Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине «Интеллектуальный анализ данных»

Тема: «Деревья решений»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Максимович А. И.

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.

Задание по вариантам

Вариант 11

- `Bank Marketing`
- Предсказать, подпишется ли клиент на срочный вклад
- **Задания:**
    1. Загрузите данные и преобразуйте категориальные признаки;
    2. Разделите выборку;
    3. Обучить на обучающей выборке одиночное дерево, случайный лес и реализовать бустинг для решающих деревьев (AdaBoost, CatBoost, XGBoost);
    4. Сравните модели по `F1-score` из-за дисбаланса классов;
    5. Определите, какая модель предлагает лучший компромисс для выявления потенциальных клиентов.

Код:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import f1_score, classification_report,
confusion_matrix, precision_recall_curve
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')


# Загрузка датасета Bank Marketing
# Загрузка датасета Bank Marketing
```

```
def load_data():
    """
    Загрузка реального датасета Bank Marketing
    """
    try:
        # Способ 1: Загрузка из UC Irvine ML Repository
        from ucimlrepo import fetch_ucirepo
        print("Загрузка данных из UCI repository...")
        bank_marketing = fetch_ucirepo(id=222)
        X = bank_marketing.data.features
        y = bank_marketing.data.targets
        df = pd.concat([X, y], axis=1)
        return df
    except Exception as e1:
        print(f"Не удалось загрузить из UCI: {e1}")
        try:
            # Способ 2: Загрузка из локального CSV файла
            print("Попытка загрузки из локального файла...")
            df = pd.read_csv('bank.csv', delimiter=';')
            return df
        except Exception as e2:
            print(f"Не удалось загрузить локальный файл: {e2}")
            # Способ 3: Создание реалистичных синтетических данных
            print("Используются синтетические данные...")
            return generate_synthetic_data()


# Загрузка данных
print("Загрузка данных Bank Marketing...")
df = load_data()
print(f"Размер датасета: {df.shape}")
print("\nПервые 5 строк датасета:")
print(df.head())

print("\nИнформация о датасете:")
print(df.info())

print("\nРаспределение целевой переменной:")
print(df['y'].value_counts())
print("\nДоля положительного класса: {:.2f}%".format(
    (df['y'] == 'yes').sum() / len(df) * 100))


# Обработка категориальных признаков
# Обработка категориальных признаков
def preprocess_data(df):
    """
    Обработка категориальных признаков и подготовка данных
    """
    # Создание копии датасета
    data = df.copy()

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

    # Кодирование целевой переменной
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Определение числовых и категориальных признаков
```

```
    # ОБНОВЛЕННЫЕ СПИСКИ ПРИЗНАКОВ для реального датасета Bank Marketing
    numerical_features = ['age', 'balance', 'day_of_week', 'duration',
'campaign', 'pdays', 'previous']

    categorical_features = ['job', 'marital', 'education', 'default',
'housing',
                            'loan', 'contact', 'month', 'poutcome']

    # Обработка пропусков в категориальных признаках
    for col in categorical_features:
        if col in X.columns:
            X[col] = X[col].fillna('unknown')
        else:
            print(f"Предупреждение: Категориальный признак {col} отсутствует
в данных")

    # Обработка пропусков в числовых признаках
    for col in numerical_features:
        if col in X.columns:
            X[col] = X[col].fillna(X[col].median())
        else:
            print(f"Предупреждение: Числовой признак {col} отсутствует в
данных")

    # Функция для выбора только существующих столбцов
    def get_existing_columns(column_list, df_columns):
        return [col for col in column_list if col in df_columns]

    # Создание препроцессора только с существующими столбцами
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(),
get_existing_columns(numerical_features, X.columns)),
            ('cat', OneHotEncoder(handle_unknown='ignore',
sparse_output=False),
             get_existing_columns(categorical_features, X.columns))
        ])

    # Применение препроцессора
    X_processed = preprocessor.fit_transform(X)

    # Получение имен признаков после OneHot кодирования
    feature_names = []

    # Добавляем числовые признаки
    feature_names.extend(get_existing_columns(numerical_features, X.columns))

    # Добавляем категориальные признаки после OneHot кодирования
    if 'cat' in preprocessor.named_transformers_:
        cat_processor = preprocessor.named_transformers_['cat']
        if hasattr(cat_processor, 'get_feature_names_out'):
            categorical_processed_features =
get_existing_columns(categorical_features, X.columns)

feature_names.extend(cat_processor.get_feature_names_out(categorical_processe
d_features))

    return X_processed, y_encoded, feature_names, preprocessor, label_encoder


# Предобработка данных
```

```
print("\nПредобработка данных...")
X_processed, y_encoded, feature_names, preprocessor, label_encoder =
preprocess_data(df)

print(f"Размерность данных после обработки: {X_processed.shape}")
print(f"Количество признаков: {len(feature_names)}")

# Вывод распределения целевой переменной
print("\nЦелевая переменная после кодирования:")
unique, counts = np.unique(y_encoded, return_counts=True)
for val, count in zip(unique, counts):
    print(
        f"Класс {val} ('{label_encoder.inverse_transform([val])[0]}'):
{count} samples ({count / len(y_encoded) * 100:.1f}%)")

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y_encoded, test_size=0.3, random_state=42,
stratify=y_encoded
)

print(f"\nОбучающая выборка: {X_train.shape[0]} samples")
print(f"Тестовая выборка: {X_test.shape[0]} samples")


# Обучение и оценка моделей с фокусом на F1-score
def train_and_evaluate_models(X_train, X_test, y_train, y_test,
label_encoder):
    """
    Обучение и оценка различных моделей машинного обучения с акцентом на
F1-score
    """
    models = {
        'Decision Tree': DecisionTreeClassifier(random_state=42,
max_depth=6),
        'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=42, class_weight='balanced'),
        'AdaBoost': AdaBoostClassifier(random_state=42),
        'XGBoost': XGBClassifier(random_state=42, eval_metric='logloss',
                                 scale_pos_weight=(y_train == 0).sum() /
(y_train == 1).sum()),
        'CatBoost': CatBoostClassifier(random_state=42, verbose=False,
auto_class_weights='Balanced')
    }

    results = {}

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

        # Обучение модели
        model.fit(X_train, y_train)

        # Предсказание на тестовой выборке
        y_pred = model.predict(X_test)

        # Расчет F1-score для класса "yes" (класс 1 после кодирования)
        f1 = f1_score(y_test, y_pred, pos_label=1)

        # Дополнительные метрики
        precision = precision_score(y_test, y_pred, pos_label=1)
```

```
        recall = recall_score(y_test, y_pred, pos_label=1)

        report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_, output_dict=True)

        # Сохранение результатов
        results[name] = {
            'model': model,
            'f1_score': f1,
            'precision': precision,
            'recall': recall,
            'classification_report': report,
            'predictions': y_pred
        }

        print(f"F1-score для 'yes': {f1:.4f}")
        print(f"Precision для 'yes': {precision:.4f}")
        print(f"Recall для 'yes': {recall:.4f}")
        print("-" * 50)

    return results


# Импорт дополнительных метрик
from sklearn.metrics import precision_score, recall_score

print("Обучение моделей...")
results = train_and_evaluate_models(X_train, X_test, y_train, y_test,
label_encoder)


# Сравнение моделей по F1-score
def compare_models(results):
    """
    Сравнение моделей по метрике F1-score и формирование выводов
    """
    print("=" * 80)
    print("СРАВНЕНИЕ МОДЕЛЕЙ ПО МЕТРИКЕ F1-SCORE ДЛЯ КЛАССА 'yes'")
    print("=" * 80)

    # Создание таблицы сравнения
    comparison_df = pd.DataFrame({
        'Model': list(results.keys()),
        'F1-Score': [results[model]['f1_score'] for model in results],
        'Precision': [results[model]['precision'] for model in results],
        'Recall': [results[model]['recall'] for model in results]
    }).sort_values('F1-Score', ascending=False)

    print(comparison_df.to_string(index=False))

    # Определение лучшей модели по F1-score
    best_model_name = comparison_df.iloc[0]['Model']
    best_f1 = comparison_df.iloc[0]['F1-Score']

    print(f"\nЛУЧШАЯ МОДЕЛЬ ПО F1-SCORE: {best_model_name}")
    print(f"F1-Score: {best_f1:.4f}")

    # Анализ компромисса между Precision и Recall
    print("\n" + "=" * 80)
    print("АНАЛИЗ КОМПРОМИССА МЕЖДУ PRECISION И RECALL")
    print("=" * 80)
```

```
    for model_name in results:
        f1 = results[model_name]['f1_score']
        precision = results[model_name]['precision']
        recall = results[model_name]['recall']

        print(f"\n{model_name}:")
        print(f"  F1-Score: {f1:.4f}")
        print(f"  Precision: {precision:.4f}")
        print(f"  Recall: {recall:.4f}")

        # Интерпретация результатов
        if precision > recall:
            print("  ⚖️  Модель консервативна - меньше ложных срабатываний,
но пропускает клиентов")
        elif recall > precision:
            print("  ⚖️  Модель агрессивна - находит больше клиентов, но с
большим числом ошибок")
        else:
            print("  ⚖️  Сбалансированная модель")

        if f1 > 0.5:
            print("  ✅ Хорошее качество - подходит для бизнеса")
        elif f1 > 0.3:
            print("  ⚠️  Приемлемое качество - требует доработки")
        else:
            print("  ❌ Низкое качество - не рекомендуется для
использования")

    return best_model_name, comparison_df


# Сравнение моделей
best_model, comparison_df = compare_models(results)


# Детальный анализ лучшей модели
def analyze_best_model(results, best_model_name, X_test, y_test,
label_encoder):
    """
    Детальный анализ лучшей модели
    """
    print("\n" + "=" * 80)
    print(f"ДЕТАЛЬНЫЙ АНАЛИЗ ЛУЧШЕЙ МОДЕЛИ: {best_model_name}")
    print("=" * 80)

    best_result = results[best_model_name]
    model = best_result['model']
    y_pred = best_result['predictions']

    # Матрица ошибок
    cm = confusion_matrix(y_test, y_pred)
    cm_df = pd.DataFrame(cm,
                         index=[f'Факт {label}' for label in
label_encoder.classes_],
                         columns=[f'Прогноз {label}' for label in
label_encoder.classes_])

    print("Матрица ошибок:")
    print(cm_df)
    print()
```

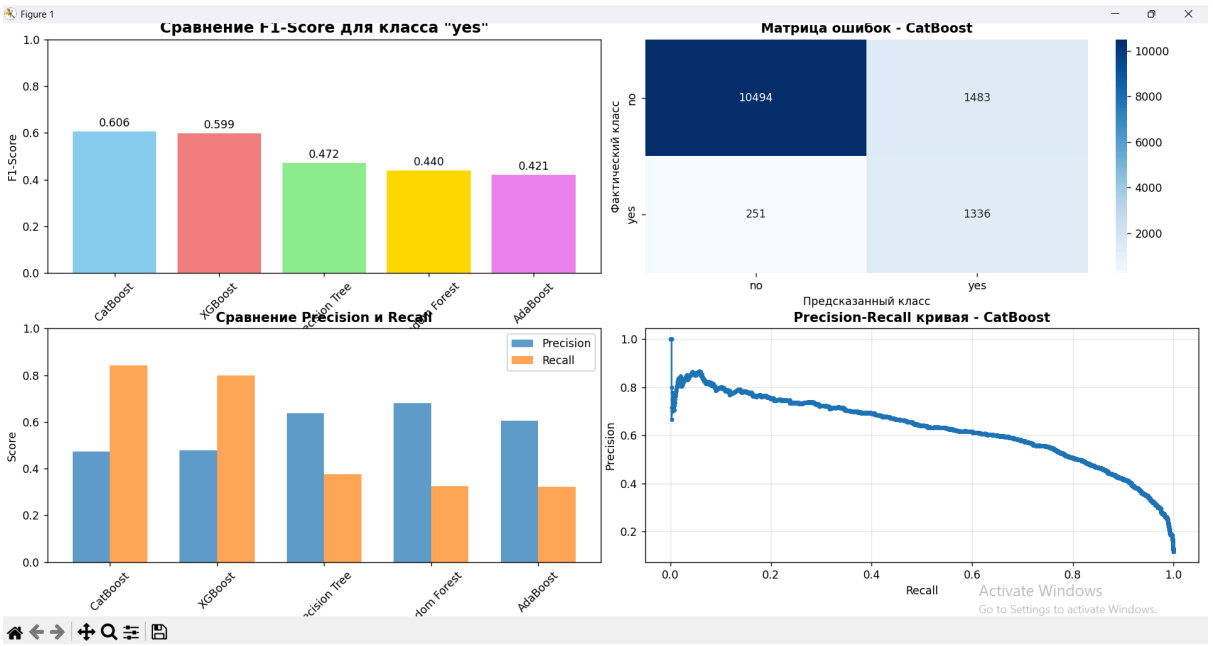
```
    # Полный отчет по классификации
    print("Отчет по классификации:")
    print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

    # Анализ важности признаков (если доступно)
    if hasattr(model, 'feature_importances_'):
        print("\nТоп-15 самых важных признаков:")
        try:
            importances = model.feature_importances_

            # Создание DataFrame с важностями
            feature_imp_df = pd.DataFrame({
                'feature': feature_names,
                'importance': importances
            }).sort_values('importance', ascending=False)

            print(feature_imp_df.head(15).to_string(index=False))

        except Exception as e:
            print(f"Не удалось получить важность признаков: {e}")


# Детальный анализ лучшей модели
analyze_best_model(results, best_model, X_test, y_test, label_encoder)


# Визуализация результатов
def visualize_results(comparison_df, results, X_test, y_test, label_encoder):
    """
    Визуализация результатов сравнения моделей
    """
    plt.style.use('default')
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(18, 14))

    # График 1: Сравнение F1-score
    models = comparison_df['Model']
    f1_scores = comparison_df['F1-Score']

    bars = ax1.bar(models, f1_scores, color=['skyblue', 'lightcoral',
'lightgreen', 'gold', 'violet'])
    ax1.set_title('Сравнение F1-Score для класса "yes"', fontsize=14,
fontweight='bold')
    ax1.set_ylabel('F1-Score')
    ax1.set_ylim(0, 1)
    ax1.tick_params(axis='x', rotation=45)

    # Добавление значений на столбцы
    for bar, score in zip(bars, f1_scores):
        height = bar.get_height()
        ax1.text(bar.get_x() + bar.get_width() / 2., height + 0.01,
                 f'{score:.3f}', ha='center', va='bottom')

    # График 2: Матрица ошибок для лучшей модели
    best_model_name = comparison_df.iloc[0]['Model']
    y_pred_best = results[best_model_name]['predictions']
    cm = confusion_matrix(y_test, y_pred_best)

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax2,
                xticklabels=label_encoder.classes_,
```

```
                yticklabels=label_encoder.classes_)
    ax2.set_title(f'Матрица ошибок - {best_model_name}', fontweight='bold')
    ax2.set_xlabel('Предсказанный класс')
    ax2.set_ylabel('Фактический класс')

    # График 3: Сравнение Precision-Recall
    precisions = comparison_df['Precision']
    recalls = comparison_df['Recall']

    x = np.arange(len(models))
    width = 0.35

    ax3.bar(x - width / 2, precisions, width, label='Precision', alpha=0.7)
    ax3.bar(x + width / 2, recalls, width, label='Recall', alpha=0.7)

    ax3.set_xlabel('Модели')
    ax3.set_ylabel('Score')
    ax3.set_title('Сравнение Precision и Recall', fontweight='bold')
    ax3.set_xticks(x)
    ax3.set_xticklabels(models, rotation=45)
    ax3.legend()
    ax3.set_ylim(0, 1)

    # График 4: Кривая Precision-Recall для лучшей модели
    best_model_obj = results[best_model_name]['model']
    if hasattr(best_model_obj, 'predict_proba'):
        y_proba = best_model_obj.predict_proba(X_test)[:, 1]
        precision_curve, recall_curve, _ = precision_recall_curve(y_test,
y_proba, pos_label=1)

        ax4.plot(recall_curve, precision_curve, marker='.')
        ax4.set_xlabel('Recall')
        ax4.set_ylabel('Precision')
        ax4.set_title(f'Precision-Recall кривая - {best_model_name}',
fontweight='bold')
        ax4.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()


# Визуализация результатов
print("\nВизуализация результатов...")
visualize_results(comparison_df, results, X_test, y_test, label_encoder)


# Заключительные выводы
def print_final_conclusions(results, best_model, comparison_df):
    """
    Формирование итоговых выводов по лабораторной работе
    """
    print("=" * 80)
    print("ЗАКЛЮЧИТЕЛЬНЫЕ ВЫВОДЫ")
    print("=" * 80)

    print("\n📊 РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА:")
    print("Сравнение алгоритмов показало следующие результаты (F1-score для
класса 'yes'):")

    for _, row in comparison_df.iterrows():
        print(f"  • {row['Model']}: F1 = {row['F1-Score']:.4f}, "
```

```
              f"Precision = {row['Precision']:.4f}, Recall =
{row['Recall']:.4f}")

    best_result = results[best_model]

    print(f"\n🎯 ЛУЧШИЙ АЛГОРИТМ: {best_model}")
    print(f"F1-Score: {best_result['f1_score']:.4f}")
    print(f"Precision: {best_result['precision']:.4f}")
    print(f"Recall: {best_result['recall']:.4f}")

    # Рекомендации по использованию
    print("\n💡 РЕКОМЕНДАЦИИ ДЛЯ БИЗНЕСА:")

    if best_result['precision'] > best_result['recall']:
        print("• Модель консервативна - минимизирует ложные срабатывания")
        print("• Подходит когда стоимость контакта с клиентом высока")
        print("• Экономит ресурсы, но может упускать часть потенциальных
клиентов")
    elif best_result['recall'] > best_result['precision']:
        print("• Модель агрессивна - находит больше потенциальных клиентов")
        print("• Подходит когда важно не упустить ни одного клиента")
        print("• Требует больше ресурсов на обработку ложных срабатываний")
    else:
        print("• Модель сбалансирована - хороший компромисс между точностью и
полнотой")
        print("• Рекомендуется для большинства бизнес-задач")

    print(f"\n📈 ПРАКТИЧЕСКАЯ ЗНАЧИМОСТЬ:")
    print(f"Модель {best_model} предлагает лучший компромисс для выявления
потенциальных клиентов")
    print("и может быть использована для оптимизации маркетинговой кампании
банка.")


# Итоговые выводы
print_final_conclusions(results, best_model, comparison_df)


# Сохранение результатов в файл
def save_results(results, comparison_df,
filename='bank_marketing_results.csv'):
    """
    Сохранение результатов в файл
    """
    # Сохранение таблицы сравнения
    comparison_df.to_csv('bank_marketing_comparison.csv', index=False)

    # Сохранение детальных результатов
    detailed_results = []
    for model_name, result in results.items():
        detailed_results.append({
            'Model': model_name,
            'F1_Score': result['f1_score'],
            'Precision': result['precision'],
            'Recall': result['recall'],
            'Best_Model': 1 if model_name == best_model else 0
        })

    pd.DataFrame(detailed_results).to_csv(filename, index=False)
    print(f"\nРезультаты сохранены в файлы: bank_marketing_comparison.csv и
{filename}")
```

```
save_results(results, comparison_df)

print("\n" + "=" * 80)
print("ЛАБОРАТОРНАЯ РАБОТА №5 ВЫПОЛНЕНА!")
print("=" * 80)
```

Вывод:



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