****

**Московский авиационный институт**

**(Национальный исследовательский университет)**

**Институт № 3**

**Кафедра 311**

**Интеллектуальные системы**

**Лабораторная работа № 5**

**«Алгоритм с ковариациями и с квадратичными описаниями»**

**Выполнил студент  
Веденеев Максим Кириллович**

**Группа М3З-501-БК**

**Дата 06.10.2023 г.**

**Принял преподаватель  
Кос Оксана Игоревна**

Оглавление

[Цель лабораторной работы 3](#__RefHeading___1)

[Глава 1. Общая схема построения алгоритмов метода группового учета аргументов (МГУА) 4](#__RefHeading___2)

[Глава 2. Реализация 6](#__RefHeading___8)

[Итоги лабораторной работы 20](#__RefHeading___7)

[Список литературы 21](#__RefHeading___6)

# Цель лабораторной работы

Изучить:

1. Алгоритм с ковариациями и с квадратичными описаниями.
2. Реализовать его.

# Глава 1. Общая схема построения алгоритмов метода группового учета аргументов (МГУА)

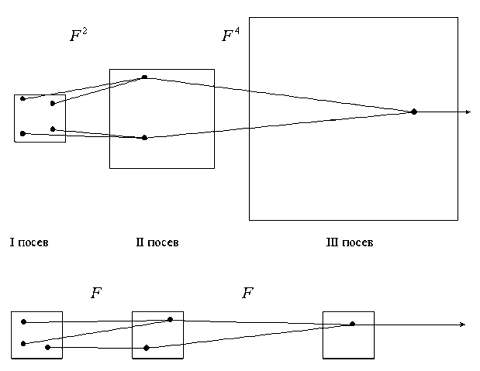


Рисунок 1. Общая схема построения алгоритмов метода группового учета аргументов

Заимствование алгоритмов переработки информации у природы является одной из основных идей кибернетики. "Гипотеза селекции" утверждает, что алгоритм массовой селекции растений или животных является оптимальным алгоритмом переработки информации в сложных задачах. При массовой селекции высевается некоторое количество семян. В результате опыления образуются сложные наследственные комбинации. Селекционеры выбирают некоторую часть растений, у которых интересующее их свойство выражено больше всего (эвристический критерий). Семена этих растений собирают и снова высевают для образования новых, еще более сложных комбинаций. Через несколько поколений селекция останавливается и ее результат является оптимальным. Если чрезмерно продолжать селекцию, то наступит "инцухт" — вырождение растений. Существует оптимальное число поколений и оптимальное количество семян, отбираемых в каждом из них.

Алгоритмы МГУА воспроизводят схему массовой селекции, показанной на Рис. 1. В них есть генераторы усложняющихся из ряда в ряд комбинаций и пороговые самоотборы лучших из них.

Так называемое “полное” описание объекта: j = f(x1,x2,x3,...,xm),

где f — некоторая элементарная функция, например степенной полином, заменяется несколькими рядами "частных" описаний:

1-ряд селекции: y1= f(x1x2), y2= f(x1x3),..., ys= f(xm-1xm),

2-ряд селекции: z1= f(y1y2), z2= f(y1y2),..., zp= f(ys-1ys), где s=c2, p=cs2 и т.д.

Входные аргументы и промежуточные переменные сопрягаются попарно, и сложность комбинаций на каждом ряду обработки информации возрастает (как при массовой селекции), пока не будет получена единственная модель оптимальной сложности.

# Глава 2. Реализация

import numpy

class Model:

def \_\_init\_\_(self, id, input1, input2, weights):

self.id = id

self.input1 = input1

self.input2 = input2

self.weights = weights

self.CR = 0.0

self.isBest = False

def f(self, xi, xj):

return self.weights[0] + self.weights[1] \* xi + self.weights[2] \* xj + self.weights[3] \* xi \* xj

class Row:

def \_\_init\_\_(self, num\_of\_models):

self.L = num\_of\_models

self.models = []

self.CR = 0.0

def countCR(self):

# ошибка ряда = min\_error

self.CR = self.models[0].CR

for model in self.models:

if self.CR > model.CR: self.CR = model.CR

class GMDH:

def \_\_init\_\_(self, m, N, NA, input\_data, y0):

self.m = m # число признаков

self.N = N # число образцов выборки (N = NA + NB)

self.NA = NA

self.NB = self.N - self.NA

self.input\_data = input\_data # таблица входных данных - [N][m]

self.y0 = y0

self.percentage = 0.4 # процент отбираемых "лучших" моделей

self.rows = []

def training(self):

# формирование рядов

F = self.m

prev\_error = 10\*\*8 # очень большое число

inputs = self.input\_data

outputs = []

row\_number = 0

while row\_number < 100:

row\_number += 1

print("Row number", row\_number)

L = self.comb2(F) # число моделей текущего ряда = C(F, 2)

row = Row(L) # создаем новый ряд

combinations = [] # (model\_num, i, j)

model\_num = 0

for i in range(0, F):

for j in range(i+1, F):

combinations.append((model\_num, i, j))

model\_num += 1

# смотрим на обучающую выборку и вычисляем весовые коэффициенты

for model\_num, i, j in combinations:

# по всем сочетаниям i и j

A\_matr = [

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0]

]

b\_matr = [0.0, 0.0, 0.0, 0.0]

for k in range(0, self.NA):

xi = inputs[k][i]

xj = inputs[k][j]

A\_matr[0][0] += 1.0

A\_matr[1][0] += xi

A\_matr[2][0] += xj

A\_matr[3][0] += xi \* xj

A\_matr[0][1] += xi

A\_matr[1][1] += xi\*\*2

A\_matr[2][1] += xi \* xj

A\_matr[3][1] += xi\*\*2 \* xj

A\_matr[0][2] += xj

A\_matr[1][2] += xi \* xj

A\_matr[2][2] += xj\*\*2

A\_matr[3][2] += xi \* xj\*\*2

A\_matr[0][3] += xi \* xj

A\_matr[1][3] += xi\*\*2 \* xj

A\_matr[2][3] += xi \* xj\*\*2

A\_matr[3][3] += xi\*\*2 \* xj\*\*2

b\_matr[0] += self.y0[k]

b\_matr[1] += self.y0[k] \* xi

b\_matr[2] += self.y0[k] \* xj

b\_matr[3] += self.y0[k] \* xi \* xj

weights = list(numpy.linalg.solve(A\_matr, b\_matr))

# добавляем модель в ряд

row.models.append(Model(model\_num, i, j, weights))

# вычисление ошибки для каждой модели

for model\_num, i, j in combinations:

row.models[model\_num].CR = 0.0

# по всем образцам из проверочной выборки

for k in range(self.NA, self.N):

xi = inputs[k][i]

xj = inputs[k][j]

row.models[model\_num].CR += (row.models[model\_num].f(xi, xj) - self.y0[k])\*\*2

#print 'f(xi, xj)', row.models[model\_num].f(xi, xj)

row.models[model\_num].CR /= self.NB

print('row.models[model\_num].CR', model\_num, row.models[model\_num].CR)

print("models before sorting:")

for model in row.models:

print(model.id, model.input1, model.input2, model.CR)

row.countCR() # вычислить ошибку ряда

# сортируем модели по возрастанию ошибки

changes = True

while changes:

changes = False

for l in range(L-1):

if row.models[l].CR > row.models[l+1].CR:

temp\_model = row.models[l]

row.models[l] = row.models[l+1]

row.models[l+1] = temp\_model;

changes = True

print("models after sorting:")

for model in row.models:

print(model.id, model.input1, model.input2, model.CR)

F = int(self.percentage \* L) # число отбираемых лучших моделей

print("L = ", L)

print("F = ", F)

# формируем вектор выходов

outputs = []

for k in range(self.N):

output = []

for l in range(F):

xi = inputs[k][row.models[l].input1]

xj = inputs[k][row.models[l].input2]

output.append(row.models[l].f(xi, xj))

outputs.append(output)

inputs = outputs

if prev\_error < row.CR:

print("overfitting")

print(row\_number, row.CR)

break

prev\_error = row.CR

self.rows.append(row)

if F <= 1:

print("no more models")

print(row\_number, row.CR)

break

# число сочетаний из n по 2 = C(n, 2)

def comb2(self, n):

if n == 0 or n == 1: return 0

comb = 0

for i in range(0, n):

for j in range(i+1, n):

comb += 1

return comb

def goBack(self):

print("### Go Back ###")

num\_of\_rows = len(self.rows)

print(num\_of\_rows)

reversed\_rows = range(num\_of\_rows)

def reversed\_rows(s):

return s[::-1]

print(reversed\_rows)

# mark models as best

best\_models = [0]

for i in range(1, num\_of\_rows+1):

new\_best = []

for j in best\_models:

new\_best.append(self.rows[-i].models[j].input1)

new\_best.append(self.rows[-i].models[j].input2)

self.rows[-i].models[j].isBest = True

best\_models = new\_best

def printGMDH(self):

print("### GMDH ###")

r = 1

for row in self.rows:

print("row ", r)

for model in row.models:

print(" ", "id =", model.id, "in1 = ", model.input1, "in2 = ", model.input2, model.isBest)

r += 1

def testGMDH(self, x):

if self.m != len(x):

print("Error! m != len(x):")

return

input = x

# for i in xrange(self.m):

# input[i] = x[i]

F = self.m

L = self.comb2(F)

combinations = [] # (model\_num, i, j)

model\_num = 0

for i in range(0, F):

for j in range(i+1, F):

combinations.append((model\_num, i, j))

model\_num += 1

# формирование рядов

F = self.m

inputs = x

outputs = []

for row in self.rows:

L = self.comb2(F) # число моделей текущего ряда = C(F, 2)

combinations = [] # (model\_num, i, j)

model\_num = 0

for i in range(0, F):

for j in range(i+1, F):

combinations.append((model\_num, i, j))

model\_num += 1

# формируем вектор выходов

outputs = []

for l in range(F):

xi = inputs[row.models[l].input1]

xj = inputs[row.models[l].input2]

outputs.append(row.models[l].f(xi, xj))

inputs = outputs

#print inputs

return inputs[0]

Тест:

import unittest

import os

from gmdh import GMDH

PROJECT\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

class Tests(unittest.TestCase):

def test\_gmdh(self):

n\_lines = 0

m = 0

N = 0

NA = 0

x = []

y = []

for line in open(os.path.join(PROJECT\_ROOT, 'datafile.txt'), 'r').readlines():

if n\_lines == 0:

N, m, NA = line.rstrip('\n').split(' ')

m = int(m)

N = int(N)

NA = int(NA)

n\_lines += 1

continue

items = line.rstrip('\n').split(' ')

class\_name = items[0]

if class\_name == 'High':

y.append(1)

elif class\_name == 'Normal':

y.append(0)

vect = items[1:]

# make it float

for i in range(len(vect)):

vect[i] = float(vect[i])

x.append(vect)

n\_lines += 1

gmdh = GMDH(m, N, NA, x, y)

gmdh.training()

gmdh.goBack()

gmdh.printGMDH()

print("Normal: ", gmdh.testGMDH([0.279, 0.856, 0.350, 0.870, 3234.801, 114920.842]))

print("High: ", gmdh.testGMDH([0.294, 2.828, 0.125, 0.205, 12712.210, 102701.833]))

print("High: ", gmdh.testGMDH([2.944, 1.905, 0.650, 0.688, 22068.982, 180123.185]))

print("### TEST ###")

for line in open(os.path.join(PROJECT\_ROOT, 'datafile\_all.txt'), 'r').readlines():

items = line.rstrip('\n').split(' ')

class\_name = items[0]

vect = items[1:]

# make it float

for i in range(len(vect)):

vect[i] = float(vect[i])

res = gmdh.testGMDH(vect)

print(class\_name, ":", res, " ->", round(res))

import unittest

import os

from lr\_5.gmdh import GMDH

PROJECT\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

class Tests(unittest.TestCase):

def test\_gmdh(self):

n\_lines = 0

m = 0

N = 0

NA = 0

x = []

y = []

for line in open(os.path.join(PROJECT\_ROOT, 'datafile.txt'), 'r').readlines():

if n\_lines == 0:

N, m, NA = line.rstrip('\n').split(' ')

m = int(m)

N = int(N)

NA = int(NA)

n\_lines += 1

continue

items = line.rstrip('\n').split(' ')

class\_name = items[0]

if class\_name == 'High':

y.append(1)

elif class\_name == 'Normal':

y.append(0)

vect = items[1:]

# make it float

for i in range(len(vect)):

vect[i] = float(vect[i])

x.append(vect)

n\_lines += 1

gmdh = GMDH(m, N, NA, x, y)

gmdh.training()

gmdh.goBack()

gmdh.printGMDH()

print("Normal: ", gmdh.testGMDH([0.279, 0.856, 0.350, 0.870, 3234.801, 114920.842]))

print("High: ", gmdh.testGMDH([0.294, 2.828, 0.125, 0.205, 12712.210, 102701.833]))

print("High: ", gmdh.testGMDH([2.944, 1.905, 0.650, 0.688, 22068.982, 180123.185]))

print("### TEST ###")

for line in open(os.path.join(PROJECT\_ROOT, 'datafile\_all.txt'), 'r').readlines():

items = line.rstrip('\n').split(' ')

class\_name = items[0]

vect = items[1:]

# make it float

for i in range(len(vect)):

vect[i] = float(vect[i])

res = gmdh.testGMDH(vect)

print(class\_name, ":", res, " ->", round(res))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

# Итоги лабораторной работы

Мы научились писать код для алгоритма с ковариациями и квадратичными отклонениями.

# Список литературы

1. ГОСТ Р 7.0.11-2011. Система стандартов по информации, библиотечному и издательскому делу. Диссертация и автореферат диссертации. Структура и правила оформления. - 19 c.
2. https://studfile.net/preview/7470285/page:16/