```
1. Code Output
− Implement a Naive Bayes classifier and use a 5*5 cross−validation
− Run Naive Bayes and ZeroR algorithms on the following 5 datasets
− The accuracy results are as following:

a. "diabetes.csv" data set
− ZeroR:
[’62.75’, ’64.05’, ’66.01’, ’68.63’, ’64.05’, ’63.40’, ’64.71’, ’66.67’, ’64.05’
, ’67.32’, ’64.05’, ’64.71’, ’68.63’, ’67.97’, ’60.13’, ’62.75’, ’68.63’, ’69.93
’, ’60.13’, ’64.71’, ’59.48’, ’70.59’, ’66.67’, ’69.28’, ’60.78’]

− NB:
[’75.82’, ’72.55’, ’79.74’, ’68.63’, ’78.43’, ’68.63’, ’77.12’, ’75.82’, ’77.78’
, ’77.12’, ’77.78’, ’75.16’, ’73.86’, ’77.12’, ’73.20’, ’75.82’, ’81.05’, ’75.16
’, ’71.90’, ’75.16’, ’80.39’, ’64.71’, ’81.70’, ’74.51’, ’73.20’]

b. "Soybean.csv" data set
− ZeroR:
[’7.35’, ’10.29’, ’11.76’, ’7.35’, ’10.29’, ’11.76’, ’10.29’, ’11.76’, ’7.35’, ’
11.76’, ’11.03’, ’9.56’, ’11.03’, ’12.50’, ’8.09’, ’11.03’, ’8.09’, ’11.03’, ’11
.03’, ’10.29’, ’9.56’, ’12.50’, ’11.76’, ’7.35’, ’7.35’]

− NB:
[’91.18’, ’92.65’, ’92.65’, ’91.91’, ’89.71’, ’93.38’, ’94.85’, ’93.38’, ’87.50’
, ’84.56’, ’87.50’, ’88.24’, ’89.71’, ’92.65’, ’93.38’, ’91.18’, ’90.44’, ’88.24
’, ’88.24’, ’91.18’, ’88.97’, ’91.91’, ’92.65’, ’91.91’, ’86.76’]

c. "iris.csv" data set
− ZeroR:
[’16.67’, ’30.00’, ’26.67’, ’23.33’, ’20.00’, ’20.00’, ’33.33’, ’26.67’, ’26.67’
, ’30.00’, ’26.67’, ’26.67’, ’33.33’, ’30.00’, ’26.67’, ’30.00’, ’26.67’, ’20.00
’, ’20.00’, ’16.67’, ’26.67’, ’26.67’, ’30.00’, ’26.67’, ’33.33’]

− NB:
[’93.33’, ’96.67’, ’93.33’, ’100.00’, ’96.67’, ’96.67’, ’86.67’, ’96.67’, ’96.67
’, ’100.00’, ’96.67’, ’90.00’, ’96.67’, ’100.00’, ’93.33’, ’100.00’, ’93.33’, ’9
6.67’, ’96.67’, ’90.00’, ’96.67’, ’93.33’, ’93.33’, ’96.67’, ’96.67’]

2. Source Codes
=============================================================================
File <lib.py>

import math
inf = 10^17
NINF = −1 * inf
PINCH = 1 / inf
PI = 3.1415926535
EE = 2.7182818284

def indexes(data):
    rows = []   #get the indexes for the data
    for i in range(len(data)):
        rows.append(i)
    return rows

def rowprint(a):
    max = len(a)
    line = ’’
    for j in range(max):
        line += (a[j] + ’,’).rjust(15)
    return line

def maybeInt(x):
    return int(x) if x % 1 == 0.0 else float(x)

def norm(x, m, s):
    s += PINCH
    return 1/math.sqrt(2*PI*s**2)*EE**(−1*(x−m)**2/(2*s**2))

=============================================================================
File <nb.py>

import lib
```

```
def nb(testT, trainT, hypotheses, k, m):
    ck = testT[’0’].klass[0] #locate the index for class col
    total = acc = 0.0
    total += len(trainT[’0’].data[ck])
    for t in range(len(testT[’0’].data[ck])):
        want = testT[’0’].data[ck][t]
        row = []
        for i in range(len(testT[’0’].data)):
            row += [testT[’0’].data[i][t]]
        got = likelihood(row, trainT, total, hypotheses, k, m)
        acc += want == got
    return 100 * acc/len(testT[’0’].data[ck])

def likelihood(row, trainT, total, hypotheses, k, m):
    like = lib.NINF
    total += k * len(hypotheses)
    best = ’’
    for h in hypotheses:
        nh = len(trainT[h].data[trainT[’0’].klass[0]])
        prior = float(nh + k)/total
        tmp = prior
        for c in trainT[h].nump:
            i = trainT[h].nump.index(c)
            x = row[c]
            if x == "?": continue
            y = lib.norm(float(x), float(trainT[h].mu[i]), float(trainT[h].sd[i]
))
            tmp *= y
        for c in trainT[h].term:
            x = row[c]
            if x == "?": continue
            y = 0.0
            for i in range(len(trainT[h].data[c])):
                if trainT[h].data[c][i] == x: y+= 1
            tmp *= (y + m*prior) / (nh +m)
        if tmp >= like:
            like = tmp
            best = h
    return best
=============================================================================
File <tablestr.py>

import lib
class Table:
    def __init__(self):
        self.data = []     #data[[col1,...],[col2,...]]
        self.name = []     #name of i−th column
        self.order = []    #order of the col
        self.nump = []     #is i−th column numeric?
        self.wordp = []    #is i−th column non−numeric?
        self.indep = []    #list of indep columns
        self.dep = []      #list of dep columns
        self.less = []     #numeric goal to be minimized
        self.more = []     #numeric goal to be maximized
        self.klass = []    #non−numeric goal
        self.term = []     #non−numeric non−goal
        self.num = []      #numeric non−goal
        # for all cols
        self.n = []        #count of things in this col
        # for wordp columns:
        self.count = []    #count of each word
        self.mode = []     #most common word
        self.most = []     #count of most common word
        # for nump columns:
        self.hi = []       #upper bound
        self.lo = []       #lower bound
        self.mu = []       #mean
        self.m2 = []       #sum of all nums
        self.sd = []       #standard deviation# −*− coding: utf−8 −*−
        # table printing format
        self.CONVFMT = ’%06d’

def centroid(table):
    "update the mode and most values for wordp type cols or update the mean and
sd values for nump cols"
    rows = [[],[]]
    for c in range(len(table.name)):
        s = table.mode[table.wordp.index(c)] if c in table.wordp else table.CONV
FMT%table.mu[table.nump.index(c)]
        rows[0].append(str(s))
        if table.n[c] == ’0’:
            s = 0.0
        else:
            s = float(table.most[table.wordp.index(c)])/table.n[c] if c in table
.wordp else table.sd[table.nump.index(c)]
        rows[1].append(str(table.CONVFMT%s))
    return rows

def tableprint(table, stats=’’):
    "print table on the console"
    print ’ ’
    if stats != ’’: table.CONVFMT = stats
    print(’ ’ + lib.rowprint(table.name)+ ’  # notes’.ljust(10))
    print(’#’ + lib.rowprint(centroid(table)[0]) + ’  # expected’.ljust(10))
    print(’#’ + lib.rowprint(centroid(table)[1]) + ’  # certainty’.ljust(10))

    for j in range(len(table.data[0])):
        line = []
        for i in range(len(table.data)):
            line.append(table.data[i][j])
        print(’ ’ + lib.rowprint(line)+ ’  #’.ljust(10))
=============================================================================
File <reader.py>

import re
import tablestr
def readcsv(filename, table):
    "read in data from csv and create a table"
    FS = ’,’                     #define field separator
    f = open(filename)
    seen  = 0
    while True:
        str = line(f)
        if str == −1:
            if seen == 0: print("WARNING: empty or missing file")
            return −1
        a = str.split(FS)        #compute the number of attributes in table
        if len(a) > 1:
            if seen: addRow(a, table)
            else: makeTable(a, table)
            seen += 1

def line(f):
    "get one line data (without comments and whitespace)"
    str = f.readline()
    if not str: return −1             #readline finds nothing, output error
    else:
        str = "".join(str.split())    #kill whitespace
        str = re.sub(r’#.*’,’’,str)   #kill comments
        if len(str) >= 1 and str[−1] == ’,’: return str + line(f)
        else: return str

def makeTable(a, table):
    "read table titles and set all corresponding parameters"
    c = 0
    for ite in range(len(a)):
        if a[ite][0] == ’?’: continue  #the col with ’?’ is ignored
        table.order.append(ite)
        x = a[ite]
        table.name.append(x)
        isNum = 1
        if x.find(’=’) != −1:
            table.dep.append(c)
            table.klass.append(c)
            isNum = 0
        elif x.find(’+’) != −1:
```

```
            table.dep.append(c)
            table.more.append(c)
        elif x.find(’−’) != −1:
            table.dep.append(c)
            table.less.append(c)
        elif x.find(’$’) != −1:
            table.indep.append(c)
            table.num.append(c)
        else:
            table.indep.append(c)
            table.term.append(c)
            isNum = 0
        table.n.append(’0’)
        if isNum:
            table.nump.append(c)
            table.hi.append(−1*10**32)
            table.lo.append(10**32)
            table.mu.append(0)
            table.m2.append(0)
            table.sd.append(0)
        else:
            table.wordp.append(c)
            table.most.append(0)
            table.count.append({})
            table.mode.append(’’)
        c += 1
    for i in range(c): table.data.append([])

def addRow(a, table):
    "add a row of data to the table"
    for c in range(len(table.name)):
        f = table.order[c]
        x = a[f]
        table.data[c].append(x)
        if x.find(’?’) == −1:
            table.n[c] = int(table.n[c]) + 1
            if c in table.wordp:
                k = table.wordp.index(c)
                if table.count[k].has_key(x): table.count[k][x] += 1
                else: table.count[k][x] = 1
                new = table.count[k][x]
                if new > table.most[k]:
                    table.mode[k] = x
                    table.most[k] = new
            else:
                k = table.nump.index(c)
                if float(x) > float(table.hi[k]): table.hi[k] = x
                if float(x) < float(table.lo[k]): table.lo[k] = x
                delta = float(x) − table.mu[k]
                table.mu[k] += delta/table.n[c]
                table.m2[k] += delta*(float(x) − table.mu[k])
                if table.n[c] > 1:
                    table.sd[k] = (table.m2[k]/(table.n[c] − 1))**0.5
            c += 1

def klasses(table):
    "generate a set of tables based on different classes"
    if len(table.klass) == 0:
        print "No labeled classes in the given data set"
        return −1
    # assume there is only one class feature in the data set
    data = table.data[table.klass[0]]
    classnames = []
    for s in data:
        if s not in classnames:
            classnames.append(s)
    tables = klass1(table, classnames, data)
    tables[’0’] = table
    tables[’names’] = classnames
    return tables

def klass1(table, classnames, data):
    tables = {}
    for s in classnames:
        tables[s] = tablestr.Table()
        makeTable(table.name, tables[s])
        for i in range(len(data)):
            if s == data[i]:
                a = []
                for j in range(len(table.order)):
                    a.append(table.data[j][i])
                addRow(a, tables[s])
    return tables
=============================================================================

File <xval.py>

import lib
import tablestr
import reader
import random

def xvals(tables, x, b):
    k = tables[’0’].order.index(tables[’0’].klass[0])
    rows = lib.indexes(tables[’0’].data[k])
    s = int(len(rows)/b)
    xvaltables = {}
    for i in range(x):      # x times
        random.shuffle(rows)
        for b1 in range(b): # b bins
            obj = xval(b1*s, (b1+1)*s, rows, tables)
            xvaltables[i*x+b1+1] = obj
    return xvaltables


def xval(start, stop, rows, tables):
    testT = tablestr.Table()
    trainT = tablestr.Table()
    reader.makeTable(tables[’0’].name, testT)
    reader.makeTable(tables[’0’].name, trainT)
    for r in range(len(rows)):
        d = rows[r]
        a = []
        for j in range(len(tables[’0’].order)):
            a.append(tables[’0’].data[j][d])
        if r >= start and r < stop: #belonging to testing data set
            reader.addRow(a, testT)
        else:
            reader.addRow(a, trainT)
    testT = reader.klasses(testT)
    trainT = reader.klasses(trainT)
    tables = {}
    tables[’train’] = trainT
    tables[’test’] = testT
    return tables
=============================================================================
File <zeror.py>

def zeror(testT, trainT, hypotheses):
    k = testT[’0’].klass[0]
    most = 0
    for h in hypotheses:
        these = len(trainT[h].data[k]) if h in trainT[’names’] else 0
        if these > most:
            most = these
            got = h
    #print "#got", got
    acc = len(testT[got].data[k]) if got in testT[’names’] else 0
    num = 0
    for h in hypotheses: num += len(testT[h].data[k]) if h in testT[’names’] els
e 0
    return got,100*float(acc)/num
=============================================================================
File <main.py>

import reader
```

```
import tablestr
import zeror
import xval
import nb
if __name__ == "__main__":
    filename = ’data/iris.csv’
    table = tablestr.Table()             #create raw data structure
    reader.readcsv(filename,table )      #read the .csv data set
    f = ’%4.2f’                          #set the formatting for the output
    tables = reader.klasses(table)
    b = x = 5                            #set cross−validation parameters
    k = 1
    m = 2
    xvaltables = xval.xvals(tables, x, b) #generate the cross validation tables
    re_zeror = []
    re_nb = []
    for s in range(x*b):
       s += 1
       got, acc_zeror = zeror.zeror(xvaltables[s][’test’], xvaltables[s][’train’
], tables[’names’])
       acc_nb = nb.nb(xvaltables[s][’test’], xvaltables[s][’train’], tables[’nam
es’], k, m)
       re_zeror += [str(f%acc_zeror)]
       re_nb += [str(f%acc_nb)]
    print re_zeror
    print re_nb
=============================================================================

3. Illustration
− xvaltables is a nested dictionary structure that stores all the results from c
ross−validation process
− xvaltables = {’i’:{’train’:{’0’:table0, ’klassname1’:table1, ’klassname2’:tabl
e2,...,’names’:list of classnames in table0},
                     ’test’ :{’0’:table0, ’klassname1’:table1, ’klassname2’:tabl
e2,...,’names’:list of classnames in table0}},
                ...
                }
        * most outlier key i: value from 1 to x*b are the separated groups of tr
aining and testing dataset
        * second outlier key: ’train’ or ’test’ indicate the datasets under the
same group i are used for training or testing
        * ’0’:table1 contain all the data designed to group i’s training or test
ing dataset
        * ’klassnamei’:tablei contain all the data in table0 with class value eq
uals to klassnamei
        * ’names’: a list that include all the classnames in table0
```