Exercise 5

Applications of Data Analysis

Marco Willgren 502606 [mahewi@utu.fi](mailto:mahewi@utu.fi)

Jarno Vuorenmaa 503618 [jkivuo@utu.fi](mailto:jkivuo@utu.fi)

# Data preprocessing and creating dependencies matrix

We read feature data into variable x and labels into y. Dependencies matrix is created in generateIndexDep – method. Matrix has information about dependencies between objects. One column has every index of pairs where object (responding to the column) is one of the member.

basepath = os.path.dirname(\_\_file\_\_)

featurepath = os.path.abspath(os.path.join(basepath,

*"../Data5/proteins.features"*)

labelpath = os.path.abspath(os.path.join(basepath, *"../Data5/proteins.labels"*))

x = np.genfromtxt(featurepath, delimiter=*','*)

y = np.genfromtxt(labelpath, delimiter=*','*)

def **generateIndexDep**():

indexDeps = []

for i in range(20):

iDep = []

for j in range(i\*20,i\*20+20):

iDep.append(j)

k = i

while(k < 400):

iDep.append(k)

k = k + 20

indexDeps.append(iDep)

return indexDeps

dependencies = generateIndexDep()

# Unmodified Leave-one-out cross-validation

Method takes each instance of the training set and uses it as a test instance. For every test instance method predicts the label using 1-neareast-neighbor. In inferNeighbors - method, the euclidean distance between test instance and each training instance is calculated and sorted into distances – array. The method returns 1-nearest neighbors.

def **LooCV**(modified):

yPredictions = []

for i in range(len(x)):

if modified:

trainSet,trainLabels = filterTrainSet(i)

trainSet.append(x[i])

trainLabels.append(y[i])

else:

trainSet = x

trainLabels = y

yPredictions.append(inferNeighbors(trainSet,x[i],trainLabels))

return yPredictions

def **inferNeighbors**(trainSet,testInstance,labels):

distances = []

for i in range(len(trainSet)):

distances.append((ssd.euclidean(trainSet[i], testInstance), labels[i]))

distances.sort(key=operator.itemgetter(0))

return distances[1][1]

# Modified Leave-one-out Cross-validation

Modified leave-one-out cross-validation is similar as unmodified except the traning set is filtered. Method filterTrainSet() gets index of the pair as an argument. The method calculates members of this pair and uses dependencies matrix to filter shared objects out of training set.

def **filterTrainSet**(index):

lowerBound = (index/10)\*10

if lowerBound % 20 != 0:

lowerBound = lowerBound - 10

indexOfSecondPair = lowerBound / 20

indexOfFirstPair = index - lowerBound

print indexOfFirstPair

testIndexes = dependencies[indexOfFirstPair] + dependencies[indexOfSecondPair]

#CREATE TRAINING SET AND LABELS

trainSet = []

trainLabels = []

for i in range(len(x)):

if not i in testIndexes:

trainSet.append(x[i])

trainLabels.append(y[i])

return trainSet,trainLabels

# Calculating C-index

def **calculateCIndex**(predictions, labels):

n = 0

h\_sum = 0

for i in range(len(labels)):

t = labels[i]

p = predictions[i]

for j in range(i+1,len(labels)):

nt = labels[j]

np = predictions[j]

if t != nt:

n = n + 1

if (p < np and t < nt) or (p > np and t > nt):

h\_sum = h\_sum + 1

elif (p < np and t > nt) or (p > np and t < nt):

h\_sum = h\_sum + 0

elif (p == np):

h\_sum = h\_sum + 0.5

if n == 0:

return 0

else:

return h\_sum/n

C-index is calculated as shown in Ileana Montoya’s presentation slides (I.Montoya, Prediction of the metal ion content from multi-parameter data, 2015).

# Results

Concordance index of unmodified CV

0.98820754717

Concordance index of modified CV

0.524287434765

# Code

*'''*

*Authors: Marco Willgren, 502606*

*Jarno Vuorenmaa, 503618*

*'''*

import os

import numpy as np

import operator

import scipy.spatial.distance as ssd

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

pass

basepath = os.path.dirname(\_\_file\_\_)

featurepath = os.path.abspath(os.path.join(basepath, *"../Data5/proteins.features"*))

labelpath = os.path.abspath(os.path.join(basepath, *"../Data5/proteins.labels"*))

x = np.genfromtxt(featurepath, delimiter=*','*)

y = np.genfromtxt(labelpath, delimiter=*','*)

def **generateIndexDep**():

indexDeps = []

for i in range(20):

iDep = []

for j in range(i\*20,i\*20+20):

iDep.append(j)

k = i

while(k < 400):

iDep.append(k)

k = k + 20

indexDeps.append(iDep)

return indexDeps

dependencies = generateIndexDep()

def **filterTrainSet**(index):

lowerBound = (index/10)\*10

if lowerBound % 20 != 0:

lowerBound = lowerBound - 10

indexOfSecondPair = lowerBound / 20

indexOfFirstPair = index - lowerBound

testIndexes = dependencies[indexOfFirstPair] + dependencies[indexOfSecondPair]

#CREATE TRAINING SET AND LABELS

trainSet = []

trainLabels = []

for i in range(len(x)):

if not i in testIndexes:

trainSet.append(x[i])

trainLabels.append(y[i])

return trainSet,trainLabels

def **LooCV**(modified):

yPredictions = []

for i in range(len(x)):

if modified:

trainSet,trainLabels = filterTrainSet(i)

trainSet.append(x[i])

trainLabels.append(y[i])

else:

trainSet = x

trainLabels = y

yPredictions.append(inferNeighbors(trainSet,x[i],trainLabels))

return yPredictions

def **inferNeighbors**(trainSet,testInstance,labels):

distances = []

for i in range(len(trainSet)):

distances.append((ssd.euclidean(trainSet[i], testInstance), labels[i]))

distances.sort(key=operator.itemgetter(0))

return distances[1][1]

def **calculateCIndex**(predictions, labels):

n = 0

h\_sum = 0

for i in range(len(labels)):

t = labels[i]

p = predictions[i]

for j in range(i+1,len(labels)):

nt = labels[j]

np = predictions[j]

if t != nt:

n = n + 1

if (p < np and t < nt) or (p > np and t > nt):

h\_sum = h\_sum + 1

elif (p < np and t > nt) or (p > np and t < nt):

h\_sum = h\_sum + 0

elif (p == np):

h\_sum = h\_sum + 0.5

if n == 0:

return 0

else:

return h\_sum/n

def **main**():

predictedLabels = LooCV(False)

print *'Concordance index of unmodified CV'*

print calculateCIndex(predictedLabels, y)

print

predictedLabels = LooCV(True)

print *'Concordance index of modified CV'*

print calculateCIndex(predictedLabels, y)

main()