Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина: «Системы искусственного интеллекта»

Лабораторная работа №3

«Линейная регрессия»

Выполнил:

студент группы ИВТАПбд-41

Князев А.Д.

Проверил:

преподаватель кафедры «ВТ»

Святов К.В.

Ульяновск, 2025

Оглавление

[1. Постановка задачи 3](#__RefHeading___Toc165_3389309196)

[2. Реализация 4](#__RefHeading___Toc165_3389309196_%25D0%2)

[3. Тестирование 8](#__RefHeading___Toc165_3389309196_%25D0%1)

[4. Вывод 9](#__RefHeading___Toc165_3389309196_%25D0%3)

# **1. Постановка задачи**

Написать программу, которая разделяет исходную выборку на обучающую и тестовую (training set, test set). Использовать стандартные функции (train\_test\_split и др. нельзя).

С использованием библиотеки scikit-learn обучить модель линейной регрессии по обучающей выборке пример

Проверить точность модели по тестовой выборке

Построить модель с использованием полиномиальной функции пример. Построить графики зависимости точности на обучающей и тестовой выборке от степени полиномиальной функции.

Построить модель с использованием регуляризации пример. На основе экспериментов подобрать параметры для регуляризации. Построить графики зависимости точности модели на обучающей и тестовой выборках от коэффициента регуляризации.

Вариант по заданию:

8) Communities and Crimes

# **2****. Реализация**

Для начала происходит загрузка и подготовка данных: Загружается датасет из UCI репозитория

Далее идёт разделение данных. Данные случайно делятся на тренировочную и тестовую выборки в пропорции 80/20 для оценки качества моделей.

Обучение базовой линейной регрессии: На тренировочных данных обучается простая линейная регрессия, которая служит базовым эталоном.

Создание и обучение моделей с полиномиальными признаками: Для степеней полинома от 1 до 4 создаются новые признаки (включая взаимодействия и степени исходных признаков), и обучаются линейные модели на этих расширенных признаках. Считаются ошибки на обучающей и тестовой выборках.

Обучение регуляризованных моделей Ridge и Lasso: Для разных значений параметра регуляризации alpha обучаются модели Ridge и Lasso регрессий, которые помогают бороться с переобучением и выбирают значимые признаки. Анализируются ошибки на обучающей и тестовой выборках.

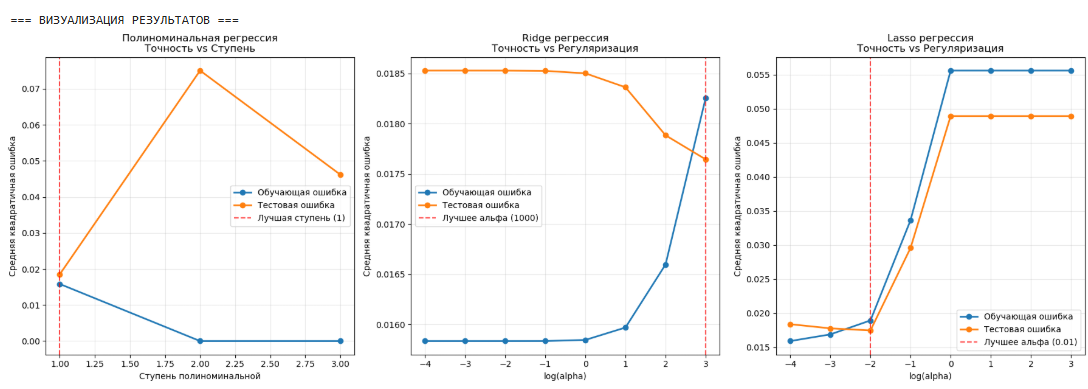
Сравнение моделей: Определяются лучшие параметры для полиномиальной регрессии (степень), Ridge и Lasso (alpha) по минимальному среднеквадратичному отклонению (MSE) на тестовой выборке. Выбирается лучшая модель среди них.

Визуализация результатов: Строятся графики зависимости ошибки от степени полинома и от параметра регуляризации (log(alpha)) для Ridge и Lasso, что помогает визуально оценить переобучение и недообучение.  
  
**Листинг 1.** Код программы

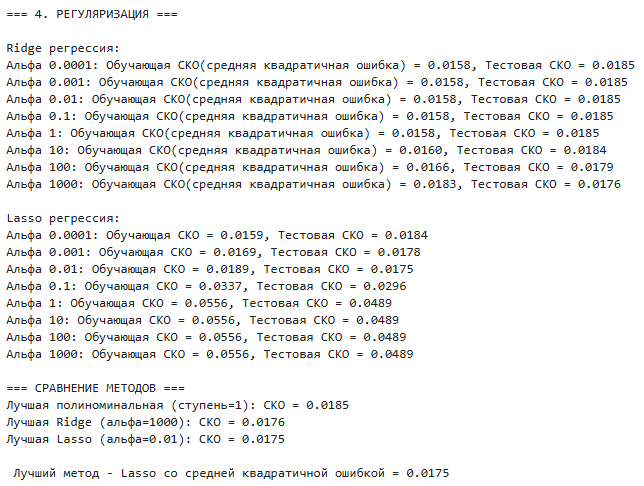
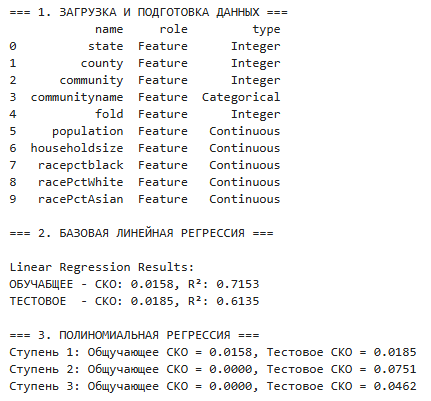
|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.preprocessing import PolynomialFeatures, StandardScaler  from ucimlrepo import fetch\_ucirepo  from sklearn.linear\_model import LinearRegression, Ridge, Lasso  from sklearn.metrics import mean\_squared\_error, r2\_score  import matplotlib.pyplot as plt  def data\_definition():  communities\_crime = fetch\_ucirepo(id=183)  print(communities\_crime.variables[['name', 'role', 'type']].head(10))  return communities\_crime.data.features, communities\_crime.data.targets  def splitter(X, y):  np.random.seed(42)  selection = np.random.permutation(len(X))  size = int(len(X) \* 0.2)  test\_selection = selection[:size]  train\_selection = selection[size:]  x\_train, x\_test = X.iloc[train\_selection], X.iloc[test\_selection]  y\_train, y\_test = y.iloc[train\_selection], y.iloc[test\_selection]  return x\_train, x\_test, y\_train, y\_test  def preprocess\_data(X):  """Предобработка данных"""  X\_processed = X.copy()    # Пропущенные значения меняются с '?' на NaN  X\_processed = X\_processed.replace('?', np.nan)    # Преобразование колонок в числовой формат  for col in X\_processed.columns:  X\_processed[col] = pd.to\_numeric(X\_processed[col], errors='coerce')    # Заполнение пропущенных значений медианой  X\_processed = X\_processed.fillna(X\_processed.median())    # Удаление констант  constant\_columns = [col for col in X\_processed.columns if X\_processed[col].nunique() <= 1]  X\_processed = X\_processed.drop(columns=constant\_columns)    return X\_processed  if \_\_name\_\_ == '\_\_main\_\_':  # 1. Загрузка и подготовка данных  print("=== 1. ЗАГРУЗКА И ПОДГОТОВКА ДАННЫХ ===")  x, y = data\_definition()    # Предобработка данных  x = preprocess\_data(x)    # Удаление строк с пропущенными значениями в целевой переменной  valid\_indices = ~y.isnull().any(axis=1)  x = x[valid\_indices]  y = y[valid\_indices]    x\_train, x\_test, y\_train, y\_test = splitter(x, y)    # Масштабирование для улучшения сходимости  scaler = StandardScaler()  x\_train\_scaled = scaler.fit\_transform(x\_train)  x\_test\_scaled = scaler.transform(x\_test)  # 2. БАЗОВАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ  print("\n=== 2. БАЗОВАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ ===")  regressor = LinearRegression().fit(x\_train\_scaled, y\_train)    # Оценка точности  y\_pred\_train = regressor.predict(x\_train\_scaled)  y\_pred\_test = regressor.predict(x\_test\_scaled)    print("\nLinear Regression Results:")  print(f"ОБУЧАБЩЕЕ - СКО: {mean\_squared\_error(y\_train, y\_pred\_train):.4f}, R²: {r2\_score(y\_train, y\_pred\_train):.4f}")  print(f"ТЕСТОВОЕ - СКО: {mean\_squared\_error(y\_test, y\_pred\_test):.4f}, R²: {r2\_score(y\_test, y\_pred\_test):.4f}")  # 3. ПОЛИНОМИАЛЬНАЯ РЕГРЕССИЯ  print("\n=== 3. ПОЛИНОМИАЛЬНАЯ РЕГРЕССИЯ ===")  train\_errors\_poly = []  test\_errors\_poly = []  train\_r2\_poly = []  test\_r2\_poly = []  degrees = range(1, 4)  for degree in degrees:  polynom = PolynomialFeatures(degree=degree)  x\_train\_poly = polynom.fit\_transform(x\_train\_scaled)  x\_test\_poly = polynom.transform(x\_test\_scaled)  model = LinearRegression()  model.fit(x\_train\_poly, y\_train)  y\_train\_pred = model.predict(x\_train\_poly)  y\_test\_pred = model.predict(x\_test\_poly)  train\_errors\_poly.append(mean\_squared\_error(y\_train, y\_train\_pred))  test\_errors\_poly.append(mean\_squared\_error(y\_test, y\_test\_pred))  train\_r2\_poly.append(r2\_score(y\_train, y\_train\_pred))  test\_r2\_poly.append(r2\_score(y\_test, y\_test\_pred))    print(f"Ступень {degree}: Общучающее СКО = {train\_errors\_poly[-1]:.4f}, Тестовое СКО = {test\_errors\_poly[-1]:.4f}")  # 4. РЕГУЛЯРИЗАЦИЯ  print("\n=== 4. РЕГУЛЯРИЗАЦИЯ ===")  alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]    # Ridge регрессия  ridge\_train\_errors = []  ridge\_test\_errors = []  ridge\_train\_r2 = []  ridge\_test\_r2 = []    # Lasso регрессия  lasso\_train\_errors = []  lasso\_test\_errors = []  lasso\_train\_r2 = []  lasso\_test\_r2 = []  print("\nRidge регрессия:")  for alpha in alphas:  # Ridge  ridge = Ridge(alpha=alpha)  ridge.fit(x\_train\_scaled, y\_train)    ridge\_train\_pred = ridge.predict(x\_train\_scaled)  ridge\_test\_pred = ridge.predict(x\_test\_scaled)    ridge\_train\_errors.append(mean\_squared\_error(y\_train, ridge\_train\_pred))  ridge\_test\_errors.append(mean\_squared\_error(y\_test, ridge\_test\_pred))  ridge\_train\_r2.append(r2\_score(y\_train, ridge\_train\_pred))  ridge\_test\_r2.append(r2\_score(y\_test, ridge\_test\_pred))    print(f"Альфа {alpha}: Обучающая СКО(средняя квадратичная ошибка) = {ridge\_train\_errors[-1]:.4f}, Тестовая СКО = {ridge\_test\_errors[-1]:.4f}")  print("\nLasso регрессия:")  for alpha in alphas:  # Lasso  lasso = Lasso(alpha=alpha, max\_iter=10000)  lasso.fit(x\_train\_scaled, y\_train.values.ravel())    lasso\_train\_pred = lasso.predict(x\_train\_scaled)  lasso\_test\_pred = lasso.predict(x\_test\_scaled)    lasso\_train\_errors.append(mean\_squared\_error(y\_train, lasso\_train\_pred))  lasso\_test\_errors.append(mean\_squared\_error(y\_test, lasso\_test\_pred))  lasso\_train\_r2.append(r2\_score(y\_train, lasso\_train\_pred))  lasso\_test\_r2.append(r2\_score(y\_test, lasso\_test\_pred))    print(f"Альфа {alpha}: Обучающая СКО = {lasso\_train\_errors[-1]:.4f}, Тестовая СКО = {lasso\_test\_errors[-1]:.4f}")  # Сравнение результатов  print("\n=== СРАВНЕНИЕ МЕТОДОВ ===")    # Лучшая полиномиальная модель  best\_poly\_idx = np.argmin(test\_errors\_poly)  best\_poly\_degree = degrees[best\_poly\_idx]  best\_poly\_error = test\_errors\_poly[best\_poly\_idx]    # Лучшая Ridge модель  best\_ridge\_idx = np.argmin(ridge\_test\_errors)  best\_ridge\_alpha = alphas[best\_ridge\_idx]  best\_ridge\_error = ridge\_test\_errors[best\_ridge\_idx]    # Лучшая Lasso модель  best\_lasso\_idx = np.argmin(lasso\_test\_errors)  best\_lasso\_alpha = alphas[best\_lasso\_idx]  best\_lasso\_error = lasso\_test\_errors[best\_lasso\_idx]    print(f"Лучшая полиноминальная (ступень={best\_poly\_degree}): СКО = {best\_poly\_error:.4f}")  print(f"Лучшая Ridge (альфа={best\_ridge\_alpha}): СКО = {best\_ridge\_error:.4f}")  print(f"Лучшая Lasso (альфа={best\_lasso\_alpha}): СКО = {best\_lasso\_error:.4f}")    # Определение лучшего метода  methods = {  'Linear': mean\_squared\_error(y\_test, y\_pred\_test),  'Polynomial': best\_poly\_error,  'Ridge': best\_ridge\_error,  'Lasso': best\_lasso\_error  }  best\_method = min(methods, key=methods.get)  print(f"\n Лучший метод - {best\_method} со средней квадратичной ошибкой = {methods[best\_method]:.4f}")  # ВИЗУАЛИЗАЦИЯ  print("\n=== ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ===")  plt.figure(figsize=(18, 6))    # 3. График полиномиальной регрессии  plt.subplot(1, 3, 1)  plt.plot(degrees, train\_errors\_poly, 'o-', label='Обучающая ошибка', linewidth=2)  plt.plot(degrees, test\_errors\_poly, 'o-', label='Тестовая ошибка', linewidth=2)  plt.axvline(x=best\_poly\_degree, color='red', linestyle='--', alpha=0.7, label=f'Лучшая ступень ({best\_poly\_degree})')  plt.xlabel('Ступень полиноминальной')  plt.ylabel('Средняя квадратичная ошибка')  plt.title('Полиноминальная регрессия\nТочность vs Ступень')  plt.legend()  plt.grid(True, alpha=0.3)    # 4. График Ridge регрессии  plt.subplot(1, 3, 2)  plt.plot(np.log10(alphas), ridge\_train\_errors, 'o-', label='Обучающая ошибка', linewidth=2)  plt.plot(np.log10(alphas), ridge\_test\_errors, 'o-', label='Тестовая ошибка', linewidth=2)  plt.axvline(x=np.log10(best\_ridge\_alpha), color='red', linestyle='--', alpha=0.7,  label=f'Лучшее альфа ({best\_ridge\_alpha})')  plt.xlabel('log(alpha)')  plt.ylabel('Средняя квадратичная ошибка')  plt.title('Ridge регрессия\nТочность vs Регуляризация')  plt.legend()  plt.grid(True, alpha=0.3)    # 4. График Lasso регрессии  plt.subplot(1, 3, 3)  plt.plot(np.log10(alphas), lasso\_train\_errors, 'o-', label='Обучающая ошибка', linewidth=2)  plt.plot(np.log10(alphas), lasso\_test\_errors, 'o-', label='Тестовая ошибка', linewidth=2)  plt.axvline(x=np.log10(best\_lasso\_alpha), color='red', linestyle='--', alpha=0.7,  label=f'Лучшее альфа ({best\_lasso\_alpha})')  plt.xlabel('log(alpha)')  plt.ylabel('Средняя квадратичная ошибка')  plt.title('Lasso регрессия\nТочность vs Регуляризация')  plt.legend()  plt.grid(True, alpha=0.3)    plt.tight\_layout()  plt.show() |

# **3****. Тестирование**

Пример работы программы можно наблюдать на рисунках 1 и 2.



**Рис.1** Результат работы программы



**Рис.2** Результат работы программы

# **4.** **Вывод**

В данной лабораторной работе была написана программа для разделения исходной выборки на обучающую и тестовую. С использованием библиотеки scikit-learn , мы обучили модель линейной регрессии по обучающей выборке. Проверили точность модели на тестовой выборке. И построили модели с использованием полиномиальной функции и регуляризации.