LAPORAN OBSERVASI TUGAS PARALEL 2 : DECISION TREE MENGGUNAKAN ALGORITMA GENETIKA

CLARISA HASYA YUTIKA | 1301174256 | IF 41 02

1. **Deskripsi Masalah**
   Decision tree adalah model prediksi menggunakan struktur pohon atau struktur berhirarki. Untuk membuat decision tree adalah menggunakan genetic algorithm untuk mengklasifikasi data dalam jumlah besar menjadi data yang lebih simple dan dibentuknya suatu aturan.
2. **Objektif yang dibangun**
   Membangun system klasifikasi decision tree menggunakan algoritma genetika untuk beberapa data uji
3. **Analisis Algoritma**
   - Rule
     Membuat list rule dengan panjang 5bit dengan bilangan integer random antara 0 sampai 3 untuk atribut dengan 3 kemungkinan, untuk bilangan integer random antara 0 sampai 4 untuk atribut dengan 4 kemungkinan, untuk bilangan integer random antara 0 sampai 1 untuk atribut terbang atau tidak.

```
def rule():
    rule = []
    rule.append(random.randint(0,3))  #suhu
    rule.append(random.randint(0,4))  #waktu
    rule.append(random.randint(0,4))  #kondisi langit
    rule.append(random.randint(0,3))  #kelembapan
    rule.append(random.randint(0,1))  #terbang/tidak
    return rule
```

   - Kromosom
     Membuat list kromosom dengan isi rule antara 2 sampai 3 rule

```
def chromosome():  #bikin chromosome dari rule dengan jumlah rule random dari 2 sampai 3
    chr = []
    for i in range(random.randint(2,3)):
        chr.append(rule())
    return chr
```

   - Populasi
     Membuat list populasi dengan isi kromosom sebanyak random dari 5 sampai 20 kromosom

```
def population():
    pop = []
    for i in range(random.randint(5,20)):
        pop.append(chromosome())
    return pop
```

   - Data Train
     Membaca 80 datatrain dari file data_latih_opsi_2.csv, dan disimpan di list

```
def loadData(file):
    data = []
    with open(file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count=0
        for row in csv_reader:
            data.append(row)
    return data
```

   - Fitness
     Untuk mendapatkan nilai fitness adalah membandingkan rule dari setiap kromosom dengan 80 rule di datatrain. Setiap rule yang sama nilai fitness ditambah satu, kemudian hasil jumlah nya dibagi 80.

```
def compare(rule,data):  #membandingkan populasi dengan datatrain
    suhu, waktu, langit, lembab, terbang = False, False, False, False, False
    if (rule[0]==int(data[0]) or rule[0]==3):
        suhu = True
    if (rule[1]==int(data[1]) or rule[1]==4):
        waktu = True
    if (rule[2]==int(data[2]) or rule[2]==4):
        langit = True
    if (rule[3]==int(data[3]) or rule[3]==3):
        lembab = True
    if (rule[4]==int(data[4])):
        terbang = True
    if suhu and waktu and langit and lembab and terbang:
        com = True
    else:
        com = False
    return com

def fitness(chr):
    fit = 0
    for rule in chr:
        for data in loadData('data_latih_opsi_2.csv'):
            # print('rule',rule)
            # print('data',data)
            if compare(rule,data):
                fit += 1
    return fit/80
```

   - Pemilihan Orang Tua
     Dalam pemilihan orang tua dapat menggunakan beberapa cara, salah satunya Roulette Wheel Selection.

```
def RouletteWheelSelection(pop,fit,total):
    r = random.random()
    i = 0
    # print("random",r)
    while (r>0):
        r -= fit[i]/total
        i += 1
        if (i == (len(pop)-1)): #berhentiin loop kalo udah sampe batas populasi
            break
    parent = pop[i]
    return parent
```

   - Crossover
     Hasil dari pemilihan orang tua kemudian di crossover dengan probabilitas 0,7. Kemudian dipilih random untuk menggunakan crossover increase-decrease atau increase. Kemudian jika terpilih crossover increase-decrease akan dipilih random lagi dengan titik potong yang berbeda, pada increase-

decrease isi dari beberapa array parent 1 akan dipindahkan ke parent 2. Jika terpilih crossover increase akan dipilih random lagi dengan titik potong yang berbeda, pada increase isi dari beberapa array parent 1 akan disalin ke parent 2.

```
def crossoverBasic(parent1,parent2,point1,point2):
    cross=[]
    cross1 = parent1[:point1] + parent2[point1:point2] + parent1[point2:]
    cross2 = parent2[:point1] + parent1[point1:point2] + parent2[point2:]
    cross.append(cross1)
    cross.append(cross2)
    return cross
```

```
def crossoverIncrease(parent1,parent2,point1,point2,point3,point4):
    cross=[]
    prob = random.randint(0,2)
    print('increase',prob)
    if prob==0 or (point2==point3 and point1==point4):
        cross = crossoverBasic(parent1,parent2,point1,point2)
    elif prob==1 and point2!=point3:
        cross1 = parent1[:point1] + parent2[point1:point3] + parent1[point3:]
        cross2 = parent2[:point1] + parent1[point1:point2] + parent2[point1:]
        cross.append(cross1)
        cross.append(cross2)
    elif prob==2 and point1!=point4:
        cross1 = parent1[:point4] + parent2[point4:point2] + parent1[point2:]
        cross2 = parent2[:point4] + parent1[point1:point2] + parent2[point2:]
        cross.append(cross1)
        cross.append(cross2)
    else:
        cross.append(cross1)
        cross.append(cross2)
    return cross
```

```
def crossoverIncreaseDecrease(parent1,parent2,point1,point2,point3,point4):
    cross=[]
    prob = random.randint(0,2)
    print('increase-decrease',prob)
    if prob==0 or (point2==point3 and point1==point4):
        cross = crossoverBasic(parent1,parent2,point1,point2)
    elif (prob==1 and point2!=point3):
        cross1 = parent1[:point1] + parent2[point1:point3] + parent1[point2:]
        cross2 = parent2[:point1] + parent1[point1:point2] + parent2[point1:]
        cross.append(cross1)
        cross.append(cross2)
    elif prob==2 and point1!=point4:
        cross1 = parent1[:point4] + parent2[point4:point2] + parent1[point2:]
        cross2 = parent2[:point1] + parent1[point1:point2] + parent2[point2:]
        cross.append(cross1)
        cross.append(cross2)
    else:
        cross.append(cross1)
        cross.append(cross2)
    return cross
```

```
def crossover(parent1,parent2):
    cross1, cross2 = [], []
    cross = []
    prob = random.random()
    print('prob',prob)
    if (prob < 0.9):
        if len(parent1) >= len(parent2):
            minim = len(parent2)
        else:
            minim = len(parent1)
        print('p1',len(parent1))
        print('p2',len(parent2))
        print('minim',minim)
        point1 = random.randint(0,minim//2)
        point2 = random.randint((minim//2)+1,minim)
        selisih = point2-point1
        mod = selisih % 5
        point3 = point1 + mod #untuk point kedua
        point4 = point2 - mod #untuk point pertama
        print('p1',point1,'p2',point2,'p3',point3,'p4',point4)
        prob1 = random.randint(0,1)
        if prob1==0:
            cross = crossoverIncrease(parent1,parent2,point1,point2,point3,point4)
            print('increase')
        elif prob1==1:
            cross = crossoverIncreaseDecrease(parent1,parent2,point1,point2,point3,point4)
            print('increase-decrease')
    else:
        cross.append(parent1)
        cross.append(parent2)
    return cross
```

- Mutasi
  Hasil dari crossover kemudian di mutasi dengan probabilitas 0.2, kemudian akan di cek per allel dengan probabilitas 0.1 untuk nilai nya diganti dengan bilangan integer random antara 0 sampai 3 untuk atribut dengan 3 kemungkinan, untuk bilangan integer random antara 0 sampai 4 untuk atribut dengan 4 kemungkinan, untuk bilangan integer random antara 0 sampai 1 untuk atribut terbang atau tidak

```
def mutation(cross1,cross2):
    prob = random.random()
    if (prob < 0.9):
        for i in range (len(cross1)):
            p = random.random()
            if (p < 0.1):
                if (i % 5 == 0):
                    cross1[i] = random.randint(0,3)
                elif (i % 5 == 1):
                    cross1[i] = random.randint(0,4)
                elif (i % 5 == 2):
                    cross1[i] = random.randint(0,4)
                elif (i % 5 == 3):
                    cross1[i] = random.randint(0,3)
                elif (i % 5 == 4):
                    cross1[i] = random.randint(0,1)
```

- Pergantian Generasi
  Untuk mendapatkan generasi yang bagus, dilakukan perulangan sebanyak 100 kali
- Kromosom Terbaik
  Untuk mendapatkan kromosom terbaik dari semua generasi yang didapatkan adalah membandingkan nilai fitness paling besar dari seluruh kromosom
- Target Uji
  Untuk mendapatan target uji adalah dengan membandingan setiap atribut dalam rule dengan atribut setiap rule dari kromosom terbaik. Ketika sama, dimasukkan isi atribut terbang/tidak dari kromosom terbaik ke dalam file csv

```
def validation(chr):
    fit = 0
    target=[]
    for rule in chr:
        for data in loadData('data_uji_opsi_2.csv'):
            terbang = getTerbang(rule,data)
            target.append(terbang)
    return target
```

4. **Kesimpulan**
   Kesimpulnnya adalah genetic algorithm adalah salah satu metode klasifikasi decision tree, semakin banyak generasi yang dibuat, semakin banyak rule dalam satu kromosom, semakin bagus hasil decision tree nya