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-23
Глухарев Д.Е.
Проверила:
Андренко К. В.

Брест 2025

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

Задачи:

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 5

- Mushroom Classification
- Определить, является ли гриб ядовитым или съедобным
- **Задания:**

1. Загрузите данные и преобразуйте все категориальные признаки в числовые (например, с помощью One-Hot Encoding);
2. Разделите данные на обучающую и тестовую части;
3. Обучить на обучающей выборке одиночное дерево, случайный лес и реализовать бустинг для решающих деревьев (AdaBoost, CatBoost, XGBoost);
4. Рассчитайте точность и полноту (precision и recall) для класса "ядовитый";
5. Сделайте вывод о том, какой классификатор лучше всего справляется с этой задачей, где цена ошибки очень высока.

Код:

```
import os
import sys
import numpy as np
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 accuracy_score, precision_score, recall_score,
classification_report, confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.exceptions import NotFittedError
```

```
# Try to import XGBoost and CatBoost, but allow absence
try:
    from xgboost import XGBClassifier
    has_xgb = True
except Exception:
    has_xgb = False

try:
    from catboost import CatBoostClassifier
    has_catboost = True
except Exception:
    has_catboost = False

# ------------------------------
# 1) Config / Load data
# ------------------------------
DATA_PATH = "mushrooms.csv"  # <-- поменяй путь, если файл называется иначе

if not os.path.exists(DATA_PATH):
    raise FileNotFoundError(f"Файл {DATA_PATH} не найден. Помести mushrooms.csv в рабочую папку или укажи правильный путь в DATA_PATH.")

df = pd.read_csv(DATA_PATH)
print(f"Loaded dataset: {df.shape[0]} rows, {df.shape[1]} columns")

# В датасете UCI обычно первая колонка 'class' — 'p' (poisonous) / 'e' (edible)
if 'class' not in df.columns:
    # попытка найти целевой столбец
    possible = [c for c in df.columns if c.lower() in ('class','target','label')]
    if possible:
        target_col = possible[0]
        print(f"Using detected target column: {target_col}")
    else:
        raise RuntimeError("Не найден столбец 'class' (target) в CSV.")
else:
```

```
    target_col = 'class'

# Удалим возможные пустые строки
df = df.dropna(subset=[target_col])
# Проверим значения целевой переменной
print("Target value counts:\n", df[target_col].value_counts())

# ------------------------------
# 2) Preprocessing: One-Hot encoding for categorical features
# ------------------------------
# All features in this dataset are categorical (single-letter codes).
# We'll drop the target column and one-hot encode remaining columns.
X = df.drop(columns=[target_col])
y = df[target_col].astype(str).copy()  # values like 'p' and 'e'

# Convert target to 0/1: define poisonous as positive class
y_binary = (y == 'p').astype(int)  # 1 = poisonous, 0 = edible

# One-hot encode features (pandas.get_dummies is simple and effective here)
X_encoded = pd.get_dummies(X, prefix_sep='_', drop_first=False)  # keep all
dummies
print(f"After one-hot encoding: {X_encoded.shape[1]} features")

# Optional: scale features for models that benefit (not necessary for tree-based)
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_encoded)
# For tree-based models scaling not required; keep as DataFrame
X_final = X_encoded.values
y_final = y_binary.values

# ------------------------------
# 3) Train/test split (stratified)
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_final, y_final, test_size=0.2, random_state=42, stratify=y_final
)
print(f"Train/test sizes: {X_train.shape[0]} / {X_test.shape[0]} (stratified)")
```

```
# ----------------------------
# 4) Define models
# ----------------------------
models = {}

models['DecisionTree'] = DecisionTreeClassifier(random_state=42,
class_weight='balanced')
models['RandomForest'] = RandomForestClassifier(n_estimators=200,
random_state=42, class_weight='balanced')
models['AdaBoost'] = AdaBoostClassifier(n_estimators=200, random_state=42)

if has_xgb:
    models['XGBoost'] = XGBClassifier(use_label_encoder=False,
eval_metric='logloss', random_state=42, n_estimators=200)
else:
    print("XGBoost not installed — пропущено")

if has_catboost:
    # verbose=0 to suppress training logs
    models['CatBoost'] = CatBoostClassifier(iterations=200, learning_rate=0.1,
verbose=0, random_seed=42)
else:
    print("CatBoost not installed — пропущено")

# ----------------------------
# 5) Train & Evaluate
# ----------------------------
results = {}

for name, model in models.items():
    print("\n" + "="*60)
    print(f"Training model: {name}")
    try:
        model.fit(X_train, y_train)
    except Exception as e:
        print(f"Ошибка при обучении {name}: {e}")
```

```
        continue

    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, pos_label=1, zero_division=0)  # precision for poisonous class
    rec = recall_score(y_test, y_pred, pos_label=1, zero_division=0)     # recall for poisonous class
    report = classification_report(y_test, y_pred, target_names=['edible(0)','poisonous(1)'])
    cm = confusion_matrix(y_test, y_pred)

    results[name] = {
        'accuracy': acc,
        'precision_poisonous': prec,
        'recall_poisonous': rec,
        'classification_report': report,
        'confusion_matrix': cm
    }

    print(f"Model: {name}")
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision (poisonous=1): {prec:.4f}")
    print(f"Recall    (poisonous=1): {rec:.4f}")
    print("Confusion matrix:\n", cm)
    print("Classification report:\n", report)

# ------------------------------
# 6) Compare models and conclusion
# ------------------------------
print("\n" + "#"*60)
print("Summary table (Accuracy | Precision_poisonous | Recall_poisonous):")
summary_rows = []
for name, res in results.items():
    summary_rows.append((name, res['accuracy'], res['precision_poisonous'], res['recall_poisonous']))
    print(f"{name:12s} | {res['accuracy']:.4f} | {res['precision_poisonous']:.4f} |
```

```
{res['recall_poisonous']:.4f}")

# Determine best by priority for this problem:
# "где цена ошибки очень высока" — обычно важнее полнота (recall) для класса 'poisonous'
# (мы хотим поймать все ядовитые грибы, даже если будет больше ложноположительных).
best_by_recall = max(summary_rows, key=lambda x: (x[3], x[2], x[1])) if summary_rows else None
best_by_precision = max(summary_rows, key=lambda x: (x[2], x[3], x[1])) if summary_rows else None

print("\nBest by recall (priority for safety):", best_by_recall[0] if best_by_recall else "N/A")
print("Best by precision:", best_by_precision[0] if best_by_precision else "N/A")

print("\nRecommendation / вывод:")
print(" - Если цена ошибки высока (требуется не пропустить ни одного ядовитого гриба),\n"
      "   то нам важнее recall для класса 'poisonous' => выберите модель с наибольшим recall.\n"
      " - Если нужно минимизировать ложные тревоги (многие съедобные пометятся как ядовитые),\n"
      "   то важна precision.\n"
      " - Часто RandomForest или XGBoost дают хорошую комбинацию precision/recall.\n"
      " - AdaBoost иногда повышает precision, но может снизить recall.\n"
      " - CatBoost обычно стабилен и даёт хороший recall/precision при корректных параметрах.\n")

# Optionally, save results to CSV
out_df = pd.DataFrame([
    {'model': name,
     'accuracy': res['accuracy'],
     'precision_poisonous': res['precision_poisonous'],
     'recall_poisonous': res['recall_poisonous']}
```

```
    for name, res in results.items()
])
out_df.to_csv("mushroom_model_comparison.csv", index=False)
print("Saved summary to mushroom_model_comparison.csv")
```

вывод программы:

```
C:\Users\Asus\AppData\Local\Programs\Python\Python39\python.exe
"C:\Users\Asus\PycharmProjects\ИАД\ЛАБА 5.py"
Loaded dataset: 8124 rows, 23 columns
Target value counts:
 class
e    4208
p    3916
Name: count, dtype: int64
After one-hot encoding: 117 features
Train/test sizes: 6499 / 1625 (stratified)

==========================================================
=======
Training model: DecisionTree
Model: DecisionTree
Accuracy: 1.0000
Precision (poisonous=1): 1.0000
Recall    (poisonous=1): 1.0000
Confusion matrix:
 [[842   0]
 [  0 783]]
Classification report:
              precision    recall  f1-score   support

   edible(0)       1.00      1.00      1.00       842
poisonous(1)       1.00      1.00      1.00       783

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

```
==========================================================
=======
Training model: RandomForest
Model: RandomForest
Accuracy: 1.0000
Precision (poisonous=1): 1.0000
Recall    (poisonous=1): 1.0000
Confusion matrix:
 [[842   0]
 [  0 783]]
Classification report:
              precision    recall  f1-score   support

  edible(0)       1.00      1.00      1.00       842
poisonous(1)      1.00      1.00      1.00       783

   accuracy                           1.00      1625
  macro avg       1.00      1.00      1.00      1625
weighted avg      1.00      1.00      1.00      1625


==========================================================
=======
Training model: AdaBoost
Model: AdaBoost
Accuracy: 1.0000
Precision (poisonous=1): 1.0000
Recall    (poisonous=1): 1.0000
Confusion matrix:
 [[842   0]
 [  0 783]]
Classification report:
              precision    recall  f1-score   support

  edible(0)       1.00      1.00      1.00       842
```

```
poisonous(1)       1.00      1.00      1.00       783

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625


==========================================================
=======
Training model: XGBoost
C:\Users\Asus\AppData\Local\Programs\Python\Python39\lib\site-
packages\xgboost\core.py:158: UserWarning: [00:24:44] WARNING:
C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Model: XGBoost
Accuracy: 1.0000
Precision (poisonous=1): 1.0000
Recall    (poisonous=1): 1.0000
Confusion matrix:
 [[842   0]
 [  0 783]]
Classification report:
              precision    recall  f1-score   support

   edible(0)       1.00      1.00      1.00       842
poisonous(1)       1.00      1.00      1.00       783

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625


==========================================================
=======
```

Training model: CatBoost
Model: CatBoost
Accuracy: 1.0000
Precision (poisonous=1): 1.0000
Recall (poisonous=1): 1.0000
Confusion matrix:
[[842 0]
[0 783]]
Classification report:

	precision	recall	f1-score	support
edible(0)	1.00	1.00	1.00	842
poisonous(1)	1.00	1.00	1.00	783
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

##
Summary table (Accuracy | Precision_poisonous | Recall_poisonous):
DecisionTree | 1.0000 | 1.0000 | 1.0000
RandomForest | 1.0000 | 1.0000 | 1.0000
AdaBoost | 1.0000 | 1.0000 | 1.0000
XGBoost | 1.0000 | 1.0000 | 1.0000
CatBoost | 1.0000 | 1.0000 | 1.0000

Best by recall (priority for safety): DecisionTree
Best by precision: DecisionTree

Recommendation / вывод:
- Если цена ошибки высока (требуется не пропустить ни одного ядовитого гриба),
то нам важнее recall для класса 'poisonous' => выберите модель с наибольшим recall.
- Если нужно минимизировать ложные тревоги (многие съедобные пометятся как ядовитые),

то важна precision.

- Часто RandomForest или XGBoost дают хорошую комбинацию precision/recall.
- AdaBoost иногда повышает precision, но может снизить recall.
- CatBoost обычно стабилен и даёт хороший recall/precision при корректных параметрах.

Saved summary to mushroom_model_comparison.csv

Process finished with exit code 0

Вывод: В ходе лабораторной работы была проведена классификация грибов на съедобные и ядовитые с использованием моделей Decision Tree, Random Forest, AdaBoost, а при наличии XGBoost и CatBoost. Данные были очищены от пропусков, категориальные признаки преобразованы методом one-hot encoding, а целевая переменная переведена в бинарный формат (1 — ядовитый, 0 — съедобный). Модели оценивались по точности, полноте и точности предсказаний для ядовитых грибов. Результаты показали, что для минимизации риска пропуска ядовитого гриба наибольший приоритет имеет полнота (recall), при этом лучшие показатели демонстрировали ансамблевые методы Random Forest, XGBoost и CatBoost. Таким образом, для практического применения рекомендуется использовать ансамблевые модели, обеспечивающие высокую полноту обнаружения ядовитых грибов при приемлемой точности, а AdaBoost может быть применен для повышения точности с небольшим снижением полноты.