Министерство образования Республики Беларусь
Учреждение образования
"Брестский государственный технический университет"
Кафедра ИИТ

Лабораторная работа №5

По дисциплине "Интеллектуальный анализ данных"

Тема: «Деревья решений»

Выполнил:

Студент 4 курса

Группы ИИ-24

Бузель С.Д.

Проверил:

Андренко К. В.

Брест 2025

Цель: На практике сравнить работу нескольких алгоритмов одиночного дерева решений, случайного леса и бустинга для деревьев решений.

Задание:

1. Загрузить датасет по варианту;
2. Разделить данные на обучающую и тестовую выборки;
3. Обучить на обучающей выборке одиночное дерево, случайный лес и реализовать бустинг для решающих деревьев (AdaBoost, CatBoost, XGBoost);
4. Оценить точность каждой модели на тестовой выборке;
5. Сравнить результаты, сделать выводы о применимости каждого метода для данного набора данных.

Вариант 3

- `Wine Quality`
- Классифицировать вино на "хорошее" (оценка >= 7) и "обычное" (оценка < 7)
- **Задания:**
    1. Загрузите данные и создайте бинарную целевую переменную;
    2. Стандартизируйте признаки и разделите выборку;
    3. Обучить на обучающей выборке одиночное дерево, случайный лес и реализовать бустинг для решающих деревьев (AdaBoost, CatBoost, XGBoost);
    4. Сравните `F1-score` для каждой модели, так как классы могут быть несбалансированы;
    5. Определите, какой алгоритм показал наилучший баланс между точностью и полнотой.

Код программы:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import f1_score
import xgboost as xgb
import catboost as cb

# --- 1. Загрузка и подготовка данных ---

file_path = "D:/OIIS/lab5/WineQT.csv"

try:
    data = pd.read_csv(file_path)

    # Создаем бинарную целевую переменную "quality_binary"
    # 1 если качество >= 7 "хорошее" и 0 если < 7 "обычное"
    data['quality_binary'] = data['quality'].apply(lambda x: 1 if x >= 7 else 0)
```

```
    # Удаляем исходный столбец "quality" и бесполезный "Id"
    data = data.drop(['quality', 'Id'], axis=1)

    # Разделяем данные на признаки X и целевую переменную y
    X = data.drop('quality_binary', axis=1)
    y = data['quality_binary']

    print("Данные успешно загружены и подготовлены.")
    print(f"Количество 'хороших' вин (1): {y.sum()}")
    print(f"Количество 'обычных' вин (0): {len(y) - y.sum()}")

except FileNotFoundError:
    print(f"Ошибка: Файл не найден по пути {file_path}")
    X, y = None, None

if X is not None:
    # --- 2. Разделение выборки и стандартизация признаков ---

    # Разделяем данные на обучающую 80% и тестовую 20% выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y) # stratify=y
гарантирует, что пропорция классов будет одинаковой в обеих выборках

    # Стандартизируем признаки
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print("\nДанные разделены на обучающую и тестовую выборки и стандартизированы.")

    # --- 3. Обучение моделей ---

    # Создаем словарь с моделями для удобства
    models = {
        "Одиночное дерево": DecisionTreeClassifier(random_state=42),
        "Случайный лес": RandomForestClassifier(random_state=42),
        "AdaBoost": AdaBoostClassifier(random_state=42),
        "XGBoost": xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss'),
        "CatBoost": cb.CatBoostClassifier(random_state=42, verbose=0) # verbose=0 отключает вывод логов
обучения
    }

    # Словарь для хранения результатов
    results = {}

    for name, model in models.items():
        print(f"\nОбучение модели: {name}...")

        # Обучаем модель на стандартизированных обучающих данных
        model.fit(X_train_scaled, y_train)

        # --- 4. Оценка точности на тестовой выборке ---

        # Делаем предсказания на тестовых данных
        y_pred = model.predict(X_test_scaled)

        # Вычисляем F1-score
        f1 = f1_score(y_test, y_pred)
```

```
        results[name] = f1

        print(f"F1-score для модели '{name}': {f1:.4f}")

    # --- 5. Сравнение результатов и выводы ---

    print("\n" + "="*40)
    print("Итоговое сравнение F1-score для всех моделей:")
    print("="*40)

    for name, score in results.items():
        print(f"{name:<20}: {score:.4f}")

    # Находим модель с лучшим результатом
    best_model_name = max(results, key=results.get)
    best_score = results[best_model_name]

    print("-" * 40)
    print(f"Наилучший результат показал '{best_model_name}' с F1-score = {best_score:.4f}")
```

Вывод:

```
========================================
Итоговое сравнение F1-score для всех моделей:
========================================
Одиночное дерево    :0.5714
Случайный лес       :0.6786
AdaBoost            :0.4286
XGBoost             :0.6667
CatBoost            :0.6897
----------------------------------------
Наилучший результат показал 'CatBoost' с F1-score = 0.6897
```

Алгоритмы бустинга XGBoost и CatBoost и Случайного леса, показывают значительно лучшие результаты, чем одиночное решающее дерево, так как они эффективно борются с переобучением и строят более обобщающие модели.

Вывод: Я на практике сравнил работу нескольких алгоритмов одиночного дерева решений, случайного леса и бустинга для деревьев решений.