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.

$\eta = f^T(x) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$
$x_1 \in [-1, 1]$
$x_2 \in [-2, 2]$
$x_3 \in [-100, 100]$
$x_4 \in [-10, 10]$
$x_5 \in [-1, 1]$
$x_6 = 2x_4 - 3x_5 + N(0, 0.01)$
$\rho = 0.15, n = 100, m = 6$

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
   $|X^T X| = \prod_{i=1}^{m} \lambda_i = 59987206$
2. минимальное собственное число матрицы
   $\lambda_1 = \lambda_{min}(X^T X) = 2.1923124189023676e - 06$
3. мера обусловленности матрицы по Нейману-Голдстейну
   $\frac{\lambda_{max}(X^T X)}{\lambda_{min}(X^T X)} = 704048310351$
4. максимальная парная сопряжённость
   $\max\limits_{i,j} x|r_{ij}|, i \neq j$
   $R = (r_{ij})$ – матрица сопряжённости
   $r_{ij} = cos(\underline{x}_i, \underline{x}_j) = \sum \frac{X_{it} X_{jt}}{\sqrt{(\sum X_{it}^2 \sum X_{jt}^2)}}$
   $\max\limits_{i,j} x|r_{ij}| = 0.082419$
5. максимальная сопряжённость
   В качестве меры мультиколлинеарности возьмём
   $\max\limits_{i} |R_i|$,где:
   $\mathrm{R}_i = \sqrt{1 - \frac{1}{R_{ii}^{-1}}}, i = 1, m$
   $\max\limits_{i} |R_i| = 0.926$

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

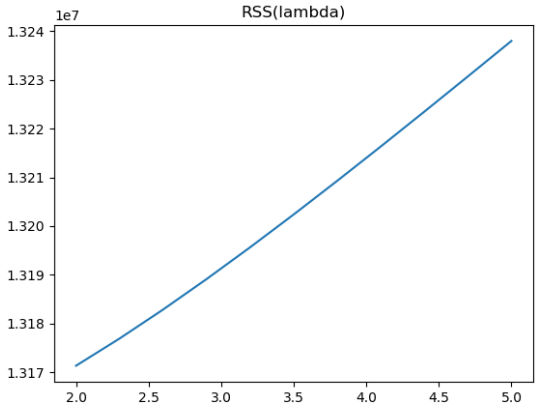
$C = \sum_{i=1}^{n}(y_i - f(X_i)\theta)^2 + \sum_{j=1}^{m}\lambda_j\theta_j^2$

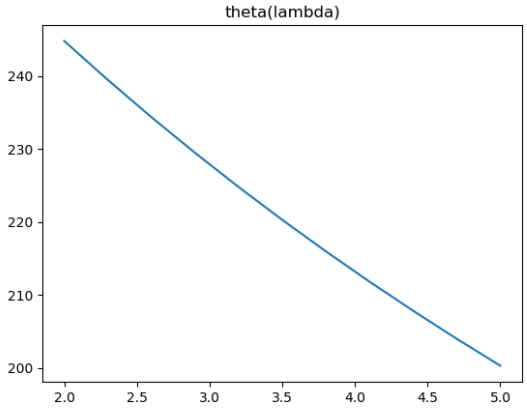
где второе слагаемое рассматривается как штраф при условии, что $\lambda_j \geq 0$

Минимизируя С, 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* :$y_t^* = b_1 z_{1t} + ... + b_l z_{lt} + E_t \cap b = (Z^TZ)^-1Z^Ty^*$

theta = -55.3567551900075, 41.0306349638696, 3.23326603674220, 15.9659268561517, 297.857791838169
7.93397529930516

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
theta_true = [1, 1, 1, 1, 1, 1]
n, m = 30, 6              # число точек, параметров и осей
# a, b = -1, 1
a = [-10, -2, -100, -10, -1]
b = [ 10,  2,  100,  10,  1]

# x6 = 2*x2 + x3 + 4*x4
rho = 0.15                        # шум в диапазоне [0.05, 0.15] или [0.50, 0.70]


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)
    # print(x[5])
    return x.transpose()


def f(th, x_vector):
    return  th[0] * x_vector[0] + \
            th[1] * x_vector[1] + \
            th[2] * x_vector[2] + \
            th[3] * x_vector[3] + \
            th[4] * x_vector[4] + \
            th[5] * x_vector[5]


def f_vector_(x_vector):
    return [x_vector[0],
            x_vector[1],
            x_vector[2],
            x_vector[3],
            x_vector[4],
            x_vector[5]]


def f_vector(th, x_vector):
    return [th[0] * x_vector[0],
            th[1] * x_vector[1],
```

```
            th[2] * x_vector[2],
            th[3] * x_vector[3],
            th[4] * x_vector[4],
            th[5] * x_vector[5] ]
```

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('Запущен 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 mpl_toolkits.mplot3d import Axes3D
from scipy import stats
from sympy import *
from sys import argv, path
import math


def dotproduct(v1, v2):
    return sum((a*b) for a, b in zip(v1, v2))


def length(v):
    return math.sqrt(abs(dotproduct(v, v)))


def angle2(v1, v2):
```

```
    return math.acos(abs(dotproduct(v1, v2)) / (length(v1) * length(v2)))


def angle(v1, v2):
    return np.dot(a, b) / (np.linalg.norm(a)*np.linalg.norm(b))

# ____________________________________________________________________________

# ____________________________________________________________________________

# ____________________________________________________________________________


class lab5():
    def __init__(self):
        self.x = np.ndarray((n, m))
        self.X = np.ndarray((n, m))
        self.u = np.ndarray(n)
        self.y = np.ndarray(n)

        with open('1/report/table_1_u_y_' + str(int(rho*100)) + '.txt', 'r') as file:
            line = file.readline().rstrip()
            for i in range(n):
                line = file.readline().rstrip().rsplit()
                for j in range(m):
                    self.x[i][j] = float(line[j + 1])
                self.u[i] = float(line[m + 1])
                self.y[i] = float(line[m + 2])

        for i in range(n):
            self.X[i] = f_vector_(self.x[i])

    def calc_det_of_inf_matrix(self):
        X = Matrix(self.X)
        self.inf_matr = np.array(X.T * X).astype(np.float64)
        d = np.linalg.det(self.inf_matr)
        print('det = ', d)

    # def calc_det_of_inf_matrix(self):
    #     X = Matrix(self.X)
    #     self.inf_matr_sp = X.T * X
    #     self.inf_matr_sp = self.inf_matr_sp / Trace(self.inf_matr_sp)
    #     print(self.inf_matr_sp)
    #     self.inf_matr = np.array(self.inf_matr_sp).astype(np.float64)
    #     d = np.linalg.det(self.inf_matr)
    #     print('det = ', d)

    def calc_min_eig_val(self):
        self.eig_vals = np.linalg.eigvals(self.inf_matr)
        print('lambda_min = ', np.min(self.eig_vals))

    def calc_cond_number(self):
        self.cond_number = np.max(self.eig_vals) / np.min(self.eig_vals)
        print('cond_number = ', self.cond_number)

```

```
    def calc_pair_sopr(self):
        X = self.X.transpose()
        self.R = np.ndarray((m, m))
        self.pair_sopr = -9000.0
        for i in range(m):
            for j in range(m):
                self.R[i][j] = 0.0
                for t in range(n):
                    numerator = X[i][t] * X[j][t]
                    denominator1 = 0.0
                    for t1 in range(n):
                        denominator1 += X[i][t1]**2
                    denominator2 = 0.0
                    for t2 in range(n):
                        denominator2 += X[i][t2]**2
                self.R[i][j] += numerator / np.sqrt(denominator1*denominator2)
            self.R[i][i] = 1.0

        #print(max(abs(self.R.max()), abs(self.R.min())))

        for i in range(m):
            for j in range(m):
                if i != j and (abs(self.R[i][j]) > self.pair_sopr):
                    self.pair_sopr = abs(self.R[i][j])

        # print(self.R)
        print('pair_sopr = %f' % self.pair_sopr)

    # def calc_pair_sopr(self):
    #     X = self.x.transpose()
    #     # print(X)
    #     self.R = np.ndarray((m, m))
    #     self.pair_sopr = -9000
    #     for i in range(m):
    #         for j in range(m):
    #             self.R[i][j] = angle(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.pair_sopr:
    #                 self.pair_sopr = abs(self.R[i][j])

    #     # print(self.R)
    #     print('pair_sopr = %d' % self.pair_sopr)

    def calc_max_sopr(self):
        # R_inv = Matrix(self.R)**-1
        R_inv = np.linalg.inv(self.R)
        self.R_list = np.ndarray(m)
        # print(R_inv)
        self.max_sopr = -9000.0
        for i in range(m):
            R_i = 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(max(abs(self.R.max()), abs(self.R.min())))

        print('max_sopr = %f' % self.max_sopr)

```

```
    def calc_theta_ridge(self):
        X = Matrix(self.X)
        y = Matrix(self.y)
        _theta_true = Matrix(theta_true)
        points = 11
        RSS = np.ndarray(points)
        theta_norm = np.ndarray(points)
        theta_calc_ridge = np.ndarray((points, m))
        lambda_params = np.linspace(2, 5, points)
        i = 0
        for lambda_param in lambda_params:
            lambdas = Matrix(np.diag(np.full(m, lambda_param)))
            _th = ((X.T*X + lambdas) ** -1) * X.T * y
            _RSS = (y - X*_th).T * (y - X*_th)
            _theta_norm = (_th - _theta_true).norm()

            RSS[i] = np.array(_RSS).astype(np.float64)
            theta_norm[i] = np.array(_theta_norm).astype(np.float64)
            theta_calc_ridge[i] = np.array(_th.T).astype(np.float64)
            i += 1

        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):
        X = self.X.T
        for i in range(m):
            X[i] -= np.mean(X[i])
        X_centered = Matrix(X.T)
        X_inf_matr_centered = X_centered.T * X_centered
        X_centered_np = np.array(X_inf_matr_centered).astype(np.float64)

        V = Matrix(np.diag(np.linalg.eigvals(X_centered_np)))
        y = Matrix(self.y)
        Z = X_centered * V
        b = ((Z.T*Z) ** -1) * Z.T * y
        self.theta_calc_mc = V.dot(b)
        print(self.theta_calc_mc)


# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------

if __name__ == "__main__":
    path.insert(1, '')
```

```
186     from model_6parameters import *
187
188     global rho
189     rho = float(argv[1])
190     print('Запушен код 5 лабораторной работы: {:d} параметров, шум {:d}%'.format(
191         m, int(rho*100)))
192     l5 = lab5()
193     l5.calc_det_of_inf_matrix()
194     l5.calc_min_eig_val()
195     l5.calc_cond_number()
196     l5.calc_pair_sopr()
197     l5.calc_max_sopr()
198     l5.calc_theta_ridge()
199     l5.calc_theta_main_components()
```