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

def dist(this,that,data,z,indep,nump):
    tot = 0.0
    for k in indep:
        ind = colname[z].index(k)
        v1 = this[ind]
        v2 = that[ind]
        if v1 ≡ "?" ∧ v2 ≡ "?":
            tot+=1
        elif k in nump:
            aLittle = 0.0000001
            if v1 ≡ "?":
                v1 = 1 if v2 < 0.5 else 0
            else:
                v1 = (v1 - lo[z][k])/ (hi[z][k] - lo[z][k] + aLittle)
            if v2 ≡ "?":
                v2 = 1 if v1 < 0.5 else 0
            else:
                v2 = (v2 - lo[z][k])/ (hi[z][k] - lo[z][k] + aLittle)
            tot += (v2-v1)**2
        else:
            if v1 ≡ "?":
                tot += 1
            elif v2 ≡ "?":
                tot += 1
            elif v1 ≠ v2:
                tot += 1
            else:
                tot += 0
    ret = tot**0.5 / (len(indep))**0.5
    return ret
```

```
    #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
    #
```

```
#! /usr/bin/env python
from reader import *
from table import *
from dist import *
import sys
def knn(test,data,z,a,k):
    acc = 0.0
    if a ≡ "−−once":
        print "#the training data"
        tableprint1(z)
    where = klassAt(z)
    for t in test:
        want = t[where]
        got = knn1(t,data,z,a,where,k)
        if want ≡ got:
            acc+=1.0
        if a ≡ "−−once":
            print want,got
            sys.exit()
    print '%0.2f' % round(100*acc/len(test),2),

def knn1(t,data,z,a,where,k):
    seen = {}
    #lst = [{} for i in range(0,len(data[z]))]
    lst = []
    sort = neighbors(t, data, z, lst)
    if a ≡ "−−once":
        print "#the test row"
        print rowprint(t)
        print "#the distances"
        for i in range(0,len(sort)):
            print i,round(sort[i]['x'],5),rowprint(sort[i]['d'])
    nearestk(k,data,z,where,sort,seen)
    return mostSeen(seen)

def neighbors(t,data,z,lst):
    for d in data[z]:
        ind = data[z].index(d)
        dic = {}
        dic['x'] = dist(t,d,data,z,indep[z],nump[z])
        dic['d'] = d
        lst.append(dic)
        #lst[ind]['x'] = dist(t,d,data,z,indep[z],nump[z])
        #lst[ind]['d'] = d
    """print "lstttttttttttttttttttttttt"
  for i in range(0,len(lst)):
    try:
      print lst[i]
      print lst[i]['x'];
      print i
    except KeyError:
      lst[i]['x'] = −1
      lst[i]['d'] = []
      """
    sort = sorted(lst, key=lambda lst: lst['x'])
    return sort

def nearestk(k,data,z,where,sort,seen):
    for i in range(0,k):
        that = sort[i]['d']
        ind = data[z].index(that)
        got = data[z][ind][where]
        try:
            seen[got] += 1
        except KeyError:
            seen[got] = 1

def mostSeen(seen):
    maxi = −10**23
    for x in seen:
        if seen[x] > maxi:
            maxi = seen[x]
```



```
            out = x
    return out
```

```
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
```

```
WARNING: empty or missing file
#the training data
       outlook    $temperature       $humidity           windy           !play
   #
          sunny            69.0            70.0           FALSE             yes
   #
          sunny            75.0            70.0            TRUE             yes
   #
          sunny            72.0            95.0           FALSE              no
   #
          sunny            80.0            90.0            TRUE              no
   #
          rainy            75.0            80.0           FALSE             yes
   #
          rainy            71.0            91.0            TRUE              no
   #
          rainy            68.0            80.0           FALSE             yes
   #
#the test row
        overcast            81.0            75.0           FALSE             yes
   #
#the distances
0 0.56789           rainy            75.0            80.0           FALSE
      yes    #
1 0.71414           sunny            69.0            70.0           FALSE
      yes    #
2 0.74204           sunny            72.0            95.0           FALSE
       no    #
3 0.74391           rainy            68.0            80.0           FALSE
      yes    #
4 0.75664           sunny            75.0            70.0            TRUE
      yes    #
5 0.76924           sunny            80.0            90.0            TRUE
       no    #
6 0.88091           rainy            71.0            91.0            TRUE
       no    #
yes yes

diabetes

WARNING: empty or missing file
nb
78.39 74.48 78.13 74.48
knn
66.15 68.49 69.79 70.57

soybean

WARNING: empty or missing file
nb
41.35 53.37 53.37 45.16
knn
86.80 83.87 87.10 86.22
```

```
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 "nb"
xvals(data,2,2,'knn',argv[2],2,2)
print ""
print "knn"
#uxvals(data,2,2,'nb',argv[2],2,2,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)
```

```
#! /usr/bin/env 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 = []
    hypotheses = {}
    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),
```