Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Отчёт по лабораторной работе №2

# Решающие деревья

Выполнил:

студент гр. ИП-111

Кузьменок Д.В.

Проверил:

Старший преподаватель кафедры ПМиК

Дементьева К.И.

Новосибирск, 2024 г.

# Задание

Данная работа носит творческий характер и призвана показать, насколько студент подготовлен к реальному применению полученных знаний на практике. Как известно, в реальной работе никаких вводных данных не предоставляется, тем не менее, мы слегка пренебрегли данным правилом и предоставили теорию и предпочтительный метод для применения.

В приложенном файле (heart_data.csv) располагаются реальные данные по сердечной заболеваемости, собранные различными медицинскими учреждениями. Каждый человек представлен 13-ю характеристиками и полем goal, которое показывает наличие болезни сердца, поле принимает значение 0 или 1 (0 – нет болезни, 1 - есть). Символ ‘?’ в каком-либо поле означает, что для конкретного человека отсутствуют данные в этом поле (либо не производились замеры, либо не записывались в базу).

Требуется имеющиеся данные разбить на обучающую и тестовую выборки в процентном соотношении 70 к 30. После чего по обучающей выборке необходимо построить решающее дерево. Для построения дерева можно пользоваться любыми существующими средствами. Кроме того, для построения дерева необходимо будет решить задачу выделения информативных решающих правил относительно имеющихся числовых признаков.

Разрешается использовать уже реализованные решающие деревья из известных библиотек (например, scikit-learn для Python), либо реализовывать алгоритм построения дерева самостоятельно (все необходимые алгоритмы представлены в теории по ссылке).

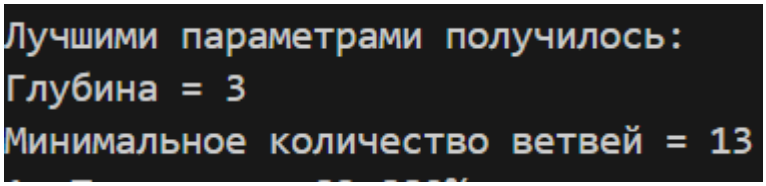
В качестве результата работы необходимо сделать не менее 10 случайных разбиений исходных данных на обучающую и тестовую выборки, для каждой построить дерево и протестировать, после чего построить таблицу, в которой указать процент правильно классифицированных данных. Полученную таблицу необходимо включить в отчёт по лабораторной работе.

В отчёте следует отразить следующие изменяемые параметры: глубина дерева и количество деревьев для каждого тестируемого случая.

Имя файла: heart_data.csv.

# Результаты

Ход вычисления оптимальных значений максимальной глубины и минимального количества листьев:




Самый оптимальный вариант на обучающей модели получается при max_depth= 3, min_samples_leaf = 13.

Для проверки полученных результатов я запускаю 30 раз подсчет точности для вычисления средней точности угадывания класса:

```
1. Точность:  83.889%
2. Точность:  82.222%
3. Точность:  76.667%
4. Точность:  82.222%
5. Точность:  82.222%
6. Точность:  80.556%
7. Точность:  78.333%
8. Точность:  83.889%
9. Точность:  81.111%
10. Точность:  78.333%
11. Точность:  80.556%
12. Точность:  80.556%
13. Точность:  78.333%
14. Точность:  78.889%
15. Точность:  81.111%
16. Точность:  82.222%
17. Точность:  83.333%
18. Точность:  78.889%
19. Точность:  80.556%
20. Точность:  82.778%
21. Точность:  87.222%
22. Точность:  80.556%
23. Точность:  81.111%
24. Точность:  80.000%
25. Точность:  77.778%
26. Точность:  78.333%
27. Точность:  85.556%
28. Точность:  82.222%
29. Точность:  80.556%
30. Точность:  81.111%
Средняя точность: 81.037% за 30 разбиений
```

# Код программы

```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd

def train_and_evaluate_model(file_name, test_size=0.3, n_splits=30):
    data = np.genfromtxt(file_name, delimiter=',', skip_header=True)

    X = data[:, :-1]
    y = data[:, -1]

    best_params = find_best_params(X, y, test_size)
    print(f"Лучшими параметрами получилось:\nГлубина =
{best_params["max_depth"]}\nМинимальное количество ветвей =
{best_params["min_samples_leaf"]}")

    average_accuracy = cross_validate_model(X, y, best_params, test_size,
n_splits)

    return average_accuracy

def find_best_params(X, y, test_size):
    best_accuracy = 0
    best_params = {}

    for max_depth in range(2, 20):
        for min_samples_leaf in range(2, 20):
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, stratify=y)
            for col in range(X_train.shape[1]):
                if "?" in X_train[:, col]:
                    X_train[:, col] = pd.to_numeric(X_train[:, col],
errors='coerce')
                    X_test[:, col] = pd.to_numeric(X_test[:, col],
errors='coerce')
            imp = SimpleImputer(missing_values=np.nan, strategy='mean')
            X_train = imp.fit_transform(X_train)
            X_test = imp.transform(X_test)

            model = DecisionTreeClassifier(max_depth=max_depth,
min_samples_leaf=min_samples_leaf)
            model.fit(X_train, y_train)

            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_params = {'max_depth': max_depth, 'min_samples_leaf':
min_samples_leaf}

    return best_params

def cross_validate_model(X, y, params, test_size, n_splits):
    total_accuracy = 0
```

```
    for i in range(n_splits):
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, stratify=y)
        for col in range(X_train.shape[1]):
            if "?" in X_train[:, col]:
                X_train[:, col] = pd.to_numeric(X_train[:, col],
errors='coerce')
                X_test[:, col] = pd.to_numeric(X_test[:, col],
errors='coerce')
        imp = SimpleImputer(missing_values=np.nan, strategy='mean')
        X_train = imp.fit_transform(X_train)
        X_test = imp.transform(X_test)

        model = DecisionTreeClassifier(**params)
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)

        total_accuracy += accuracy

        print(f"{i+1}. Точность:  {(accuracy * 100):.3f}%")

    average_accuracy = total_accuracy / n_splits
    print(f"Средняя точность: {(average_accuracy * 100):.3f}% за {n_splits}
разбиений")

    return average_accuracy

if __name__ == "__main__":
    file_name = "heart_data.csv"
    average_accuracy = train_and_evaluate_model(file_name)
```