Exercise 5

Applications of Data Analysis

Marco Willgren 502606 mahewi@utu.fi

Jarno Vuorenmaa 503618 jkivuo@utu.fi

1 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()
```

2 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.

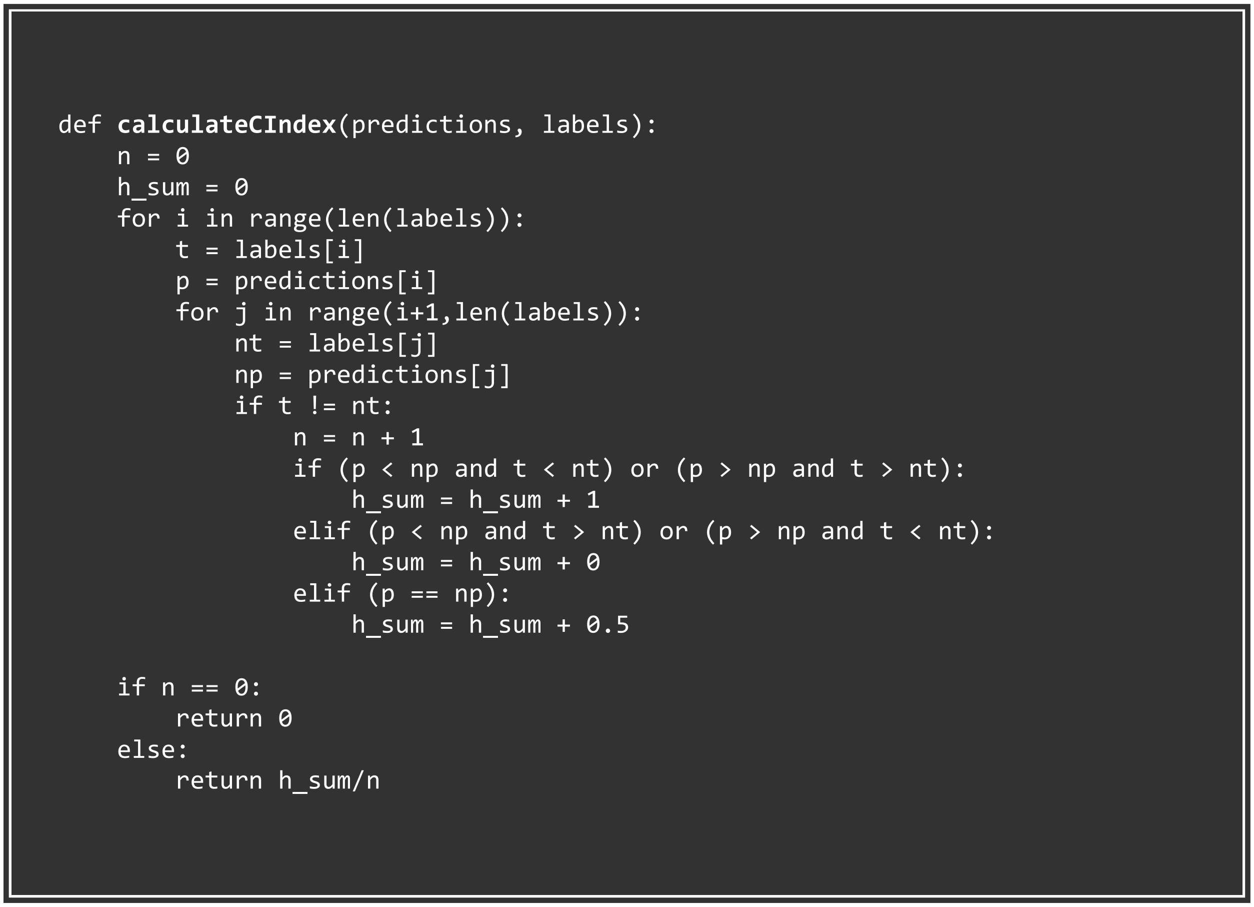
```
def LooCV(modified):
    yPredictions = []
    for i in range(len(x)):
        if modified:
            trainSet,trainLabels = filterTrainSet(i)
            trainSet.append(x[i])
            trainLabels.append(y[i])
        else:
            trainSet = x
            trainLabels = y
        yPredictions.append(inferNeighbors(trainSet,x[i],trainLabels))

    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]
```

3 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
```

4 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).

5 RESULTS

```
Concordance index of unmodified CV
0.98820754717

Concordance index of modified CV
0.524287434765
```

6 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()
```