Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 5

Специальность ИИ-23

Выполнил:

Тутина Е.Д.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» _____ 2025

Брест 2025

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

Общее 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: 11

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

Код программы:

```
import os
import warnings
warnings.filterwarnings("ignore")


import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score,
recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from catboost import CatBoostClassifier, Pool
import xgboost as xgb
```

```
import matplotlib.pyplot as plt
import joblib

RANDOM_STATE = 42
CSV_PATHS = ["bank.csv"]
MODEL_OUT_DIR = Path("models_out")
MODEL_OUT_DIR.mkdir(exist_ok=True)
TEST_SIZE = 0.2
DO_HYPERSEARCH = False
N_JOBS = -1

def load_csv_try(paths):
    for p in paths:
        if Path(p).exists():
            return pd.read_csv(p)
    raise FileNotFoundError(f"Не найден ни один из файлов: {paths}. Поместите bunk.csv рядом со скриптом.")

df = load_csv_try(CSV_PATHS)
print("Data shape:", df.shape)
print("Columns:", df.columns.tolist())

print(df.head(3))

TARGET_COL = 'deposit'
if TARGET_COL not in df.columns:
    possible = [c for c in df.columns if c.lower() in ('deposit', 'y', 'subscribed', 'target')]
    if possible:
        TARGET_COL = possible[0]
    else:
        raise ValueError("Не найдена целевая колонка (deposit). Переименуйте её или укажите явно в коде.")

df[TARGET_COL] = df[TARGET_COL].astype(str).str.strip().str.lower()
pos_label = 'yes'
df['y_bin'] = (df[TARGET_COL] == pos_label).astype(int)

X = df.drop(columns=[TARGET_COL, 'y_bin'])
y = df['y_bin']

cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
```

```
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
print(f"Categorical ({len(cat_cols)}):", cat_cols)
print(f"Numerical ({len(num_cols)}):", num_cols)

high_card = [c for c in cat_cols if X[c].nunique() > 20]
low_card = [c for c in cat_cols if X[c].nunique() <= 20]
print("High-cardinality categorical:", high_card)
print("Low-cardinality categorical:", low_card)

preprocessor = ColumnTransformer(transformers=[
    ('num', 'passthrough', num_cols),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output=False), low_card),
    ('ord', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1), high_card)
], remainder='drop')



X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, stratify=y, random_state=RANDOM_STATE
)
print("Train/test shapes:", X_train.shape, X_test.shape, y_train.sum(), "/", y_test.sum())

def fit_and_eval(pipeline, X_train, y_train, X_test, y_test, name="model", save_model=True):
    print(f"\n=== TRAIN & EVAL: {name} ===")
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    report = classification_report(y_test, y_pred, digits=4, output_dict=True)
    print(classification_report(y_test, y_pred, digits=4))
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion matrix:\n", cm)
    f1_macro = f1_score(y_test, y_pred, average='macro')
    f1_pos = f1_score(y_test, y_pred, pos_label=1)
    prec_pos = precision_score(y_test, y_pred, pos_label=1)
    rec_pos = recall_score(y_test, y_pred, pos_label=1)
    summary = {
        'name': name,
        'f1_macro': f1_macro,
        'f1_pos': f1_pos,
        'precision_pos': prec_pos,
        'recall_pos': rec_pos
    }
```

```
    if save_model:
        joblib.dump(pipeline, MODEL_OUT_DIR / f"{name}.joblib")
    return summary, report



results = []

dt_clf = DecisionTreeClassifier(random_state=RANDOM_STATE, class_weight='balanced')
dt_pipe = Pipeline([('pre', preprocessor), ('clf', dt_clf)])
res_dt, report_dt = fit_and_eval(dt_pipe, X_train, y_train, X_test, y_test, name="decision_tree")
results.append(res_dt)

rf_clf = RandomForestClassifier(n_estimators=200, random_state=RANDOM_STATE, class_weight='balanced',
n_jobs=N_JOBS)
rf_pipe = Pipeline([('pre', preprocessor), ('clf', rf_clf)])
res_rf, report_rf = fit_and_eval(rf_pipe, X_train, y_train, X_test, y_test, name="random_forest")
results.append(res_rf)

adb_clf = AdaBoostClassifier(n_estimators=100, random_state=RANDOM_STATE)
adb_pipe = Pipeline([('pre', preprocessor), ('clf', adb_clf)])
res_adb, report_adb = fit_and_eval(adb_pipe, X_train, y_train, X_test, y_test, name="adaboost")
results.append(res_adb)

X_pre = preprocessor.fit_transform(X_train)
X_pre_test = preprocessor.transform(X_test)
dtrain = xgb.DMatrix(X_pre, label=y_train)
dtest = xgb.DMatrix(X_pre_test, label=y_test)
xgb_params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    'use_label_encoder': False,
    'seed': RANDOM_STATE,
    'verbosity': 0,
}
xgb_clf = xgb.XGBClassifier(**{k:v for k,v in xgb_params.items() if k!='verbosity'}, n_estimators=200,
n_jobs=N_JOBS, random_state=RANDOM_STATE)
xgb_clf.fit(X_pre, y_train)
y_pred_xgb = xgb_clf.predict(X_pre_test)
from sklearn.metrics import classification_report as cr
print("\n=== XGBoost ===")
print(cr(y_test, y_pred_xgb, digits=4))
```

```
res_xgb = {
    'name': 'xgboost',
    'f1_macro': f1_score(y_test, y_pred_xgb, average='macro'),
    'f1_pos': f1_score(y_test, y_pred_xgb, pos_label=1),
    'precision_pos': precision_score(y_test, y_pred_xgb, pos_label=1),
    'recall_pos': recall_score(y_test, y_pred_xgb, pos_label=1)
}
joblib.dump((preprocessor, xgb_clf), MODEL_OUT_DIR / "xgboost_preproc_clf.joblib")
results.append(res_xgb)


cat_idx = [i for i, col in enumerate(X.columns) if col in cat_cols]
cb_clf = CatBoostClassifier(iterations=500, learning_rate=0.05, eval_metric='F1',
random_seed=RANDOM_STATE, verbose=False)
cb_clf.fit(X_train, y_train, cat_features=cat_idx)
y_pred_cb = cb_clf.predict(X_test).astype(int)
print("\n=== CatBoost ===")
print(cr(y_test, y_pred_cb, digits=4))
res_cb = {
    'name': 'catboost',
    'f1_macro': f1_score(y_test, y_pred_cb, average='macro'),
    'f1_pos': f1_score(y_test, y_pred_cb, pos_label=1),
    'precision_pos': precision_score(y_test, y_pred_cb, pos_label=1),
    'recall_pos': recall_score(y_test, y_pred_cb, pos_label=1)
}
cb_clf.save_model(str(MODEL_OUT_DIR / "catboost.cbm"))
results.append(res_cb)


res_df = pd.DataFrame(results).set_index('name')
print("\n=== SUMMARY ===")
print(res_df.sort_values('f1_pos', ascending=False))


plt.figure(figsize=(8,5))
res_df['f1_pos'].sort_values(ascending=True).plot.barh()
plt.xlabel("F1 (positive class = подписка)")
plt.title("Сравнение моделей по F1 для положительного класса")
plt.tight_layout()
plt.savefig("model_f1_pos.png")
print("Saved chart: model_f1_pos.png")


feature_names = []
feature_names += num_cols
```

```
ohe = preprocessor.named_transformers_['ohe']
if hasattr(ohe, 'get_feature_names_out'):
    ohe_names = list(ohe.get_feature_names_out(low_card))
else:
    ohe_names = []
feature_names += ohe_names
feature_names += high_card

rf_model = rf_clf
try:
    importances = rf_model.feature_importances_
    if len(importances) == len(feature_names):
        imp_df = pd.Series(importances, index=feature_names).sort_values(ascending=False).head(30)
        print("\nTop features (RandomForest):")
        print(imp_df)
        imp_df.head(20).plot.barh(figsize=(8,6))
        plt.tight_layout()
        plt.savefig("rf_feature_importance.png")
        print("Saved: rf_feature_importance.png")
    else:
        print("Не совпадает длина feature_importances_ и feature_names; пропускаю вывод.")
except Exception as e:
    print("Ошибка при RF importances:", e)

try:
    xgb_imp = xgb_clf.get_booster().get_score(importance_type='gain')
    xgb_imp_s = pd.Series(xgb_imp).sort_values(ascending=False).head(30)
    print("\nTop features (XGBoost, raw indices):")
    print(xgb_imp_s)
except Exception as e:
    print("Ошибка при XGBoost importance:", e)

try:
    cb_imp = cb_clf.get_feature_importance(prettified=True)
    print("\nCatBoost feature importance (top):")
    print(cb_imp[:20])
except Exception as e:
    print("Ошибка при CatBoost importance:", e)

best_by_f1pos = res_df['f1_pos'].idxmax()
```

```
print(f"\nRecommendation: лучшая модель по F1 для положительного класса: {best_by_f1pos}")


res_df.to_csv("models_comparison_summary.csv")

print("Saved summary to models_comparison_summary.csv")
```

Результат работы программы:

```
Data shape: (11162, 17)

Columns: ['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact',
'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'deposit']

   age         job  marital  education default  balance housing loan  contact  \

0   59      admin.  married  secondary      no     2343     yes   no  unknown

1   56      admin.  married  secondary      no       45      no   no  unknown

2   41  technician  married  secondary      no     1270     yes   no  unknown


   day month  duration  campaign  pdays  previous poutcome deposit

0    5   may      1042         1     -1         0  unknown     yes

1    5   may      1467         1     -1         0  unknown     yes

2    5   may      1389         1     -1         0  unknown     yes

Categorical (9): ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month',
'poutcome']

Numerical (7): ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

High-cardinality categorical: []

Low-cardinality categorical: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month', 'poutcome']

Train/test shapes: (8929, 16) (2233, 16) 4231 / 1058


=== TRAIN & EVAL: decision_tree ===

              precision    recall  f1-score   support


           0     0.8025    0.8196    0.8109      1175

           1     0.7948    0.7760    0.7853      1058


    accuracy                         0.7989      2233

   macro avg     0.7986    0.7978    0.7981      2233

weighted avg     0.7988    0.7989    0.7988      2233


Confusion matrix:

 [[963 212]

 [237 821]]


=== TRAIN & EVAL: random_forest ===
```

```
              precision    recall  f1-score   support

           0     0.8939    0.8315    0.8616      1175
           1     0.8263    0.8904    0.8571      1058

    accuracy                         0.8594      2233
   macro avg     0.8601    0.8609    0.8593      2233
weighted avg     0.8619    0.8594    0.8595      2233

Confusion matrix:
 [[977 198]
 [116 942]]

=== TRAIN & EVAL: adaboost ===
              precision    recall  f1-score   support

           0     0.8315    0.8187    0.8250      1175
           1     0.8020    0.8157    0.8088      1058

    accuracy                         0.8173      2233
   macro avg     0.8168    0.8172    0.8169      2233
weighted avg     0.8175    0.8173    0.8174      2233

Confusion matrix:
 [[962 213]
 [195 863]]

=== XGBoost ===
              precision    recall  f1-score   support

           0     0.8854    0.8485    0.8666      1175
           1     0.8392    0.8781    0.8582      1058

    accuracy                         0.8625      2233
   macro avg     0.8623    0.8633    0.8624      2233
weighted avg     0.8635    0.8625    0.8626      2233


=== CatBoost ===
              precision    recall  f1-score   support
```

```
           0     0.8975    0.8417    0.8687      1175
           1     0.8355    0.8932    0.8634      1058

    accuracy                         0.8661      2233
   macro avg     0.8665    0.8674    0.8660      2233
weighted avg     0.8681    0.8661    0.8662      2233
```

=== SUMMARY ===

```
               f1_macro    f1_pos  precision_pos  recall_pos
name
catboost       0.866047  0.863408       0.835544    0.893195
xgboost        0.862389  0.858199       0.839205    0.878072
random_forest  0.859347  0.857143       0.826316    0.890359
adaboost       0.816926  0.808810       0.802045    0.815690
decision_tree  0.798109  0.785270       0.794773    0.775992
Saved chart: model_f1_pos.png

Top features (RandomForest):
duration               0.330812
balance                0.078214
age                    0.072111
day                    0.066242
campaign               0.034961
poutcome_success       0.030213
pdays                  0.029822
contact_unknown        0.022842
previous               0.020100
contact_cellular       0.017401
housing_yes            0.016087
housing_no             0.015509
month_apr              0.013934
poutcome_unknown       0.012934
month_may              0.012590
month_mar              0.011707
month_aug              0.010780
month_oct              0.010391
month_jun              0.010149
education_secondary    0.009702
```

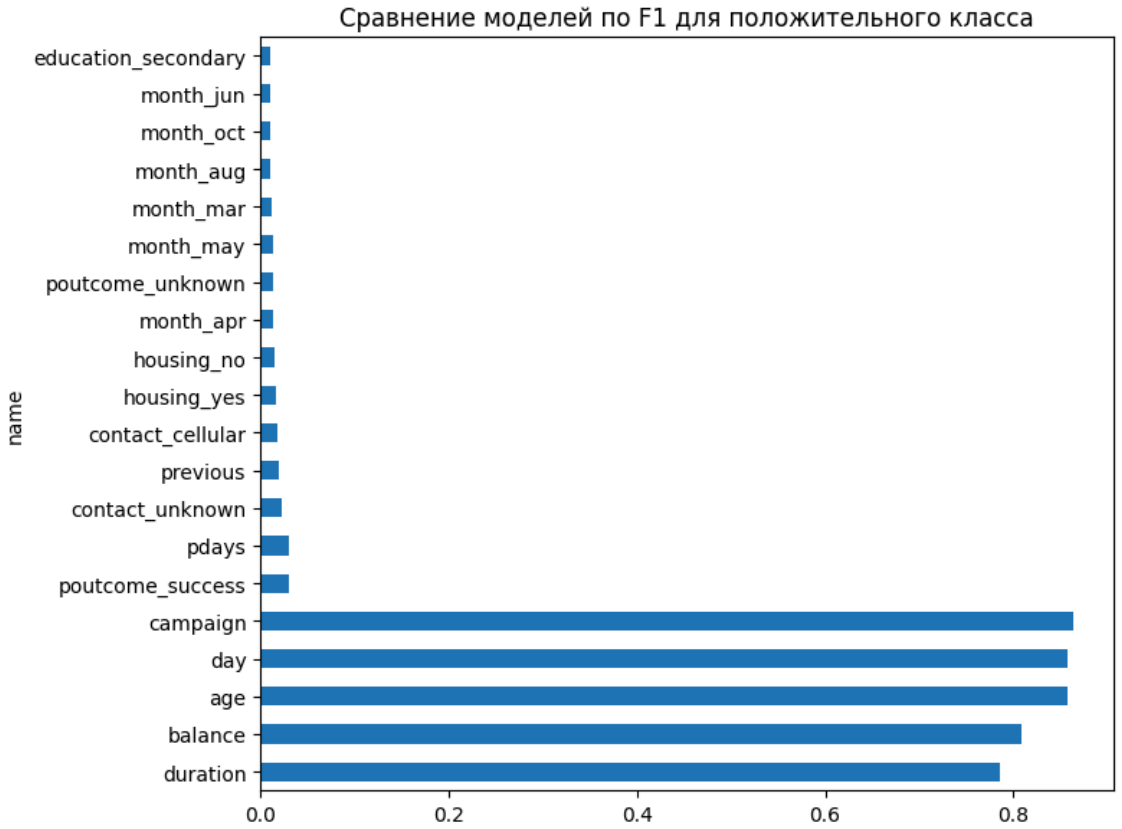
```
marital_married        0.009666
education_tertiary     0.009333
month_feb              0.009184
marital_single         0.008834
month_jul              0.008827
month_nov              0.008686
job_technician         0.008354
job_blue-collar        0.008248
job_management         0.008019
job_admin.             0.007689
dtype: float64
Saved: rf_feature_importance.png

Top features (XGBoost, raw indices):
f49    24.258430
f34    17.364965
f42    14.117775
f45     7.139676
f28     6.240437
f3      5.918829
f46     5.573921
f44     5.201066
f41     4.927559
f35     4.681825
f6      4.479147
f36     4.148084
f38     3.947485
f37     3.811176
f43     3.575876
f40     3.456488
f30     3.178613
f8      2.580063
f5      2.432737
f32     2.409832
f33     2.067122
f2      2.062860
f15     1.952493
f39     1.799488
f24     1.739828
f13     1.675115
```

```
f47     1.665975
f14     1.630935
f0      1.625223
f19     1.600963
dtype: float64


CatBoost feature importance (top):
   Feature Id  Importances
0    duration    29.808389
1       month    16.744350
2     contact     9.403571
3         day     8.425794
4    poutcome     8.403496
5         age     4.323723
6     balance     3.748753
7     housing     3.668291
8         job     3.520164
9   education     2.605224
10      pdays     2.602575
11   campaign     2.447914
12   previous     1.565346
13    marital     1.483031
14       loan     1.109491
15    default     0.139888


Recommendation: лучшая модель по F1 для положительного класса: catboost
Pipeline finished. Посмотрите файлы: models_comparison_summary.csv, model_f1_pos.png, rf_feature_importance.png.
```

Рассмотрите баланс между precision и recall в зависимости от бизнес-цели:

- Если важно не пропустить потенциального клиента (recall важнее), выбирайте модель с более высоким recall_pos.

- Если важно уменьшить ложные срабатывания (чтобы не тратить ресурсы на незаинтересованных), ориентируйтесь на precision_pos.



Для более удобного просмотра организовала данные в таблицу

Модель	F1_macro	F1 (положит.)	Precision	Recall
CatBoost	**0.8660**	**0.8634**	0.8355	**0.8932**
XGBoost	0.8624	0.8582	0.8392	0.8781
RandomForest	0.8593	0.8571	0.8263	0.8904
AdaBoost	0.8169	0.8088	0.8020	0.8157
DecisionTree	0.7981	0.7853	0.7948	0.7760

CatBoost дал наилучшее качество по F1-score и самый высокий recall — значит, он лучше всех распознаёт клиентов, которые действительно откликнутся (минимум пропущенных потенциальных клиентов). XGBoost и RandomForest тоже показали высокие результаты — разница с CatBoost минимальна (<0.01 по F1). AdaBoost и одиночное дерево сильно отстают, что ожидаемо.

Для выбора подходящего варианта нужно опираться на цель :

Если цель бизнеса — не упустить потенциальных клиентов (максимизировать recall):

1. выбираем CatBoost (recall_pos = 0.893) или RandomForest (0.890).
2. они «чувствительнее» и находят почти всех, кто согласится на вклад.

Если цель — не беспокоить лишних клиентов (важнее precision):

1. тогда XGBoost (precision_pos = 0.839) или CatBoost (0.835) оптимальны.
2. они меньше ошибаются, но всё ещё сохраняют высокий recall.

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