Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет
Кафедра теоретической и прикладной информатики

Статистические методы анализа данных

Лабораторная работа №5

Факультет: ФПМИ
Группа: ПМ-63
Студенты: Кожекин М.В.
Майер В.А.
Назарова Т.А.
Утюганов Д.С.
Вариант: 1

Новосибирск
2019

1. Цель работы

Разработать программу для определения мультиколлинеарности модели.

2. Решение

1. В соответствии с вариантом задания сгенерировать экспериментальные данные, в которых в явном виде присутствует эффект мультиколлинеарности.

Согласно заданию регерссия на 6 факторах. Эффект мультиколлинеарности создают 3 фактора. Имеется разброс в масштабах факторов.

$\eta = f^T(x) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$
$x_1 \in [-1, 1]$

$x_2 \in [-1, 1]$

$x_3 \in [-1, 1]$

$x_4 \in [-1, 1]$

$x_5 \in [-1, 1]$

$x_6 = 2x_4 - 3x_5 + N(0, 0.01)$

$\rho = 0.15, n = 100, m = 6$

2. Рассчитать ряд показателей, характеризующих эффект мультиколлинеарности. Определить факторы, ответсвенные за возникновение эффекта мультиколлинеарности.

В качестве мер измерения эффекта мультиколлинеарности рассмотрим следующее:

1. определитель информационной матрицы

$$|X^T X| = \prod_{i=1}^{m} \lambda_i = 7.22$$

2. минимальное собственное число матрицы

$$\lambda_1 = \lambda_{min}(X^T X) = 4.77e - 06$$

3. мера обусловленности матрицы по Нейману-Голдстейну

$$\frac{\lambda_{max}(X^T X)}{\lambda_{min}(X^T X)} = 34699460.47$$

4. максимальная парная сопряжённость

$\max\limits_{i,j} x|r_{ij}|, i \neq j$
$R = (r_{ij})$ – матрица сопряжённости

$$r_{ij} = cos(\underline{x}_i, \underline{x}_j) = \frac{(x_i, x_j)}{|x_i||x_j|}$$

$$\max_{i,j} x|r_{ij}| = 0.84$$

5. максимальная сопряжённость

   В качестве меры мультиколлинеарности возьмём

$$\max_i |R_i|, \text{ где :}$$

$$R_i = \sqrt{1 - \frac{1}{R_{ii}^{-1}}}, i = 1, m$$

$$\max_i |R_i| = 0.99$$

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

Один из способов оценивания параметров в условиях мултиколлинеарности состоит в управлении масштабом полученных оценок. Однако это смещение значительно меньше, чем у обычных МНК оценок.

С целью управления масштабом оценок введём в расмотрение функцию стоимости

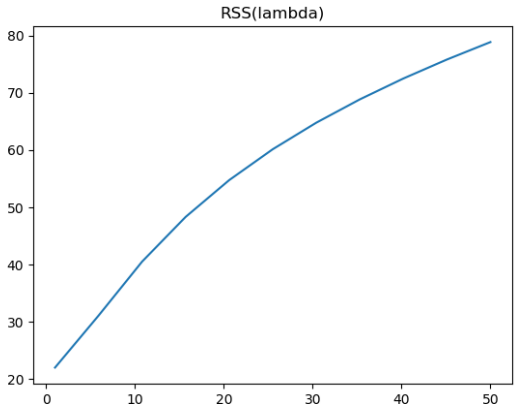
$$C = \sum_{i=1}^{n}(y_i - f(X_i)\theta)^2 + \sum_{j=1}^{m}\lambda_j\theta_j^2$$

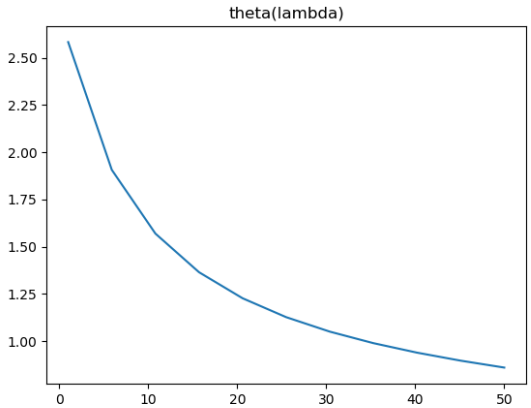
где второе слагаемое рассматривается как штраф при условии, что $\lambda_j \geq 0$

Минимизируя C, получим $\hat{\theta} = (X^TX + \Lambda)^{-1}X^Ty$, которые известны как ридж-оценки. Часто матрицу Λ задают диагональной в виде $\Lambda_{ii} = \lambda(X^TX)_{ii}, \lambda \geq 0$.

Оптимальное значение $\lambda = m\hat{\sigma}^2/||\hat{\beta}||^2$

Графики:





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

Для начала мы центрируем нашу исходнную модель

$$y_t = \Theta_0 + \Theta_1 x_{1t} + ... + \Theta_k x_{kt} + E_t$$

Тогда модель наблюдения будет иметь вид

$$y_t^* = \beta_0 + \beta_1 x_{1t}^* + ... + \beta_k x_{kt}^* + E_t$$

Т.к. меру изменчивости можно измерить при помощи собственных значений, то мы можем получить матрицу главных компонент через матрицу

$$V = (v_1, ..., v_k)$$
$$Z = X^* V$$

После отбора главных компонент можно оценить регрессию y^* на главные компоненты:

$$y_t^* = b_1 z_{1t} + ... + b_l z_{lt} + E_t$$
$$\hat{b} = (Z^T Z)^{-1} Z^T y^*$$

$\theta = 1, 1, 1, 1, 1, 1$
$\hat{\theta} = 1.78, 3.58, 0.56, 1.30, 1.13, 0.63$

3. Текст программы

Файл run_lab5.py

```
import os

# os.system('python 1/lab1.py 0.15 6 False')
os.system('python 5/lab5.py 0.15')
```

Файл model_6parameters.py

```
# файл параметров модели
import numpy as np


# параметры уравнения
# th0*x1 + th1*x2 + th2*x3 + th3*x4 + th4*x5 + th5*x6
# x6 = 2*x2 + x3 + 4*x4
theta_true = [1, 1, 1, 1, 1, 1]
n, m = 30, 6                # число точек, параметров и осей
rho = 0.15                        # шум в диапазоне [0.05, 0.15] или [0.50, 0.70]
a = [-1, -1, -1, -1, -1]
b = [ 1,  1,  1,  1,  1]


def sample_x():
    x = np.ndarray((m,n))
    x[0] = np.random.uniform(low=a[0], high=b[0], size=n)
    x[1] = np.random.uniform(low=a[1], high=b[1], size=n)
    x[2] = np.random.uniform(low=a[2], high=b[2], size=n)
    x[3] = np.random.uniform(low=a[3], high=b[3], size=n)
```

```
    x[4] = np.random.uniform(low=a[4], high=b[4], size=n)
    x[5] = 2*x[0] - 3*x[3] + np.random.uniform(0, 0.01)
    return x.transpose()


def f(theta, x):
    return np.dot(theta, x)

def f_vector_T(x):
    return np.array(x)
```

Файл lab1.py

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from scipy import stats
from sympy import *
from sys import argv, path


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
class lab1():
    def __init__(self):
        self.x = sample_x()
        self.u = np.ndarray(n)
        for i in range(n):
            self.u[i] = f(theta_true, self.x[i])
        u_mean = np.full((n), np.mean(self.u))
        w_squared = np.dot((self.u - u_mean), (self.u - u_mean)) / (n-1)
        self.dist = rho * w_squared
        self.y = np.copy(self.u)
        for i in range(n):
            self.y[i] += np.random.normal(0, self.dist)

    def save_table_1(self):
        with open('1/report/table_1_u_y_' + str(int(rho*100)) + '.txt', 'w') as
    file:
            file.write('i\t')
            for i in range(1, m+1):
                file.write('x%d\t' % i)
            file.write('u\ty\n')
            for i in range(n):
                file.write('{:d}\t'.format(i))
                for j in range(m):
                    file.write('{:.17f}\t'.format(self.x[i][j]))

                file.write('{:.17f}\t{:.17f}\n'.format(self.u[i], self.y[i]))

    def draw(self, doDrawWithNoize, title, name, elevation, azimuth):
        path_to_save = '1/pics/' + name + '_' + \
            str(int(rho*100)) + '_' + str(elevation) + '_' + str(azimuth) + '.
    png'
        fig = plt.figure('1')
        ax = fig.gca(projection='3d')
        tmp_range = np.linspace(a, b, sqrt(n))
        x1, x2 = np.meshgrid(tmp_range, tmp_range)
        u = f_2_parameters(theta_true, x1, x2)
        fig.subplots_adjust(bottom=-0.05, top=1, left=-0.05, right=1.05)
```

```
        ax.view_init(elevation, azimuth)
        ax.plot_surface(x1, x2, u, zorder=2, alpha=0.2)
        if doDrawWithNoize:
            title += ' ' + str(int(rho * 100)) + '%'
            ax.scatter(self.x1, self.x2, self.y, c='black', zorder=1)
        else:
            ax.scatter(self.x1, self.x2, self.u, c='black', zorder=1)
        plt.title(title, fontsize=19)
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.grid(alpha=0.5)
        plt.savefig(path_to_save)
        plt.clf()


#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
if __name__ == "__main__":
    path.insert(1, '')
    global rho
    rho = float(argv[1])

    params_count = int(argv[2])
    if params_count == 3:
        from model_3parameters import *
    elif params_count == 4:
        from model_4parameters import *
    elif params_count == 6:
        from model_6parameters import *

    doDraw = bool(argv[3])

    print('Запушен код 1 лабораторной работы: {:d} параметров, шум {:d}%'.format(int
    (argv[2]), int(rho*100)))

    l1 = lab1()
    l1.save_table_1()
    doDraw = False
    if doDraw:
        l1.draw(False, 'исходная модель', 'before', None, None)
        l1.draw(False, 'исходная модель', 'before', 0, 0)
        l1.draw(False, 'исходная модель', 'before', 0, 90)
        l1.draw(True, 'помеха', 'after', None, None)
        l1.draw(True, 'помеха', 'after', 0, 0)
        l1.draw(True, 'помеха', 'after', 0, 90)
```

Файл lab5.py

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import inv, det, eigvals, norm
from sys import argv, path


def cos(a, b):
    ''' косинус между углами (a,b) = |a|*|b|*cos(a,b) '''
    return np.dot(a, b) / (norm(a)*norm(b))

#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
```

```
#-----------------------------------------------------------------------------
class lab5():
    def __init__(self):
        ''' Выделение памяти под массивы '''
        self.x = np.ndarray((n, m))
        self.X = np.ndarray((n, m))
        self.u = np.ndarray(n)
        self.y = np.ndarray(n)
        self.R = np.ndarray((m, m))

    def read_from_file(self, filename):
        ''' Считываем данные из файла '''
        data = np.loadtxt(filename,float,skiprows=1).transpose()
        self.x = data[1:m+1].transpose()
        self.u = data[-2].transpose()
        self.y = data[-1].transpose()
        for i in range(n):
            self.X[i] = f_vector_T(self.x[i])

    def calc_det_of_inf_matrix(self):
        ''' Вычисляем определитель информационной матрицы '''
        X = self.X
        self.inf_matr = X.T @ X
        d = det(self.inf_matr)
        print('определитель информационной матрицы: ', d)

    def calc_min_eig_val(self):
        ''' Поиск минимального собственного значения '''
        self.eig_vals = eigvals(self.inf_matr)
        self.min_eig_val = np.min(self.eig_vals)
        self.max_eig_val = np.max(self.eig_vals)
        print('минимальное собственное значение: ', self.min_eig_val)

    def calc_cond_number(self):
        ''' Мера обусловленности по НеймануГолдстейну- '''
        self.cond_number = self.max_eig_val / self.min_eig_val
        print('число обусловленности: ', self.cond_number)

    def calc_max_pair_sopr(self):
        '''
        Определяем максимальную парную сопряжённость.
        3 вектора могут быть коллинеарны, а попарно нет
        Максимальный коэффициент сопряжённости не несёт на себе эффекта масштаба
        '''
        X = self.X.transpose()
        self.max_pair_sopr = -9000.0
        # строим матрицу сопряжённости
        for i in range(m):
            for j in range(m):
                self.R[i][j] = cos(X[i], X[j])
            self.R[i][i] = 1.0

        for i in range(m):
            for j in range(m):
                if i != j and (abs(self.R[i][j]) > self.max_pair_sopr):
                    self.max_pair_sopr = abs(self.R[i][j])
        print('максимальная парная сопряжённость: ', self.max_pair_sopr)

    def calc_max_sopr(self):
        ''' Максимальная сопряжённость '''
```

```
        R_inv = inv(self.R)
        self.max_sopr = -9000.0
        for i in range(m):
            R_i = np.sqrt(1.0 - (1.0 / R_inv[i][i]))
            if R_i > self.max_sopr and R_i != float('inf'):
                self.max_sopr = R_i
        print('максимальная сопряжённость: ', self.max_sopr)

    def calc_theta_ridge(self):
        '''
        Риджоценка-
        Мультиколлинеарные( данные выглядят как хребты на карте)
        '''
        X = self.X
        y = self.y
        points = 11
        RSS = np.ndarray(points)
        theta_norm = np.ndarray(points)
        theta_calc_ridge = np.ndarray((points, m))
        lambda_params = np.linspace(1, 50, points)
        i = 0
        for lambda_param in lambda_params:
            lambdas = np.diag(np.full(m, lambda_param))
            theta = inv(X.T @ X + lambdas) @ X.T @ y
            RSS[i] = (y - X @ theta).T @ (y - X @ theta)
            theta_norm[i] = norm(theta)
            theta_calc_ridge[i] = theta.T
            i += 1

        # print(theta_calc_ridge)

        path_to_save = '5/pics/RSS_from_lambda.png'
        plt.title('RSS(lambda)')
        plt.plot(lambda_params, RSS)
        plt.savefig(path_to_save)
        plt.clf()

        path_to_save = '5/pics/theta_from_lambda.png'
        plt.title('theta(lambda)')
        plt.plot(lambda_params, theta_norm)
        plt.savefig(path_to_save)
        plt.clf()

    def calc_theta_main_components(self):
        '''
        Метод главных компонент
        '''
        X = self.X.T
        y = self.y
        for i in range(m):
            X[i] -= np.mean(X[i])
        X_centered = X.T
        X_inf_matr_centered = X_centered.T @ X_centered

        V = np.diag(eigvals(X_inf_matr_centered))
        Z = X_centered @ V
        b = inv(Z.T @ Z) @ Z.T @ y
        self.theta_calc_mc = V.dot(b)
        print(self.theta_calc_mc)

```

```
133
134 #-----------------------------------------------------------------------------------------------------------------------
135 #-----------------------------------------------------------------------------------------------------------------------
136 #-----------------------------------------------------------------------------------------------------------------------
137 if __name__ == "__main__":
138     path.insert(1, '')
139     from model_6parameters import *
140
141     global rho
142     rho = float(argv[1])
143     print('5 Лабораторная работа:', m, 'параметров, шум', int(rho*100), '\n')
144
145     l5 = lab5()
146     l5.read_from_file('1/report/table_1_u_y_' + str(int(rho*100)) + '.txt')
147     l5.calc_det_of_inf_matrix()
148     l5.calc_min_eig_val()
149     l5.calc_cond_number()
150     l5.calc_max_pair_sopr()
151     l5.calc_max_sopr()
152     l5.calc_theta_ridge()
153     l5.calc_theta_main_components()
```