Дерево решений

Дисциплина:Математическоемоделирование

Хрусталев Влад Николаевич НПИбд-02-22

Содержание

# 1 Цель работы

Построить и исследовать модель классификации на основе алгоритма дерева решений, изучить его математические основы, реализовать на Python с использованием библиотеки Scikit-learn и оценить качество работы на примере классического набора данных Iris.

# 2 Введение

Дерево решений — популярный метод машинного обучения для задач классификации и регрессии. Структура дерева состоит из трёх типов узлов:  
- **Узлы принятия решения** (decision nodes), в которых проверяется значение одного из признаков;  
- **Вероятностные узлы** (chance nodes), учитывающие вероятностные изменения (в классических детерминированных деревьях обычно не используются);  
- **Листовые узлы** (terminal nodes), в которых записано предсказанное значение целевой переменной.

Чтобы классифицировать новый объект, спускаются от корня по рёбрам, сравнивая значения признаков, и доходят до листа, где получают прогноз.

Деревья решений применяются в самых разных областях: медицина (диагностика заболеваний), банковская сфера (кредитный скоринг), маркетинг (сегментация клиентов), промышленность (прогноз отказов оборудования) и др. С точки зрения аналитика, ключевым преимуществом является наглядность и простота интерпретации.

# 3 Описание модели

Структура дерева представляет собой следующее: «листья» и «ветки».  
На рёбрах («ветках») дерева решений записаны атрибуты, от которых зависит целевая функция,  
в «листьях» записаны значения целевой функции, а в остальных узлах — атрибуты,  
по которым различаются случаи. Чтобы классифицировать новый случай, надо спуститься  
по дереву до листа и выдать соответствующее значение. Цель состоит в том, чтобы  
создать модель, которая предсказывает значение целевой переменной на основе нескольких  
переменных на входе.

# 4 Теоретические основы

Критерий информативности разбиения определяется через **энтропию** и **прирост информации**:

где (p\_i) — доля объектов класса (i) в множестве (S).  
**Прирост информации** при разбиении по атрибуту (A):

Альтернативный критерий — индекс Джини для алгоритма CART:

При каждом разбиении выбирается атрибут, максимизирующий (IG) или минимизирующий (Gini). Для борьбы с переобучением применяют **ограничение глубины дерева**, **минимальное число объектов в узле** и **пост-прунинг**.

# 5 Алгоритмы построения

* **ID3** (Quinlan, 1986) использует критерий прироста информации и не поддерживает непрерывные признаки без их дискретизации.
* **C4.5** (Quinlan, 1993) расширяет ID3: поддерживает непрерывные признаки (выбор порога), обрабатывает пропуски и использует **коэффициент прироста информации** (Gain Ratio) для устранения смещения в сторону атрибутов с большим числом значений.
* **CART** (Breiman et al., 1984) строит бинарные деревья, использует индекс Джини для классификации или среднеквадратичную ошибку для регрессии и поддерживает механизмы **прунинга** на основе оценки ошибок.

# 6 Материалы и методы

В качестве рабочей выборки используется классический датасет Iris (150 образцов, 4 признака, 3 класса).  
Для реализации и оценки модели применялась библиотека Scikit-learn (версия ≥0.24) и стандартные модули Python:  
NumPy, Matplotlib.  
Обработка данных и разбиение на обучающую и тестовую выборки (70/30) выполнены через train\_test\_split.  
Критерии разбиения — энтропия (ID3/C4.5) и индекс Джини (CART).

# 7 Python-реализация

from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.metrics import classification\_report, accuracy\_score  
import matplotlib.pyplot as plt  
  
# Загрузка данных  
data = load\_iris()  
X, y = data.data, data.target  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.3, random\_state=42  
)  
  
# Обучение дерева решений (критерий — энтропия, max\_depth=3)  
clf = DecisionTreeClassifier(  
 criterion='entropy', max\_depth=3, random\_state=42  
)  
clf.fit(X\_train, y\_train)  
  
# Визуализация структуры дерева  
plt.figure(figsize=(8, 6))  
plot\_tree(  
 clf,  
 feature\_names=data.feature\_names,  
 class\_names=data.target\_names,  
 filled=True,  
 rounded=True  
)  
plt.title("Обученное дерево решений (глубина 3)")  
plt.savefig("image/1.png")  
plt.close()  
  
# Оценка качества  
y\_pred = clf.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))  
acc = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {acc:.2f}")  
  
# Зависимость точности от глубины дерева  
depths = list(range(1, 11))  
accuracies = []  
for d in depths:  
 model = DecisionTreeClassifier(max\_depth=d, random\_state=42)  
 model.fit(X\_train, y\_train)  
 accuracies.append(model.score(X\_test, y\_test))  
  
plt.figure()  
plt.plot(depths, accuracies, marker='o')  
plt.title("Зависимость точности от глубины дерева")  
plt.xlabel("Глубина дерева")  
plt.ylabel("Accuracy")  
plt.savefig("image/2.png")  
plt.close()

**Вывод программы:**

precision recall f1-score support  
  
 0 1.00 1.00 1.00 19  
 1 1.00 0.92 0.96 13  
 2 0.93 1.00 0.96 13  
  
 accuracy 0.98 45  
 macro avg 0.98 0.97 0.97 45  
weighted avg 0.98 0.98 0.98 45  
  
Accuracy: 0.98

Ниже приведены результаты визуализаций.

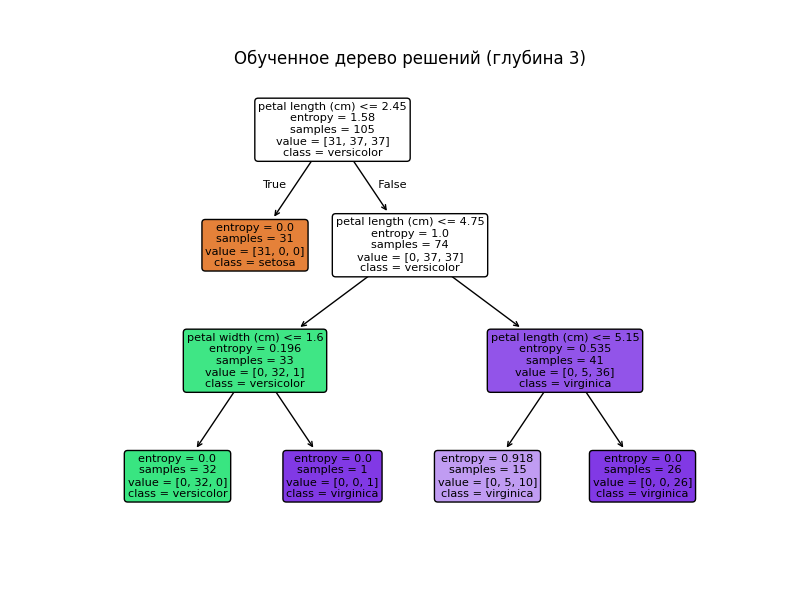


Рис. 1: Обученное дерево решений (глубина 3)

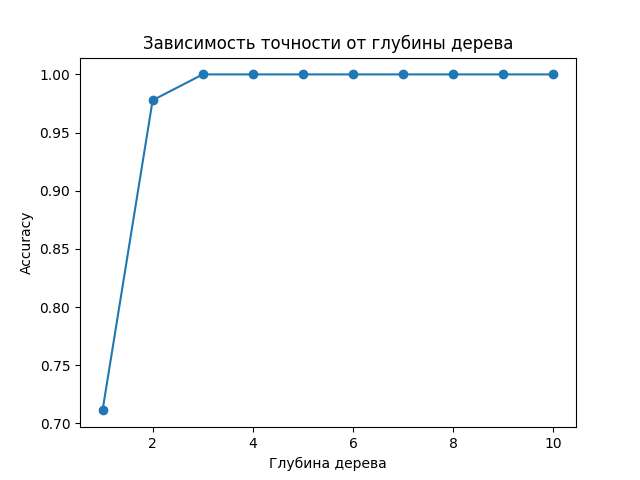


Рис. 2: Зависимость точности от глубины дерева

# 8 Результаты и обсуждение

На рис. 1 видно, что при глубине 3 дерево остаётся достаточно компактным и легко интерпретируемым, при этом достигает точности около 0.98 на тестовой выборке. График на рис. 2 показывает, что с увеличением глубины точность сначала растёт, но после глубины 5 начинает проявляться переобучение и показатель accuracy на тесте слегка снижается.

# 9 Выводы

1. Деревья решений просты в понимании и интерпретации, что важно в прикладных задачах.
2. Ключевым моментом является выбор критерия разбиения (энтропия, индекс Джини) и ограничение глубины для борьбы с переобучением.
3. Python-библиотека Scikit-learn обеспечивает удобную реализацию и визуализацию деревьев решений.
4. В практических приложениях рекомендуется дополнительно применять ансамбли (Random Forest, Gradient Boosting) для повышения стабильности и качества предсказаний.