

NaiveBayes35

February 1, 2019

```
In [2]: ### CREATION DES ECHANTILLONS DE REFERENCE
```

```
import time as cl
import random as rd
import numpy as np
import pickle

def GenPermutation(n): # Création d'une permutation de [0,1,2, ..., n-1]
    L1 = list(range(n))
    L = []
    m = n
    for k in range(n):
        nouv = rd.randint(0,m-1)
        m -= 1
        L.append(L1.pop(nouv))
    return L

def PartitionHomogene(X,ident,p):
    # Utiliser la fonction VectorisationAmb pour avoir X et ident
    deb = 0
    nX = []
    nY = []
    nXn = []
    nYn = []
    n = 0
    for couple in ident:
        nbTextes = couple[1]
        TailleSample = int(nbTextes * p)
        L = GenPermutation(nbTextes)

        for k in range(TailleSample):
            nX.append(X[ deb+L[k] ])
            nY.append(n)

        for k in range(TailleSample,nbTextes):
```

```

        nXn.append(X[ deb + L[k] ])
        nYn.append(n)

    deb += nbTextes
    n+=1
    return nX, nY, nXn, nYn

def GenEchantillons(n,p,Xt,ident):
    Xtot = []
    c1=c1.clock()

    for k in range(n):
        nX,nY,nXn,nYn = PartitionHomogene(Xt,ident,p)
        Xtot.append((nX,nY,nXn,nYn))
    c2=c1.clock()
    print(c2-c1)

    return Xtot

def GenGamme(n,pas):
    Interv = np.linspace(0,1,pas)

    Banque=[]

    response = VectorisationAmb()
    Vec,ident = response
    X = []
    for vec in Vec:
        X.append(list(vec))
    Y = []

    for k in range(len(ident)):
        for i in range(ident[k][1]):
            Y.append(k)
    dim = 30
    Xt,pca = ReductionDim(X,dim)
    xt = []
    for a in Xt:
        xt.append(list(a))
    Xt = xt

    for p in Interv[1:(pas-1)]:
        Banque.append(GenEchantillons(n,p,Xt,ident))

    return Banque

```

```

##Banque = GenGamme(20,11)

def moyenne(X):
    n = len(X)
    tot = 0
    for x in X:
        tot+=x
    moy = tot/n
    variance = 0
    for x in X:
        elem = (moy-x)**2
        variance += elem/n
    ecartType = variance*(1/2)
    incertitude = ecartType/(n**(1/2))
    return moy,incertitude

# Extraction d'un fichier binaire
def readbinary(adresse):

    with open(adresse, "rb") as file:
        s = file.read()
    return s

def register(Banque,direction):
    serialBanque = pickle.dumps(Banque)

    fichiertxt = open(direction,mode="xb")
    fichiertxt.write(serialBanque)
    fichiertxt.close()

def recuperation(direction):
    c1 = cl.clock()

    serial_Banque= readbinary(direction)

    Banque= pickle.loads(serial_Banque)
    c2 = cl.clock()
    print(c2-c1)
    return Banque

##Banque = recuperation("Banque")

```

```
In [3]: DecisionTreeRes = recuperation("/Users/NAIT/classification/pact35/modules/C")
```

0.0018180000000000973

```
In [4]: # Import Library of Gaussian Naive Bayes model
        from sklearn.naive_bayes import GaussianNB

        import numpy as np
        import pylab as pl
        import random as rd

        # Partition de tous les vecteurs
        # Y est la liste correspondante aux ambiances des textes de X
        def partitionproba(X,Y,p):
            #X et Y des listes de même taille
            nX = []
            nY = []
            nXn = []
            nYn = []
            n = len(X)
            if len(Y) != n:
                raise Exception

            samp = []
            sampn = []
            for k in range(n):
                w = rd.random()
                if w < p:
                    nX.append(X[k])
                    nY.append(Y[k])
                    samp.append(k)
                else:
                    nXn.append(X[k])
                    nYn.append(Y[k])
                    sampn.append(k)
            return nX, nY, samp, nXn, nYn, sampn

        def efficl(X,Y,pas, iteration):
            P = np.linspace(0.4,0.8,pas)
            # P = P[1:(pas-1)]
            res = []
            for p in P:
                cl = cl.clock()
                T=[]
                for k in range(iteration):
                    nX, nY, samp, nXn, nYn, sampn = partitionproba(X,Y,p)
                    T.append(test(nX, nY, nXn, nYn))
```

```

        res.append(moyenne(T))
        c2 = c1.clock()
        print(c2-c1)
    pl.plot(P,res)
    pl.show()
    return P,res

```

Maintenant avec des échantillons d'entraînement homogènes.

```

def GenPermutation(n): # Création d'une permutation de [0,1,2, ..., n-1]
    L1 = list(range(n))
    L = []
    m = n
    for k in range(n):
        nouv = rd.randint(0,m-1)
        m -= 1
        L.append(L1.pop(nouv))
    return L

```

```

def PartitionHomogene(X,ident,p):
    # Utiliser la fonction VectorisationAmb pour avoir X et ident
    deb = 0
    nX = []
    nY = []
    nXn = []
    nYn = []
    n = 0
    for couple in ident:
        nbTextes = couple[1]
        TailleSample = int(nbTextes * p)
        L = GenPermutation(nbTextes)

        for k in range(TailleSample):
            nX.append(X[ deb+L[k] ])
            nY.append(n)

        for k in range(TailleSample,nbTextes):
            nXn.append(X[ deb + L[k] ])
            nYn.append(n)

        deb += nbTextes
        n+=1
    return nX, nY, nXn, nYn

```

```

def testNaiveBayes(nX, nY, nXn, nYn):
    #nX et nY les parties d'entraînement
    #nXn et nYn les parties de test
    model = GaussianNB()
    model.fit(nX,nY)
    n = len(nXn)
    if n == 0:
        return -1
    goal = 0
    failure = 0

    for k in range(n):
        prediction = model.predict([nXn[k]])
        if prediction[0] == nYn[k]:
            goal+=1
        else:
            failure +=1
    return goal/n

def moyenne(X):
    n = len(X)
    tot = 0
    for x in X:
        tot+=x
    moy = tot/n
    variance = 0
    for x in X:
        elem = (moy-x)**2
        variance += elem/n
    ecartType = variance*(1/2)
    incertitude = ecartType/(n**(1/2))
    return moy,incertitude

# It's measuring time
def effic2(X,ident,pas,iteration):
    P = np.linspace(0.2,0.5,pas)
    P = P[1:(pas-1)] # On enlève les cas triviaux 0 et 1
    res = []
    for p in P:
        c1 = cl.clock()
        T=[]
        for k in range(iteration):
            nX, nY, nXn, nYn = PartitionHomogene(X,ident,p)
            T.append(test(nX, nY, nXn, nYn))
        res.append(moyenne(T))
        c2 = cl.clock()
        print("Pour la proportion p = ", p , ", on met un temps de ", (c2-c1))

```

```

ResP = []
ResM = []
for couple in res:
    ResM.append(couple[0]-couple[1])
    ResP.append(couple[0]+couple[1])
pl.plot(P,ResP)
pl.plot(P,ResM)
pl.show()
return P,res

def efficNaiveBayes(Banque):
    P = np.linspace(0,1,11)
    P = P[1:10]      # On enlève les cas triviaux 0 et 1
    res = []
    OrdonneeP = []
    for i in range(len(P)):
        cl = cl.clock()
        EnsemblePartitionP = Banque[i]
        p =P[i]
        Z = []

        for (nX, nY, nXn, nYn) in EnsemblePartitionP:
            zi = testNaiveBayes(nX, nY, nXn, nYn)
            Z.append(zi)
        z,incertitude = moyenne(Z)
        OrdonneeP.append((z,incertitude))
        c2 = cl.clock()
        print("Pour la proportion p = ", p , ", on met un temps de ", (c2-cl.clock()))
    return P,OrdonneeP

#Pnb=np.array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
#Ordnb = [(0.22977137176938373, 5.395383878198053e-06), (0.2682190760059612

```

0.0.1 La fonction retourne une liste, notée NaiveBayesRes, qui est une liste de couples comportant le taux de réussite et un calcul d'incertitude; et ce pour chaque proportion p du DataTraining. À noter que ce classifieur ne nécessite pas de paramètre en entrée.

```
In [5]: NaiveBayesRes = [(0.22977137176938373, 5.395383878198053e-06), (0.2682190760059612,
```

```
In [6]: import numpy as np
import matplotlib.pyplot as plt
```

```

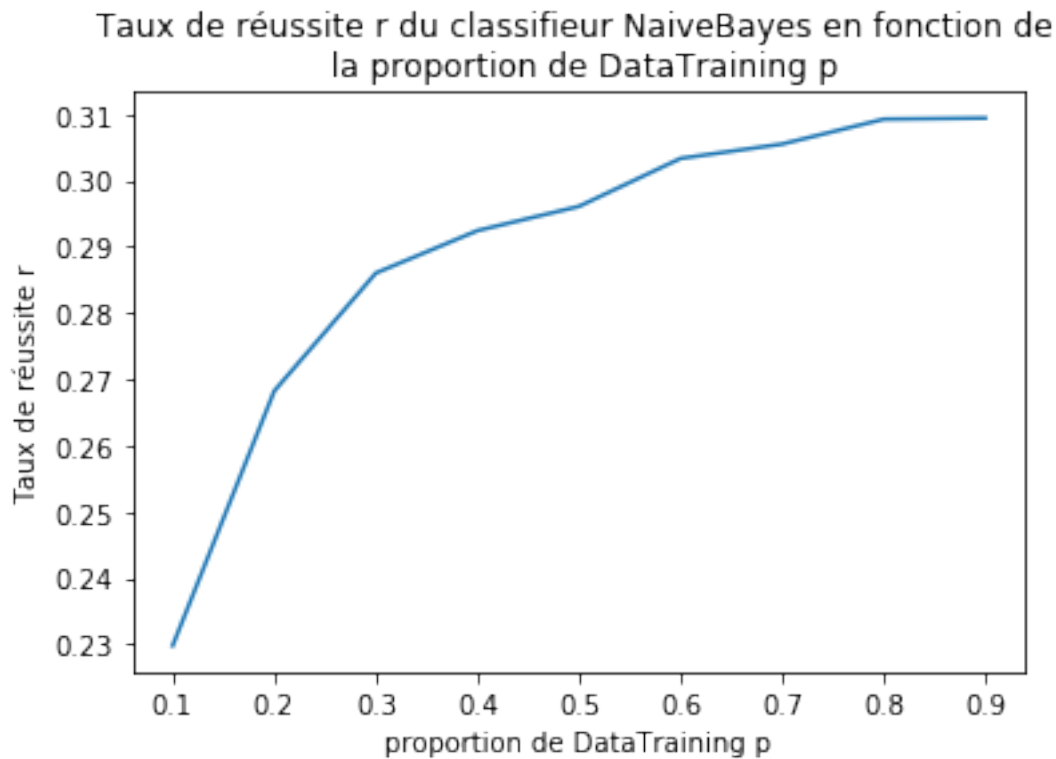
x = np.array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
y = np.array([NaiveBayesRes[i][0] for i in range(len(NaiveBayesRes))])

```

```

plt.plot(x, y)
plt.title("Taux de réussite r du classifieur NaiveBayes en fonction de \n p")
plt.xlabel("proportion de DataTraining p")
plt.ylabel("Taux de réussite r")
plt.savefig('CourbeNaiveBayesthiz.png')
plt.show() # affiche la figure a l'ecran

```



0.1 Détermination du maximum d'efficacité du Naive Bayes et des paramètres associés

```

In [7]: max = 0
        imax = 0
        k = 0
        for yi in y:
            k+=1
            if yi>max:
                max = yi
                imax = k
        print ("Efficacité maximale du classifieur par Naive Bayes = " + str(max*100))
        print ("obtenue pour une proportion de DataTraining p = " + str(imax*0.1))

```


Efficacité maximale du classifieur par Naive Bayes = 30.9428571429%
obtenue pour une proportion de DataTraining $p = 0.9$

0.2 CONCLUSION : Efficacité du classifieur Naive Bayes maximale, de maximum 30.9% de réussite avec une proportion de DataTraining $p = 0.9$

0.2.1 NB : À noter qu'on considère être une réussite le fait de renvoyer exactement l'ambiance du texte. Les rapprochements d'ambiance ne sont pas pris en compte. Notamment, on ne pondère pas selon si la deuxième ambiance trouvée se rapproche de celle souhaitée.

In [8]: *### Pour Test et Intégration*

```
def entraine(nX,nY):  
    model = GaussianNB()  
    model.fit(nX,nY)  
    return model  
  
def classifieUnTxt(monTextetxt,model): # Doit prendre 1 texte et 1 model en  
    return None
```