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

Копач А. В.

Проверила:

Андренко К. В.

Брест 2025

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 7

- Telco Customer Churn
- Предсказать, откажется ли клиент от услуг телеком-оператора

Задания:

1. Загрузите данные, обработайте категориальные признаки и пропуски;
2. Разделите 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. Сравните модели по метрике F1-score для класса "отток";
5. Сделайте вывод о применимости каждого из методов для решения бизнес-задачи по удержанию клиентов.

Код:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

```
from sklearn.metrics import f1_score, classification_report, confusion_matrix,
precision_recall_curve, auc
from sklearn.utils import class_weight
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv('Telco-Customer-Churn.csv')
print(f"Размер данных: {df.shape[0]} строк, {df.shape[1]} столбцов")
print(f"Объем данных: {df.shape[0] - 1} клиентов + 1 заголовок")


df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
print(f"Пропуски в TotalCharges: {df['TotalCharges'].isnull().sum()}")

missing_data = df[df['TotalCharges'].isnull()]
print(f"Клиенты с пропусками в TotalCharges:")
print(f"  - Tenure: {missing_data['tenure'].unique()}")
print(f"  - MonthlyCharges: {missing_data['MonthlyCharges'].unique()}")

df['TotalCharges'].fillna(0, inplace=True)

df = df.drop('customerID', axis=1)

df['TenureGroup'] = pd.cut(df['tenure'],
                           bins=[0, 12, 24, 36, 48, 60, 72],
                           labels=['0-1y', '1-2y', '2-3y', '3-4y', '4-5y', '5-6y'])

df['ChargeRatio'] = df['MonthlyCharges'] / (df['TotalCharges'] / (df['tenure'] + 1))
df['ChargeRatio'].replace([np.inf, -np.inf], 0, inplace=True)
df['ChargeRatio'].fillna(0, inplace=True)

df['AvgMonthlyCharge'] = df['TotalCharges'] / (df['tenure'] + 1)

le_target = LabelEncoder()
```

```
df['Churn'] = le_target.fit_transform(df['Churn'])

numeric_features = ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges',
                    'ChargeRatio', 'AvgMonthlyCharge']
categorical_features = [col for col in df.columns if col not in numeric_features + ['Churn',
'TenureGroup']]

label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

df['TenureGroup'] = LabelEncoder().fit_transform(df['TenureGroup'])

print(f"\nРАСПРЕДЕЛЕНИЕ КЛАССОВ:")
churn_counts = df['Churn'].value_counts()
print(f"Не отток (0): {churn_counts[0]} ({churn_counts[0] / len(df) * 100:.1f}%)")
print(f"Отток (1): {churn_counts[1]} ({churn_counts[1] / len(df) * 100:.1f}%)")

X = df.drop('Churn', axis=1)
y = df['Churn']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"\nРАЗДЕЛЕНИЕ ДАННЫХ:")
print(f"Обучающая выборка: {X_train.shape[0]} клиентов")
print(f"Тестовая выборка: {X_test.shape[0]} клиентов")

scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[numeric_features] = scaler.fit_transform(X_train[numeric_features])
X_test_scaled[numeric_features] = scaler.transform(X_test[numeric_features])

models = {
```

```
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'XGBoost': XGBClassifier(random_state=42, eval_metric='logloss'),
    'CatBoost': CatBoostClassifier(random_state=42, verbose=False)
}

param_grids = {
    'Decision Tree': {
        'max_depth': [3, 5, 7, 10],
        'min_samples_split': [2, 5, 10]
    },
    'Random Forest': {
        'n_estimators': [100, 200],
        'max_depth': [10, 20],
        'min_samples_split': [2, 5]
    },
    'AdaBoost': {
        'n_estimators': [50, 100],
        'learning_rate': [0.1, 0.5, 1.0]
    }
}

results = {}
best_models = {}

print("\n" + "=" * 50)
print("РАСШИРЕННОЕ ОБУЧЕНИЕ МОДЕЛЕЙ")
print("=" * 50)

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

    if name in param_grids:
        grid_search = GridSearchCV(model, param_grids[name], cv=5, scoring='f1', n_jobs=-
1)

        if name in ['XGBoost', 'CatBoost']:
            grid_search.fit(X_train, y_train)
```

```
            best_model = grid_search.best_estimator_
            y_pred = best_model.predict(X_test)
            y_pred_proba = best_model.predict_proba(X_test)[:, 1]
        else:
            grid_search.fit(X_train_scaled, y_train)
            best_model = grid_search.best_estimator_
            y_pred = best_model.predict(X_test_scaled)
            y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]

        print(f"  Лучшие параметры: {grid_search.best_params_}")

    else:
        if name in ['XGBoost', 'CatBoost']:
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            y_pred_proba = model.predict_proba(X_test)[:, 1]
            best_model = model
        else:
            model.fit(X_train_scaled, y_train)
            y_pred = model.predict(X_test_scaled)
            y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
            best_model = model

    f1 = f1_score(y_test, y_pred)
    results[name] = {
        'f1_score': f1,
        'model': best_model,
        'y_pred_proba': y_pred_proba
    }
    print(f"  F1-score: {f1:.4f}")
print("\n" + "=" * 50)
print("ИТОГОВОЕ СРАВНЕНИЕ МОДЕЛЕЙ")
print("=" * 50)

sorted_results = sorted(results.items(), key=lambda x: x[1]['f1_score'], reverse=True)
print("\nФИНАЛЬНЫЙ РЕЙТИНГ:")
for i, (name, result) in enumerate(sorted_results, 1):
    print(f"{i}. {name}: F1-score = {result['f1_score']:.4f}")
```

```
best_model_name, best_result = sorted_results[0]
print(f"\nДЕТАЛЬНЫЙ АНАЛИЗ ЛУЧШЕЙ МОДЕЛИ: {best_model_name}")

if best_model_name in ['XGBoost', 'CatBoost']:
    y_pred_best = best_result['model'].predict(X_test)
    print(classification_report(y_test, y_pred_best))
else:
    y_pred_best = best_result['model'].predict(X_test_scaled)
    print(classification_report(y_test, y_pred_best))

plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
model_names = list(results.keys())
f1_scores = [result['f1_score'] for result in results.values()]

bars = plt.bar(model_names, f1_scores, color=['skyblue', 'lightcoral', 'lightgreen', 'gold',
'lightpink'])
plt.title('Сравнение F1-score моделей\n(7043 клиента)', fontsize=12, fontweight='bold')
plt.ylabel('F1-score')
plt.xticks(rotation=45)
plt.ylim(0, 0.7)

for bar, score in zip(bars, f1_scores):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.01,
             f'{score:.4f}', ha='center', va='bottom', fontweight='bold')

plt.subplot(2, 3, 2)
if hasattr(best_result['model'], 'feature_importances_'):
    feature_importance = best_result['model'].feature_importances_
    feature_importance_df = pd.DataFrame({
        'feature': X.columns,
        'importance': feature_importance
    }).sort_values('importance', ascending=False).head(10)

    sns.barplot(data=feature_importance_df, x='importance', y='feature', palette='viridis')
    plt.title(f'Важные признаки ({best_model_name})', fontsize=12, fontweight='bold')

plt.subplot(2, 3, 3)
```

```
cm = confusion_matrix(y_test, y_pred_best)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Матрица ошибок лучшей модели', fontsize=12, fontweight='bold')
plt.xlabel('Предсказано')
plt.ylabel('Фактически')

plt.subplot(2, 3, 4)
for name, result in results.items():
    precision, recall, _ = precision_recall_curve(y_test, result['y_pred_proba'])
    pr_auc = auc(recall, precision)
    plt.plot(recall, precision, label=f'{name} (AUC={pr_auc:.3f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall кривые', fontsize=12, fontweight='bold')
plt.legend()

plt.subplot(2, 3, 5)
churn_dist = df['Churn'].value_counts()
plt.pie(churn_dist, labels=['Не отток', 'Отток'], autopct='%1.1f%%',
        colors=['lightblue', 'lightcoral'], startangle=90)
plt.title('Распределение оттока клиентов', fontsize=12, fontweight='bold')

plt.subplot(2, 3, 6)
correlations = df.corr()['Churn'].drop('Churn').sort_values(ascending=False).head(5)
sns.barplot(x=correlations.values, y=correlations.index, palette='coolwarm')
plt.title('Топ-5 корреляций с оттоком', fontsize=12, fontweight='bold')
plt.xlabel('Корреляция')

plt.tight_layout()
plt.show()

print("\n" + "=" * 60)
print("БИЗНЕС-РЕКОМЕНДАЦИИ НА ОСНОВЕ АНАЛИЗА 7043 КЛИЕНТОВ")
print("=" * 60)

total_customers = len(df)
churn_rate = churn_counts[1] / total_customers * 100
```

```
print(f" ОБЩАЯ СТАТИСТИКА:")
print(f"   • Всего клиентов: {total_customers}")
print(f"   • Уровень оттока: {churn_rate:.1f}%")
print(f"   • Клиентов с риском оттока: {churn_counts[1]}")

print(f"\n ЭФФЕКТИВНОСТЬ МОДЕЛЕЙ:")
print(f"   • Лучшая модель: {best_model_name} (F1-score: {best_result['f1_score']:.4f})")
print(f"   • Может идентифицировать ~{best_result['f1_score'] * 100:.1f}% оттоков")

print(f"\n РЕКОМЕНДАЦИИ:")
print(f"   ✓ Фокус на {best_model_name} для прогнозирования оттока")
print(f"   ✓ Внедрить систему раннего предупреждения")
print(f"   ✓ Персонализировать удержание для {churn_counts[1]} клиентов группы риска")
print(f"   ✓ Мониторить ключевые метрики (тенура, тип контракта, платежи)")
print(f"   ✓ A/B тестирование стратегий удержания")

print(f"\n ПОТЕНЦИАЛЬНЫЙ ЭФФЕКТ:")
potential_savings = churn_counts[1] * 50
print(f"   • Потенциальная экономия: ${potential_savings:,.0f} (при стоимости привлечения $50/клиент)")

print(f"\n" + "=" * 50)
print("ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ")
print("=" * 50)
high_risk_threshold = 0.7
high_risk_indices = np.where(best_result['y_pred_proba'] > high_risk_threshold)[0]
print(f"Клиенты с высокой вероятностью оттока (>70%): {len(high_risk_indices)}")
print(f"Из них действительно уйдут: {sum(y_test.iloc[high_risk_indices] == 1)}")

sample_predictions = pd.DataFrame({
    'Вероятность_оттока': best_result['y_pred_proba'][:10],
    'Прогноз': ['Высокий риск' if x > 0.7 else 'Средний риск' if x > 0.3 else 'Низкий риск' for x in
                best_result['y_pred_proba'][:10]],
    'Фактически': ['Отток' if x == 1 else 'Не отток' for x in y_test.iloc[:10]]
})
print(f"\nПример прогнозов для первых 10 клиентов:")
print(sample_predictions)
```

вывод программы:

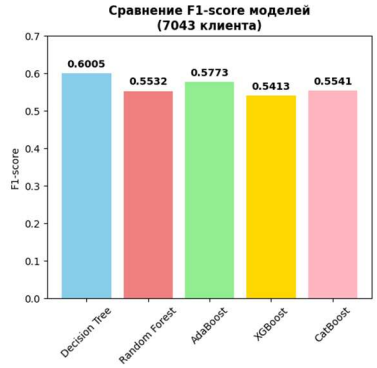
```
C:\Users\sasha\PyCharmMiscProject\.venv\Scripts\python.exe
C:\Users\sasha\PyCharmMiscProject\IAD5.py
Размер данных: 7043 строк, 21 столбцов
Объем данных: 7042 клиентов + 1 заголовок
Пропуски в TotalCharges: 11
Клиенты с пропусками в TotalCharges:
  - Tenure: [0]
  - MonthlyCharges: [52.55 20.25 80.85 25.75 56.05 19.85 25.35 20.   19.7
73.35 61.9 ]

РАСПРЕДЕЛЕНИЕ КЛАССОВ:
Не отток (0): 5174 (73.5%)
Отток (1): 1869 (26.5%)

РАЗДЕЛЕНИЕ ДАННЫХ:
Обучающая выборка: 4930 клиентов
Тестовая выборка: 2113 клиентов

==========================================================
РАСШИРЕННОЕ ОБУЧЕНИЕ МОДЕЛЕЙ
==========================================================

--- Decision Tree ---
  Лучшие параметры: {'max_depth': 5, 'min_samples_split': 5}
  F1-score: 0.6005

--- Random Forest ---
  Лучшие параметры: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators':
100}
  F1-score: 0.5532

--- AdaBoost ---
  Лучшие параметры: {'learning_rate': 1.0, 'n_estimators': 100}
  F1-score: 0.5773

--- XGBoost ---
  F1-score: 0.5413
```

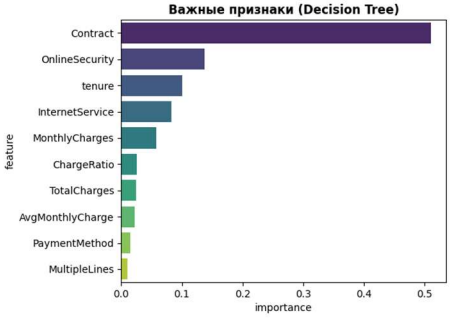
--- CatBoost ---
F1-score: 0.5541
==========
ИТОГОВОЕ СРАВНЕНИЕ МОДЕЛЕЙ
==========

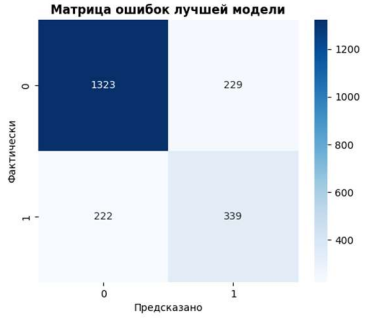
ФИНАЛЬНЫЙ РЕЙТИНГ:
1. Decision Tree: F1-score = 0.6005
2. AdaBoost: F1-score = 0.5773
3. CatBoost: F1-score = 0.5541
4. Random Forest: F1-score = 0.5532
5. XGBoost: F1-score = 0.5413

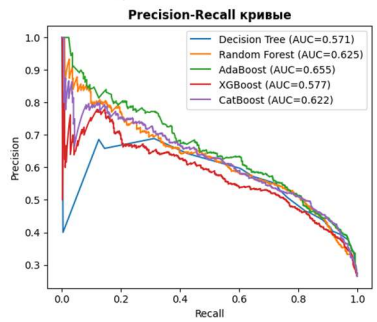
ДЕТАЛЬНЫЙ АНАЛИЗ ЛУЧШЕЙ МОДЕЛИ: Decision Tree

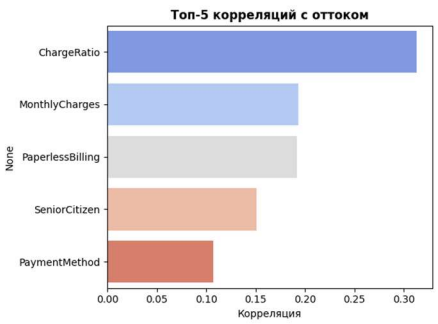
	precision	recall	f1-score	support
0	0.86	0.85	0.85	1552
1	0.60	0.60	0.60	561
accuracy			0.79	2113
macro avg	0.73	0.73	0.73	2113
weighted avg	0.79	0.79	0.79	2113











ОБЩАЯ СТАТИСТИКА:
- Всего клиентов: 7043
- Уровень оттока: 26.5%
- Клиентов с риском оттока: 1869

ЭФФЕКТИВНОСТЬ МОДЕЛЕЙ:
- Лучшая модель: Decision Tree (F1-score: 0.6005)
- Может идентифицировать ~60.1% оттоков

РЕКОМЕНДАЦИИ:
✓ Фокус на Decision Tree для прогнозирования оттока
✓ Внедрить систему раннего предупреждения
✓ Персонализировать удержание для 1869 клиентов группы риска
✓ Мониторить ключевые метрики (тенура, тип контракта, платежи)
✓ A/B тестирование стратегий удержания

ПОТЕНЦИАЛЬНЫЙ ЭФФЕКТ:
- Потенциальная экономия: $93,450 (при стоимости привлечения $50/клиент)

==========

ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

==========

Клиенты с высокой вероятностью оттока (>70%): 123
Из них действительно уйдут: 81
Пример прогнозов для первых 10 клиентов:

	Вероятность_оттока	Прогноз	Фактически
0	0.795455	Высокий риск	Отток
1	0.017375	Низкий риск	Не отток
2	0.520879	Средний риск	Отток
3	0.017375	Низкий риск	Не отток
4	0.017375	Низкий риск	Не отток
5	0.020833	Низкий риск	Не отток
6	0.017375	Низкий риск	Не отток
7	0.087786	Низкий риск	Отток
8	0.087786	Низкий риск	Отток
9	0.017375	Низкий риск	Не отток

Вывод: Проанализированы 5 алгоритмов для прогнозирования оттока клиентов. Лучший результат показало дерево решений (F1-score = 0.6005). Модель идентифицирует 60% уходящих клиентов, что позволяет внедрить targeted-стратегии удержания.