UNIVERSITEIT VAN AMSTERDAM

KANSREKENING EN STATISTIEK

LAB-2

Authors:
Abe WIERSMA (10433120)

14 april 2015

1 Deel 1

1. (a)

$$f(x) = \begin{cases} 1/7 & \text{Wanneer x binnen de verdeling.} \\ 0 & \text{Wanneer daarbuiten.} \end{cases}$$

   En dus

$$F(x) = \begin{cases} 0 & \text{als } x < a \\ (x-3)/(9-3) & \text{Wanneer x binnen de verdeling.} \\ 1 & \text{als } x \geq b \end{cases}$$

   (b) $[-10, -9, ..., 3] \vee [3, 4, ..., 9] = [3]$ $P(3) = 1/7$

   (c) $((b-a)+1) * 1/7$

2. Ik had laatst nog een artikel gelezen over dat een munt niet geladen kan zijn, heeft wel aansluiting op dit vraagstuk.Wel leuk.
   `http://www.stat.columbia.edu/~gelman/research/published/diceRev2.pdf`

   (a) $U = \{kop, munt\}$

   (b) $\binom{n}{k} p^k (1-p)^{n-k}$

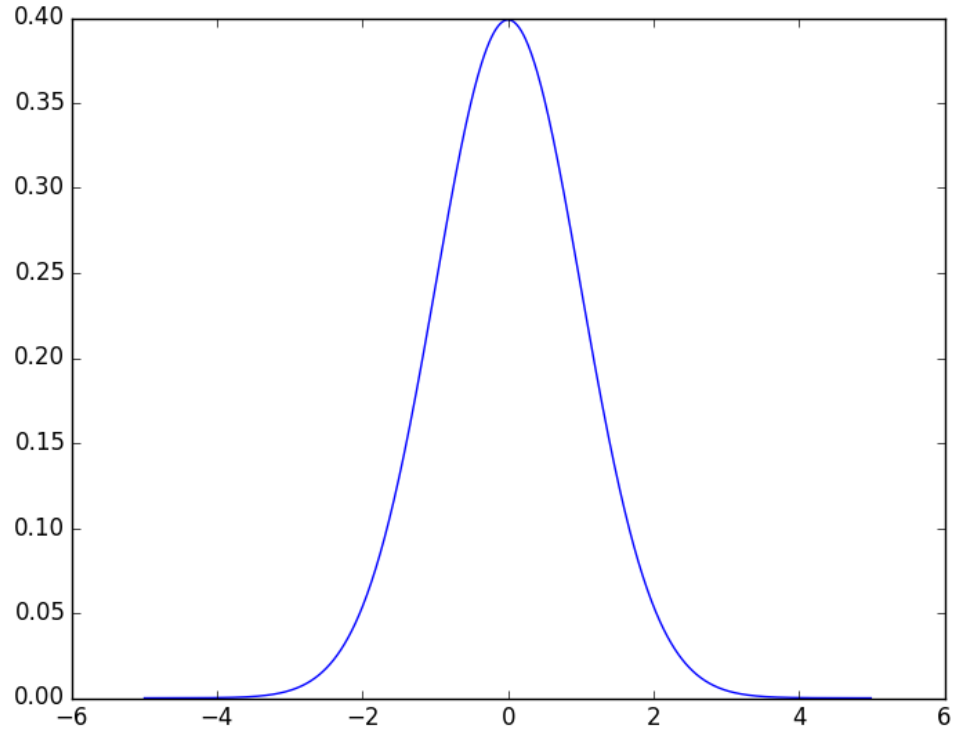
   (c) De Binomiale verdeling

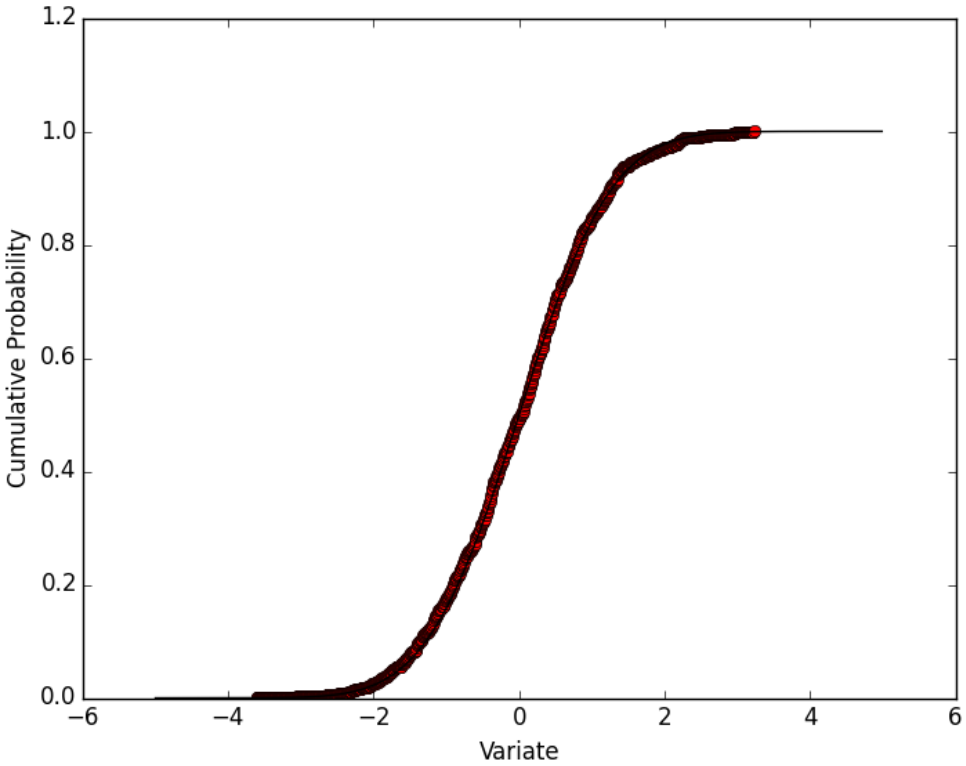
   (d) Implementatie:

```
import random

p = 0.6
upper_bound = 100000000
U = [0, 0]
for index in range(upper_bound):
    if random.random() < p:
        U[0] += 1
    else:
        U[1] += 1
test_p = U[0] / upper_bound
print(test_p, p)
```

3. Implementatie:

```
import pylab
import numpy
from scipy.stats import norm
import random


# function for standard normal equation
def sne(x):
    return numpy.power(numpy.e, -0.5 * x * x) / numpy.sqrt(2 * numpy.pi)

def main():
    # 1000 spaced numbers
    x = numpy.arange(-5.0, 5.0, 0.01)
    y1 = sne(x)

    pylab.plot(x, y1)
    pylab.show()

    y = norm.cdf(x)
    x2 = []
    for i in range(1000):
        x2.append(random.normalvariate(0, 1))

    values, base = numpy.histogram(x2, bins=1000)
    normal_values = values / float(values.sum())
    #evaluate the cumulative
    cumulative = numpy.cumsum(normal_values)
    # plot the cumulative function
    pylab.plot(base[:-1], cumulative, 'ro')

    pylab.plot(x, y, color="black")
    pylab.xlabel("Variate")
    pylab.ylabel("Cumulative Probability")
    pylab.show()

if __name__ == '__main__':
    main()
```





2 Naive Bayes Classifier

2.1 Implementatie

My python implementation:

```
import re
import numpy
import pylab

def male_or_female(Mmu, Fmu, Msigma, Fsigma, test_data):
    Mpost = normpdf(test_data[0], Mmu[0], Msigma[0]) * \
            normpdf(test_data[1], Mmu[1], Msigma[1]) * \
            normpdf(test_data[2], Mmu[2], Msigma[2])
    Fpost = normpdf(test_data[0], Fmu[0], Fsigma[0]) * \
            normpdf(test_data[1], Fmu[1], Fsigma[1]) * \
            normpdf(test_data[2], Fmu[2], Fsigma[2])

    if(Mpost > Fpost):
        return 'M'
    else:
        return 'F'


# Normal probability density function
def normpdf(x, mu, sigma):
    u = (x - mu) / abs(sigma)
    y = (1 / (numpy.sqrt(2 * numpy.pi) * abs(sigma))) * numpy.exp(-u * u / 2)
    return y


def main():
    ptrn = '^([MF])\s?,([1-9][0-9]*|0),([1-9][0-9]*|0),([1-9][0-9]*.?[0-9]|0)$'
    lines = [line.strip() for line in open('biometrie2014.csv')]
    Mdata = []
    Fdata = []
    for line in lines:
        searchResult = re.search(ptrn, line)
        if searchResult:
            if searchResult.group(1) == 'M':
                Mdata.append([float(searchResult.group(2)),
                              float(searchResult.group(3)),
                              float(searchResult.group(4))])
            if searchResult.group(1) == 'F':
                Fdata.append([float(searchResult.group(2)),
                              float(searchResult.group(3)),
                              float(searchResult.group(4))])
        else:
            print('line did not pass regex')

    Mdata = numpy.array(Mdata)
    Fdata = numpy.array(Fdata)

    Mmu = numpy.median(Mdata, axis=0)
    Fmu = numpy.median(Fdata, axis=0)

    Msigma = numpy.sqrt(numpy.var(Mdata, axis=0))
    Fsigma = numpy.sqrt(numpy.var(Fdata, axis=0))
```

```
    xWeight = numpy.arange(0, 150, 0.1)
    Mweight = normpdf(xWeight, Mmu[0], Msigma[0])
    Fweight = normpdf(xWeight, Fmu[0], Fsigma[0])

    pylab.plot(xWeight, Mweight, color='red')
    pylab.plot(xWeight, Fweight, color='blue')
    pylab.legend(['man', 'vrouw'])
    pylab.xlabel("Weight in Kg")
    pylab.show()

    xHeight = numpy.arange(140, 210, 0.1)
    Mheight = normpdf(xHeight, Mmu[1], Msigma[1])
    Fheight = normpdf(xHeight, Fmu[1], Fsigma[1])

    pylab.plot(xHeight, Mheight, color='red')
    pylab.plot(xHeight, Fheight, color='blue')
    pylab.legend(['man', 'vrouw'])
    pylab.xlabel("Height in cm")
    pylab.show()

    xSize = numpy.arange(30, 50, 0.1)
    Msize = normpdf(xSize, Mmu[2], Msigma[2])
    Fsize = normpdf(xSize, Fmu[2], Fsigma[2])

    pylab.plot(xSize, Msize, color='red')
    pylab.plot(xSize, Fsize, color='blue')
    pylab.xlabel("Shoe size")
    pylab.legend(['man', 'vrouw'])
    pylab.show()

    print(male_or_female(Mmu, Fmu, Msigma, Fsigma, [60, 183, 42]))

    Mtest = 0
    Ftest = 0

    for m in Mdata:
        if(male_or_female(Mmu, Fmu, Msigma, Fsigma, m) == 'M'):
            Mtest = Mtest + 1
        else:
            Ftest = Ftest + 1
    print(Mtest, Ftest)

    Mtest = 0
    Ftest = 0

    for f in Fdata:
        if(male_or_female(Mmu, Fmu, Msigma, Fsigma, f) == 'F'):
            Ftest = Ftest + 1
        else:
            Mtest = Mtest + 1
    print(Mtest, Ftest)

if __name__ == '__main__':
```

```
main()
```

2.2 Theorie

Naive Bayes is a conditional probabilistic model that classifies a vector to a class space. Given a vector $\mathbf{x}$ with properties $x_1 ... x_n$, it determines the probabilities of belonging to predefined classes.In our case we have the Female Class and the Male Class. And we want to check whether the probability of belonging to one class is higher than belonging to the other class.
$x \in Female$ if$(p(Female|x_1, \ldots, x_n) > p(Male|x_1, \ldots, x_n))$
or
$x \in Male$ if$(p(Male|x_1, \ldots, x_n) > p(Female|x_1, \ldots, x_n))$
Using Bayes theorem this probability can then be rewritten:

$$p((Fe)Male|\mathbf{x}) = \frac{p((Fe)Male)\ p(\mathbf{x}|(Fe)Male)}{p(\mathbf{x})}.$$

Because the denominator is a constant in both sides of the equation it can be discarded.

$$p((Fe)Male)\ p(\mathbf{x}|(Fe)Male) > p((Fe)Male)\ p(\mathbf{x}|(Fe)Male)$$

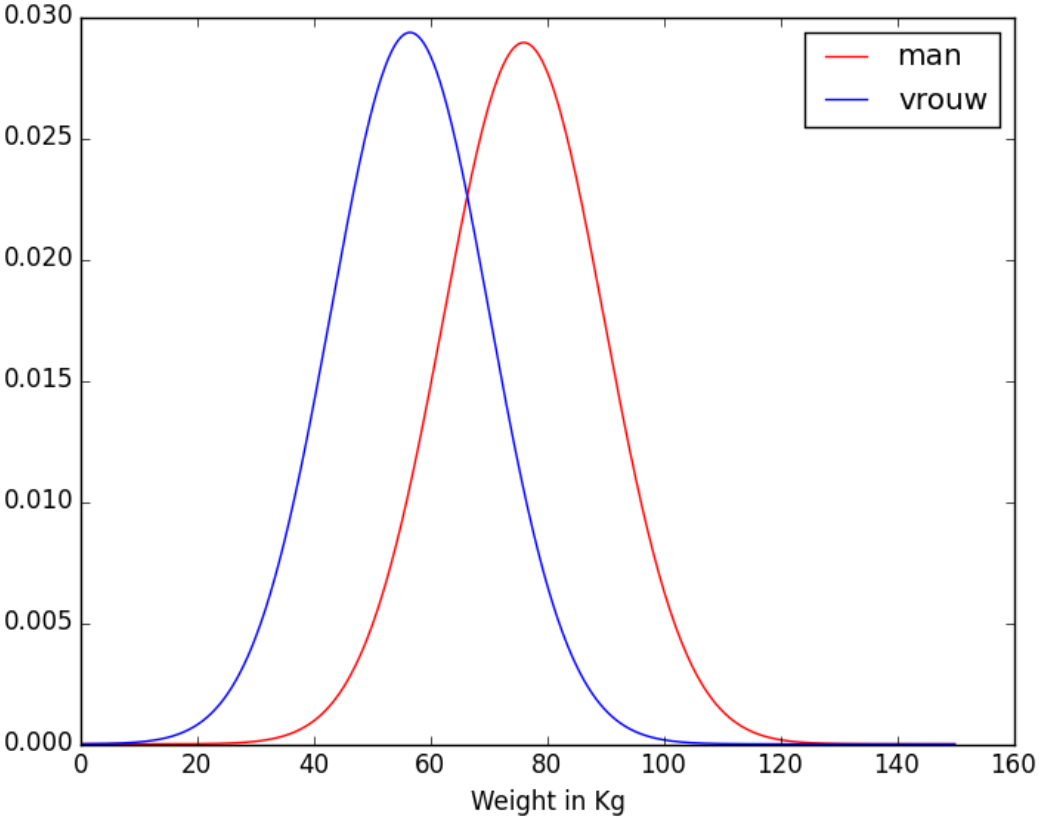
The probability to draw either a man of a woman from the population in our case is also equal and can therefore also be discarded.

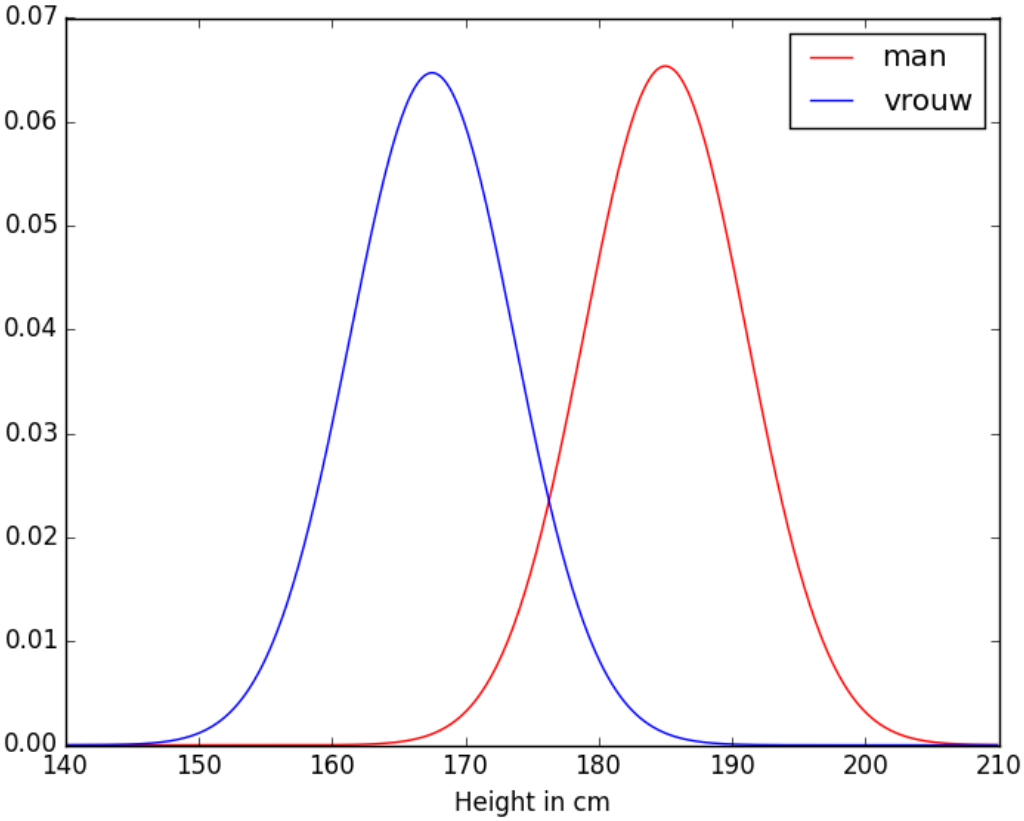
$$p(\mathbf{x}|(Fe)Male) > p(\mathbf{x}|(Fe)Male)$$

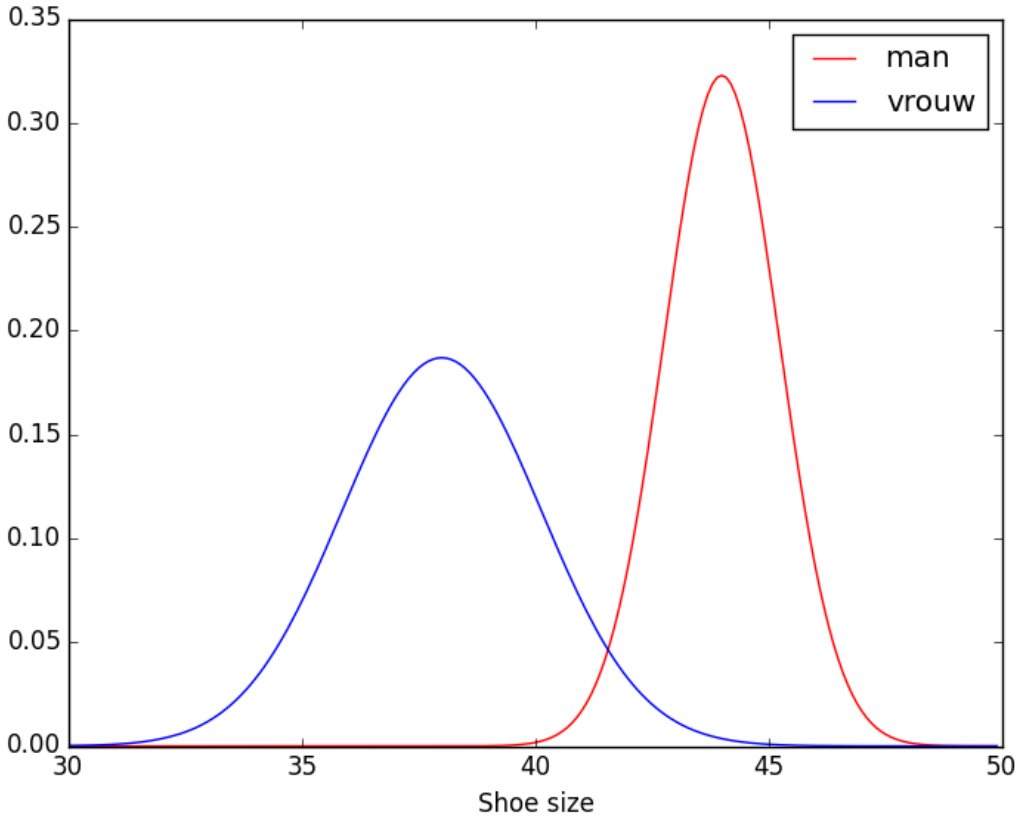
Here the navety of the classifier comes around, from here on out we assume that every property of x are conditionally independent. This means we can unbind $\mathbf{x}$ into it's properties:

$$p(x_{weight}|(Fe)Male)p(x_{length}|(Fe)Male)p(x_{shoe\_size}|(Fe)Male)$$

In our case as seen in my implementation we calculate the probability of a property of x within the class Male of Female by taking the Normal probability density function using the data from the database: biometrie2014.csv
The implementation for the classifier can be found in the $male\_or\_female$ function.


man
vrouw
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
160
Weight in Kg


man
vrouw
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
140
150
160
170
180
190
200
210
Height in cm



As you can see I tested using the first entry in the database, which correctly identifies as male.

Then i check every entry in the database for class and see we come out pretty okay.

	M	F
M	26	1
F	1	5