Постановка задачи

Реализация алгоритма **CART** для построения дерева решений. На каждом этапе работы алгоритма вычисляется подходящий признак для разбиения на основе анализа **impurity** узлов. Необходимо показать выбранный признак и его значение impurity.

Теоретическая справка

Мы будем предполагать, что из каждой вершины решающего дерева исходит либо две дуги, либо не исходит ни одной. Вершины, из которых не исходит ни одной дуги, мы будем называть **листьями**. Кроме того, существует единственная вершина, в которую не входит ни одна дуга. Эту вершину мы будем называть **корнем дерева**. Отсюда следует, что из корня дерева до любой вершины дерева существует единственный путь.

Важная числовая характеристика дерева – **глубина дерева** определяется как максимальная длина пути в этом дереве.

Пусть D – это некоторой массив, на котором определена функция

$$P : D \to \{True, False\}$$

Эта функция P называется **предикатом**. Решающее дерево применяется для решения задачи представления (описания) массива D, на котором можно задать множество предикатов. Тогда каждый предикат P делит массив D на два подмассива D_T (левая дуга) и D_F (правая) следующим образом

$$D_T = \{x \in D : P(x) = True\},$$

$$D_F = \{x \in D : P(x) = False\}.$$

В вершинах, не являющихся листьями, решающего дерева располагаются предикаты, а на дугах подмассивы массива D. Будем предполагать, что предикат в корневой вершине дерева применяется ко всему массиву D. На листьях мы имеем подмассив массива D. Если массив D конечный, то всегда можно построить решающее дерево таким образом, что листья этого дерева будут содержать по одному элементу.

Основная проблема при построении решающих деревьев состоит в том, как выбрать предикаты таким образом, чтобы глубина этого дерева была минимальной. Для решения этой задачи используются методы основанные на минимизации энтропии или максимизировании информации.

Пусть мы имеем дело с дискретными данными, тогда каждый элемент множества D имеет частоту встречаемости в массиве D, которую мы будем называть вероятностью. Пусть каждый элемент $d_k \in D$ имеет вероятность p_k. Тогда информационный **критерий Джини** (Gini impurity) определяется по формуле

$$G(j, t) = 1 - \sum_k (p_k)^2,$$

где j — признак, t — порог, по которым объекты делятся на две подвыборки.

Пусть у нас есть массив D, который необходимо разделить на два подмассива с помощью одного из предикатов P. Пусть множество всех доступных предикатов обозначено через $\prod$. Тогда P^* будет наиболее информативным, если

$$P^* = \arg\max_{P \in \prod} InformationGain(P).$$

Или иначе

$$j^*, t^* = \arg\max_{j \in J, t \in T} InformationGain(j, t).$$

О датасете

Выбранный датасет представляет собой реальные данные о классификации зерен трех разных сортов пшеницы на основе их геометрических свойств [1]. У каждого случайно выбранного для эксперимента представителя выборки выделены следующие 7 признаков:

- **Площадь** A
- **Периметр** P
- $C = \frac{4\pi A}{P^2}$
- **Длина зерна**
- **Ширина зерна**
- **Коэффициэнт ассиметрии**
- **Длина желобка**

Целевым признаком являются **сорта** пшеницы : “Kama”, “Rosa”, “Canadian”.

В обучающей выборке данные о 150 зернах (по 50 на каждый сорт). В тестовом датасете данные о 60 объектах (в каждом классе по 20).

Использование алгоритма

В результате использования реализованного алгоритма CART на описанном датасете получаем:

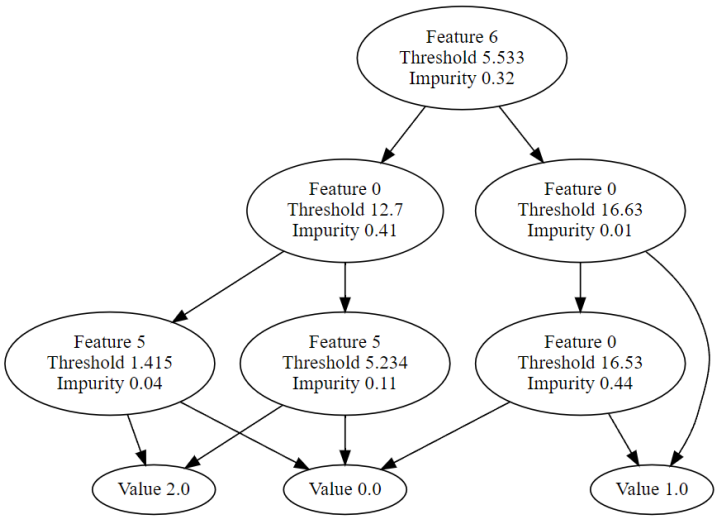

Рис. 1: Полученное дерево максимальной глубины 3

max_depth	**1**	**2**	**3**	**4**	**5**
Train	0.33	0.96	0.99	1.0	1.0
Test	0.33	0.82	0.83	0.85	0.85

Таблица 1: Точность деревьев решений для различных значений уровней

Оптимальное число уровней — 4, большее значение не дает выигрыша по accuracy.

Проверим устойчивость построенного дерева: удалим по пять записей каждого класса из обучающего множества и проверим работу (качество классификации) дерева решений на той же тестовой выборке:

	Train size	**Test size**	**Accuracy**	**Right**	**Wrong**
1	135	60	0.85	51	9
2	120		0.85	51	9
3	105		0.83	50	10

Таблица 2: Проверка устойчивости решения при max_depth = 4

Видим, что построенное решение достаточно устойчиво.

Заключение

В работе проведен анализ датасета, содержащего информацию о геометрических характеристиках зерен пшеницы. Применен алгоритм CART для построения дерева решений, которое достигло максимальной точности на тестовой выборке при глубине 4. Построенное при глубине 4 решение оказалось устойчивым при удалении части данных из обучающей выборки.

Список литературы

[1] Charytanowicz,Magorzata, Niewczas,Jerzy, Kulczycki,Piotr, Kowalski,Piotr, and Lukasik,Szymon. (2012). seeds. UCI Machine Learning Repository. DOI

Листинг программы

```
import numpy as np
import pandas as pd
from graphviz import Digraph


def split(X, k, feature):
    """ Split source data (X) into two part depends on threshold (k) and feature.
    """
    Left = X[X[:, feature] <= k]
    Right = X[X[:, feature] > k]
    return Left, Right

def Gini(X):
    """ Compute Gini information criterion of X.
    """
    _, counts = np.unique(X[:, -1], return_counts=True)
    probabilities = counts / len(X)
    impurity = 1 - np.sum(probabilities**2)
    return impurity

def IG_Gini(X, k, feature):
    """ Compute Gini information gain of child nodes.
    """
    Left, Right = split(X, k, feature)

    Left_weight = len(Left) / (len(Left) + len(Right))
    Right_weight = 1 - Left_weight

    return Gini(X) - (Left_weight * Gini(Left) + Right_weight * Gini(Right))

def get_opt_split(X):
    """ Find optimal split of X by optimal feature and optimal threshold.
    """
    Optk = sorted(np.unique(X[:, 0]))[0]
    num_features = X.shape[1] - 1
    OptFeature = 0
    OptGini = IG_Gini(X, Optk, OptFeature) # we maximize Gini information gain to find optimal

    for feature in range(num_features):
        if feature == 0:
            possible_splits = sorted(np.unique(X[:, feature]))[1:]
        else:
            possible_splits = sorted(np.unique(X[:, feature]))
        for k in possible_splits:
            G = IG_Gini(X, k, feature)

            if G > OptGini:
                OptGini = G
                Optk = k
                OptFeature = feature
    return OptGini, Optk, OptFeature

def build_tree(X, max_depth, current_depth=0, tree=None):
    """ Recursive function to build the decision tree (tree) with max possible depth (max_depth).
        Return: tree as dict
    """
    if tree is None:
        tree = {}

    OptGini, Optk, OptFeature = get_opt_split(X)

    if current_depth == max_depth or OptGini == 0:
        tree['leaf'] = True
        tree['value'] = np.round(np.mean(X[:,-1]))
        return tree

    tree['feature'] = OptFeature
    tree['threshold'] = Optk
    tree['impurity'] = OptGini

    Left, Right = split(X, Optk, OptFeature)

    tree['left'] = {}
    tree['right'] = {}

    build_tree(Left, max_depth, current_depth + 1, tree=tree['left'])
    build_tree(Right, max_depth, current_depth + 1, tree=tree['right'])
```

```
    return tree

def predict(tree, X):
    """ Predict label of X.
    """
    if 'leaf' in tree:
        return tree['value']
    else:
        if X[tree['feature']] <= tree['threshold']:
            return predict(tree['left'], X)
        else:
            return predict(tree['right'], X)

def accuracy(tree, data):
    """ Compute accuracy score.
    """
    predictions = [predict(tree, x[:-1]) for x in data]
    actual_labels = data[:, -1]

    correct_predictions = np.sum(predictions == actual_labels)
    total_samples = len(data)

    accuracy = correct_predictions / total_samples
    return accuracy


target_index = {'Kama': 0, 'Rosa': 1, 'Canadian': 2}
df = pd.read_csv('train.csv')
df['Target'] = df['Target'].apply(lambda x: target_index[x])
train = df.to_numpy()
test_df = pd.read_csv('test.csv')
test_df['Target'] = test_df['Target'].apply(lambda x: target_index[x])
test = test_df.to_numpy()
tree = build_tree(train, 4)
print(accuracy(tree, train))
print(accuracy(tree, test))

# compare our implementation with sklearn implementation
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
max_depth = 4
clf = DecisionTreeClassifier(max_depth=max_depth)
clf.fit(train[:,:-1], train[:, -1])
y_pred_1 = clf.predict(train[:,:-1])
y_pred_2 = clf.predict(test[:,:-1])
print(accuracy_score(train[:, -1], y_pred_1))
print(accuracy_score(test[:, -1], y_pred_2))

def plot_tree(tree, dot=None, parent_name=None, graph=None):
    """ Visualize tree as digraph.
    """
    if dot is None:
        dot = Digraph(comment='Decision Tree')

    if tree.get('feature') is not None:
        node_name = f"Feature {tree['feature']}\nThreshold {tree['threshold']}\nImpurity {round(tree['impurity'], 2)}"
    else:
        node_name = f"Value {tree['value']}"

    dot.node(node_name)

    if parent_name is not None:
        dot.edge(parent_name, node_name)

    for child in ('left', 'right'):
        if child in tree:
            plot_tree(tree[child], dot, node_name, graph=graph)

    return dot

plot_tree(tree)
```