Описание алгоритма линейной регрессии

Линейная регрессия - это статистический метод моделирования зависимости между одной (или несколькими) независимыми переменными (признаками) и зависимой переменной (целевым значением) с помощью линейной функции.

Основные принципы:

1. Предполагается, что зависимость между признаками и целевой переменной линейна
2. Цель - найти коэффициенты (веса), которые минимизируют ошибку предсказания
3. Чаще всего используется метод наименьших квадратов (MSE - Mean Squared Error) для нахождения оптимальных коэффициентов

Реализация на Python

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import scipy as sp

def act(x):

    return 0 if x < 0.5 else 1

def go(house, rock, attr):

    x = np.array([house, rock, attr])

    w11 = [0.3, 0.3, 0]

    w12 = [0.4, -0.5, 1]

    weight1 = np.array([w11, w12])  # матрица 2x3

    weight2 = np.array([-1, 1])     # вектор 1x2

    sum\_hidden = np.dot(weight1, x)  # вычисляем сумму на входах нейронов скрытого слоя

    print("Значения сумм на нейронах скрытого слоя: " + str(sum\_hidden))

    out\_hidden = np.array([act(x) for x in sum\_hidden])

    print("Значения на выходах нейронов скрытого слоя: " + str(out\_hidden))

    sum\_end = np.dot(weight2, out\_hidden)

    y = act(sum\_end)

    print("Выходное значение НС: " + str(y))

    return y

house = 1

rock = 0

attr = 1

res = go(house, rock, attr)

if res == 1:

    print("Ты мне нравишься")

else:

    print("Созвонимся")

Значения сумм на нейронах скрытого слоя: [0.3 1.4]

Значения на выходах нейронов скрытого слоя: [0 1]

Выходное значение НС: 1

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

from tensorflow.keras.optimizers import Adam

# 1. Генерация синтетических данных

np.random.seed(42)

time = np.arange(0, 1000, 0.1)

sine\_wave = np.sin(time \* 0.1)  # Синусоида

noise = np.random.normal(0, 0.1, len(time))  # Шум

data = (sine\_wave + noise).reshape(-1, 1)  # Форма (1000, 1)

# Визуализация данных

plt.figure(figsize=(12, 4))

plt.plot(data[:200])

plt.title("Первые 200 точек сгенерированного временного ряда")

plt.show()

# 2. Предварительная обработка данных

scaler = MinMaxScaler(feature\_range=(0, 1))

data\_scaled = scaler.fit\_transform(data)

# Функция для создания последовательностей

def prepare\_data(series, n\_steps):

    X, y = [], []

    for i in range(len(series) - n\_steps):

        X.append(series[i:i + n\_steps])

        y.append(series[i + n\_steps])

    return np.array(X), np.array(y)

n\_steps = 20

n\_features = 1

X, y = prepare\_data(data\_scaled, n\_steps)

# Разделение на train/test

train\_size = int(0.8 \* len(X))

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# 3. Построение модели RNN

model = Sequential([

    SimpleRNN(50, activation='tanh', input\_shape=(n\_steps, n\_features)),

    Dense(1)

])

model.compile(optimizer=Adam(learning\_rate=0.001),

              loss='mse',

              metrics=['mae'])

history = model.fit(X\_train, y\_train,

                    epochs=50,

                    validation\_data=(X\_test, y\_test),

                    verbose=1)

# 4. Прогнозирование и оценка

# Предсказание на тестовых данных

y\_pred = model.predict(X\_test)

# Обратное масштабирование

y\_test\_inv = scaler.inverse\_transform(y\_test)

y\_pred\_inv = scaler.inverse\_transform(y\_pred)

# Вычисление MSE

mse = mean\_squared\_error(y\_test\_inv, y\_pred\_inv)

print(f"Test MSE: {mse:.4f}")

# Визуализация кривой обучения

plt.figure(figsize=(12, 4))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Кривая обучения')

plt.ylabel('MSE')

plt.xlabel('Epoch')

plt.legend()

plt.show()

# Визуализация предсказаний

plt.figure(figsize=(12, 4))

plt.plot(y\_test\_inv[:100], 'b', label='Истинные значения')

plt.plot(y\_pred\_inv[:100], 'r--', label='Предсказания')

plt.title('Сравнение предсказаний и истинных значений (первые 100 точек)')

plt.legend()

plt.show()

# Улучшенная модель с LSTM и Dropout

from tensorflow.keras.layers import LSTM, Dropout

model = Sequential([

    LSTM(100, activation='tanh', return\_sequences=True, input\_shape=(n\_steps, n\_features)),

    Dropout(0.2),

    LSTM(50, activation='tanh'),

    Dropout(0.2),

    Dense(1)

])

# Простой трансформер для временных рядов

from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization

def transformer\_encoder(inputs, head\_size, num\_heads, ff\_dim, dropout=0):

    # Attention and Normalization

    x = MultiHeadAttention(key\_dim=head\_size, num\_heads=num\_heads, dropout=dropout)(inputs, inputs)

    x = LayerNormalization(epsilon=1e-6)(x + inputs)

    # Feed Forward Part

    y = Dense(ff\_dim, activation="relu")(x)

    y = Dense(inputs.shape[-1])(y)

    return LayerNormalization(epsilon=1e-6)(x + y)

inputs = tf.keras.Input(shape=(n\_steps, n\_features))

x = transformer\_encoder(inputs, head\_size=32, num\_heads=4, ff\_dim=64)

x = tf.keras.layers.GlobalAveragePooling1D()(x)

outputs = Dense(1)(x)

transformer\_model = tf.keras.Model(inputs, outputs)

# Добавление callback'ов

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

callbacks = [

    EarlyStopping(patience=10, restore\_best\_weights=True),

    ReduceLROnPlateau(factor=0.1, patience=5)

]

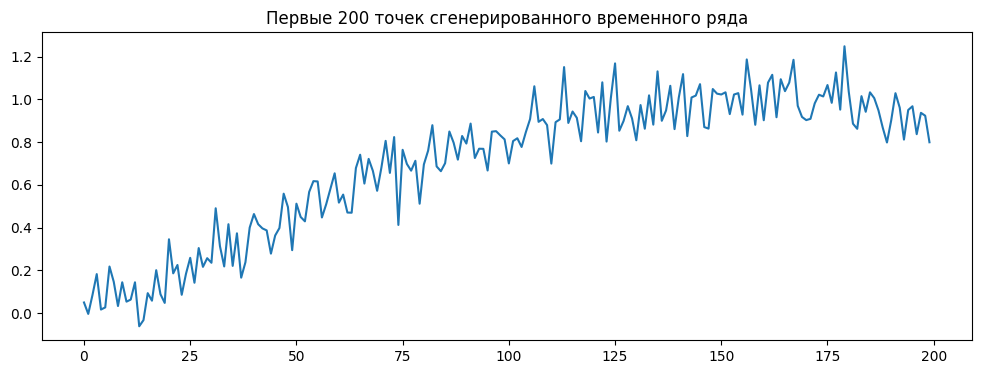
model.fit(X\_train, y\_train,

          epochs=100,

          validation\_data=(X\_test, y\_test),

          callbacks=callbacks,

          batch\_size=32)



/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(\*\*kwargs)

Epoch 1/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **4s** 8ms/step - loss: 0.0395 - mae: 0.1083 - val\_loss: 0.0020 - val\_mae: 0.0357

Epoch 2/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0021 - mae: 0.0366 - val\_loss: 0.0020 - val\_mae: 0.0353

Epoch 3/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **4s** 11ms/step - loss: 0.0020 - mae: 0.0359 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 4/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **4s** 6ms/step - loss: 0.0019 - mae: 0.0350 - val\_loss: 0.0018 - val\_mae: 0.0337

Epoch 5/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 8ms/step - loss: 0.0019 - mae: 0.0350 - val\_loss: 0.0018 - val\_mae: 0.0336

Epoch 6/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0019 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 7/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 10ms/step - loss: 0.0019 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 8/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0019 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0342

Epoch 9/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0019 - mae: 0.0351 - val\_loss: 0.0020 - val\_mae: 0.0356

Epoch 10/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0019 - mae: 0.0349 - val\_loss: 0.0018 - val\_mae: 0.0337

Epoch 11/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0350 - val\_loss: 0.0018 - val\_mae: 0.0338

Epoch 12/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 13/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 9ms/step - loss: 0.0019 - mae: 0.0351 - val\_loss: 0.0021 - val\_mae: 0.0367

Epoch 14/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0019 - mae: 0.0353 - val\_loss: 0.0020 - val\_mae: 0.0353

Epoch 15/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0019 - val\_mae: 0.0347

Epoch 16/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0019 - mae: 0.0351 - val\_loss: 0.0018 - val\_mae: 0.0342

Epoch 17/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0348 - val\_loss: 0.0019 - val\_mae: 0.0346

Epoch 18/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0020 - mae: 0.0353 - val\_loss: 0.0018 - val\_mae: 0.0337

Epoch 19/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 7ms/step - loss: 0.0019 - mae: 0.0351 - val\_loss: 0.0018 - val\_mae: 0.0336

Epoch 20/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 5ms/step - loss: 0.0018 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0334

Epoch 21/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0346 - val\_loss: 0.0018 - val\_mae: 0.0335

Epoch 22/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0348 - val\_loss: 0.0018 - val\_mae: 0.0335

Epoch 23/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0018 - mae: 0.0344 - val\_loss: 0.0019 - val\_mae: 0.0351

Epoch 24/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 10ms/step - loss: 0.0020 - mae: 0.0354 - val\_loss: 0.0019 - val\_mae: 0.0345

Epoch 25/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0019 - mae: 0.0349 - val\_loss: 0.0018 - val\_mae: 0.0334

Epoch 26/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0020 - val\_mae: 0.0356

Epoch 27/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0019 - mae: 0.0350 - val\_loss: 0.0019 - val\_mae: 0.0346

Epoch 28/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0345 - val\_loss: 0.0018 - val\_mae: 0.0335

Epoch 29/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0020 - mae: 0.0354 - val\_loss: 0.0018 - val\_mae: 0.0334

Epoch 30/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 8ms/step - loss: 0.0019 - mae: 0.0343 - val\_loss: 0.0018 - val\_mae: 0.0340

Epoch 31/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0018 - mae: 0.0343 - val\_loss: 0.0018 - val\_mae: 0.0336

Epoch 32/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - loss: 0.0018 - mae: 0.0338 - val\_loss: 0.0018 - val\_mae: 0.0341

Epoch 33/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0018 - mae: 0.0340 - val\_loss: 0.0018 - val\_mae: 0.0340

Epoch 34/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0018 - mae: 0.0340 - val\_loss: 0.0023 - val\_mae: 0.0385

Epoch 35/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0018 - val\_mae: 0.0340

Epoch 36/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 7ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0018 - val\_mae: 0.0342

Epoch 37/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0018 - mae: 0.0343 - val\_loss: 0.0018 - val\_mae: 0.0341

Epoch 38/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - loss: 0.0018 - mae: 0.0341 - val\_loss: 0.0018 - val\_mae: 0.0343

Epoch 39/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0349 - val\_loss: 0.0017 - val\_mae: 0.0334

Epoch 40/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0018 - mae: 0.0343 - val\_loss: 0.0017 - val\_mae: 0.0334

Epoch 41/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 9ms/step - loss: 0.0019 - mae: 0.0349 - val\_loss: 0.0018 - val\_mae: 0.0338

Epoch 42/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0018 - mae: 0.0339 - val\_loss: 0.0018 - val\_mae: 0.0338

Epoch 43/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 44/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 5ms/step - loss: 0.0018 - mae: 0.0341 - val\_loss: 0.0019 - val\_mae: 0.0347

Epoch 45/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0019 - mae: 0.0349 - val\_loss: 0.0018 - val\_mae: 0.0339

Epoch 46/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0343 - val\_loss: 0.0017 - val\_mae: 0.0332

Epoch 47/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 9ms/step - loss: 0.0018 - mae: 0.0343 - val\_loss: 0.0018 - val\_mae: 0.0335

Epoch 48/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0017 - val\_mae: 0.0332

Epoch 49/50

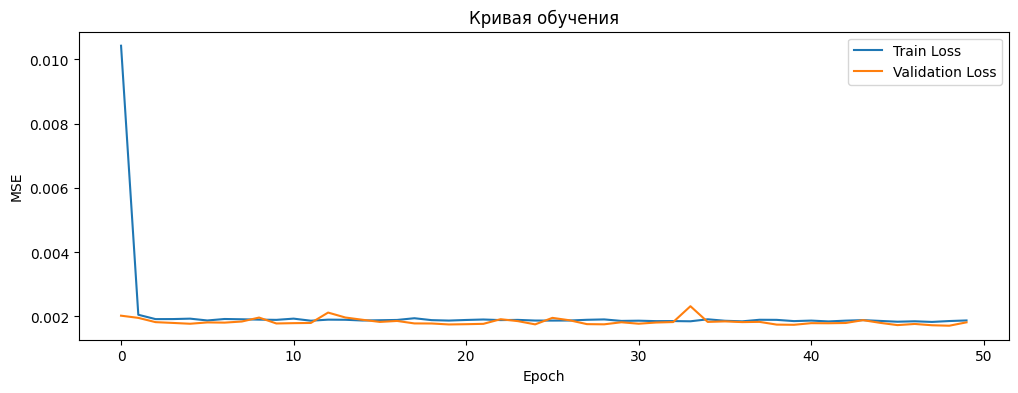
**250/250** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.0018 - mae: 0.0341 - val\_loss: 0.0017 - val\_mae: 0.0330

Epoch 50/50

**250/250** ━━━━━━━━━━━━━━━━━━━━ **2s** 7ms/step - loss: 0.0019 - mae: 0.0347 - val\_loss: 0.0018 - val\_mae: 0.0342

**63/63** ━━━━━━━━━━━━━━━━━━━━ **0s** 4ms/step

Test MSE: 0.0123





/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(\*\*kwargs)

---------------------------------------------------------------------------

import matplotlib.pylab as plt

import numpy as np

%matplotlib inline

from sklearn.linear\_model import LinearRegression

from sklearn import datasets

diabetes = datasets.load\_diabetes()

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(diabetes.data, diabetes.target, test\_size=0.2, random\_state=0)

# There are three steps to model something with sklearn

# 1. Set up the model

model = LinearRegression()

# 2. Use fit

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

plt.plot(y\_test, y\_pred, '.')

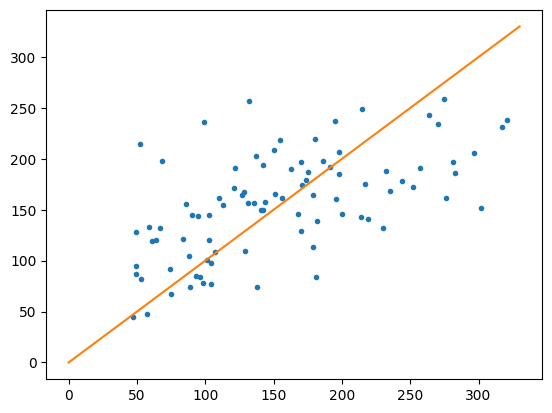
# plot a line, a perfit predict would all fall on this line

x = np.linspace(0, 330, 100)

y = x

plt.plot(x, y)

plt.show()



Пошаговое объяснение кода:

1. **Импорт библиотек**:
   * matplotlib.pylab для визуализации
   * numpy для работы с массивами
   * LinearRegression из scikit-learn для модели регрессии
   * datasets для загрузки стандартных наборов данных
   * train\_test\_split для разделения данных
2. **Загрузка данных**:
   * Используется встроенный датасет diabetes о прогрессировании диабета
3. **Разделение данных**:
   * Данные делятся на обучающую (80%) и тестовую (20%) выборки
   * random\_state=0 обеспечивает воспроизводимость результатов
4. **Создание и обучение модели**:
   * Создается экземпляр модели линейной регрессии
   * Модель обучается на тренировочных данных методом fit()
5. **Предсказание и оценка**:
   * Модель делает предсказания для тестовых данных
   * Результаты визуализируются: точки - реальные vs предсказанные значения, линия - идеальное соответствие