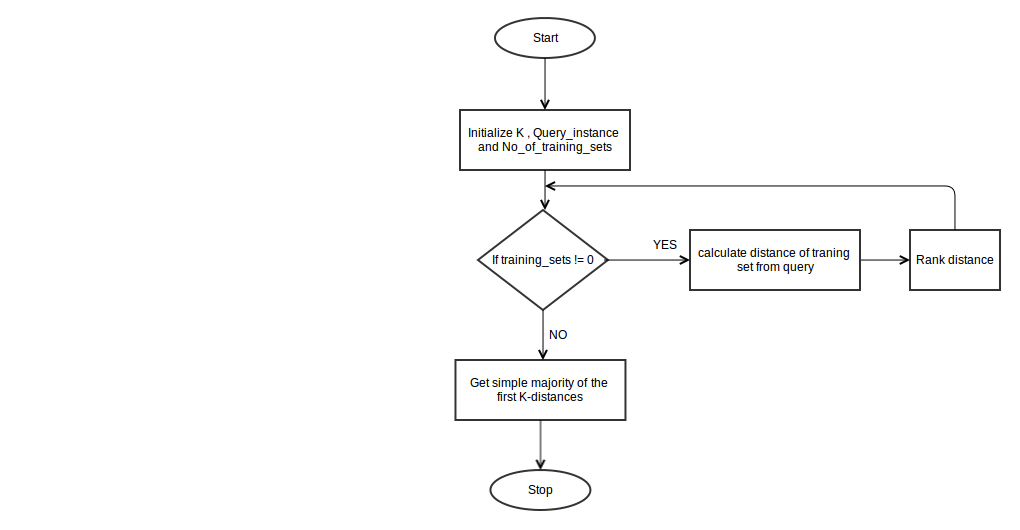
**K NEAREST NEIGHBOUR**



**ALGORITHM AND IMPLEMENTATION**

def read\_data(fn):

""" read dataset and separate into input data

and label data

"""

**# read dataset file**

with open(fn) as f:

raw\_data = np.loadtxt(f, delimiter= ',', dtype="float",

skiprows=1, usecols=None)

**# initilize list**

data = []; label = []

**#assign input data and label data**

for row in raw\_data:

data.append(row[:-1])

label.append(int(row[-1]))

**# return input data and label data**

return np.array(data), np.array(label)

def knn(k, dtrain, dtest, dtr\_label, dist=1):

""" k-nearest neighbors """

**# initialize list to store predicted class**

pred\_class = []

**# for each instance in data testing,**

**# calculate distance in respect to data training**

for ii, di in enumerate(dtest):

distances = [] # initialize list to store distance

for ij, dj in enumerate(dtrain):

**# calculate distances**

distances.append((calc\_dist(di,dj,dist), ij))

**# k-neighbors**

k\_nn = sorted(distances)[:k]

**# predict the class for the instance**

pred\_class.append(classify(k\_nn, dtr\_label))

**# return prediction class**

return pred\_class

def calc\_dist(di,dj,i=1):

""" Distance calculation for every

distance functions in use"""

if i == 1:

return ssd.euclidean(di,dj) # built-in Euclidean fn

elif i == 2:

return ssd.cityblock(di,dj) # built-in Manhattan fn

elif i == 3:

return ssd.cosine(di,dj) # built-in Cosine fn

def classify(k\_nn, dtr\_label):

""" Classify instance data test into class"""

dlabel = []

for dist, idx in k\_nn:

**# retrieve label class and store into dlabel**

dlabel.append(dtr\_label[idx])

**# return prediction class**

return np.argmax(np.bincount(dlabel))

def evaluate(result):

""" Evaluate the prediction class"""

**# create eval result array to store evaluation result**

eval\_result = np.zeros(2,int)

for x in result:

**# increment the correct prediction by 1**

if x == 0:

eval\_result[0] += 1

# increment the wrong prediction by 1

else:

eval\_result[1] += 1

# return evaluation result

return eval\_result

def main():

**""" k-nearest neighbors classifier """**

# initialize runtime

start = time.clock()

# data tests, 1 = breast cancer data test,

# 2 = iris data test

data\_tests = [1,2]

for d in data\_tests:

if d == 1:

# read dataset of breast cancer

dtrain, dtr\_label = read\_data('breast-cancer-train.csv')

dtest, true\_class = read\_data('breast-cancer-test.csv')

else:

# read dataset of breast cancer

dtrain, dtr\_label = read\_data('iris-train.csv')

dtest, true\_class = read\_data('iris-test.csv')

# initialize K

K = [1,3,7,11]

# distance function for euclidean (1), manhattan (2),

# and cosine (3)

dist\_fn = [1,2,3]

if d == 1:

print "k-NN classification results for breast cancer data set:"

else:

print "k-NN classification results for iris data set:"

print

print " Number of correct / wrong classified test records"

print "k | Euclidean dist | Manhattan dist | Cosine dist"

# run knn classifier for each k and distance function

for i in range(len(K)):

# classification result for each distance function

results = []

for j in range(len(dist\_fn)):

# predict the data test into class

pred\_class = knn(K[i], dtrain, dtest, dtr\_label, dist\_fn[j])

# evaluate the predicted result

eval\_result = evaluate(pred\_class-true\_class)

# assign the evaluated result into classification result

results.append(eval\_result[0])

results.append(eval\_result[1])

# print the classification result into the screen

print K[i], " | ", results[0], "/", results[1], \

" | ", results[2], "/", results[3], \

" | ", results[4], "/", results[5]

results = []

print

**# retrieve**

run\_time = time.clock() - start

print "Runtime:", run\_time

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

main()

**SCREENSHOT**

