НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Системы искусственного интеллекта

Лабораторная работа № 1

Выполнил студент

Набокова Алиса Владиславовна

Группа № P3320

Преподаватель: Королёва Юлия Александровна

г. Санкт-Петербург

2025

Задание:

Реализовать класс модели линейной регрессии и класс логистической регрессии без использования готовых моделей из библиотек.

Для обучения использовать следующие датасеты:

- Для линейной регрессии — датасет California_Houses.csv
- Для логистической регрессии — датасет Titanic-Dataset.csv

Выполнить предобработку данных:

- удалить или преобразовать все категориальные признаки (object), оставить только числовые типы (float, int, bool),

Провести обучение моделей

Вывести метрики качества моделей

Алгоритм линейной регрессии

y_pred = Xw + b

err = y_pred – y

dw = (2 / N) * X^T * error

db = (2 / N) * sum(error)

w -= alpha * dw

b -= alpha * db

Предобработка датасета

На рисунках 1-3 представлена матрица корреляции признаков исходного датасета, фрагмент кода для ее построения и итоговая тепловая карта после удаления зависимых параметров

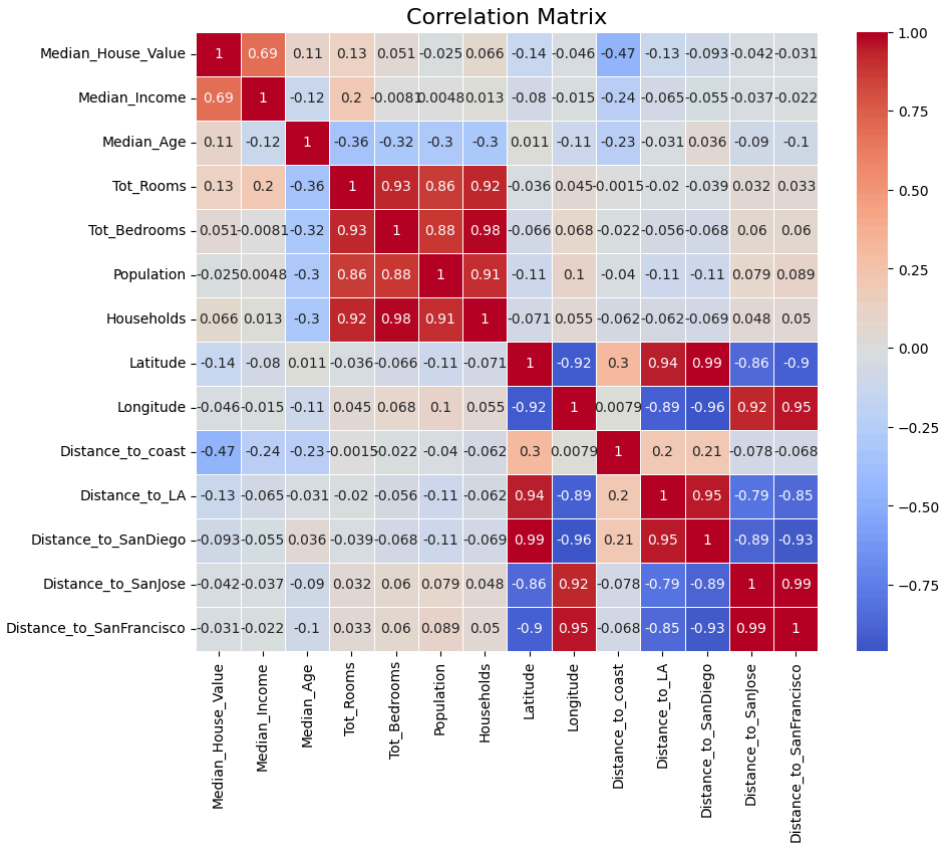


Рисунок 1. Матрица корреляции

```python
df_houses = pd.DataFrame(data_houses)

correlation_matrix_houses = df_houses.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_houses,
            annot=True,
            cmap="coolwarm",
            center=0,
            linewidths=0.5)
plt.title("Correlation Matrix", fontsize=16)
plt.show()
```

Рисунок 2. Код построения матрицы корреляции

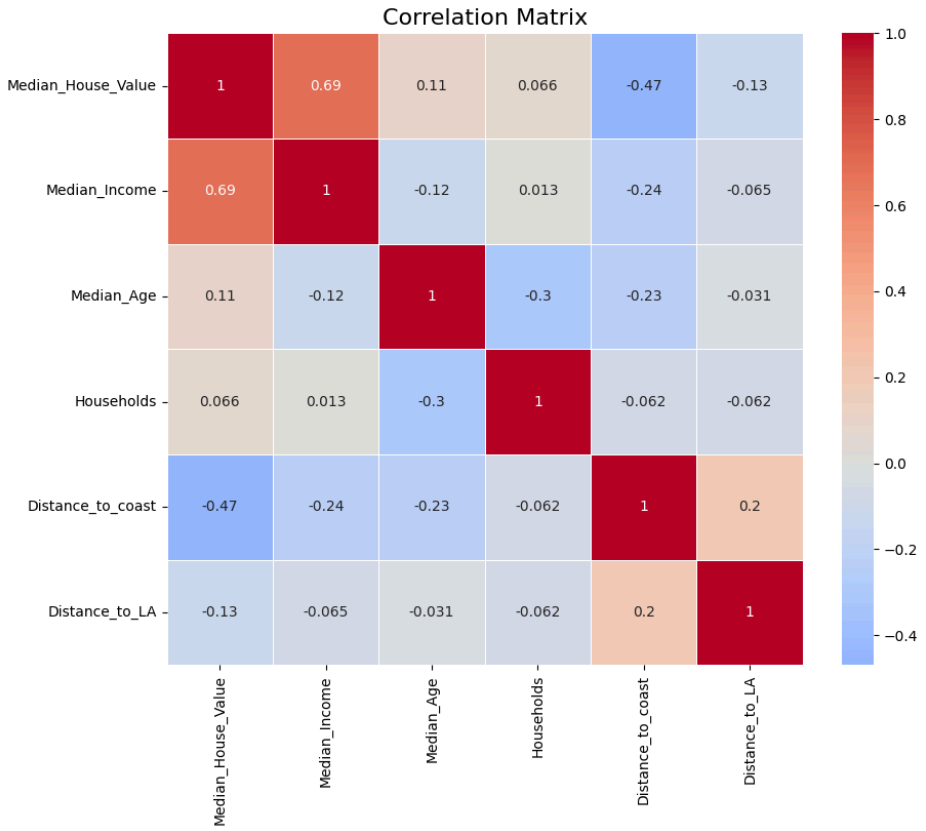


Рисунок 3. Матрица корреляции после фильтрации зависимых параметров

На рисунке 4-5 представлена подготовка данных для последующего обучения модели линейной регрессии

```
X = df_houses_filted.drop(columns='Median_House_Value')
y = df_houses['Median_House_Value']

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

Рисунок 4. Разделение данных на тренировочную и тестовую выборки

```
X_mean, X_std = np.mean(X_train, axis=0), np.std(X_train, axis=0)
X_std[X_std == 0] = 1
X_train_scaled = (X_train - X_mean) / X_std
X_test_scaled = (X_test - X_mean) / X_std
```

Рисунок 5. Стандартизация данных

На рисунках 6-7 приведен алгоритм обучения линейной регрессии и его применение

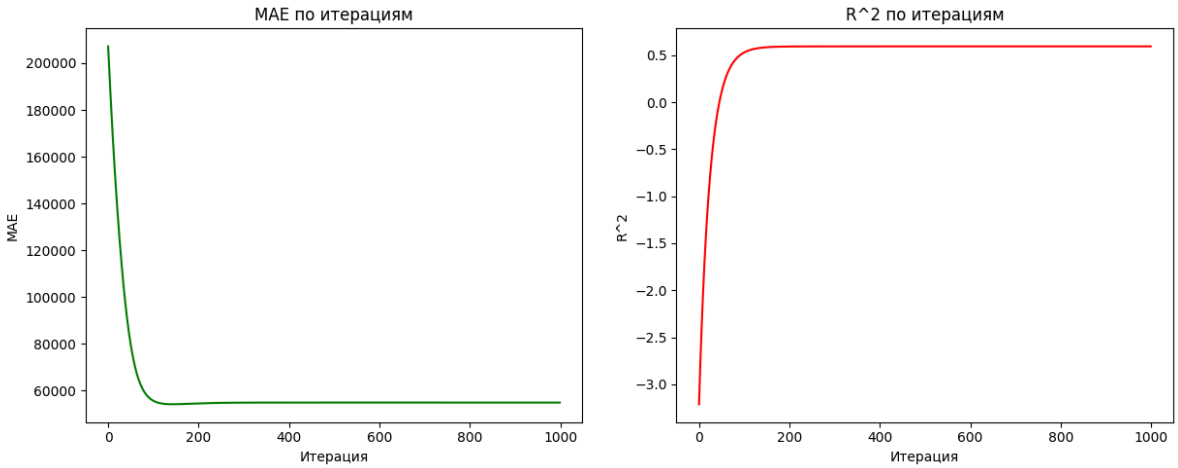
```python
class LinearRegressionMatrix:
    def __init__(self, lr=0.01, n_iters=1000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

        self.history_mae = []
        self.history_r2 = []

    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y, dtype=float).reshape(-1)

        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
          y_pred = np.dot(X, self.weights) + self.bias

          error = y_pred - y

          dw = (2 / n_samples) * np.dot(X.T, error)
          db = (2 / n_samples) * np.sum(error)

          self.weights -= self.lr * dw
          self.bias -= self.lr * db

          mse = np.mean(error**2)
          mae = np.mean(np.abs(error))
          r2 = 1 - np.sum(error**2) / np.sum((y - np.mean(y))**2)

          self.history_mae.append(mae)
          self.history_r2.append(r2)

    def predict(self, X):
        X = np.array(X, dtype=float)
        return np.dot(X, self.weights) + self.bias
```

Рисунок 6. Класс модели линейной регрессии

```python
model = LinearRegressionMatrix(lr=0.01, n_iters=1000)
model.fit(X_train_scaled, y_train)
```

Рисунок 7. Обучение модели

На рисунке 8-9 изображены графики оценки качества модели линейной регрессии по метрикам MAE и R^2, а также представлен фрагмент кода для их построения




```
MAE: 74796.8117
R^2: 0.5731
```

Рисунок 8. Оценка модели (MAE, R^2)

```python
plt.figure(figsize=(18,5))

# MAE
plt.subplot(1,2,1)
plt.plot(model.history_mae, color='green')
plt.title("MAE по итерациям")
plt.xlabel("Итерация")
plt.ylabel("MAE")

# R^2
plt.subplot(1,2,2)
plt.plot(model.history_r2, color='red')
plt.title("R^2 по итерациям")
plt.xlabel("Итерация")
plt.ylabel("R^2")

plt.tight_layout()
plt.show()
```

Рисунок 9. Построение графиков оценки модели

Алгоритм логистической регрессии

y_pred = sigmoid(Xw + b)

err = y_pred – y

dw = (X^T * err) / N

db = sum(err) / N

w -= alpha *dw

b -= alpha *db

На рисунке 10 представлена подготовка данных для последующего обучения модели логистической регрессии

```
X = df_titanic_mean_fillted.drop(columns='Survived')
y = df_titanic_mean_fillted['Survived']

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

numeric_cols = ['PassengerId', 'Age', 'SibSp', 'Parch', 'Fare']

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

X_mean = X_train[numeric_cols].mean(axis=0)
X_std = X_train[numeric_cols].std(axis=0)

X_train_scaled[numeric_cols] = (X_train[numeric_cols] - X_mean) / X_std
X_test_scaled[numeric_cols] = (X_test[numeric_cols] - X_mean) / X_std
X_std[X_std == 0] = 1
```

Рисунок 10. Разделение данных на тренировочную и тестовую выборки

На рисунках 9-10 приведен алгоритм обучения логистической регрессии и его применение

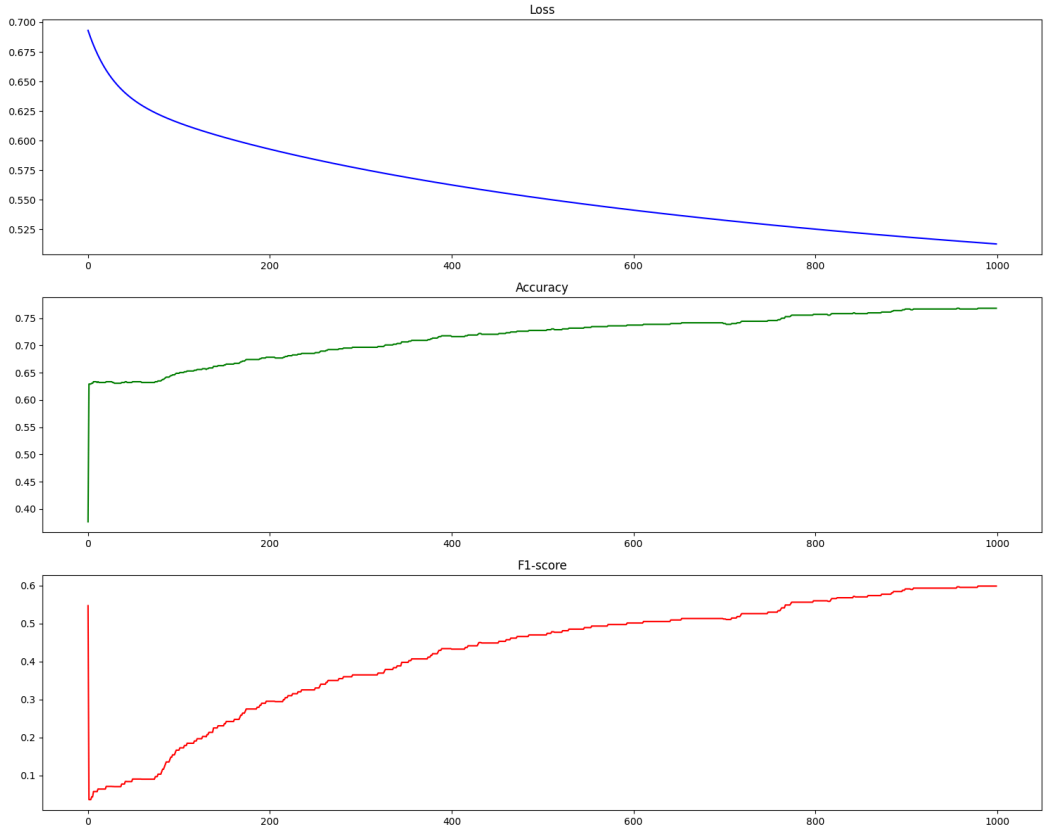
```python
class LogisticRegressionMatrix:
    def __init__(self, lr=0.01, n_iter=1000):
        self.lr = lr
        self.n_iter = n_iter
        self.weights = None
        self.bias = None

        self.loss_history = []
        self.acc_history = []
        self.f1_history = []

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y, dtype=float).reshape(-1)

        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iter):
            y_pred = self.sigmoid(np.dot(X, self.weights) + self.bias)

            err = y_pred - y

            dw = np.dot(X.T, err) / n_samples
            db = np.sum(err) / n_samples

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

            eps = 1e-15
            y_pred_clipped = np.clip(y_pred, eps, 1 - eps)
            loss = -np.mean(y * np.log(y_pred_clipped) + (1 - y) * np.log(1 - y_pred_clipped))
            self.loss_history.append(loss)


            preds = (y_pred >= 0.5).astype(int)
            acc = accuracy_score(y, preds)
            f1 = f1_score(y, preds)
            self.acc_history.append(acc)
            self.f1_history.append(f1)


    def predict_proba(self, X):
        return self.sigmoid(np.dot(X, self.weights) + self.bias)

    def predict(self, X, threshold=0.5):
        return (self.predict_proba(X) >= threshold).astype(int)
```

Рисунок 9. Класс для логической регрессии

```python
model = LogisticRegressionMatrix(lr=0.01, n_iter=1000)
model.fit(X_train_scaled, y_train)
```

Рисунок 10. Обучение модели

На рисунках 11-12 изображены графики оценки качества модели логистической регрессии по метрикам Log-Loss, Accuracy и F1-score, а также представлен фрагмент кода для их построения




```
Final Accuracy: 0.7988826815642458
Final F1-score: 0.75
```

Рисунок 11. Оценка модели (Log-Loss, Accuracy, F1_score)

```
plt.figure(figsize=(15,12))

plt.subplot(3,1,1)
plt.plot(model.loss_history, color='blue')
plt.title("Loss")

plt.subplot(3,1,2)
plt.plot(model.acc_history, color='green')
plt.title("Accuracy")

plt.subplot(3,1,3)
plt.plot(model.f1_history, color='red')
plt.title("F1-score")

plt.tight_layout()
plt.show()
```

Рисунок 12. Построение графиков оценки модели

Вывод

В ходе данной лабораторной работы я ознакомилась и реализовала алгоритмы для логистической и линейной регрессии, а также провела оценку обученным моделей.