**Computer Assignment 2.1**

ธนกิจ เหล่ายาวิระ600632014

**1.รายละเอียดของทฤษฎีหรือวิธีการต่าง ๆ ที่ใช้**

1.  ;Euclidean Distant

2. ใช้ NodeJS ในการพัฒนาซอฟท์แวร์

**2.Flowchart**

No

Start

แบ่งข้อมูลตาม cross validation

คำนวณค่าระยะทาง

สร้าง Confusion matrix

ถ้าเป็นข้อมูล validate สุดท้าย

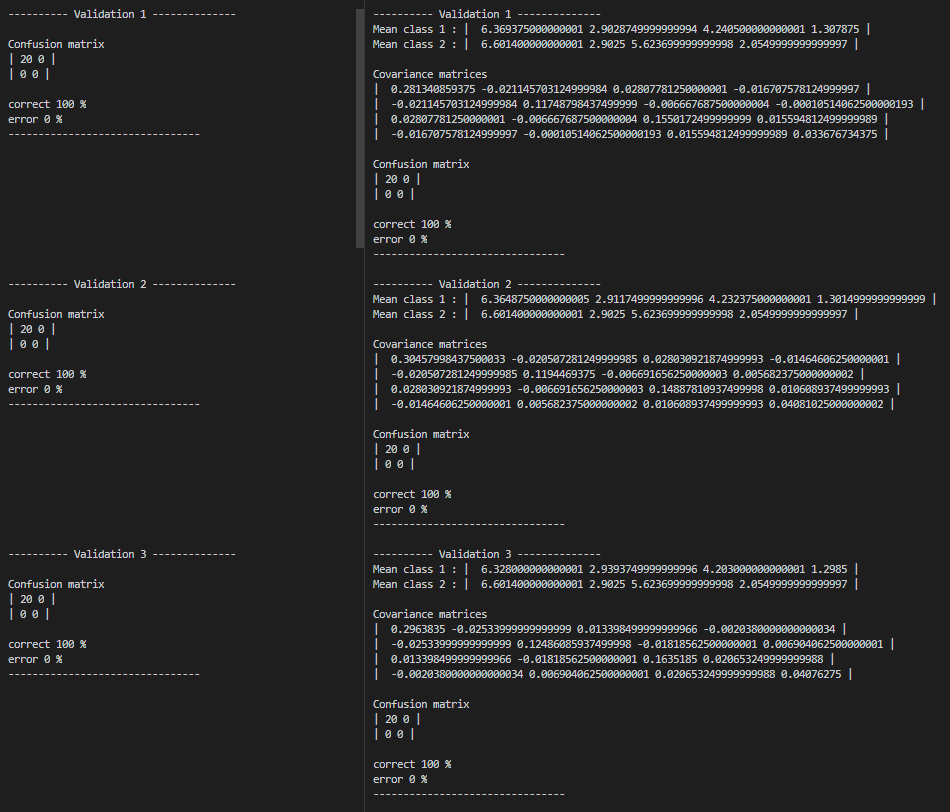
yes

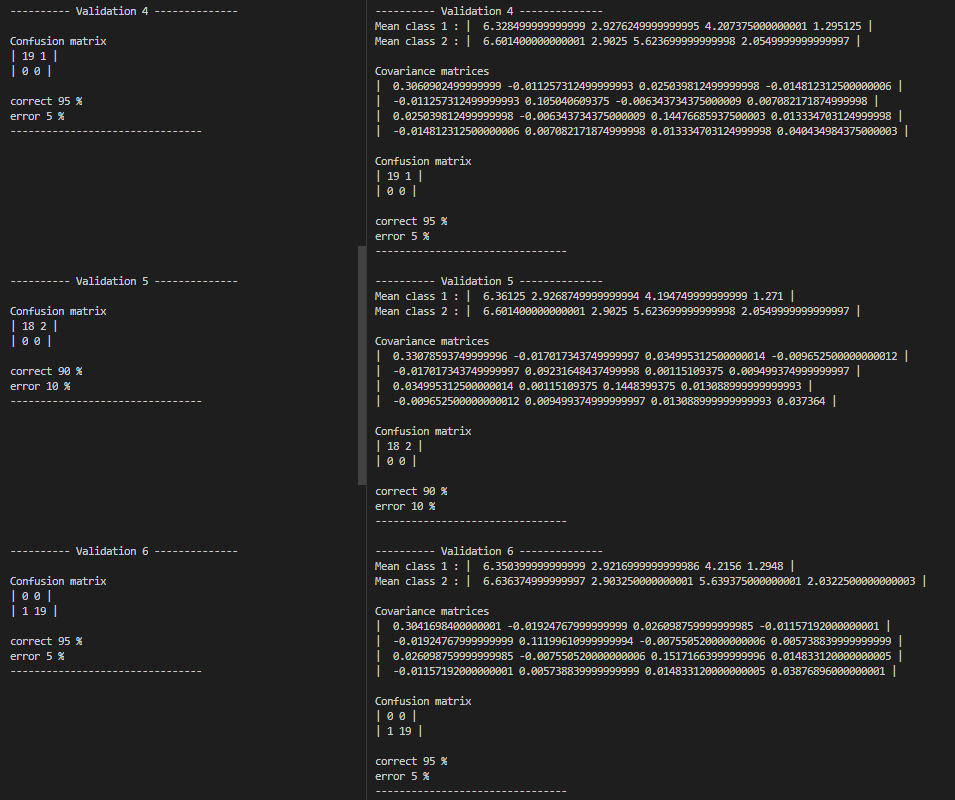
เปรียบเทียบผล

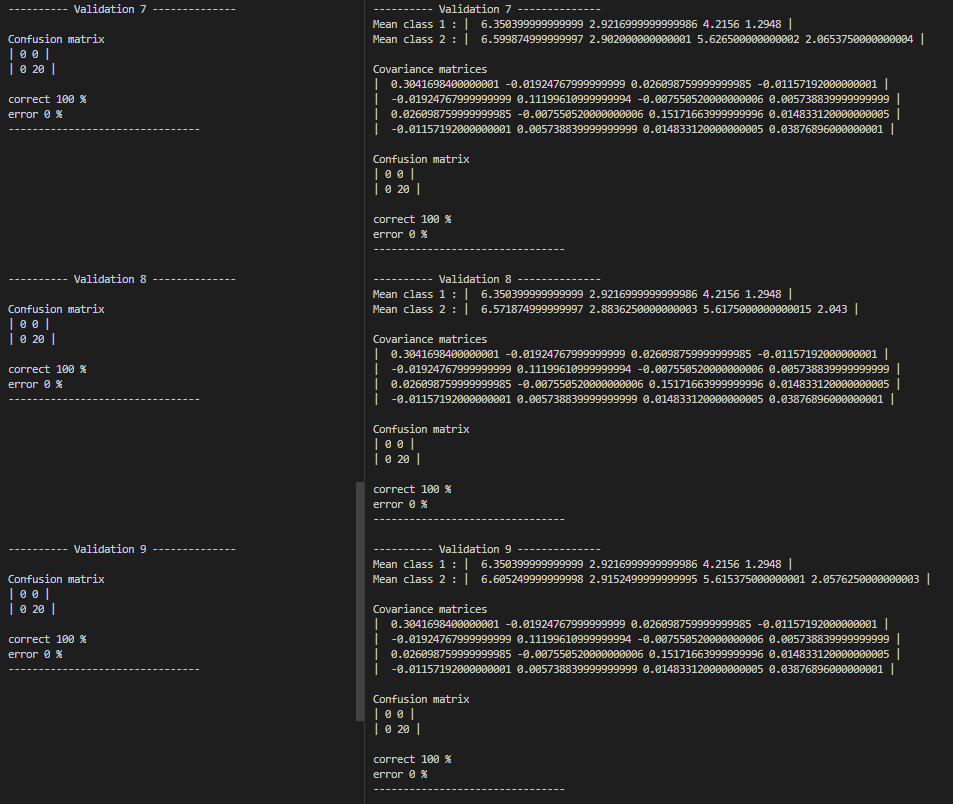
End

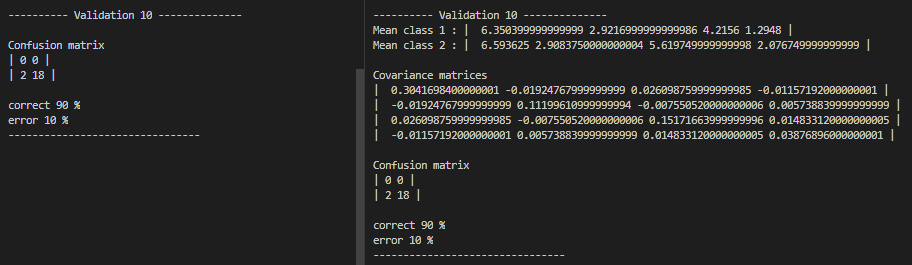
**3. ผลการทดลอง**

เมื่อทดลองกับ KNN k = 4 กับ Bayes Classifier 4 feature









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

**ภาคผนวก**

Code: KNN (NodeJS)

let fs = require('fs')

const math = require('mathjs')

let input = fs.readFileSync('TWOCLASS.dat', 'utf8')

const fNum = 4 //Change number of features here

const percentValidate = 10

const kValue = 14

const fetchData = Promise.resolve(

input.trim().split('\r\n').map(x => x.split('\t'))

)

const setUpTestData = (percentValidate, data) => {

const testDataNum = data.length / percentValidate

const round = data.length / testDataNum

let testDatas = []

for (i = 0; i < round; i++) {

testDatas.push(data.slice((i \* testDataNum), (i \* testDataNum + testDataNum)))

}

return testDatas

}

const setUpTrainData = (percentValidate, data) => {

const testDataNum = data.length / percentValidate

const round = data.length / testDataNum

let trainDatas = []

for (i = 0; i < round; i++) {

let trainData = []

for (j = 0; j < data.length; j++) {

if (j < (i \* testDataNum) || j >= (i \* testDataNum + testDataNum)) {

trainData.push(data[j])

}

}

trainDatas.push(trainData)

}

return trainDatas

}

const calDistance = (sources, destinations) => {

return destinations.map(

(destination) => {

let res = []

for (let j = 0; j < (destination.length - 1); j++) {

res.push(

Math.pow(

(parseFloat(sources[j]) - parseFloat(destination[j]))

, 2)

)

}

let resSq = Math.sqrt(res.reduce((prev, curr) => prev + curr))

res = []

res.push(resSq, destination[4])

return res

}

)

}

const main = async () => {

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

value.shift()

return value

})

const testDatas = setUpTestData(percentValidate, sourceData)

const trainDatas = setUpTrainData(percentValidate, sourceData)

for (let i = 0; i < 10; i++) {

let a = 0, b = 0, c = 0, d = 0

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

let dist = await calDistance(testDatas[i][j], trainDatas[i])

const distSorteds = dist.sort((a, b) => a[0] - b[0])

const distSliceds = distSorteds.slice(0, kValue)

const classes = [0, 0]

distSliceds.forEach(distSliced => {

if (distSliced[1] === '1') { classes[0]++ }

else if (distSliced[1] === '2') { classes[1]++ }

})

let classChoose

if (classes[0] === classes[1]) { classChoose = Math.floor((Math.random() \* 2) + 1) }

else if (classes[0] > classes[1]) { classChoose = 1 }

else if (classes[0] < classes[1]) { classChoose = 2 }

if (parseInt(testDatas[i][j][4]) === 1 && classChoose === 1) { a = a + 1 }

else if (parseInt(testDatas[i][j][4]) === 1 && classChoose === 2) { b = b + 1 }

else if (parseInt(testDatas[i][j][4]) === 2 && classChoose === 1) { c = c + 1 }

else if (parseInt(testDatas[i][j][4]) === 2 && classChoose === 2) { d = d + 1 }

}

console.log(`---------- Validation ${i + 1} --------------`)

console.log(`\nConfusion matrix`)

console.log(`| ${a} ${b} |`)

console.log(`| ${c} ${d} |`)

let correct = 100 \* (a + d) / (a + b + c + d)

let error = 100 - correct

console.log(`\ncorrect ${correct} %`)

console.log(`error ${error} %`)

console.log(`--------------------------------\n\n\n\n\n\n\n\n\n`)

}

}

main()

Code: Bayes (NodeJS)

let fs = require('fs')

const math = require('mathjs')

let input = fs.readFileSync('TWOCLASS.dat', 'utf8')

const fNum = 4 //Change number of features here

const percentValidate = 10

const pw1 = 1

const pw2 = 1

const fetchData = Promise.resolve(

input.trim().split('\r\n').map(x => x.split('\t'))

)

const setUpTestData = (percentValidate, data) => {

const testDataNum = data.length / percentValidate

const round = data.length / testDataNum

let testDatas = []

for (i = 0; i < round; i++) {

testDatas.push(data.slice((i \* testDataNum), (i \* testDataNum + testDataNum)))

}

return testDatas

}

const setUpTrainData = (percentValidate, data) => {

const testDataNum = data.length / percentValidate

const round = data.length / testDataNum

let trainDatas = []

for (i = 0; i < round; i++) {

let trainData = []

for (j = 0; j < data.length; j++) {

if (j < (i \* testDataNum) || j >= (i \* testDataNum + testDataNum)) {

trainData.push(data[j])

}

}

trainDatas.push(trainData)

}

return trainDatas

}

const separateClass = (dataSource, classLabel, f) => {

return dataSource.filter((data) => { return data[4] === classLabel }).map((val) => {

return val.slice(0, f)

})

}

const average = datas => {

let initP = []

initP = datas[0].map(a => 0)

return datas.reduce((p, c) => {

let res = []

for (let i = 0; i < p.length; i++) {

res.push(parseFloat(p[i]) + parseFloat(c[i]))

}

return res

}, initP).map((sum) => {

return sum / datas.length

})

}

const xMinusMean = (datas, means) => {

return datas.map((data) => {

let res = []

for (let i = 0; i < data.length; i++) {

res.push(parseFloat(data[i]) - parseFloat(means[i]))

}

return res

})

}

const fx = (numClass, cov, xMinusMean) => {

return (1 / (Math.sqrt(((2 \* Math.PI) ^ numClass) \* (math.det(cov))))) \* math.exp(math.multiply(math.multiply(math.multiply(math.transpose(xMinusMean), -0.5), math.inv(cov)), xMinusMean))

}

const main = async () => {

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

value.shift()

return value

})

const testDatas = setUpTestData(percentValidate, sourceData)

const trainDatas = setUpTrainData(percentValidate, sourceData)

for (i = 0; i <= 9; i++) {

let testData = testDatas[i].map((val) => {

return val.slice(0, fNum)

})

let testClass = testDatas[i].map((val) => {

return val.slice(-1)

})

let trainClass1 = separateClass(trainDatas[i], '1', fNum)

let trainClass2 = separateClass(trainDatas[i], '2', fNum)

let meanClass1 = average(trainClass1)

let meanClass2 = average(trainClass2)

let trainXMinusMean1 = xMinusMean(trainClass1, meanClass1)

let trainXMinusMean2 = xMinusMean(trainClass2, meanClass2)

let cov1 = math.multiply(math.multiply(math.transpose(trainXMinusMean1), trainXMinusMean1), (1 / trainClass1.length))

let cov2 = math.multiply(math.multiply(math.transpose(trainXMinusMean2), trainXMinusMean2), (1 / trainClass2.length))

let a = 0, b = 0, c = 0, d = 0

for (j = 0; j < testData.length; j++) {

let testXMinusMean1 = xMinusMean(testData, meanClass1)

let testXMinusMean2 = xMinusMean(testData, meanClass2)

let classChoose

let fx1 = fx(2, cov1, testXMinusMean1[j])

let fx2 = fx(2, cov2, testXMinusMean2[j])

if (fx1 \* pw1 === fx2 \* pw2) { classChoose = Math.floor((Math.random() \* 2) + 1) }

else if (fx1 \* pw1 > fx2 \* pw2) { classChoose = 1 }

else if (fx1 \* pw1 < fx2 \* pw2) { classChoose = 2 }

if (parseInt(testClass[j]) === 1 && classChoose === 1) { a = a + 1 }

else if (parseInt(testClass[j]) === 1 && classChoose === 2) { b = b + 1 }

else if (parseInt(testClass[j]) === 2 && classChoose === 1) { c = c + 1 }

else if (parseInt(testClass[j]) === 2 && classChoose === 2) { d = d + 1 }

}

console.log(`---------- Validation ${i + 1} --------------`)

let smeanClass1 = meanClass1.reduce((res, mean) => res + ' ' + mean, '')

let smeanClass2 = meanClass2.reduce((res, mean) => res + ' ' + mean, '')

console.log(`Mean class 1 : | ${smeanClass1} |`)

console.log(`Mean class 2 : | ${smeanClass2} |`)

console.log(`\nCovariance matrices`)

cov1.forEach((row) => {

let cov = row.reduce((res, r) => res + ' ' + r, '')

console.log(`| ${cov} |`)

})

console.log(`\nConfusion matrix`)

console.log(`| ${a} ${b} |`)

console.log(`| ${c} ${d} |`)

let correct = 100 \* (a + d) / (a + b + c + d)

let error = 100 - correct

console.log(`\ncorrect ${correct} %`)

console.log(`error ${error} %`)

console.log(`--------------------------------\n`)

}

}

main()