Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 **Прикладная информатика**

О Т Ч Е Т

по лабораторной работе № 1

Название: Прогнозирование моделью линейной регрессии

Дисциплина: Прикладной анализ данных

Студент ИУ6-55Б _____ Т.А. Китаев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ М.А. Кулаев
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

1 Цель работы

Исследовать аналитический и градиентный способы расчета коэффициентов модели линейной регрессии, изучить библиотеки для градиентного способа, изучить способы оценки качества построенных моделей.

2 Ход работы

1. Загрузка данных

Перед нормированием и расчетом весов необходимо загрузить данные с помощью библиотеки numpy (у нас будут X – матрица признаков, y – матрица целевых значений). В листинге 1 представлен код, с помощью которого читаются данные из файлов, а на листинге 2 представлена часть матрицы X.

```
import numpy as np

from sklearn.metrics import r2_score

# получение данных из файлов
X = np.genfromtxt("./datasets/lab1_X.csv")
y = np.genfromtxt("./datasets/lab1_y.csv").reshape(X.shape[0],
1)

print(f"Матрицы X и y:\n{X}\n{y}")
```

Листинг 1 – Чтение данных

```
      Матрицы X и y:
[[7.000e+00 1.590e+01 8.200e+00 5.100e+00 1.380e+01 2.290e+02 1.
720e+02
  2.000e+01 2.104e+03]
 [7.200e+00 1.820e+01 7.400e+00 6.100e+00 1.430e+01 1.460e+02 1.
670e+02
  2.910e+01 2.489e+03]
 [7.900e+00 1.970e+01 6.400e+00 4.700e+00 1.980e+01 1.740e+02 1.
440e+02
  2.280e+01 2.428e+03]
 [7.700e+00 2.080e+01 6.900e+00 5.200e+00 1.710e+01 1.280e+02 1.
110e+02
  4.270e+01 2.494e+03]
 [9.200e+00 1.590e+01 7.800e+00 5.300e+00 1.670e+01 1.690e+02 1.
480e+02
  2.270e+01 2.094e+03]
 [7.600e+00 1.640e+01 6.700e+00 4.700e+00 1.550e+01 1.440e+02 1.
500e+02
```

```
  2.790e+01 1.768e+03]
 [7.300e+00 1.830e+01 6.300e+00 4.900e+00 1.960e+01 1.380e+02 1.
330e+02
  3.370e+01 1.982e+03]
      ...
```

Листинг 2 – Загруженные данные

Теперь разделим все данные на выборку для обучения и выборку для валидации (X_train, y_train, X_test и y_test соотвественно). Данные будут делиться в соотношении 70% на 30%. В листинге 3 представлен код, отвечающий за разделение данные.

```
# разделяем данные так: 70% для обучения, 30% для валидации
split = round(X.shape[0] * 0.7)

# Разделение для матрицы признаков
X_train = X[:split]
X_test = X[split:]

# Разделение для матрицы целевых значений
y_train = y[:split]
y_test = y[split:]
```

Листинг 3 – Разделение данных

Имеем следующие размерности матриц: X_test – 15x9, X_train – 35x9, y_test – 15x1, y_train – 35x1.

2. Нормирование (масштабирование) исходных данных.

Теперь необходимо нормализовать данные, поскольку разброс значений данных слишком большой. Будет использоваться Z-нормализация, ее формула:

$$X_{new} = \frac{X_{old} - mean(X_{old})}{std(X_{old})}$$

Код для нормализации представлен в листинге 4, а часть результата нормализации – в листинге 5.

```
# функции для нахождения среднего и стандартного распределения
X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train,
axis=0)
X_test = (X_test - np.mean(X_test, axis=0)) / np.std(X_test,
axis=0)

print(X_train)
```

Листинг 4 – Нормализация данных

```
[[-0.86580164  0.12296253  1.55757652  0.91545987 -1.66258049  0
.94244347
   0.9941688  -1.24316968  0.76671086]
 [-0.79287009  0.94101901  0.35944073  2.02032522 -1.43859928 -0
.27652282
   0.80669085 -0.26936362  1.51428169]
 [-0.53760968  1.4745341  -1.13822899  0.47351372  1.02519397  0
.13469473
  -0.05570773 -0.94353704  1.39583541]
 [-0.61054123  1.86577851 -0.38939413  1.0259464  -0.18430453 -0
.54087695
  -1.29306223  1.18599488  1.52399041]
 [-0.06355463  0.12296253  0.95850863  1.13643294 -0.3634895   0
.06126302
   0.09427463 -0.95423821  0.74729344]
 [-0.647007    0.30080089 -0.68892807  0.47351372 -0.90104439 -0
.3058955
   0.16926581 -0.39777761  0.11428541]
 [-0.75640432  0.97658668 -1.28799597  0.69448679  0.93560149 -0
.39401355
  -0.46815923  0.22288999  0.52981829]
 [-0.53760968  0.30080089 -0.5391611   0.80497333  0.03967667  0
.47248057
   0.35674376 -0.53689276 -0.17115072]
      ...
```

Листинг 5 – Результат нормализации

После нормализации к данным были добавлены единичные столбцы (параметр w_0). Код добавления столбца представлен в листинге 6.

```
X_train = np.append(X_train,
np.ones(X_train.shape[0]).reshape((X_train.shape[0], 1)),
axis=1)
X_test = np.append(X_test,
np.ones(X_test.shape[0]).reshape((X_test.shape[0], 1)), axis=1)

print(X_train)
```

Листинг 6 – Добавление единичного столбца

3. Расчет весов линейной регрессии по аналитической формуле.

После нормализации данных можно рассчитать веса с помощью аналитической формуле (здесь также будут использоваться функции numpy, matmul для произведения матриц, transpose для транспонирования матрицы, linalg.inv для нахождения обратной матрицы). Аналитическая формула:

$$w = (X^T X)^{-1} X^T Y$$

В листинге 7 представлен код, отвечающий за нахождение весов линейной регрессии по аналитической формуле, а в листинге 8 представлены сами весы.

```
cov_mat = np.matmul(np.transpose(X_train), X_train) # получение ковариационной матрицы
temp = np.matmul(np.linalg.inv(cov_mat), np.transpose(X_train)) # произведение обратной ковариационной матрицы и X транспонированной
w = np.matmul(temp, y_train) # получение матрицы весов

print(f"Полученные веса:\n{w}")
```

Листинг 7 – Нахождение весов

```
Полученные веса:
[[-1.56018400e-01]
 [-2.23529642e+00]
 [ 6.68635399e-01]
 [-8.35658594e-03]
 [ 4.98044131e-01]
 [ 3.28450435e-01]
 [-6.46361963e-01]
 [ 1.75903141e-01]
 [ 7.07629979e-01]
 [ 5.92542857e+01]]
```

Листинг 8 – Полученные веса

Можно увидеть, что наименее значимым является параметр w_4, а наиболее значимым - w_2. Свободный член равен примерно 59.

4. Построение и интерпретация корреляционной матрицы. Определение степени мультиколлинеарности на основе числа обусловленности.

Теперь определим степень мультиколлинеарности на основе числа

обусловленности, также построим матрицу корреляционную матрицу с помощью библиотеки pandas. В листинге 9 представлен код для вывода корреляционной матрицы и нахождения числа обусловленности. В листинге 10 представлен вид корреляционной матрицы и вычисленное число обусловленности.

```
import pandas as pd

print(f"Кореляционная матрица:\n{pd.DataFrame(X_train).corr()}")
print(f"Число обусловленности: {np.linalg.cond(X_train, p =
2)}")
```

Листинг 9 – Нахождение корреляционной матрицы и числа обусловленности

```
Кореляционная матрица:
          0         1         2         3         4         5         6  \
0  1.000000 -0.854247 -0.011641 -0.808624  0.066136 -0.327963 -0.629259
1 -0.854247  1.000000 -0.056221  0.791387  0.026760  0.294823  0.540912
2 -0.011641 -0.056221  1.000000  0.333781 -0.043589  0.297351  0.290214
3 -0.808624  0.791387  0.333781  1.000000 -0.080584  0.287546  0.588136
4  0.066136  0.026760 -0.043589 -0.080584  1.000000 -0.175363  0.016271
5 -0.327963  0.294823  0.297351  0.287546 -0.175363  1.000000  0.601711
6 -0.629259  0.540912  0.290214  0.588136  0.016271  0.601711  1.000000
7  0.517301 -0.543087 -0.015963 -0.434231  0.301653 -0.565339 -0.659190
8 -0.610383  0.601123 -0.302931  0.509551  0.076973 -0.076500  0.203852
9       NaN       NaN       NaN       NaN       NaN       NaN       NaN

          7         8   9
0  0.517301 -0.610383 NaN
1 -0.543087  0.601123 NaN
2 -0.015963 -0.302931 NaN
3 -0.434231  0.509551 NaN
4  0.301653  0.076973 NaN
5 -0.565339 -0.076500 NaN
6 -0.659190  0.203852 NaN
7  1.000000 -0.234360 NaN
8 -0.234360  1.000000 NaN
9       NaN       NaN NaN
Число обусловленности: 6.790224663844371
```

Листинг 10 – Корреляционная матрица и число обусловленности

Из корреляционной матрицы можно заметить сильную отрицательную корреляцию (-0,85) между «Рождаемостью на 1000 человек» и «Смертностью на 1000 человек», что логично, так как, чем больше рождаемость, тем меньше смертность. Сильную отрицательную корреляцию (-0,80) можно увидеть между «Рождаемостью на 1000 человек» и «Числом разводов на 1000 человек», потому что, чем больше разводов, тем меньше рождаемость. А

сильную положительную корреляцию (0,79) можно заметить между «Смертностью на 1000 человек» и «Числом разводов на 1000 человек» (развод вызывает серьезные стресс и потрясение в жизни, что может увеличивать смертность).

Число обусловленности равно 6,79, что говорит об отсутствии мультиколлинеарности (меньше 10).

5. Анализ регрессионных остатков.

Найдем среднеквадратичную ошибку (MSE) и коэффициент детерминации (R^2) для матрицы X_train, будем искать регрессионные остатки относительно матрицы y_train. В листинге 11 представлен код, который находит эти значения.

```
# оценочные значения целевого набора обучающих данных
y_pred_train = np.matmul(X_train, w)

# MSE для обучающий данных
Eps_train = y_pred_train - y_train
MSE_train = np.matmul(np.transpose(Eps_train), Eps_train) /
X_train.shape[0]

# коэффициент детерминации для обучающих данных
r2_train = r2_score(y_train, y_pred_train)

print(f"MSE: {MSE_train[0][0]}\nR2: {r2_train}")
```

Листинг 11 – Анализ регрессионных остатков для X_train и y_train

Полученные значения – MSE: 2,446, R2: 0,676. Среднеквадратичная ошибка довольно низкая, а значение коэффициента детерминации говорит о том, что полученная модель достаточно неплохая (ближе к 1, нежели к 0).

Теперь найдем MSE для X_test и y_test. В листинге 12 представлен код, отвечающий за нахождение среднеквадратичной ошибки.

```
# оценочные значения целевого набора данных для валидации
y_pred_test = np.matmul(X_test, w)
```

```
# MSE для валидационных данных
Eps_test = y_pred_test - y_test
MSE_test = np.matmul(np.transpose(Eps_test), Eps_test) /
X_test.shape[0]

print(f"MSE: {MSE_test[0][0]}")
```

Листинг 12 – Нахождение MSE

Полученное значение – MSE: 20,389. Эта ошибка больше, чем для обучающих данных, поскольку данные для валидации никак не участвовали в алгоритме расчета весов. При этом само количество данных для обучения было не особо большим (а именно 35 наборов признаков).

6. Определение весов линейной регрессии градиентным методом. Проанализировать изменение ошибки от итерации к итерации.

Сначала необходимо инициализировать весы (каждый параметр будет равен единице). В листинге 13 представлена данная инициализация.

```
# 1. Инициализация весов
w = np.ones((X_train.shape[1], 1))

print(w)
```

Листинг 13 – Инициализация весов

Теперь на протяжении 500 итераций будем рассчитывать оценочные значения по весам, градиент функции потерь и новые значения весов. В листинге 14 представлен код, реализующий алгоритм градиентного метода. А в листинге 15 представлены первые 10 итераций алгоритма.

```
learning_rate = 0.1
S = []

for i in range(500):
    # 2. Расчет предсказанного значения y_pred по весам w
    y_pred = np.matmul(X_train, w)
    Eps_train = y_train - y_pred

    # 3. Расчет ошибки и градиента функции потерь
```

```
    S.append((1 / X_train.shape[0]) *
np.matmul(np.transpose(Eps_train), Eps_train)[0][0])

    # 4. Установка новых значений весов
    dS = (-2 / X_train.shape[0]) *
np.transpose(np.matmul(np.transpose(Eps_train), X_train))
    w -= learning_rate * dS

[print(f"Эпоха {_ + 1}: {S[_]}") for _ in range(len(S))]
```

Листинг 14 – Градиентный метод

```
Эпоха 1: 3414.5719234274197
Эпоха 2: 2177.29186039666
Эпоха 3: 1394.4373505344688
Эпоха 4: 893.6640155037337
Эпоха 5: 573.1673431870605
Эпоха 6: 368.0374233041753
Эпоха 7: 236.74062770154924
Эпоха 8: 152.6966339480131
Эпоха 9: 98.89458277709178
Эпоха 10: 64.44780696269203
```

Листинг 15 – Первые 10 итераций алгоритма

Можно заметить, что с каждой итерацией MSE становится меньше, однако, чем ближе локальный минимум, тем меньше уменьшение этой ошибки. Это видно на итерациях, начиная с 50, где ошибка равна 2,54. А ошибка на 500 итерации равна 2,44.

Теперь найдем MSE и R2 для обучающих данных. В листинге 16 представлен код, для нахождения этих значений.

```
y_pred_train = np.matmul(X_train, w)

Eps_train = y_train - y_pred_train
MSE_train = (1 / X_train.shape[0]) *
np.matmul(np.transpose(Eps_train), Eps_train)

r2_train = r2_score(y_train, y_pred_train)

print(f"MSE: {MSE_train[0][0]}\nR2: {r2_train}")
```

Листинг 16 – Нахождение MSE и R2

Полученные значения: MSE: 2,446, R2: 0,676. Эти значения практически совпадают с теми, которые получились после использования аналитического

метода.

Теперь найдем MSE для валидационных данных. В листинге 17 представлен код для расчета.

```
Eps_test = y_test - np.matmul(X_test, w) # регресионный остатки
MSE_test = (1 / X_test.shape[0]) *
np.matmul(np.transpose(Eps_test), Eps_test) # среднеквадратичная
ошибка

print(f"MSE: {MSE_test[0][0]}")
```

Листинг 17 – Нахождение MSE

Можно заметить, что полученная MSE (20,389) практически совпадает с той, которая была подсчитана после использования аналитического метода.

7. Сравнение результатов по аналитическому и градиентному методу.

Если сравнивать ошибки на валидационных данных и данных для обучения, то можно увидеть, что коэффициент детерминации и среднеквадратичные ошибки практически совпадают.

8. С помощью библиотеки sklearn сделать fit-predict модели линейной регрессии. Сравнить результаты с ранее полученными.

Теперь создадим модель через scikit-learnю Для этого воспользуемся одноименной библиотекой и функцией LinearRegression. Код, в котором происходит fit-predict, показан в листинге 18.

```
from sklearn.linear_model import LinearRegression

# инициализация модели
model = LinearRegression()

# обучение модели
model.fit(X_train, y_train)

# получение оценочных значений
y_pred = model.predict(X_test)

# регрессионные остатки и MSE
Eps_test = y_test - y_pred
```

```
MSE_test = (1 / X_test.shape[0]) *
np.matmul(np.transpose(Eps_test), Eps_test)

print(f"MSE: {MSE_test[0][0]}")
```

Листинг 18 – Fit-predict модели

MSE для валидационных данных составила 20,389, что практически совпадает с MSE, посчитанной после градиентного и аналитического методов.

Теперь найдем MSE и R2 для обучающих данных. Код, который делает это, представлен в листинге 19.

```
y_pred_train = model.predict(X_train)

Eps_train = y_train - y_pred_train
MSE_train = (1 / X_train.shape[0]) *
np.matmul(np.transpose(Eps_train), Eps_train)

r2_train = r2_score(y_train, y_pred_train)

print(f"MSE: {MSE_train[0][0]}\nR2: {r2_train}")
```

Листинг 19 – Вычисление MSE и R2

Можно заметить, что MSE (2,446) и R2 (0,676) практически совпадает с ранее подсчитанными (после аналитического и градиентного методов).

9. С помощью библиотеки statmodels получить «эконометрический» результат обучения модели линейной регрессии. Проинтерпретировать все его составляющие (в т.ч. те, которые изучались только теоретически), сравнить с предыдущими результатами.

Здесь используем библиотеку statsmodel, а именно statsmodel.api. Для обучения будет использоваться метод fit, а для получения отчета - summary. Модель для обучения - OLS, Ordinary Linear Regression (Обыкновенная линейная регрессия), которая использовалась в предыдущих заданиях. В листинге 20 представлен код, который создает модель и отчет. В листинге 21 представлен сам отчет.

```
import statsmodels.api as sm

# обучение модели
results = sm.OLS(y_train, X_train).fit()

# получение отчета
print(results.summary())
```

Листинг 20 – Использование statsmodel

```
OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.677
Model:                            OLS   Adj. R-squared:                  0.560
Method:                 Least Squares   F-statistic:                     5.813
Date:                Wed, 04 Oct 2023   Prob (F-statistic):           0.000226
Time:                        20:56:13   Log-Likelihood:                -65.319
No. Observations:                  35   AIC:                             150.6
Df Residuals:                      25   BIC:                             166.2
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.1560      0.760     -0.205      0.839      -1.722       1.410
x2            -2.2353      0.764     -2.926      0.007      -3.809      -0.662
x3             0.6686      0.488      1.370      0.183      -0.337       1.674
x4            -0.0084      0.791     -0.011      0.992      -1.637       1.620
x5             0.4980      0.379      1.315      0.200      -0.282       1.278
x6             0.3285      0.443      0.741      0.465      -0.584       1.241
x7            -0.6464      0.583     -1.109      0.278      -1.846       0.554
x8             0.1759      0.531      0.331      0.743      -0.918       1.270
x9             0.7076      0.475      1.489      0.149      -0.271       1.687
const         59.2543      0.313    189.421      0.000      58.610      59.899
==============================================================================
Omnibus:                       15.657   Durbin-Watson:                   2.621
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               25.905
Skew:                           1.038   Prob(JB):                     2.37e-06
Kurtosis:                       6.668   Cond. No.                         6.79
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
```

Листинг 21 – Отчет от statsmodel

Из этого отчета можно увидеть, что коэффициент детерминации (R-squared) совпадает с тем, который был подсчитан для предыдущих моделей. А также, коэффициенты w0..w9 примерно совпадают с теми, которые были подсчитаны аналитическим способом. Также число обусловленности (Cond. No.) совпадает с подсчитанным ранее.

Рассмотрим данные, которые представлены в отчете.

No. Observations – количество наборов признаков (в обучающих данных их 35);

Df Residuals – степень свободы (количество наборов признаков - количество параметров - 1 = 25);

Df Model – количество переменных параметров модели (в нашем случае 9);

Covariance Type – Тип ковариации (по умолчанию);

R-squared – коэффициент детерминации (в нашем случае он совпадает с подсчитанным ранее);

Adj. R-squared – коэффициент детерминации с штрафом за корреляцию (из-за того, что между некоторыми значениями сильная корреляция, Adj. R-squared меньше, чем R-squared);

F-statistic – Критерий Фишера;

Prob (F-statistics) – вероятность равенства коэффициентов при всех переменных нулю (в нашем случае стремится к нулю);

Log-likelihood – функция правдоподобия;

AIC/BIC – Информационный критерий Акаике и Информационный критерий Бейеса для выбора модели среди конечного набора моделей;

Omnibus – проверка равенства нулю сразу нескольких параметров;

Prob (Omnibus) – статический тест, измеряющий вероятность нормального распределения остатков (если 1, то идеально нормально распределены);

skew – мера асимметрии распределения вероятностей действительной случайной величины (если стремится к нулю, то симметричны, у нас нет);

Kurtosis – острота распределения остатков (коэффициент эксцесса);

Durbin-Watson – критерий проверки наличия автокорреляции (если автокорреляция отсутствует, то критерий равен 2, если есть и положительная, то стремится к 0, если есть и отрицательная, то стремится к 4), в нашем случае равна 2,621, что говорит о наличии отрицательной корреляции;

Jarque-Bera – статический тест, проверяющий ошибки наблюдений на нормальность посредством сверки асимметрии (skew) и эксцесса (Kurtosis) и показывающий насколько распределение ошибок близко к нормальному;

Cond. No. – число обусловленности (описано ранее), примерно совпадает с подсчитанным ранее.

9. Сравнить качество получаемых моделей на основе коэффициента детерминации и MSE.

Коэффициент детерминации R2 примерно одинаковый для всех моделей и равен 0.677, что близко к единице и говорит о том, что модель хорошо обучена. Для обучающего набора данных среднеквадратическая ошибка равна 2.446, это очень немного. А для валидационного набора ошибка составила 20.389, что намного больше. Это произошло из-за того, что эти данные никак не участвовали в алгоритмах расчета весов.

10. Сделать итоговый вывод касательно причин различия в результатах при выполнении работ разными методами, а также по получаемым моделям в целом. Провести сравнительный анализ.

Результаты примерно одинаковые для всех моделей, а именно MSE и R2 для обучающих данных и MSE для валидационных данных практически совпадают.

Вывод:

В результате выполнения лабораторной работы я исследовал аналитический и градиентный способы расчета коэффициентов модели линейной регрессии, изучил библиотеки для градиентного способа, изучил способы оценки качества построенных моделей.