Running Kotlin Scripts on CBSU

Terry Casstevens (tmc46@cornell.edu)
November 1, 2022

Kotlin Requirement

- cd /workdir/<userid>
- Look here for latest Kotlin Version https://github.com/JetBrains/kotlin/releases
- wget https://github.com/JetBrains/kotlin/releases/download/v1.7.20/kotlin-compiler-1.7.20.zip
- unzip kotlin-compiler-1.7.20.zip
- export PATH=/workdir/<userid>/<kotlin folder>/bin:$PATH

Kotlin Script Extension

- Name file with extension <filename>.main.kts

Increase Heap Size for Execution

- export _JAVA_OPTIONS=-Xmx50g

Common Dependencies (put at top of script)

// https://mvnrepository.com/artifact/net.maizegenetics/tassel
@file:DependsOn("net.maizegenetics:tassel:5.2.86")

// https://mvnrepository.com/artifact/net.maizegenetics/phg
@file:DependsOn("net.maizegenetics:phg:1.2")

// https://mvnrepository.com/artifact/org.biokotlin/biokotlin
@file:DependsOn("org.biokotlin:biokotlin:0.05.01")

Executing Script

- kotlinc -script <filename>.main.kts <input args…>

Example Script using TASSEL KinshipPlugin

```
@file:DependsOn("net.maizegenetics:tassel:5.2.84")
@file:DependsOn("org.apache.commons:commons-math3:3.6.1")

import net.maizegenetics.analysis.distance.KinshipPlugin
import net.maizegenetics.dna.map.Position
import net.maizegenetics.dna.map.PositionList
import net.maizegenetics.dna.map.PositionListBuilder
import net.maizegenetics.dna.snp.GenotypeTableUtils
import net.maizegenetics.dna.snp.ImportUtils
import net.maizegenetics.taxa.distance.DistanceMatrix
import net.maizegenetics.util.LoggingUtils
import net.maizegenetics.util.Utils
import org.apache.commons.math3.stat.correlation.PearsonsCorrelation
import java.io.File

LoggingUtils.setupDebugLogging()

if (args.size != 5) {
    println("Usage: MerrittKinshipMatrices <gene names file> <VCF file> <tab
delimited gene location file> <tab delimited gene location file> <Output
File>")
    System.exit(0)
}

// Genotype file (i.e. VCF, HapMap, etc.)
val inFile = args[0]

// Gene names file listing one gene name per line
val geneNames = args[1]

// Tab delimited file with columns: Chromosome, Start, End, Gene Name
val geneLocationsFile1 = args[2]

// Tab delimited file with columns: Chromosome, Start, End, Gene Name
val geneLocationsFile2 = args[3]

// Output file
val outFile = args[4]


val genotypes = ImportUtils.read(inFile, false, false)

val genes = File(geneNames).readLines()

val positionsByName1 = nameToPositions(geneLocationsFile1)
val positionsByName2 = nameToPositions(geneLocationsFile2)
```

```
Utils.getBufferedWriter(outFile).use { writer ->

    writer.write("Gene Name\tPearson Correlation\tNumber of SNPs First\tNumber
Entries in First Gene Location File\tNumber of SNPs Second\tNumber Entries in
Second Gene Location File\n")

    genes.forEach { gene ->

        val positions1 = positionsByName1[gene]?.first
        val numEntries1 = positionsByName1[gene]?.second
        val positions2 = positionsByName2[gene]?.first
        val numEntries2 = positionsByName2[gene]?.second

        if (positions1 != null && positions2 != null) {
            // val genotype1 =
FilterSiteBuilderPlugin().positionList(positions1).runPlugin(genotypes)
            // val genotype2 =
FilterSiteBuilderPlugin().positionList(positions2).runPlugin(genotypes)
            val genotype1 = GenotypeTableUtils.filterSitesByChrPos(genotypes,
positions1, true)
            val genotype2 = GenotypeTableUtils.filterSitesByChrPos(genotypes,
positions2, true)
            if (genotype1.numberOfSites() != 0 && genotype2.numberOfSites() !=
0) {
                val kinship1 = KinshipPlugin(null, false).runPlugin(genotype1)
                val kinship2 = KinshipPlugin(null, false).runPlugin(genotype2)
                val correlation =
PearsonsCorrelation().correlation(flattenMatrix(kinship1),
flattenMatrix(kinship2))


writer.write("${gene}\t$correlation\t${genotype1.numberOfSites()}\t${numEntries
1}\t${genotype2.numberOfSites()}\t${numEntries2}\n")
            }
        }

    }
}

/**
 * Flattens a distance matrix into a 1D array. Only the upper triangle is
returned.
 * Not including the diagonal.
 */
fun flattenMatrix(matrix: DistanceMatrix): DoubleArray {
    val result = DoubleArray(matrix.numberOfTaxa() * matrix.numberOfTaxa())
    var index = 0
    for (column in 0 until matrix.numberOfTaxa()) {
        for (row in column + 1 until matrix.numberOfTaxa()) {
            result[index++] = matrix.getDistance(row, column).toDouble()
```

```
        }
    }
    return result
}

/**
 * Converts a tab delimited file to a map of gene names to position lists.
 * The file is expected to have the following columns: chromosome, start, end,
gene name
 * The positions are 1-based and inclusive.
 */
fun nameToPositions(filename: String): Map<String, Pair<PositionList, Int>> {
    val temp = mutableMapOf<String, Pair<PositionListBuilder, Int>>()
    File(filename).forEachLine { line ->
        val parts = line.split("\t")
        val name = parts[3]
        val chr = parts[0]
        val start = parts[1].toInt()
        val end = parts[2].toInt()
        val builderCount = temp.getOrPut(name) { Pair(PositionListBuilder(), 0)
}
        val builder = builderCount.first
        temp.put(name, Pair(builder, builderCount.second + 1))
        for (i in start..end) {
            builder.add(Position.of(chr, i))
        }
    }
    return temp.mapValues { Pair(it.value.first.build(), it.value.second) }
```