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 №4**

**«Линейная регрессия»**

**Выполнил**:

студент гр. ИВТАПбд-41

Князев А.Д.

**Проверил работу:**

Святов К.В.

Ульяновск 2025

# Цель работы:

1. Написать программу, которая разделяет исходную выборку на обучающую и тестовую (training set, test set).
2. С использованием библиотеки scikit-learn обучить модель линейной регрессии по обучающей выборке.
3. Проверить точность модели по тестовой выборке.
4. Построить модель с использованием полиномиальной функции. Построить графики зависимости точности на обучающей и тестовой выборке от степени полиномиальной функции.
5. Построить модель с использованием регуляризации. На основе экспериментов подобрать параметры для регуляризации. Построить графики зависимости точности модели на обучающей и тестовой выборках от коэффициента регуляризации.

# Описание данных

Данный датасет содержит социо-экономические и демографические характеристики городских сообществ США, объединенные со статистикой преступности за 1990 год. Исходная информация позволяет исследовать взаимосвязи между социально-экономическими факторами и уровнем преступности, выявлять значимые предикторы и строить прогнозные модели.

Данный набор данных широко используется в задачах прогнозного моделирования, анализа социальных детерминант преступности и разработки политики общественной безопасности, что делает его ценным ресурсом для исследований в области криминологии и социальных наук. Особую ценность представляет возможность анализа комплексных взаимосвязей между множественными социальными факторами и различными типами преступности.

# Описание реализации

Загрузка и подготовка данных.

Для получения набора данных используется функция fetch\_ucirepo из репозитория UCI с идентификатором 183 (Communities and Crime). Функция data\_definition() выводит информацию о первых 10 переменных, включая их названия, роли и типы. Данные разделяются на признаки (features) и целевую переменную (targets). Предобработка включает замену пропущенных значений '?' на NaN, преобразование всех колонок в числовой формат, заполнение пропусков медианными значениями и удаление константных колонок.

*Листинг 1.*

|  |
| --- |
| python  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 |

Разделение данных на обучающую и тестовую выборки.

Для разделения данных используется функция splitter(), которая случайным образом перемешивает индексы наблюдений с фиксированным seed для воспроизводимости и распределяет 20% данных в тестовую выборку, а остальные 80% - в обучающую. Такой подход обеспечивает репрезентативность выборок и объективную оценку моделей.

*Листинг 2.*

|  |
| --- |
| python  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 |

Предобработка данных.

Функция preprocess\_data() выполняет комплексную очистку данных: заменяет специальные символы '?' на NaN, преобразует данные к числовому формату, заполняет пропущенные значения медианами и удаляет константные признаки. Дополнительно применяется стандартизация данных через StandardScaler для улучшения сходимости алгоритмов.

*Листинг 3.*

|  |
| --- |
| python  def preprocess\_data(X):  X\_processed = X.copy()  X\_processed = X\_processed.replace('?', np.nan)  for col in X\_processed.columns:  X\_processed[col] = pd.to\_numeric(X\_processed[col], errors='coerce')  X\_processed = X\_processed.fillna(X\_processed.median())  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 |

Обучение базовой модели линейной регрессии.

Базовая модель линейной регрессии обучается на стандартизированных данных с использованием LinearRegression из scikit-learn. Модель оценивается по метрикам MSE и R² на обучающей и тестовой выборках, что позволяет получить базовые показатели точности.

*Листинг 4.*

|  |
| --- |
| python  regressor = LinearRegression().fit(x\_train\_scaled, y\_train)  y\_pred\_train = regressor.predict(x\_train\_scaled)  y\_pred\_test = regressor.predict(x\_test\_scaled)  print(f"ОБУЧАЮЩЕЕ - 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}") |

Построение полиномиальной регрессии.

Для учета нелинейных зависимостей применяется полиномиальное преобразование признаков с помощью PolynomialFeatures для степеней от 1 до 3. Для каждой степени строится отдельная модель линейной регрессии, вычисляются MSE и R² на обеих выборках. Анализ ошибок позволяет определить оптимальную сложность модели.

*Листинг 5.*

|  |
| --- |
| python  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)) |

Применение регуляризации.

Для борьбы с переобучением используются Ridge и Lasso регрессии с различными значениями параметра alpha. Ridge-регрессия (L2-регуляризация) и Lasso (L1-регуляризация) обучаются на одних и тех же данных, после чего сравниваются их ошибки на обучающей и тестовой выборках для подбора оптимального параметра регуляризации.

*Листинг 6.*

|  |
| --- |
| python  # Ridge регрессия  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)  # Lasso регрессия  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) |

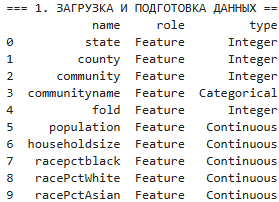
Сравнение методов и визуализация.

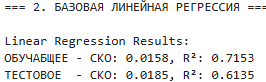
Проводится сравнение всех методов по минимальной тестовой ошибке, определяется лучшая модель. Результаты визуализируются на трех графиках, показывающих зависимость ошибки от степени полинома и параметра регуляризации, что позволяет наглядно оценить влияние сложности модели на ее обобщающую способность.

*Листинг 7.*

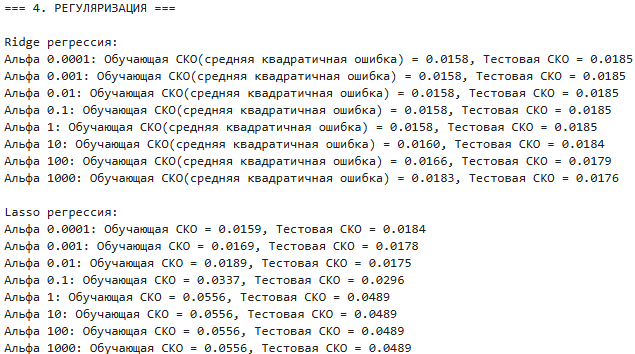
|  |
| --- |
| python  methods = {  'Linear': mean\_squared\_error(y\_test, y\_pred\_test),  'Polynomial': best\_poly\_error,  'Ridge': best\_ridge\_error,  'Lasso': best\_lasso\_error  }  best\_method = min(methods, key=methods.get)  print(f"Лучший метод - {best\_method} со средней квадратичной ошибкой = {methods[best\_method]:.4f}") |

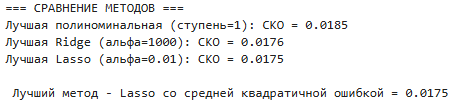
# Результат работы программы

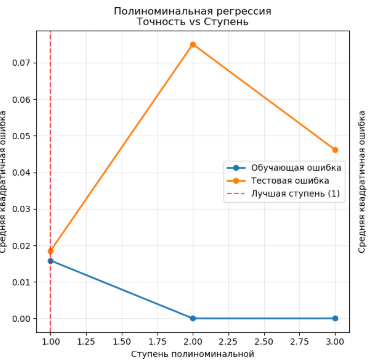


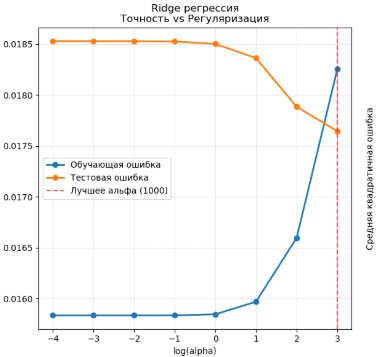


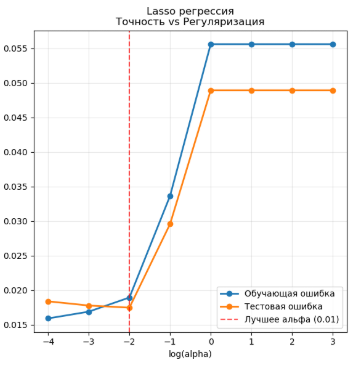












# Вывод

В данной лабораторной работе реализованы методы линейной, полиномиальной, Ridge и Lasso регрессии для прогнозирования уровня преступности в сообществах. Сравнительный анализ продемонстрировал эффективность методов регуляризации (Ridge и Lasso) в достижении баланса между точностью на обучающих данных и обобщающей способностью на тестовой выборке. Эксперименты показали, что полиномиальное преобразование признаков позволяет улучшить качество модели за счет учета нелинейных зависимостей, однако требует применения регуляризации для предотвращения переобучения.