Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №2
По дисциплине: «ОМО»
Тема:” Линейные модели для задач регрессии и классификации”

Выполнил:
Студент 3-го курса
Группы АС-66
Савинец М.Д.
Проверил:
Крощенко А.А.

Брест 2025

Цель: Изучить применение линейной и логистической регрессии для решения практических задач. Научиться обучать модели, оценивать их качество с помощью соответствующих метрик и интерпретировать результаты.

Вариант 2

Вариант 2

• Регрессия (Прогнозирование медицинских расходов)

1. Medical Cost Personal Datasets

2. Предсказать страховые выплаты (charges)

3. Задания:

§ загрузите и обработайте категориальные признаки (например, sex, smoker);

§ обучите модель линейной регрессии для предсказания charges;

§ рассчитайте MAE (Mean Absolute Error) и R2;

§ визуализируйте зависимость charges от bmi (индекс массы тела) с помощью диаграммы рассеяния и линии регрессии.

• Классификация (Диагностика заболеваний сердца)

1. Heart Disease UCI

2. Предсказать наличие у пациента болезни сердца (target)

3. Задания:

§ загрузите данные и разделите их на обучающую и тестовую выборки;

§ обучите модель логистической регрессии;

§ оцените модель с помощью Accuracy, Precision, Recall и F1-score;

§ постройте матрицу ошибок.

```
import json
import pathlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import (
    mean_absolute_error, r2_score,
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix
)
from sklearn.pipeline import Pipeline

OUT_DIR = pathlib.Path("outputs")
OUT_DIR.mkdir(exist_ok=True, parents=True)
```

```
INSURANCE_PATH = pathlib.Path("medical_cost_personal_dataset.csv")
HEART_PATH = pathlib.Path("heart_disease_uci.csv    ")


def require_file(path: pathlib.Path):
    if not path.exists():
        raise FileNotFoundError(f"Файл не найден: {path}. Поместите
{path.name} рядом со скриптом.")


def save_confusion_matrix(cm: np.ndarray, classes, path: pathlib.Path, title:
str = "Confusion Matrix"):
    fig = plt.figure(figsize=(5, 4), dpi=120)
    ax = fig.add_subplot(111)
    im = ax.imshow(cm, interpolation='nearest')
    ax.set_title(title)
    plt.colorbar(im)
    tick_marks = np.arange(len(classes))
    ax.set_xticks(tick_marks, classes)
    ax.set_yticks(tick_marks, classes)
    ax.set_ylabel('True label')
    ax.set_xlabel('Predicted label')

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], 'd'),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    fig.savefig(path, bbox_inches="tight")
    plt.close(fig)


# -------------------- Регрессия --------------------
def run_regression():
    print("\n=== Регрессия: Medical Cost — предсказание charges ===")
    require_file(INSURANCE_PATH)
    df = pd.read_csv(INSURANCE_PATH)

    print("\nПервые строки данных (insurance.csv):")
    print(df.head())

    target_col = "charges"
    categorical = ["sex", "smoker", "region"]
    numeric = [c for c in df.columns if c not in categorical + [target_col]]
# age, bmi, children

    X = df[categorical + numeric]
    y = df[target_col]

    preprocessor = ColumnTransformer(
        transformers=[
            ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"),
categorical),
            ("num", "passthrough", numeric)
        ]
    )

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    pipe = Pipeline([("prep", preprocessor), ("model", LinearRegression())])
    pipe.fit(X_train, y_train)
```

```
    y_pred = pipe.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    r2  = r2_score(y_test, y_pred)

    print(f"\nMAE: {mae:.3f}")
    print(f"R2 : {r2:.3f}")

    # Рассеяние charges vs bmi + линия регрессии
    df_plot = df[["bmi", "charges"]].dropna().sort_values("bmi")
    X_bmi = df_plot[["bmi"]].values
    y_ch  = df_plot["charges"].values

    uni_model = LinearRegression().fit(X_bmi, y_ch)
    y_line = uni_model.predict(X_bmi)

    fig = plt.figure(figsize=(6, 4), dpi=120)
    ax = fig.add_subplot(111)
    ax.scatter(df_plot["bmi"], df_plot["charges"], s=10, alpha=0.6)
    ax.plot(df_plot["bmi"], y_line, linewidth=2)
    ax.set_xlabel("BMI (индекс массы тела)")
    ax.set_ylabel("Charges (страховые выплаты)")
    ax.set_title("Зависимость charges от BMI и линия регрессии")
    fig.tight_layout()
    out_path = OUT_DIR / "regression_bmi_charges.png"
    fig.savefig(out_path, bbox_inches="tight")
    plt.close(fig)

    with open(OUT_DIR / "regression_metrics.json", "w", encoding="utf-8") as
f:
        json.dump({"MAE": float(mae), "R2": float(r2)}, f,
ensure_ascii=False, indent=2)

    print(f"[OK] График сохранён: {out_path}")
    print(f"[OK] Метрики сохранены: {OUT_DIR / 'regression_metrics.json'}")


# -------------------- Классификация (робастная к вариантам) ----------------
----
def derive_binary_target(df: pd.DataFrame) -> pd.Series:
    """
    Возвращает бинарную цель 'target' по правилам:
    - Если 'target' есть:
        * Если двоичная (nunique==2) — берём как есть (пытаемся привести к
{0,1}).
        * Если многоклассовая — bin: (target > 0) -> 1, else 0.
    - Иначе, если есть 'num' (классический столбец UCI) — target = (num > 0).
    - Иначе — KeyError.
    """
    if "target" in df.columns:
        t = df["target"]
        # Попробуем привести к числовому виду, если это строки
        t_num = pd.to_numeric(t, errors="coerce")
        if t.nunique(dropna=True) == 2:
            # двоичный как есть (если строки — попробуем map в 0/1)
            if t_num.notna().all():
                return (t_num > 0).astype(int) if set(t_num.unique()) - {0,1}
else t_num.astype(int)
            # строковые метки: попытаемся маппить "yes/no",
"presence/absence", т.п.
            lower = t.astype(str).str.lower()
            if set(lower.unique()) <= {"yes","no"}:
                return (lower=="yes").astype(int)
            if set(lower.unique()) <= {"present","absent"}:
```

```
                return (lower=="present").astype(int)
            # в общем случае — категориальный to codes в {0,1} по порядку
            codes = pd.Categorical(t).codes
            # нормируем к {0,1}
            return (codes == codes.max()).astype(int)
        else:
            # многоклассовая — как в UCI: наличие болезни = target>0
            if t_num.notna().any():
                return (t_num > 0).astype(int)
            # если не числа — попробуем категориально: последние классы
считаем "болезнь"
            codes = pd.Categorical(t).codes
            return (codes > codes.min()).astype(int)

    if "num" in df.columns:
        t_num = pd.to_numeric(df["num"], errors="coerce")
        if t_num.isna().any():
            # Если есть NaN — заменим их на 0 (как "нет болезни") или
отбросим
            # Здесь возьмём простой вариант — NaN -> 0
            t_num = t_num.fillna(0)
        return (t_num > 0).astype(int)

    raise KeyError("Не найден ни 'target', ни 'num' для построения целевой
переменной.")


def run_classification():
    print("\n=== Классификация: Heart Disease UCI — предсказание target ===")
    require_file(HEART_PATH)
    df = pd.read_csv(HEART_PATH)

    print("\nПервые строки данных (heart.csv):")
    print(df.head())
    print(f"\n[INFO] Колонки: {list(df.columns)}")

    # Чистка "?" → NaN
    df = df.replace("?", np.nan)

    # Создадим бинарный target
    try:
        y = derive_binary_target(df)
        print(f"[INFO] Целевая переменная сформирована. Положительных
классов: {int(y.sum())} из {len(y)}")
    except Exception as e:
        raise ValueError(f"Не удалось сформировать бинарную цель: {e}")

    # Признаки: всё кроме исходных столбцов цели ('target'/'num'), если они
есть
    drop_cols = [c for c in ["target", "num"] if c in df.columns]
    X = df.drop(columns=drop_cols, errors="ignore")

    # Приводим числовые столбцы к float, остальное оставляем как object
    X_numeric = X.select_dtypes(include=[np.number]).columns.tolist()

    # Попробуем добить числовые через to_numeric (на случай, если числовые
лежат строками)
    for c in X.columns:
        if c not in X_numeric:
            coerced = pd.to_numeric(X[c], errors="coerce")
            # если после приведения осталось достаточно не-NaN — считаем
числовым
            if coerced.notna().sum() > 0 and (coerced.notna().mean() > 0.7):
                X[c] = coerced
```

```
    # Обновим списки
    X_numeric = X.select_dtypes(include=[np.number]).columns.tolist()
    X_categorical = [c for c in X.columns if c not in X_numeric]

    # Удалим строки, где есть пропуски (простой вариант)
    Xy = pd.concat([X, y.rename("target")], axis=1)
    before = len(Xy)
    Xy = Xy.dropna()
    after = len(Xy)
    if after < before:
        print(f"[INFO] Удалено строк с пропусками: {before - after} (из
{before})")
    X = Xy.drop(columns=["target"])
    y = Xy["target"].astype(int)

    # Конвейер: OneHot для категорий, StandardScaler для числовых
    transformers = []
    if X_categorical:
        transformers.append(("cat", OneHotEncoder(handle_unknown="ignore"),
X_categorical))
    if X_numeric:
        transformers.append(("num", StandardScaler(), X_numeric))

    if not transformers:
        raise ValueError("Не обнаружены признаки для обучения (все колонки
пустые после очистки).")

    preprocessor = ColumnTransformer(transformers=transformers)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    pipe = Pipeline([("prep", preprocessor), ("model",
LogisticRegression(max_iter=1000, solver="lbfgs"))])
    pipe.fit(X_train, y_train)

    y_pred = pipe.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec  = recall_score(y_test, y_pred, zero_division=0)
    f1   = f1_score(y_test, y_pred, zero_division=0)

    print(f"\nAccuracy : {acc:.3f}")
    print(f"Precision: {prec:.3f}")
    print(f"Recall   : {rec:.3f}")
    print(f"F1-score : {f1:.3f}")

    cm = confusion_matrix(y_test, y_pred)
    cm_path = OUT_DIR / "classification_confusion_matrix.png"
    save_confusion_matrix(cm, classes=("No disease", "Disease"),
path=cm_path, title="Heart Disease — Confusion Matrix")

    with open(OUT_DIR / "classification_metrics.json", "w", encoding="utf-8")
as f:
        json.dump({"Accuracy": float(acc), "Precision": float(prec),
"Recall": float(rec), "F1": float(f1)}, f, ensure_ascii=False, indent=2)

    print(f"[OK] Матрица ошибок сохранена: {cm_path}")
    print(f"[OK] Метрики сохранены: {OUT_DIR /
'classification_metrics.json'}")
```

```
def main():
    run_regression()
    run_classification()


if __name__ == "__main__":
    main()
```

График регрессия:

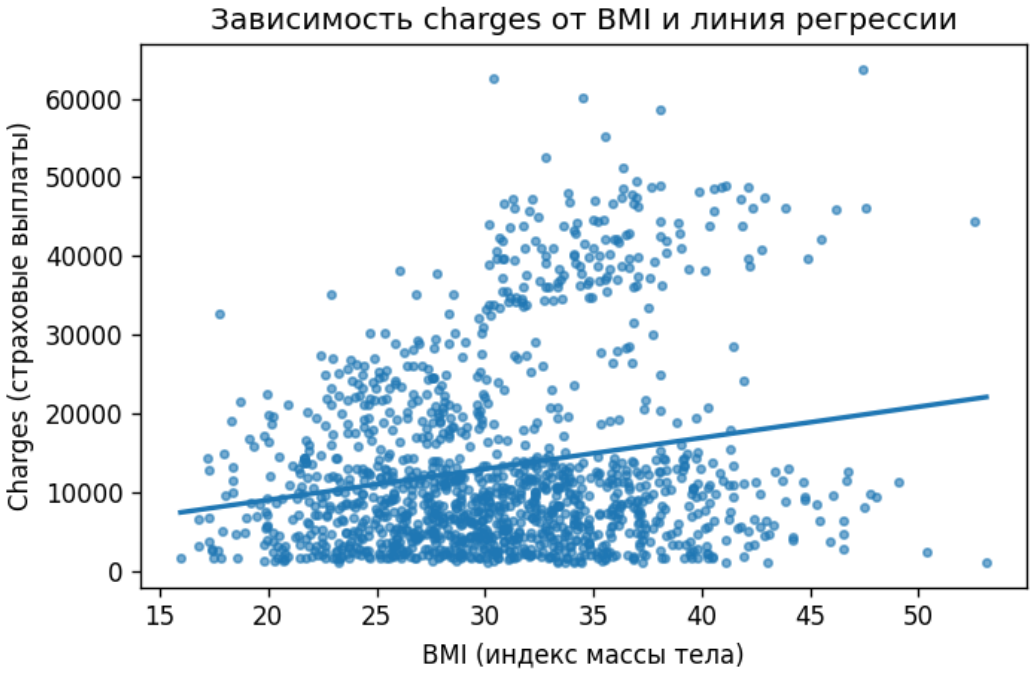
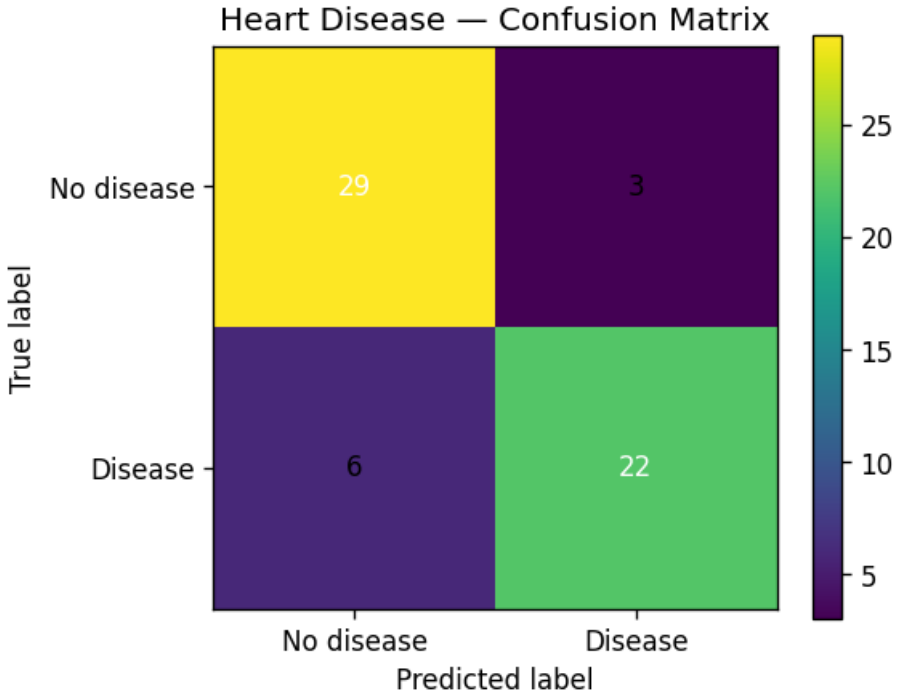


График классификация:



Результаты:

```
=== Регрессия: Medical Cost — предсказание charges ===

Первые строки данных (insurance.csv):
   age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520

MAE: 4181.194
R2 : 0.784
```

```
=== Классификация: Heart Disease UCI — предсказание target ===

Первые строки данных (heart.csv):
   id  age     sex    dataset  ...        slope   ca              thal num
0   1   63    Male  Cleveland  ...  downsloping  0.0       fixed defect   0
1   2   67    Male  Cleveland  ...         flat  3.0            normal   2
2   3   67    Male  Cleveland  ...         flat  2.0  reversable defect   1
3   4   37    Male  Cleveland  ...  downsloping  0.0            normal   0
4   5   41  Female  Cleveland  ...    upsloping  0.0            normal   0

[5 rows x 16 columns]

[INFO] Колонки: ['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']
[INFO] Целевая переменная сформирована. Положительных классов: 509 из 920
[INFO] Удалено строк с пропусками: 621 (из 920)

Accuracy : 0.850
Precision: 0.880
Recall   : 0.786
F1-score : 0.830
```

Вывод: в результате выполнения данной лабораторной работы изучили применение линейной и логистической регрессии для решения практических задач. Научились обучать модели, оценивать их качество с помощью соответствующих метрик и интерпретировать результаты.