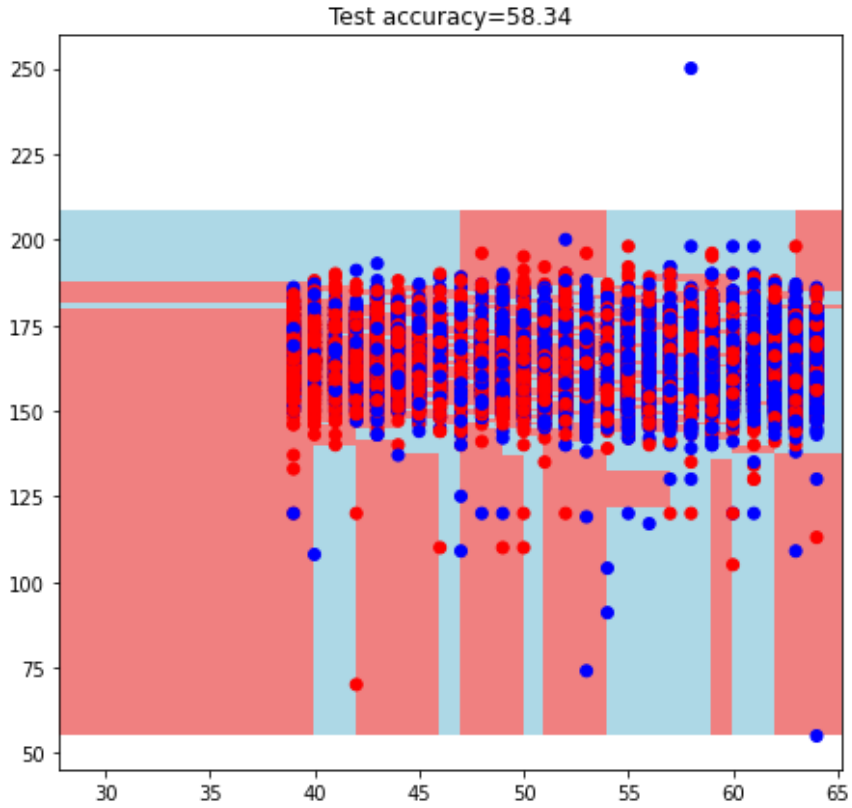
Ввод [67]:
```
train_accuracy = accuracy_metric(train_labels, train_answers)
test_accuracy = accuracy_metric(test_labels, test_answers)

print(f'Train accuracy', train_accuracy)
print(f'Test accuracy', test_accuracy)
```

```
Train accuracy 61.381632653061224
Test accuracy 58.34285714285714
```

Ввод [68]:
```
%%time
visualize(train_data, test_data)
```

```
CPU times: user 27.6 s, sys: 208 ms, total: 27.8 s
Wall time: 28.8 s
```





Ввод [69]:
```
dt = DecisionTreeClassifier()
dt.fit(train_data, train_labels)

train_answers = dt.predict(train_data)
test_answers = dt.predict(test_data)

train_accuracy = accuracy_metric(train_labels, train_answers)
test_accuracy = accuracy_metric(test_labels, test_answers)

print(f'Train accuracy', train_accuracy)
print(f'Test accuracy', test_accuracy)
```

```
Train accuracy 61.559183673469384
Test accuracy 58.357142857142854
```

Ввод [70]:
```
accuracy_score(train_answers, train_labels) * 100
```

Out[70]:
```
61.559183673469384
```

Работа деревьев в случае пропущенных значений

Иногда в реальных задачах бывает так, что не для всех объектов известно значение того или иного признака. Одним из преимуществ деревьев решений является возможность обрабатывать такие случаи.

Допустим, требуется вычислить функционал качества для разбиения $[x_j \leq t]$, но в выборке X_m для некоторого подмножества объектов V_j неизвестно значение j-го признака. В этом случае функционал качества рассчитывается без учета этих объектов (обозначим выборку без их учета как $X_m \setminus V_j$), с поправкой на потерю информации:

$$Q_{X_m,j,t} = \frac{|X_m \setminus V_j|}{|X_m|} Q(X_m \setminus V_j, j, t).$$

Если такое разбиение окажется лучшим, объекты из V_j помещаются в оба образованных поддерева.

На этапе применения дерева выполняется похожая операция. Если объект попал в вершину, в которой нельзя вычислить критерий разбиения из-за отсутствия значения необходимого признака, прогнозы для него вычисляются в обоих поддеревьях, а затем усредняются с весами, пропорциональными числу объектов в них.

$$\frac{|X_l|}{|X_m|}a_l(x) + \frac{|X_r|}{|X_m|}a_r(x),$$

где a - прогноз веорятности отнесения объекта x к одному из классов.

Добавим в выборку пропущенные значения

Ввод [71]:
```
random_indices = np.random.randint(0, df.shape[0], 1000)
df.loc[random_indices, ['height']] = np.nan
df.isna().sum()
```

Out[71]:
```
age         0
height    990
cardio      0
dtype: int64
```

Получим значения критерия Джини

Ввод [148]:
```
root_gini = gini(df['cardio'])
root_gini
```

Out[148]:
```
0.49999982000000004
```

Ввод [149]:
```
t = df['height'].median()

df_clean = df[~df['height'].isna()]
```

Ввод [150]:
```
df1 = df_clean[df_clean['height'] <= t]
df2 = df_clean[df_clean['height'] > t]
```

Ввод [151]:
```
gini1 = gini(df1['cardio'])
gini2 = gini(df2['cardio'])
gini1, gini2
```

Out[151]:
```
(0.4999896791652597, 0.49997134448585223)
```

Получим значение прироста информации

Ввод [152]:
```
current_gain = gain(df1['cardio'], df2['cardio'], root_gini)
current_gain
```

Out[152]:
```
1.7925477692748437e-05
```

Сделаем поправку на потерю информации

Ввод [153]:
```
df_clean.shape[0] / df.shape[0] * current_gain
```

Out[153]:
```
1.7925477692748437e-05
```

Если разбиение лучшее, то наблюдения с nan добавляются в обе ветки

Ввод [78]:
```
df1 = df1.append(df[df['height'].isna()])
df2 = df2.append(df[df['height'].isna()])
```

Работа деревьев с категориальными признаками

Кроме вещественных и бинарных признаков в задаче могут иметь место категориальные признаки (делящиеся на конечное число категорий, например, цвета автомобилей). Самый простой способ учета категориальных признаков в алгоритме деревьев состоит в разбитии вершины на столько поддеревьев, сколько имеется возможных значений признака. В этом случае дерево называется *n-арным*. Условие разбиения будет простым (отнесение признака к той или иной категории), однако здесь появляется риск получения конечного дерева с очень большим числом листьев. В случае такого дерева критерий ошибки Q будет состоять из n слагаемых (или из $(n+1)$) в случае максимизируемого критерия, который мы использовали.

Есть и другой подход, заключающийся в формировании бинарных деревьев путем разделения множества значений признака $C = \{c_1, \ldots, c_n\}$ на два непересекающихся подмножества C_1 и C_2. После такого разделения условием разбиения в узле будет проверка принадлежности признака одному из подмножеств $[x \in C_1]$.

Задача остается в выборе оптимального варианта разбиения исходного множества на два подмножества, так как обычный перебор всех вариантов может быть крайне затруднительным из-за большого количества вариантов разбиения. В случаях с бинарной классификацией и регрессией используют следующий метод: все возможные значения категориального признака сортируются по определенному принципу, затем заменяются на натуральные числа.

В случае бинарной классификации признаки упорядочиваются на основе того, какая доля объектов с такими признаками относится к классу +1. Если обозначить множество объектов в узле m, у которых j-й признак имеет значение c, через $X_m(c)$, а через $N_m(c)$ количество таких объектов, получим:

$$\frac{1}{N_m(c_1)} \sum_{x \in X_m(c_1)} [y_i = +1] \leq \ldots \leq \frac{1}{N_m(c_n)} \sum_{x \in X_m(c_n)} [y_i = +1],$$

и после замены категории c_i на натуральное число ищется разбиение как для вещественного признака.

В случае задачи регрессии сортировка происходит схожим образом, но вместо доли объектов положительного класса среди объектов с таким значением признака вычисляется средний ответ по объектам с соответствующим значением категориального признака:

$$\frac{1}{N_m(c_1)} \sum_{x \in X_m(c_1)} y_i \leq \ldots \leq \frac{1}{N_m(c_n)} \sum_{x \in X_m(c_n)} y_i.$$

Для классификации

Ввод [79]:
```
colors = ['gray', 'blue', 'green']
new_feature = []
for i in range(df.shape[0]):
    new_feature.append(np.random.choice(colors, p=['0.5', '0.2', '0.3']))

df['eye_color'] = new_feature

df
```

Out[79]:

	age	height	cardio	eye_color
0	50	168.0	0	gray
1	55	156.0	1	gray
2	51	165.0	1	green
3	48	169.0	1	blue
4	47	156.0	0	gray
...	...	...	...	...
69995	52	168.0	0	gray
69996	61	158.0	1	gray
69997	52	183.0	1	green
69998	61	163.0	1	blue
69999	56	170.0	0	green

70000 rows × 4 columns

Ввод [80]:
```
df[df['cardio'] == 1]['eye_color'].value_counts()
```

Out[80]:
```
gray     17350
green    10569
blue      7060
Name: eye_color, dtype: int64
```

Ввод [81]:
```
df['eye_color'].replace({'gray': 3, 'green': 2, 'blue': 1})
```

Out[81]:
```
0        3
1        3
2        2
3        1
4        3
        ..
69995    3
69996    3
69997    2
69998    1
69999    2
Name: eye_color, Length: 70000, dtype: int64
```

Для регрессии

Ввод [82]:
```
df.groupby('eye_color').mean()['age'].sort_values(ascending=False)
```

Out[82]:
```
eye_color
green    52.859112
blue     52.839765
gray     52.829851
Name: age, dtype: float64
```

Ввод [83]:
```
df['eye_color'].replace({'gray': 1, 'green': 3, 'blue': 2})
```

Out[83]:
```
0        1
1        1
2        2
3        3
4        1
        ..
69995    1
69996    1
69997    2
69998    3
69999    2
Name: eye_color, Length: 70000, dtype: int64
```

Домашнее задание

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

2. *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.

Проект*:

1. https://www.kaggle.com/c/gb-tutors-expected-math-exam-results (https://www.kaggle.com/c/gb-tutors-expected-math-exam-results) регрессия
2. https://www.kaggle.com/c/gb-choose-tutors (https://www.kaggle.com/c/gb-choose-tutors) классификация

Дополнительные материалы

1. Энтропия (https://habr.com/ru/post/305794/)

2. Энтропия - теоретическое обоснование (https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%
3. Cost-Complexity Pruning (http://mlwiki.org/index.php/Cost-Complexity_Pruning)
4. Реализация дерева решений в функциональном стиле (https://github.com/random-forests/tutorials/blob/master/decision_tree.ipynb)
5. ООП-реализация дерева решений (https://github.com/curiousily/Machine-Learning-from-Scratch/blob/master/3_decision_trees.ipynb)
6. Пример работы дерева решений в задаче регрессии (https://habr.com/ru/company/ods/blog/322534/#derevo-resheniy-v-zadache-

Summary

- Решающее дерево - последовательное построение узлов, разбивающих множество входящих объектов согласно принципу минимизации неопределенности узла
- Вопросы для разбиения в узле выбираются из всего множества признаков
- Деревья могут легко переобучиться под выборку, если не ограничивать их глубину
- Деревья очень чувствительны к небольшим изменениям в выборке (шумам)

Определения

Дерево решений

Дерево решений — это математическая модель в виде графа, которая отображает точки принятия решений, предшествующие им события и последствия.

Вершина — узел, показывающий решение, которое нужно принять.

Лист — терминальный узел, показывающий конечный результат пути решения.

Энтропия — мера неопределённости некоторой системы.

Прирост информации — величина обратная энтропии, чем выше прирост информации, тем меньше энтропия, меньше неучтенных данных и лучше решение.