```
    #Info for table
    #
    csvindex = -1 #initialized to -1 as lists start at zero
    colname = {k: [] for k in range(1)} #stores dict of names of columns
 5  data = {k: [] for k in range(1)} #stores dict of list of lists containing each r
    ow
    test = [] #stores test data
    #
    #metadata
    #
10  order = {k:dict.fromkeys(colname) for k in range(1)} #stores colnames and index
    of column in csv
    klass = {k: [] for k in range(1)} #dict of list of klass columns
    more = {k: [] for k in range(1)} #dict of list of more columns
    less = {k: [] for k in range(1)} #dict of list of less columns
    num = {k: [] for k in range(1)} #dict of list of num columns
15  term = {k: [] for k in range(1)} #dict of list of term columns
    dep = {k: [] for k in range(1)} #dict of list of dependent columns
    indep = {k: [] for k in range(1)} #dict of list of independent columns
    nump = {k: [] for k in range(1)} #dict of list containing nump column names
    wordp = {k: [] for k in range(1)} #dict of list containing wordp column names
20  #
    #for nump values
    #
    hi = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing highest va
    lues of nump columns
    lo = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing lowest val
    ues of nump columns
25  mu = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing mean value
    s of nump columns
    m2 = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing m2 values
    of nump columns
    sd = {k:dict.fromkeys(nump) for k in range(1)} #dictionary containing std dev of
     nump columns
    #
    #for wordp values
30  #
    mode = {k:dict.fromkeys(wordp) for k in range(1)} #dictionary containing mode of
     wordp columns
    most = {k:dict.fromkeys(wordp) for k in range(1)} #dictionary containing most oc
    cured item of wordp columns
    count = {k:dict(dict.fromkeys(wordp)) for k in range(1) }#dictionary of dictiona
    ries of count of each item in each wordp column
    #
35  #for all
    #
    n = {k:dict.fromkeys(colname) for k in range(1)} #stores number of elements in e
    ach column
    isnum = True
    #
40  #for the zeror
    #
    hypotheses = {}
    #
    #for naive bayes
45  l = {} #dictionary of likelyhood
    #
```

```
import re
from globfile import *
from random import *
from math import *
PI = 3.1415926535
EE = 2.7182818284

def line(csvfile): #returns formatted line from the csvfile
    l = csvfile.readline()
    endcommare = re.compile('.*,$')
    if l ≠ '':
        l = l.split('#')[0]
        l = l.replace('\t','')
        l = l.replace('\n','')
        l = l.replace(' ','')
        endcomma = endcommare.match(l)
        if endcomma:
            return l+line(csvfile)
        else:
            return l
    else:
        return -1

def rowprint(row): #returns neat rows
    columns = [ "%15s" % cell for cell in row]
    columns.append("%4s" % '#')
    return ' '.join(columns)

def expected(row,z): #returns expected outcome
    out = [c for c in colname[z]]
    for c in row:
        if c in wordp[z]:
            out[colname[z].index(c)] = str(mode[z][c])
        else:
            out[colname[z].index(c)] = str('%0.2f' % round(mu[z][c],2))
    return out

def indexes(lst):
    out = []*len(lst)
    for i in lst:
        out[i] = i
    return out

def newdlist(name, key):
    name[key] = []

def newddict(name,key):
    name[key] = {}

def newddictdict(name,key,c):
    name[key][c] = {}

def indexes(data,z):
    return data[z]

def shuffled(rows):
    shuffle(rows)

def norm(x,m,s):
    s += 1/10**23
    a = 1/sqrt(2*pi*s**2)
    b = (x-m)**2/(2*s**2)
    return a*e**(-1*b)
```

```
#! /usr/bin/env python
from lib import *
from reader import *
from xval import *
from math import *

def nb(test,data,hypotheses,z,k,m):
    total = 0.0
    acc = 0.0
    for h in hypotheses:
        total += len(data[h])
    where = klassAt(z)
    for t in test:
        want = t[where]
        got = likelyhood(t,data,total,hypotheses,l,z,k,m)
        if want ≡ got:
            acc+=1.0
    print '%0.2f' % round(100*acc/len(test),2),

def likelyhood(t,data,total,hypotheses,l,z,k,m):
    like = -0.1*10**23
    best = ''
    total += k*len(hypotheses)
    for h in hypotheses:
        nh = len(data[h])*0.1
        prior = (nh+k) / total
        tmp = log(prior)
        for c in term[h]:
            try:
                ind = colname[h].index(c)
                x = t[ind]
                if x ≡ '?':
                    continue
                y = count[h][c][x]
                tmp += log((y + m*prior)/(nh+m))
            except KeyError:
                continue
        for c in num[h]:
            ind = colname[h].index(c)
            x = t[ind]
            if x ≡ '?':
                continue
            y = norm(x, mu[h][c], sd[h][c])
            tmp += log(y)
        l[h] = tmp
        if tmp ≥ like:
            like = tmp;
            best = h
    return best
```

```
diabetes
zeror
WARNING: empty or missing file
69.28 66.01 62.09 67.32 61.44 62.09 66.67 64.71 60.78 71.24 63.40 66.01 58.17 64
.05 73.20 67.97 60.78 60.13 71.24 65.36 67.32 67.97 60.13 65.36 64.71
diabetes
nb
WARNING: empty or missing file
79.08 75.16 72.55 81.05 74.51 71.24 75.16 79.74 77.78 77.12 76.47 79.08 75.82 74
.51 74.51 81.05 74.51 72.55 75.16 77.78 73.20 85.62 74.51 72.55 75.16
soybean
zeror
WARNING: empty or missing file
12.50 12.50 11.76 13.97 13.24 13.24 13.97 13.97 10.29 16.18 13.24 12.50 13.97 13
.24 13.97 13.97 9.56 9.56 16.18 18.38 15.44 13.97 8.09 14.71 14.71
soybean
nb
WARNING: empty or missing file
50.00 72.79 76.47 78.68 80.88 83.82 86.76 80.15 85.29 89.71 89.71 86.76 84.56 89
.71 86.76 86.03 87.50 89.71 90.44 86.76 89.71 88.24 86.76 89.71 86.76
iris
zeror
WARNING: empty or missing file
20.00 43.33 30.00 26.67 36.67 26.67 26.67 13.33 40.00 40.00 20.00 26.67 30.00 30
.00 33.33 26.67 26.67 30.00 33.33 30.00 30.00 26.67 30.00 36.67 40.00
iris
nb
WARNING: empty or missing file
100.00 93.33 93.33 100.00 96.67 96.67 90.00 100.00 96.67 93.33 96.67 96.67 93.33
 100.00 93.33 90.00 96.67 96.67 100.00 96.67 93.33 100.00 96.67 93.33 96.67
```

```
from reader import *
from table import *
from sys import argv
from xval import *

csvfile = open('../data/'+argv[1]+'.csv','r')
readCsv(csvfile,argv[2]) #takes predicted value as arguement
xvals(data,5,5,'nb',argv[2],1,2)

#tableprint(argv[1])
```

```
import re
from lib import *

def makeTable(lst,z):
    newdlist(klass,z)
    newddict(order,z)
    newdlist(less,z)
    newdlist(num,z)
    newdlist(term,z)
    newdlist(dep,z)
    newdlist(indep,z)
    newdlist(nump,z)
    newdlist(wordp,z)
    newdlist(colname,z)
    newdlist(data,z)
    newddict(count,z)
    newddict(n,z)
    newddict(mode,z)
    newddict(most,z)
    newddict(hi,z)
    newddict(lo,z)
    newddict(mu,z)
    newddict(m2,z)
    newddict(sd,z)
    newdlist(data,z)

    csvindex = -1
    for csvcol in lst:
        isnum=True
        csvindex+=1
        ignore = re.match('\?.*$',csvcol)
        if ignore:
            continue
        else:
            colname[z].append(csvcol)
            order[z][csvcol] = csvindex
            klasschk = re.match('=.*$',csvcol)
            morechk = re.match('\+.*$',csvcol)
            lesschk = re.match('-.*$',csvcol)
            numchk = re.match('\$.*$',csvcol)
            if klasschk:
                dep[z].append(csvcol)
                klass[z].append(csvcol)
                isnum = False
            elif morechk:
                dep[z].append(csvcol)
                more[z].append(csvcol)
            elif lesschk:
                dep[z].append(csvcol)
                less[z].append(csvcol)
            elif numchk:
                indep[z].append(csvcol)
                num[z].append(csvcol)
            else:
                indep[z].append(csvcol)
                term[z].append(csvcol)
                isnum = False
            n[z][csvcol] = 0
            if isnum:
                nump[z].append(csvcol)
                hi[z][csvcol] = 0.1 * (-10**13)
                lo[z][csvcol] = 0.1 * (10**13)
                mu[z][csvcol] = 0.0
                m2[z][csvcol] = 0.0
                sd[z][csvcol] = 0.0
            else:
                wordp[z].append(csvcol)
                count[z][csvcol] = {}
                mode[z][csvcol] = 0
                most[z][csvcol] = 0

def addRow(lst,z):
    temp = []
```



```
    skip = False
    for c in klass[z]:
        csvindex = order[z][c]
        item = lst[csvindex]
        if item ≠ z:
            skip = True
        if z ≡ "both":
            skip = False
    for c in colname[z]:
        csvindex = order[z][c]
        item = lst[csvindex]
        uncertain = re.match('\?',str(item))
        if skip:
            return
        if uncertain:
            temp.append(item)
        else:
            n[z][c] += 1
            if c in wordp[z]:
                temp.append(item)
                try:
                    new = count[z][c][item] = count[z][c][item] + 1
                    if new > most[z][c]:
                        most[z][c] = new
                        mode[z][c] = item
                except KeyError:
                    count[z][c][item] = 1
                    if count[z][c][item] > most[z][c]:
                        most[z][c] = 1
                        mode[z][c] = item
            else:
                item = float(item)
                temp.append(item)
                if item > hi[z][c]:
                    hi[z][c] = item
                if item < lo[z][c]:
                    lo[z][c] = item
                delta = item - mu[z][c]
                mu[z][c] += delta / n[z][c]
                m2[z][c] += delta * (item - mu[z][c])
                if n[z][c] > 1:
                    sd[z][c] = (m2[z][c] / (n[z][c] - 1))**0.5
    data[z].append(temp)

def readCsv(csvfile,z):
    seen = False
    FS = ','
    while True:
        lst = line(csvfile)
        if lst ≡ -1:
            print 'WARNING: empty or missing file'
            return -1
        lst = lst.split(FS)
        if lst ≠ ['']:
            if seen:
                addRow(lst,z)
            else:
                seen = True
                makeTable(lst,z)
```

```
from globfile import *
from lib import *
def tableprint(z): #prints table with the summary
    print rowprint(colname[z]),'%10s' % 'notes'
    print rowprint(expected(colname[z],z)), '%10s' % 'expected'
    temp = [ c for c in range(len(colname[z]))]
    for c in colname[z]:
        if c in nump[z]:
            temp[colname[z].index(c)] = str('%0.2f' % round(sd[z][c],2))
        else:
            temp[colname[z].index(c)] = str('%0.2f' % round(float(most[z][c])/fl
oat(n[z][c]),2))
    print rowprint(temp),'%10s' % 'certainity'
    for row in data[z]:
        print rowprint(row)

def klass1(data, z):
    for k in klass[z]:
        return data[colname[z].index(k)]

def klassAt(z):
    for k in klass[z]:
        return colname[z].index(k)
```

```
#! /bin/python
from lib import *
from reader import *
from table import *
from zeror import *
from nb import *

def xvals(data,x,b,f,z,k,m):
    rows = indexes(data,z)
    s = int(len(rows)/b)
    while x>0:
        shuffled(rows)
        for b1 in range(0,b):
            xval(b1*s, (b1+1)*s, data, rows, f, z, k, m)
        x=x-1

def  xval(start, stop, data, rows, f, z, k, m):
    rmax = len(rows)
    test = []
    temp = ""
    newddict(data,z)
    for r in range(0, rmax):
        d = rows[r]
        if r ≥ start ∧ r < stop:
            test.append(d)
        else:
            temp = klass1(d, z)
            try:
                hypotheses[temp] += 1
                if hypotheses[temp] ≡ 1:
                    makeTable(colname[z],temp)
                addRow(d,temp)
            except KeyError:
                hypotheses[temp] = 1
                if hypotheses[temp] ≡ 1:
                    makeTable(colname[z],temp)
                addRow(d,temp)
    #zeror(test, data, hypotheses, z)
    #xvalTest1(test,data,hypotheses)
    nb(test,data,hypotheses,z,k,m)


def xvalTest1(test,data,hypotheses):
    print "\n================================="
    for h in hypotheses:
        tableprint(h)
```

```
from reader import *
from  xval import *
from lib import *

def zeror(test,data,hypotheses,z):
    hmost = -10**23
    acc = 0
    got = ""
    for h in hypotheses:
        these = len(data[h])
        if these > hmost:
            hmost = these
            got = h
    #print "#got: ",got
    where = klassAt(z)
    for t in test:
        want = t[where]
        if want == got:
            acc+=1.0
    print '%0.2f' % round(100*acc/len(test),2),
```