|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования Описание: Описание: FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 5 | | |
| по дисциплине «Статистические методы анализа данных» | | |
| ОЦЕНИВАНИЕ ПАРАМЕТРОВ МОДЕЛИ РЕГРЕССИИ В УСЛОВИЯХ МУЛЬТИКОЛЛИНЕАРНОСТИ | | |
|  | | |
|  | Факультет | фпми |
|  | Группа | пми - 12 |
|  |  |
| Вариант | 10 |
| Студенты | Швадченко А. В.  субботин Д.А |
| Преподаватели | Попов а. а. |
|  |  |
| Новосибирск, 2024 | | |

1. **Задание**

Регрессия на 7 факторах. Эффект мультиколлинеарности создают 4 фактора. Разброса в масштабах факторов нет.

1. В соответствии с вариантом задания сгенерировать экспериментальные данные, в которых в явном виде присутствует эффект мультиколлинеарности.
2. Рассчитать ряд показателей, характеризующих эффект мультиколлинеарности. Определить факторы, ответственные за возникновение эффекта мультиколлинеарности.
3. Построить ридж-оценки параметров при различных значениях параметра регуляризации. Выбрать оптимальное значение параметра регуляризации. Построить графики изменения квадрата евклидовой нормы оценок параметров и остаточной суммы квадратов от параметра регуляризации.
4. Провести оценивание модели регрессии по методу главных компонентов. Перейти к описанию в исходном пространстве факторов. Сравнить решение с ридж-оцениванием по смещению оценок и точности предсказания отклика.
5. **Выбор модели**

Выберем имитационную модель η( x, θ):

η(x, θ) = θT f(x)

Эффект мультиколлинеарности создают 4 фактора. Разброса в масштабах факторов нет:

Выберем такие значения , чтобы выполнялись условия задачи:

θ = (1, 1, 1, 1, 1, 1, 1)

n = 100 – размер выборки, m = 8 – число параметров

1. **Ход работы**
2. Рассчитать ряд показателей, характеризующих эффект мультиколлинеарности. Определить факторы, ответственные за возникновение эффекта мультиколлинеарности.

***Определитель информационной матрицы , нормированный на след:***

Определитель близок к нулю, следовательно, и минимальное собственное значение близко к нулю, что говорит о плохой обусловленности матрицы.

***Минимальное собственное число матрицы :***

Чем меньше минимальное собственное значение матрицы, тем сильнее мультиколлинеарность.

***Мера обусловленности по Нейману-Голдстейну:***

Так как имеется линейная зависимость, то данная мера не отражает разницу в масштабах.

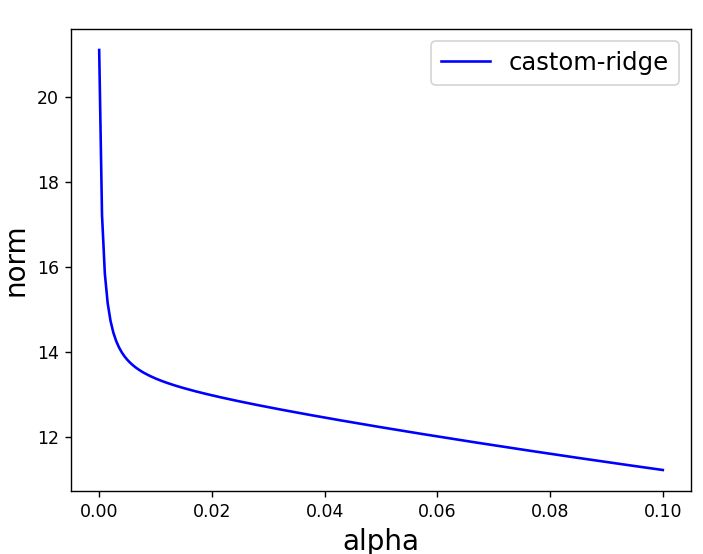
***Максимальная парная сопряженность:***

***Максимальная сопряженность***:

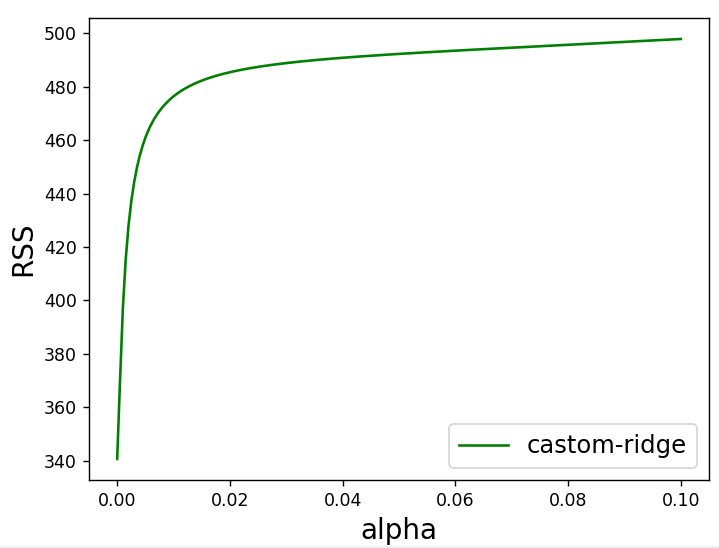
Максимальная сопряженность тоже довольно близка к 1.

1. Построить ридж-оценки параметров при различных значениях параметра регуляризации. Выбрать оптимальное значение параметра регуляризации. Построить графики изменения квадрата евклидовой нормы оценок параметров и остаточной суммы квадратов от параметра регуляризации.

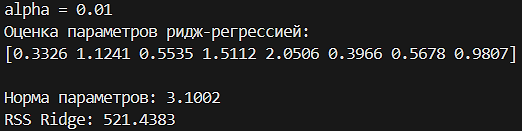
**График зависимости квадрата нормы параметров модели от параметра регуляризации alpha:**



**График зависимости остаточной суммы квадратов от параметра регуляризации alpha:**



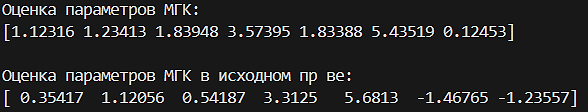
Оптимальным выглядит значение параметра регуляризации alpha = 0.01



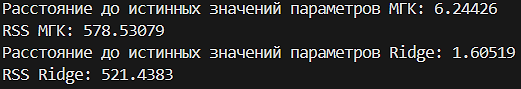
1. Провести оценивание модели регрессии по методу главных компонентов. Перейти к описанию в исходном пространстве факторов. Сравнить решение с ридж-оцениванием по смещению оценок и точности предсказания отклика.



Уберем последнюю компоненту из дальнейшего анализа, поскольку она описывает изменение свободного члена, что является нулевым. При построении информационной матрицы с полученными главными компонентами, окажется что она невырожденная.



Характеристики оценки МГК и Ridge:



1. **Вывод**

Оценка, полученная с помощью ридж-оценки получилась более точной, чем с помощью МГК.

1. **Текст программы**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from math import ceil

def make\_X(x\_array: np.array):

return np.array([[x\_array[0][i],

x\_array[1][i],

x\_array[2][i],

x\_array[3][i],

x\_array[4][i],

x\_array[5][i],

x\_array[6][i],

1,] for i in range(len(x\_array[0]))])

theta = np.array([1, 1, 1, 1, 1, 1, 1, 1])

def u\_calc(x1, x2, x3, x4, x5, x6, x7):

return theta[0] \* x1 + theta[1] \* x2 + theta[2] \* x3 + theta[3] \* x4 + theta[4] \* x5 + theta[5] \* x6 + theta[6] \* x7 + theta[7]

def make\_experiments(n: int):

x1 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x2 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x3 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x4 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x5 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x6 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

x7 = np.linspace(-1.0, 1.0, ceil(n\*\*(1/8)))

#np.random.seed(123)

u = np.array([])

x1\_array = np.array([])

x2\_array = np.array([])

x3\_array = np.array([])

x4\_array = np.array([])

x5\_array = np.array([])

x6\_array = np.array([])

x7\_array = np.array([])

error\_array = np.array([])

y\_array = np.array([])

for i1 in x1:

for i2 in x2:

for i3 in x3:

for i4 in x4:

for i5 in x5:

for i6 in x6:

i7 = i4 +2\*i5 -i6 + np.random.normal(0, 0.1)

x1\_array = np.append(x1\_array, i1)

x2\_array = np.append(x2\_array, i2)

x3\_array = np.append(x3\_array, i3)

x4\_array = np.append(x4\_array, i4)

x5\_array = np.append(x5\_array, i5)

x6\_array = np.append(x6\_array, i6)

x7\_array = np.append(x7\_array, i7)

u = np.append(u, u\_calc(i1, i2, i3, i4, i5, i6, i7))

u\_avg = np.average(u)

omega = (u - u\_avg).T @ (u - u\_avg) / len(u)

for i in range(len(u)):

error = np.random.normal(0.0, np.sqrt(0.6\*omega))

error\_array = np.append(error\_array, error)

y\_array = np.append(y\_array, u[i] + error)

df = pd.DataFrame({'x1': x1\_array, 'x2': x2\_array, 'x3': x3\_array, 'x4': x4\_array, 'x5': x5\_array, 'x6': x6\_array, 'x7': x7\_array, 'u': u, 'e': error\_array, 'y': y\_array})

df.to\_csv('data.csv', index=False)

return df

#Класс регрессии по МНК

class OLS():

def \_\_init\_\_(self):

self.theta\_array = np.array([])

self.X = np.array([])

self.Y = np.array([])

def fit(self, X: np.array, Y: np.array):

self.X = X

self.Y = Y

self.theta\_array = np.linalg.inv(self.X.T @ self.X) @ self.X.T @ self.Y

def predict(self, X: np.array):

return X @ self.theta\_array

#Класс ридж-регрессии

class Ridge():

def \_\_init\_\_(self, alpha: np.float64 = 1):

self.alpha = alpha

self.theta\_array = np.array([])

self.X = np.array([])

self.Y = np.array([])

def fit(self, X: np.array, Y: np.array):

self.X = X

self.Y = Y

self.Lambda = np.diag(np.diag(self.X.T @ self.X)) \* self.alpha

self.theta\_array = np.linalg.inv(self.X.T @ self.X + self.Lambda) @ self.X.T @ self.Y

def predict(self, X: np.array):

return X @ self.theta\_array

def norm(self):

return np.linalg.norm(self.theta\_array)

def RSS(self):

return np.sum((self.X @ self.theta\_array - self.Y)\*\*2)

##Класс тестов мультиколлинеарности

class MulticollinearityTest():

def \_\_init\_\_(self, X: np.array):

self.X = X

def Det(self):

return np.linalg.det(self.X.T @ self.X/np.trace(self.X.T @ self.X))

def MinW(self):

return np.min(np.linalg.eig(self.X.T @ self.X)[0])

def Neumann\_Goldstein(self):

return np.max(np.linalg.eig(self.X.T @ self.X)[0])/self.MinW()

def MaxPairConjugacy(self):

x\_t = self.X.T

max\_k = 0

max\_comb = (0, 0)

for i in range(len(x\_t)):

for j in range(i + 1, len(x\_t)):

k = np.dot(x\_t[i], x\_t[j]) / (np.linalg.norm(x\_t[i]) \* np.linalg.norm(x\_t[j]))

if k > max\_k:

max\_k = k

max\_comb = (i, j)

return (max\_k, max\_comb)

def MaxConjugacy(self):

x\_t = self.X.T

R = np.zeros((9, 9))

for i in range(len(x\_t)):

for j in range(i + 1, len(x\_t)):

if i != j:

k = np.dot(x\_t[i], x\_t[j]) / (np.linalg.norm(x\_t[i]) \* np.linalg.norm(x\_t[j]))

R[i, j] = k

R[j, i] = k

return np.max(R)

#Класс метода главных компонент

class PCA:

def \_\_init\_\_(self, n\_components=None):

self.n\_components = n\_components

def fit\_transform(self, X):

n\_samples, n\_features = X.shape

if self.n\_components is None:

self.n\_components = min(n\_samples, n\_features)

X\_centered = X - X.mean(axis=0)

self.U, self.V = np.linalg.eig(X\_centered.T @ X\_centered)

self.explained\_variance = (self.U[:self.n\_components] \*\* 2) / (n\_samples - 1)

self.explained\_variance\_ratio = self.explained\_variance / np.sum(self.explained\_variance)

return X\_centered[:, : self.n\_components] @ self.V.T[:self.n\_components, : self.n\_components]

df = make\_experiments(10)

print(df)

#получение необходимых оценок

alpha = 0.01

# 1. Вычисляем X и Y перед использованием в ridge.fit()

X = make\_X(np.array([np.array(df['x1']), np.array(df['x2']), np.array(df['x3']), np.array(df['x4']), np.array(df['x5']), np.array(df['x6']), np.array(df['x7'])]))

Y = np.array(df['y'])

ridge = Ridge(alpha)

ridge.fit(X, Y)

print(f'alpha = {alpha}')

print(f'Оценка параметров ридж-регрессией: \n{ridge.theta\_array.round(4)}\n')

print(f'Норма параметров: {ridge.norm().round(4)}')

print(f'RSS Ridge: {ridge.RSS().round(4)}')

pca = PCA(8)

Xt = pca.fit\_transform(X)

print(f'Значимость признаков МГК:\n{pca.explained\_variance\_ratio.round(5)}')

#обрезаем последний 0, что бы матрица X^T\*X была вырожденной

pca.n\_components = 7

Xt = pca.fit\_transform(X)

Yt = Y - np.mean(Y)

ols = OLS()

ols.fit(Xt, Yt)

print(f'Оценка параметров МГК:\n{ols.theta\_array.round(5)}\n')

est\_theta = pca.V.T[:pca.n\_components, : pca.n\_components] @ ols.theta\_array

print(f'Оценка параметров МГК в исходном пр ве:\n{(est\_theta).round(5)}')

print(f'Расстояние до истинных значений параметров МГК: {np.linalg.norm(theta[:pca.n\_components] - est\_theta).round(5)}')

print(f'RSS МГК: {(((Y - X[:, :pca.n\_components]@est\_theta.T).T)@(Y - X[:, :pca.n\_components]@est\_theta.T)).round(5)}')

print(f'Расстояние до истинных значений параметров Ridge: {np.linalg.norm(theta - ridge.theta\_array).round(5)}')

print(f'RSS Ridge: {ridge.RSS().round(4)}')

#Тесты коллинеарности

X = make\_X(np.array([np.array(df['x1']), np.array(df['x2']), np.array(df['x3']), np.array(df['x4']), np.array(df['x5']), np.array(df['x6']), np.array(df['x7'])]))

Test = MulticollinearityTest(X)

print(f'Определитель X^T\*X: {Test.Det()}')

print(f'Минимальное собственное число: {Test.MinW().round(3)}')

print(f'Тест Неймана-Голдстейна: {Test.Neumann\_Goldstein().round(3)}')

print(f'Максимальная парная сопряженность: значение - {Test.MaxPairConjugacy()[0].round(3)}, пара факторов - {Test.MaxPairConjugacy()[1]}')

print(f'Максимальная сопряженность: {Test.MaxConjugacy().round(3)}')

#Формирования графиков зависимости нормы параметров и квадратов остатков от параметра регуляризации

norm\_1, rss\_1, norm\_2, rss\_2 = [], [], [], []

alpha = np.linspace(0.00001, 0.1, 200)

for i in alpha:

X = make\_X(np.array([np.array(df['x1']), np.array(df['x2']), np.array(df['x3']), np.array(df['x4']), np.array(df['x5']), np.array(df['x6']), np.array(df['x7'])]))

Y = np.array(df['y'])

rols = Ridge(i)

rols.fit(X, Y)

norm\_1.append(rols.norm().round(5)\*\*2)

rss\_1.append(rols.RSS().round(5))

plt.plot(alpha, norm\_1, color='b', label='castom-ridge')

plt.xlabel(r'alpha', fontsize=16)

plt.ylabel(r'norm', fontsize=16)

plt.legend(fontsize=14)

plt.show()

plt.plot(alpha, rss\_1, color='g', label='castom-ridge')

plt.xlabel(r'alpha', fontsize=16)

plt.ylabel(r'RSS', fontsize=16)

plt.legend(fontsize=14)

plt.show()