Харківський національний університет імені В.Н. Каразіна

Факультет математики і інформатики

Кафедра прикладної математики

Кваліфікаційна робота

на тему: **Про аналітичні розв'язки деякої задачі класифікації**

Виконав: студент групи МП61 II курсу (магістерського рівня), спеціальності 113 Прикладна математика
Майко І.В.

Керівник: доктор фіз.-мат. наук, професор кафедри прикладної математики
Фардигола Л.В.

Харків — 2020

Зміст

Анотація

У кваліфікаційній роботі розглядається аналітичний підхід до задачі класифікації, заданої у вигляді матриці параметрів, заданих дійсними числами, й параметром класу, також надано один з можливих розв'язків задачі $p53Mutants$ за допомогою описаних алгоритмів.

Abstract

The qualification work considers the analytical approach to the classification problem given in the form of a matrix of parameters given by real numbers and a class parameter, also provides one of the possible solutions of the problem $p53Mutants$ using the described algorithms.

1. Вступ

Задача класифікації існує стільки ж, скільки й живі істоти. Саме здатність до розпізнавання образів як механізм виживання задає тон розвитку всього живого. Люди підняли розпізнавання образів на новий рівень, поступаючись миттєвими поривами, розуміючи наслідки свого вибору. Разом з розвитком людини розвивалася і ускладнювалася сфера прийняття рішень. Оптимальне рішення приймає той, хто знає більше, але з часом обсяг інформації перестав піддаватися аналізу голіруч, таким чином півстоліття тому почали розвиватися методи аналізу даних за допомогою комп'ютерів. Наразі, в цю сферу залучено великі ресурси, й всі вони йдуть на алгоритми машинного навчання. В цій роботі я б хотів звернути увагу на те, що існують аналітичні алгоритми, які за умов наявності розв'язку працюють швидше й надають кращий результат. Моя робота в певній мірі буде звітом мого вкладу у роботу групи, яка влітку 2020 року займалася задачею: $p53Mutants$ [5] ; в якості підсумкової роботи з курсу: "Data analytics and data driven decision"в університеті Л'Аквіли (Італія).

2. Постановка задачі

Задача класифікації — формалізована задача, яка містить множину об'єктів (ситуацій), поділених певним чином на класи. Задана скінченна множина об'єктів й класів, до яких вони належать. Така множина називається вибіркою. Ціль — знайти такий алгоритм, що буде здатний класифікувати довільний об'єкт з вихідної множини.

Задача $p53Mutants$ задається матрицею початкових даних:

$$\begin{pmatrix} x_{11} & \dots & x_{1p} & y_1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{n1} & \dots & x_{np} & y_n \end{pmatrix};$$

де x_{ij} — матриця вхідних параметрів (множина об'єктів), $i = 1, \dots, n$ (n - кількість експериментів), $j = 1, \dots, p$ (p - кількість вхідних параметрів одного об'єкта), y_i — вихідні параметри (класи), $\forall i = 1, \dots, n : y_i \in 0, 1$. Загалом кожен стовпчик є статистикою випадкової величини.

Дані представляють собою 31422 експерименти (n), з 5409 параметрами в кожному ($p + 1$). Ці експерименти описують біофізичні моделі мутантних білків $p53$, які можна використовувати для прогнозування транскрипційної активності $p53$. Усі мітки класів визначаються за допомогою аналізів *in vivo*.

Атрибути 1–4826 представляють $2D$ електростатичні та поверхневі особливості. Атрибути 4827–5408 представляють $3D$ особливості, що базуються на відстані. Атрибут 5409 — це атрибут класу, який є активним або неактивним.

Ціллю є знайти f — функцію, для якої виконується рівність: $y = f(x_1, x_2, \dots, x_p)$.

3. Пошук розв'язків серед гіперплощин

Функцію f ми будемо шукати у лінійному вигляді, залежному від двох змінних. Тобто:

$$y = f(x_1, x_2, \dots, x_p) = \theta_0 + \theta_i x_i + \theta_j x_j, \tag{3.1}$$

де $\theta_0, \theta_i, \theta_j \in \mathbb{R},\ i = 1, \dots, p,\ j = 1, \dots, p$.

Розв'язки такого типу можуть й не існувати, або їх може існувати

декілька, але у подальшому ми припускаємо існування, що найменш, одного розв'язку такого виду.

Для початку введемо й доведемо теорему для обґрунтування майбутніх кроків:

Теорема 3.1. *Нехай z_1, z_2 і y — випадкові величини, також $r_{z_1z_2} = \pm 1$. Тут*

$$r_{z_1z_2} = \frac{\operatorname{cov}(z_1, z_2)}{\sqrt{S^2_{z_1} S^2_{z_2}}},$$

є коефіцієнтом кореляції Пірсона,

$$\operatorname{cov}(z_1, z_2) = \sum_{i=1}^{n} (z_{1i} - \overline{z}_1)(z_{2i} - \overline{z}_2),$$
$$S^2_{z_k} = \sum_{i=1}^{n} (z_{ki} - \overline{z}_k)^2;$$

$\overline{z}_1, \overline{z}_2$ є вибірковими середніми z_1, z_2; $S^2_{z_k}$ є вибірковими дисперсіями.

Тоді $r_{z_2y} = \pm r_{z_1y}$.

Доведення. Оскільки

$$r_{z_1z_2} = \pm 1,$$

то

$$\theta_0 + \theta_1 z_1 = z_2;$$

Оскільки

$$S^2_{z_2} = \theta_1^2 S^2_{z_1},$$

то

$$r_{z_2y} = \frac{\operatorname{cov}(z_2, y)}{\sqrt{S^2_{z_2} S^2_y}} = \frac{\operatorname{cov}(\theta_0 + \theta_1 z_1, y)}{\sqrt{\theta_1^2 S^2_{z_1} S^2_y}} = \frac{\theta_1 \operatorname{cov}(z_1, y)}{|\theta_1| \sqrt{S^2_{z_1} S^2_y}} = \operatorname{sgn}(\theta_1) r_{z_1y}. \quad \square$$

За допомогою цієї теореми можемо подивитися на модуль кореляції між усіма x_i й y. Тоді можна згрупувати їх за близькими значеннями й

високою кореляцією між параметрами, а потім шукати зв'язок у вигляді:

$$X_\alpha \theta \approx y, \tag{3.2}$$

де $\alpha = \alpha_1, \ldots, \alpha_k \subset 1, \ldots, p$, X_α - матриця зі стовпчиків з X за номерами з α.

Якщо є зв'язок, тоді можливо знайти невідомий вектор θ за формулою:

$$\theta = \left(X_\alpha^T X_\alpha\right)^{-1} X_\alpha^T y. \tag{3.3}$$

Такий метод також може давати θ, що не дають розв'язку первинної задачі, тобто після отримання цих коефіцієнтів потрібно перевірити їх шляхом перевірки рівняння (3.2).

4. Розв'язки більшого степеня

Якщо подивитися на алгоритм для пошуку парного, лінійного розв'язку, стає зрозумілим, що пару лінійних змінних в рівнянні (3.1) можливо замінити на пару елементів степеневого ряду. Новий вигляд функції, у вигляді якої ми шукаємо відповідь, буде:

$$y \approx \theta_0 + \theta_\alpha x^\alpha + \theta_\beta x^\beta;$$

де $x = (x_1, \ldots, x_n)$, $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\beta = (\beta_1, \ldots, \beta_n)$, $x^\alpha = x_1^{\alpha_1} \ldots x_n^{\alpha_n}$, $\forall\, i, j \in \{1, \ldots, n\} : \alpha_i, \beta_j \in \{0\} \cup \mathbb{N}$.

5. Реалізація алгоритму з використанням Python

У цій реалізації використовуються стандартні бібліотеки, більшість з яких або вже встановлені з будь-яким компілятором Python, або є загально вживаними.

Лістинг 1: Перелік необхідних бібліотек

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
from sklearn.metrics import confusion_matrix
```

У наступній частині дані зчитуються з файлу. Через невдалу архітектуру устаткування для реалізації проекту аналіз проводився на 5000 строках, але додаткова перевірка проводилася на всьому масиві даних. "*K9.data*" - це ім'я файлу, в якому зберігаються данні для аналізу за посиланням [5].

Лістинг 2: Завантаження даних

```
link = "K9.data"

def read_data(link):
    with open(link) as myfile:
        k = []
        text = myfile.read()
        lines = text.split(sep = "\n")
        for i in range(5000):
            k.append(lines[i].split(sep = ","))
        k = pd.DataFrame(k)
    return k

dataset = read_data(link)
```

У цій частині видаляється непотрібний стовпчик, що є наслідком методу зчитування, усі данні перетворюються на числа й видаляються стовпчики, що мають пропущенні значення параметрів.

Лістинг 3: Попередня обробка даних

```
#Last column is garbage, so we drop it:
dataset.drop(5409, axis=1, inplace=True)
#Transform our data from string to numbers or Nan:
dataset.iloc[:,:-1] = dataset.iloc[:,:-1].apply(pd.to_numeric,
    errors='coerce')
#Drop Nan values:
dataset.dropna(inplace=True)
#Change our output to numerical: 'inactive' --> 0 and 'active'
    --> 1:
dataset.replace({'inactive': 0, 'active': 1}, inplace=True)
```

Лістинг 4: Підготовка необхідних елементів для аналізу

```
#Output column extraction:
test = pd.DataFrame(dataset[5408])

#Computation of correlation of the input columns with output
    column:
corr_with_res = []

for i in range(5408):
    corr_with_res.append((test.join(dataset[i])).astype(float)
        .corr()[5408][i])

whole_corr = pd.DataFrame(corr_with_res)

#Collecting "active" rows:
active = dataset[dataset[5408] == 1]
```

За допомогою наведеної нижче функції буде здійснено аналіз даних у наступній частині:

Лістинг 5: Функція для здійснення аналізу

```
def corr_mean(alpha = 0, beta = 0.4, drop = pd.Index([])):
```

```
    signif_col = (whole_corr[whole_corr[0].abs() > alpha].
        index &\\
                          whole_corr[whole_corr[0].abs() \\
                          < beta].index).difference(drop)
    active_signif = active[signif_col]
    active_signif_corr = active_signif.T.astype(float).corr()
    mean = 0

    for i in range(len(active_signif_corr) - 1):
        sum_row_corr = active_signif_corr.iloc[i:i+1,i+1:].T.
            abs().sum().values[0]
        mean += sum_row_corr
    mean = mean/(((len(active_signif_corr) - 1)*len(
        active_signif_corr))/2)
    return mean, signif_col
```

Далі, використовуючи *corr_mean*, в ручну звужувалися відрізки, поки середній показник кореляції зростав. Й друкувалася кореляція параметрів та їхні номери, починаючи відлік з нуля.

Лістинг 6: Пошук потенційних розв'язків

```
# In[8]:

alpha = 0.22
beta = 0.225
mean, save = corr_mean(alpha = 0.22 , beta = 0.225)
print(mean)
print(save)

>> 1.0
>> Int64Index([66, 3261], dtype='int64')

# In[9]:
```

```
alpha = 0.0115
beta = 0.0116
drop1 = save
mean, save1 = corr_mean(alpha , beta, drop1)
print(mean)
print(save1)

>> 0.9692503034215866
>> Int64Index([ 241,  315,  484,  559,  606,  737, 1405, 1514,
    1883, 2091, 2429,
            2697, 2720, 2747, 2966, 2997, 3293, 3300, 3455,
               3502, 4058, 4193,
            4343, 5274, 5335],
           dtype='int64')

# In[10]:

alpha = 0.0029
beta = 0.0032
drop2 = save | save1
mean, save2 = corr_mean(alpha , beta, drop2)
print(mean)
print(save2)

>> 0.9790097084253334
>> Int64Index([  19,   23,   82,   99,  105,  113,  138,  142,
     148,  150,
            ...
            4567, 4892, 4973, 5002, 5008, 5016, 5086, 5098,
               5190, 5314],
           dtype='int64', length=116)

# In[11]:
```

```
alpha = 0.0121
beta = 0.01225
drop3 = drop2 | save2
mean, save3 = corr_mean(alpha , beta, drop3)
print(mean)
print(save3)

>> 0.9903499303437886
>> Int64Index([ 575,  577, 1269, 1349, 1407, 1682, 1763, 2226,
     2592, 2984, 3429,
             3728, 3761, 3792, 4124, 4187, 4190, 4202, 4248,
                5348],
            dtype='int64')

# In[12]:

alpha = 0.01099
beta = 0.011
drop4 = drop3 | save3
mean, save4 = corr_mean(alpha , beta, drop4)
print(mean)
print(save4)

>> 0.9762724814621782
>> Int64Index([2755, 3477, 4770], dtype='int64')

# In[13]:

alpha = 0.01098
beta = 0.01099
drop5 = drop4 | save4
mean, save5 = corr_mean(alpha , beta, drop5)
print(mean)
print(save5)
```

```
>> 0.9432420789854078
>> Int64Index([1839, 2520, 2594], dtype='int64')

# In[14]:

alpha = 0.0109
beta = 0.01096
drop6 = drop5 | save5
mean, save6 = corr_mean(alpha , beta, drop6)
print(mean)
print(save6)

>> 0.9624046916662425
>> Int64Index([ 438,  530, 1342, 2185, 2421, 2764, 3462, 3948,
     4123, 4148, 4219,
            4358, 4415, 4801, 5293],
           dtype='int64')

# In[15]:

alpha = 0.01089
beta = 0.0109
drop7 = drop6 | save6
mean, save7 = corr_mean(alpha , beta, drop7)
print(mean)
print(save7)

>> 1.0
>> Int64Index([1383, 3693], dtype='int64')

# In[16]:

alpha = 0.01088
```

```
beta = 0.01089
drop8 = drop7 | save7
mean, save8 = corr_mean(alpha , beta, drop8)
print(mean)
print(save8)

>> 1.0
>> Int64Index([1701, 4589], dtype='int64')

# In[17]:

alpha = 0.01086
beta = 0.01087
drop9 = drop8 | save8
mean, save9 = corr_mean(alpha , beta, drop9)
print(mean)
print(save9)

>> 0.9968008237650818
>> Int64Index([1132, 2613, 4091], dtype='int64')

# In[18]:

alpha = 0.010851
beta = 0.010856
drop10 = drop9 | save9
mean, save10 = corr_mean(alpha , beta, drop10)
print(mean)
print(save10)

>> 1.0
>> Int64Index([433, 5196], dtype='int64')

# In[19]:
```

```
alpha = 0.01084
beta = 0.01085
drop11 = drop10 | save10
mean, save11 = corr_mean(alpha , beta, drop11)
print(mean)
print(save11)

>> 1.0
>> Int64Index([562, 2288], dtype='int64')

# In[20]:

alpha = 0.01083
beta = 0.01084
drop12 = drop11 | save11
mean, save12 = corr_mean(alpha , beta, drop12)
print(mean)
print(save12)

>> 1.0
>> Int64Index([3497, 3645], dtype='int64')

# In[21]:

alpha = 0.01082
beta = 0.01083
drop13 = drop12 | save12
mean, save13 = corr_mean(alpha , beta, drop13)
print(mean)
print(save13)

>> 0.9984817507816726
>> Int64Index([529, 3456, 4087], dtype='int64')
```

```
# In[22]:

alpha = 0.01081
beta = 0.01082
drop14 = drop13 | save13
mean, save14 = corr_mean(alpha , beta, drop14)
print(mean)
print(save14)

>> 0.9404479726015302
>> Int64Index([798, 3478, 5301], dtype='int64')

# In[23]:

alpha = 0.379
beta = 0.385
drop15 = drop14 | save14
mean, save15 = corr_mean(alpha , beta, drop15)
print(mean)
print(save15)

>> 1.0
>> Int64Index([1150, 1151], dtype='int64')

# In[24]:

alpha = 0.355
beta = 0.4
drop16 = drop15 | save15
mean, save16 = corr_mean(alpha , beta, drop16)
print(mean)
print(save16)
```

```
>> 1.0
>> Int64Index([1149, 1160], dtype='int64')
```

Об'єднавши всі такі параметри в єдину множину, ми за алгоритмом Звичайної Лінійної Регресії отримуємо розв'язок у вигляді (3.2). Також було перевірено, що кожен з 17 наборів потенційних розв'язків, також з використанням Звичайної Лінійної Регресії, дає самостійний розв'язок. Тобто кожен з наборів дає нам функцію, що відповідає умовам розв'язку класифікаційної задачі, разом з будь-якою їх комбінацією.

Лістинг 7: Побудування розв'язку

```
#Grouping all indexes:
res_for_s16_index = drop16 | save16

#Number of indexes:
len(drop16 | save16)

>>207

#Simple Linear Regression scheme:
X_dataset_for_s16 = dataset[drop16 | save16].drop_duplicates()
    .values
y = dataset[5408].values.reshape([len(dataset), 1])
X_dataset_for_s16 = np.hstack((y, X_dataset_for_s16))
I = np.linalg.inv(X_dataset_for_s16.T.dot(X_dataset_for_s16))
theta=I.dot(X_dataset_for_s16.T.dot(y))

#Solution check:
count = 0
X_dataset_for_s16 = dataset[drop16 | save16].values
y = dataset[5408].values.reshape([len(dataset), 1])
X_dataset_for_s16 = np.hstack((y, X_dataset_for_s16))
#Results obtained by linear regression:
test_out_theta_s16_dataset = X_dataset_for_s16.dot(theta)
```

```
for i in range(len(dataset)):
    if math.isclose(test_out_theta_s16_dataset[i][0], y[i][0],
        abs_tol=1e-12):
        count += 1
print('The length of all set is', len(dataset), \\
                'close guesses length set is', count)
print(pd.DataFrame(confusion_matrix(test_out_theta_s16_dataset
    .round(), y),\\
                   index=['inactive', 'active'], columns= ['
                       inactive', 'active']))

>> The length of all set is 4921 close guesses length set is
    4921
          inactive  active
inactive      4885       0
active           0      36
```

Матриця невідповідностей елементів демонструє нам, що цей алгоритм дав нам декілька повних розв'язків. Якщо говорити про ефективність, то цей алгоритм має значний недолік: відповіді у формі, в якій ми шукаємо, може й не бути; але якщо цей розв'язок існує, його пошук займає стільки ж часу, скільки й пошук зворотньої матриці для пошуку коефіцієнтів Звичайної Лінійної Регресії. Наразі для розв'язку подібного роду задач використовуються алгоритми Машинного Навчання (Machine Learning), але на прикладі цієї задачі через низку факторів, наприклад, нерівномірне представництво класів, отримані результати не дають потрібної точності, й пошук цих розв'язків займає набагато більше часу. Ця ж задача розглядалась в статті [4] з використанням алгоритмів Глибокого Навчання (Deep Learning). Перевірка за наданим у роботі алгоритмом на наявність відповідного розв'язку може звести багато часів роботи іте-

ративного алгоритму до виклику однієї функції, з найліпшою можливою відповіддю, як у випадку $p53Mutants$, знаходженням функції зі 100% точністю.

6. Необхідні для самостійного побудування розв'язку індекси

За потреби, з цієї секції можуть бути взяті індекси параметрів, на котрих проводилася лінійна регресія, відлік починається з нуля:
[19, 23, 66, 82, 99, 105, 113, 138, 142, 148, 150, 208, 225, 238, 241, 244, 248, 274, 297, 315, 336, 433, 438, 452, 466, 471, 475, 484, 495, 497, 529, 530, 559, 562, 567, 575, 577, 601, 606, 682, 707, 737, 770, 772, 798, 809, 824, 845, 893, 910, 927, 949, 957, 961, 989, 1031, 1063, 1132, 1149, 1150, 1151, 1160, 1229, 1269, 1342, 1349, 1380, 1383, 1405, 1407, 1420, 1421, 1431, 1451, 1460, 1486, 1514, 1519, 1538, 1540, 1545, 1547, 1682, 1701, 1702, 1763, 1789, 1796, 1839, 1883, 1905, 1937, 1939, 1988, 1993, 2014, 2053, 2067, 2081, 2089, 2091, 2104, 2105, 2132, 2185, 2226, 2254, 2288, 2293, 2311, 2313, 2348, 2351, 2354, 2358, 2388, 2411, 2421, 2429, 2520, 2568, 2592, 2594, 2601, 2613, 2625, 2697, 2714, 2720, 2747, 2755, 2764, 2829, 2864, 2940, 2966, 2984, 2997, 3007, 3233, 3261, 3293, 3300, 3301, 3336, 3348, 3365, 3429, 3455, 3456, 3462, 3477, 3478, 3491, 3497, 3502, 3645, 3693, 3727, 3728, 3761, 3792, 3886, 3944, 3948, 3949, 3985, 3996, 4058, 4087, 4091, 4123, 4124, 4133, 4148, 4187, 4190, 4193, 4202, 4219, 4248, 4278, 4343, 4358, 4408, 4415, 4426, 4544, 4567, 4589, 4770, 4801, 4892, 4973, 5002, 5008, 5016, 5086, 5098, 5190, 5196, 5274, 5293, 5301, 5314, 5335, 5348]

7. Висновок

Сформовано алгоритм пошуку розв'язків задачі класифікації, заданої у вигляді матриці параметрів, заданих дійсними числами, й параметром класу з цільовою функцією у вигляді степеневого ряду з двома доданками залежними від параметрів й константою. Надана комп'ютерна реалізація алгоритму й її результати, які вирішують задачу $p53Mutants$.

Література

[1] Danziger, S.A., Baronio, R., Ho, L., Hall, L., Salmon, K., Hatfield, G.W., Kaiser, P., and Lathrop, R.H. (2009) Predicting Positive p53 Cancer Rescue Regions Using Most Informative Positive (MIP) Active Learning, PLOS Computational Biology, 5(9), e1000498

[2] Danziger, S.A., Zeng, J., Wang, Y., Brachmann, R.K. and Lathrop, R.H. (2007) Choosing where to look next in a mutation sequence space: Active Learning of informative p53 cancer rescue mutants, Bioinformatics, 23(13), 104-114.

[3] Danziger, S.A., Swamidass, S.J., Zeng, J., Dearth, L.R., Lu, Q., Chen, J.H., Cheng, J., Hoang, V.P., Saigo, H., Luo, R., Baldi, P., Brachmann, R.K. and Lathrop, R.H. (2006) Functional census of mutation sequence spaces: the example of p53 cancer rescue mutants, IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM, 3, 114-125.

[4] Dima S., Malek A., Wael E., Ghazi A.; Dima Suleiman et al. Empirical Evaluation of the Classification of Deep Learning under Big Data Processing Platforms, International Journal of Advanced Trends in Computer Science and Engineering, 9(5),September-October 2020, 9189-9196, ISSN 2278-3091.

[5] Richard https://archive.ics.uci.edu/ml/datasets/p53+Mutants , Richard H. Lathrop, UC Irvine, http://www.ics.uci.edu/ rickl