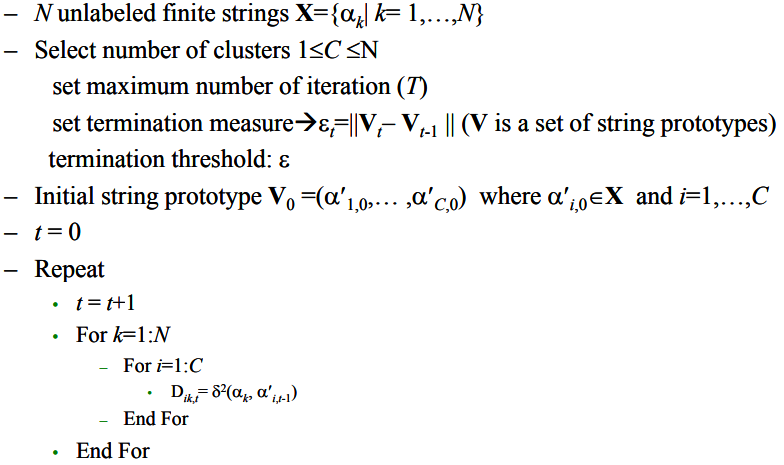
**Computer Assignment 2.1**

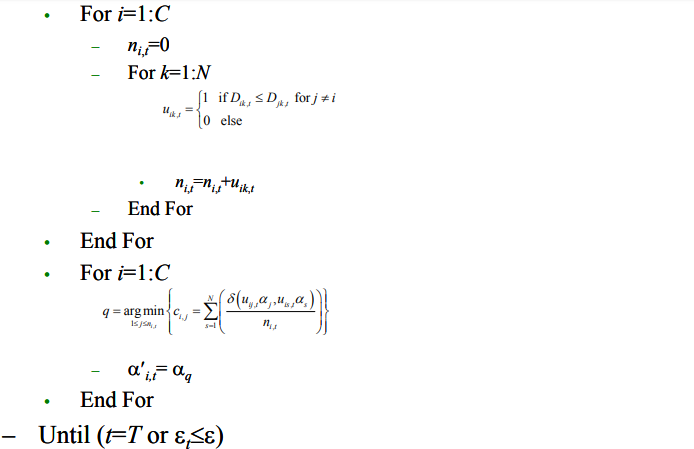
ธนกิจ เหล่ายาวิระ600632014

**1.รายละเอียดของทฤษฎีหรือวิธีการต่าง ๆ ที่ใช้**

1. Levenshtein Distance คือ ขั้นตอนวิธีการนี้จะเป็นการนำชุดอักขระ 2 ชุด มาเปรียบเทียบจำนวนความแตกต่างกัน โดยจะพิจารณาดังนี้  
- การแทรก เป็นการนำเอาอักขระตัวใด ๆ มา เพื่อให้ชุดอักขระชุดนั้นเหมือนกับอีกชุดอักขระหนึ่งในภายหลัง  
- การตัดออก เป็นการตัดอักขระออกครั้งละ 1 ตัว จากชุดอักขระตัวหนึ่ง เพื่อให้ชุดอักขระชุดนั้นเหมือนกับอีกชุดอักขระหนึ่งในภายหลัง  
- การแทนที่ เป็นการนำอักขระของชุดอักขระหนึ่งไปแทนอักขระของอีกชุดอักขระหนึ่ง เพื่อให้ชุดอักขระชุดนั้นเหมือนกับอีกชุดอักขระหนึ่งในภายหลัง

2. ใช้ NodeJS ในการพัฒนาซอฟท์แวร์

**2. Algorithm**



**3. ผลการทดลอง**

เมื่อทดลองกับ KNN k = 4 กับ Bayes Classifier 4 feature

**4. สรุปผลการทดลอง** เมื่อใช้จำนวน K และ จำนวน feature เท่ากัน ผลของการเลือกคลาสจะเหมือนกัน

**ภาคผนวก**

Code: String Grammar Hard C-Means (NodeJS)

let fs = require('fs')

const levenshtein = require('js-levenshtein');

const cNum = 3 //Change number of clusters here

const maxT = 100

let e = 0.5

const fetchData = new Promise((resolve) => {

let res = []

for (let i = 1; i <= 22; i++) {

let input = fs.readFileSync(`chrom/dif${i}da`, 'utf8').trim().split('\r\n').map(x => x.split('\t'))

res = res.concat(input)

}

resolve(res)

})

const main = async () => {

let prototype = []

let p1 = null

let p2 = null

let v1 = 0

let v2 = 0

let cluster = []

let eCal = 99999

let sourceData = await fetchData.then((values) => {

return values.map((value) => value[1])

})

// random choose prototype

for (let i = 0; i < cNum; i++) {

let rand = Math.floor(Math.random() \* sourceData.length)

prototype.push(rand)

cluster.push([])

}

p1 = prototype

for (let t = 0; t < maxT; t++) {

cluster = []

for (let i = 0; i < cNum; i++) {

cluster.push([])

}

for (let n = 0; n < sourceData.length; n++) {

let minDist = 999999

let chooseC = null

for (let i = 0; i < cNum; i++) {

let dist = levenshtein(sourceData[n], sourceData[prototype[i]])

if (dist < minDist) {

chooseC = i

minDist = dist

}

}

cluster[chooseC].push(n)

}

console.log(`round ${t + 1}`)

prototype = []

let minDist = 999999

let chooseV = null

let sumDist = 0

let cen = 0

for (let i = 0; i < cNum; i++) {

minDist = 999999

chooseV = null

for (let j = 0; j < cluster[i].length; j++) {

sumDist = 0

for (let k = 0; k < cluster[i].length; k++) {

if (j != k) {

let dist = levenshtein(sourceData[cluster[i][j]], sourceData[cluster[i][k]])

sumDist = sumDist + dist

}

}

cen = sumDist / cluster[i].length

if (cen < minDist) {

chooseV = cluster[i][j]

minDist = cen

}

}

prototype.push(chooseV)

}

if (t === 0) {

p2 = prototype

}

else if (t === 1) {

p1 = p2

p2 = prototype

}

else {

p1 = p2

p2 = prototype

}

console.log(`prototype 1: ${p1} prototype 2: ${p2}`)

if (t > 0) {

let sumPDist = 0

for (let i = 0; i < cNum; i++) {

let dist = levenshtein(sourceData[p2[i]], sourceData[p1[i]])

sumPDist = sumPDist + Math.pow(dist, 2)

}

if (t === 0) {

v1 = 0

v2 = Math.sqrt(sumPDist)

}

else if (t === 1) {

v1 = v2

v2 = Math.sqrt(sumPDist)

eCal = Math.abs(v1 - v2)

console.log(`v1 ${v1} v2 ${v2}`)

console.log(`e ${eCal}`)

}

else {

v1 = v2

v2 = Math.sqrt(sumPDist)

eCal = Math.abs(v1 - v2)

console.log(`v1 ${v1} v2 ${v2}`)

console.log(`e ${eCal}`)

}

}

console.log(`cluster 1: ${cluster[0].length} cluster 2: ${cluster[1].length} cluster 3: ${cluster[2].length}`)

console.log('cluster all', cluster[0].length + cluster[1].length + cluster[2].length, '\n')

if (t > 0 && eCal <= e) break

}

}

main()