```
diabetes
zeror
62.09 71.24 70.59 59.48 62.75 69.28 63.40 69.28 65.36 58.17 67.97 64.05 62.75 63
.40 66.67 69.93 62.75 58.17 70.59 64.71 67.97 68.63 62.09 62.75 63.40
nb
77.78 78.43 73.86 70.59 76.47 70.59 69.93 88.24 71.90 74.51 73.20 77.78 71.90 71
.90 83.01 70.59 72.55 76.47 79.74 77.78 76.47 79.74 77.78 69.93 76.47

iris
zeror
30.00 30.00 23.33 23.33 30.00 26.67 26.67 26.67 26.67 26.67 23.33 26.67 23.33 23
.33 23.33 26.67 26.67 20.00 26.67 23.33 23.33 26.67 23.33 30.00 30.0
nb
100.00 96.67 93.33 86.67 96.67 96.67 96.67 93.33 96.67 96.67 93.33 90.00 96.67 9
6.67 100.00 100.00 93.33 100.00 93.33 90.00 96.67 90.00 93.33 96.67 96.67

soybean
zeror
12.50 11.03 9.56 6.62 11.03 11.76 11.03 9.56 11.03 11.03 10.29 10.29 10.29 11.03
 12.50 11.03 11.76 10.29 9.56 11.76 8.82 12.50 11.03 7.35 11.03
nb
64.71 66.91 69.85 65.44 66.91 62.50 58.09 72.79 69.12 75.00 63.24 70.59 64.71 66
.91 70.59 71.32 62.50 69.12 57.35 66.91 65.44 61.76 73.53 61.76 69.12
```

```
    #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+=0.00001
    a = 1/sqrt(2*pi*(s**2))
    b = (x-m)**2/(2*s**2)
    return a*e**(-1*b)

def numberp(x):
    return isinstance(x,int)

def l2s(l,sep,form):
    form = form if form ≠ "" else "%5.3f"
    s = ""
    n = len(l)
    for i in range(0,n):
        one = l[i]
```



```
        if numberp(one):
            one = format(one,form)
        s = s+sep+str(one)
    return s

def l2sd(d,sep,form):
    form = form if form ≠ "" else "%5.3f"
    s = ""
    for i in d:
        one = d[i]
        s = s+sep+str(round(one,3))
    return s

def pairs(lst):
    tmp = {}
    i = 0
    while(i < len(lst)):
        tmp[lst[i]] = lst[i+1]
        i += 2
    return tmp
```

```
    from nb import *
    from zeror import *
```

```
#! /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
```

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

csvfile = open('../data/'+argv[1]+'.csv','r')
readCsv(csvfile,argv[2]) #takes predicted value as arguement
a = argv[3]
print argv[1]
print "zeror"
xvals(data,5,5,'zeror',argv[2],2,1)
print ""
print "nb"
xvals(data,5,5,'nb',argv[2],2,1)
#print "knn"
#uxvals(data,5,5,'knasd',argv[2],2,1,a)

#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)
                klasschk1 = re.match('=.*$',csvcol)
                morechk = re.match('\+.*$',csvcol)
                lesschk = re.match('-.*$',csvcol)
                numchk = re.match('\$.*$',csvcol)
                if klasschk ∨ klasschk1:
                    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" ∨ z ≡ "train":
                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 tableprint1(z):
        print rowprint(colname[z])
        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 *
from knn import *

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

def uxval(start,stop,data,rows,f,z,m,k,a):
    rmax = len(rows)
    test = []
    temp = ""
    makeTable(colname[z],"train")
    for r in range(0, rmax):
        d = rows[r]
        if r ≥ start ∧ r < stop:
            test.append(d)
        else:
            addRow(d,"train")
    #zeror(test, data, hypotheses, z)
    #xvalTest1(test,data,hypotheses)
    #nb(test,data,hypotheses,z,k,m)
    knn(test,data,"train",a,k)

def uxvalTest1(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,k,m):
    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),
```