ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		Поляк М.Д.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

построение моделей линейной регрессии

по дисциплине: Основы машинного обучения

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4134к		Н.А. Костяков
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

Цель работы

Получение навыков решения задачи регрессионного анализа и оптимизации функций методом градиентного спуска

листинг

```
### BEGIN YOUR CODE

Student_ID = 4

### END YOUR CODE
datasets = [('Diabetes dataset','https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes'), ('California Housing','https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html#fetch-california-housing'), ('Auto MPG', 'https://archive.ics.uci.edu/dataset/9/auto+mpg'), ('Forest Fires', 'https://archive.ics.uci.edu/dataset/162/forest+fires'), ('Concrete Compressive Strength', 'https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength')]

dataset_id = None if Student_ID is None else Student_ID % len(datasets)
if dataset_id is None:
  print("ОШИБКА! Не указан порядковый номер студента в списке группы.")
else:
  print(f"Информация о датасете '{datasets[dataset_id][0]}' доступна по следующей ссылке: {datasets[dataset_id][1]}")
### BEGIN YOUR CODE

import pandas as pd

### END YOUR CODE
### BEGIN YOUR CODE

#!wget  -O dataset.zip https://archive.ics.uci.edu/static/public/165/concrete+compressive+strength.zip
# The dataset was incorrect, and it should've been Concrete_Data.xls

#!unzip dataset.zip
!ls
# !tar ...
# !gunzip ...

# added sep=';' and header=None because the dataset is not comma-separated, nor has a header
dataset = pd.read_excel("Concrete_Data.xls")

### END YOUR CODE
### BEGIN YOUR CODE
!du -sh Concrete_Data.xls
print(dataset.shape)
```

```
print(f"Number of features: {dataset.shape[1]}")
print(dataset.info())
print(dataset.describe())
# Check for missing values and sum them up for each column
missing_values = dataset.isnull().sum()
print("Counting nulls",missing_values)
### END YOUR CODE
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# ... (your code to load the dataset into 'dataset') ...

# Calculate the correlation matrix
correlation_matrix = dataset.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Print the correlation with the target variable ('mpg')
# Instead of using the full column name, try accessing it using dataset.columns:
target_variable = dataset.columns[-1]  # Assuming the target variable is the last
column
print(correlation_matrix[target_variable].sort_values(ascending=False))
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
import numpy as np
# Разделение данных на признаки (X) и целевую переменную (y)
X = dataset.iloc[:, :-1]  # Все столбцы, кроме последнего
y = dataset.iloc[:, -1]   # Последний столбец - целевая переменная

X = X.drop(columns=['Blast Furnace Slag (component 2)(kg in a m^3 mixture)'])
#X = X.drop(columns=['Fly Ash (component 3)(kg in a m^3 mixture)'])
#X = X.drop(columns=['Coarse Aggregate  (component 6)(kg in a m^3 mixture)'])
#X = X.drop(columns=['Fine Aggregate (component 7)(kg in a m^3 mixture)'])
# Проверка пропущенных значений и применение заполнения (если бы были)
imputer = SimpleImputer(strategy='mean')

# Масштабирование данных
scaler = StandardScaler()

# Создание пайплайна для обработки данных
pipeline = Pipeline([
    ('imputer', imputer),  # На случай, если появятся пропущенные значения
    ('scaler', scaler)     # Нормализация данных
```

```
])

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Применение пайплайна к обучающим данным
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)

# Проверка результатов
print("Обработанные данные (первые строки):")
print(X_train[:5])
print("Среднее значение каждого признака после нормализации (должно быть ~0):")
print(np.mean(X_train, axis=0))
print("Стандартное отклонение (должно быть ~1):")
print(np.std(X_train, axis=0))
# Внимание: нельзя использовать библиотечный код для реализации линейной
регрессии, напишите свой!
# Не забудьте поменять значения self.attribute = 0 на более подходящие или (лучше)
задать их перед началом обучения

class LinearRegressionModel:
    """
    Класс для выполнения линейной регрессии
    """

    def __init__(self):
        """
        Инициализация модели
        """
        self.theta = None  # Вектор параметров модели
        self.alpha = 0.01  # Скорость обучения (по умолчанию)
        self.cost = 0      # Значение функции стоимости

    def _compute_cost(self, X, y, theta):
        """
        Вычисление значения функции стоимости (MSE)
        :param X: Матрица признаков с x0 = 1
        :param y: Вектор истинных значений
        :param theta: Вектор параметров
        :return: Значение функции стоимости
        """
        m = len(y)  # Количество обучающих примеров
        predictions = np.dot(X, theta)  # Предсказания модели
        errors = predictions - y
        cost = (1 / (2 * m)) * np.dot(errors, errors)
        return cost

    def gradient_descent(self, X_train, y_train, theta, alpha, iters):
```

```
    """
    Реализация градиентного спуска
    :param X_train: Матрица признаков обучающей выборки (с x0 = 1)
    :param y_train: Вектор истинных значений
    :param theta: Начальные параметры модели
    :param alpha: Скорость обучения
    :param iters: Количество итераций
    :return: Обновлённое значение вектора параметров и значение функции стоимости
    """
    m = len(y_train)  # Количество обучающих примеров
    for i in range(iters):
        predictions = np.dot(X_train, theta)  # Предсказания модели
        errors = predictions - y_train        # Ошибки предсказаний
        gradient = (1 / m) * np.dot(X_train.T, errors)  # Градиент функции стоимости
        theta -= alpha * gradient  # Обновление параметров

        # Вывод промежуточной стоимости каждые 100 итераций
        if i % 100 == 0:
            self.cost = self._compute_cost(X_train, y_train, theta)
            print(f"Iteration {i}, Cost: {self.cost}")

    # Финальная стоимость
    self.cost = self._compute_cost(X_train, y_train, theta)
    return theta, self.cost

  def fit_with_GD(self, X_train, y_train, iters=200):
    """
    Обучение модели методом градиентного спуска
    :param X_train: Матрица признаков
    :param y_train: Вектор истинных значений
    :param iters: Количество итераций (по умолчанию 200)
    :return: Итоговое значение функции стоимости
    """
    # Добавляем столбец единиц для x0
    X_b = np.hstack([np.ones((X_train.shape[0], 1)), X_train])

    # Инициализация параметров
    self.theta = np.zeros(X_b.shape[1])

    # Выполнение градиентного спуска
    self.theta, self.cost = self.gradient_descent(X_b, y_train, self.theta,
self.alpha, iters)
    return self.cost

  def fit_with_normal_equations(self, X_train, y_train):
    """
    Обучение с помощью нормальных уравнений (МНК)
    :param X_train: Матрица признаков
    :param y_train: Вектор истинных значений
    :return: Значение функции стоимости
```

```
    """
    # Добавляем столбец единиц для x0
    X_b = np.hstack([np.ones((X_train.shape[0], 1)), X_train])

    # Нормальные уравнения: θ = (X.T * X)^(-1) * X.T * y
    self.theta = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_train

    # Вычисление стоимости
    self.cost = self._compute_cost(X_b, y_train, self.theta)
    return self.cost

  def predict(self, X_test):
    """
    Предсказание значений на тестовых данных
    :param X_test: Матрица признаков тестовой выборки
    :return: Прогнозы модели
    """
    # Добавляем столбец единиц для x0
    X_b = np.hstack([np.ones((X_test.shape[0], 1)), X_test])
    # Вычисляем предсказания
    y_predict = np.dot(X_b, self.theta)
    return y_predict

  def fit_with_normal_equations(self, X_train, y_train):
    """
    Обучение с помощью нормальных уравнений (МНК)
    :param X_train: Матрица признаков
    :param y_train: Вектор истинных значений
    :return: Значение функции стоимости
    """
    # Добавляем столбец единиц для x0
    X_b = np.hstack([np.ones((X_train.shape[0], 1)), X_train])

    # Нормальные уравнения: θ = (X.T * X)^(-1) * X.T * y
    self.theta = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_train

    # Вычисление стоимости (MSE)
    self.cost = self._compute_cost(X_b, y_train, self.theta)
    return self.cost


  def __str__(self):
    """
    Вывод всех параметров модели при вызове функции print()
    """
    return f"Вектор параметров: {self.theta}\nЗначение функции стоимости:
{self.cost}"

```

```
my_model = LinearRegressionModel()
# поместите сюда ваш код для вызова экземпляра класса LinearRegressionModel

print(my_model)
import matplotlib.pyplot as plt
import numpy as np


# Создание модели
my_model = LinearRegressionModel()

plt.figure(figsize=(12, 8))

# Параметры для графика
alphas = [0.01, 0.03, 0.1, 0.3, 0.5]  # Скорости обучения
iters = list(range(0, 1000, 50))      # Количество итераций

# Построение графиков для каждой скорости обучения
for alpha in alphas:
    my_model.alpha = alpha
    costs = []
    for num_iters in iters:
        my_model.fit_with_GD(X_train, y_train, num_iters)
        costs.append(my_model.cost)
    plt.plot(iters, costs, label=f"α = {alpha}")

# Настройки графика
plt.xlabel('Количество итераций', fontsize=14)
plt.ylabel(r'Функция стоимости $Q(\theta)$', fontsize=14)
plt.title('Зависимость функции стоимости от количества итераций', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True)
plt.show()
# Реализуйте метод fit_with_normal_equations класса LinearRegressionModel перед
тем,
# как выполнить приведенный ниже код

my_normal_equations_model = LinearRegressionModel()
cost = my_normal_equations_model.fit_with_normal_equations(X_train, y_train)

print(cost)
print(my_normal_equations_model)

my_GD_model = LinearRegressionModel()
cost = my_normal_equations_model.fit_with_GD(X_train, y_train)
print(cost)
print(my_normal_equations_model)
```

```
from sklearn.metrics import mean_squared_error, r2_score

# Использование встроенной реализации метода стохастического градиентного спуска
для построения модели
from sklearn.linear_model import SGDRegressor
my_sgd_model = SGDRegressor()
my_sgd_model.fit(X_train, y_train)
y_predict = my_sgd_model.predict(X_test)
mse = mean_squared_error(y_test, y_predict)
my_sgd_model_rmse = np.sqrt(mse)
my_gd_r2 = r2_score(y_test, y_predict)
print("SGD:", my_sgd_model_rmse)
print("R2 :", my_gd_r2)

# поместите сюда ваш код
my_Lin = LinearRegressionModel()
my_Lin.fit_with_normal_equations(X_train, y_train)
y_predict = my_Lin.predict(X_test)
mse = mean_squared_error(y_test, y_predict)
my_Lin_rmse = np.sqrt(mse)
my_Lin_r2 = r2_score(y_test, y_predict)
print("Lin:", my_Lin_rmse)
print("R2 :", my_Lin_r2)

my_Lin_GD = LinearRegressionModel()
my_Lin_GD.fit_with_GD(X_train, y_train)
y_predict = my_Lin_GD.predict(X_test)
mse = mean_squared_error(y_test, y_predict)
my_Lin_GD_rmse = np.sqrt(mse)
my_Lin_GD_r2 = r2_score(y_test, y_predict)
print("Lin_GD:", my_Lin_GD_rmse)
print("R2 :", my_Lin_GD_r2)
```

Вывод

Я получил навыки решения задачи регрессионного анализа и оптимизации функций методом градиентного спуска