LinearSVC35

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=cl.clock()

    for k in range(n):
        nX,nY,nXn,nYn = PartitionHomogene(Xt,ident,p)
        Xtot.append((nX,nY,nXn,nYn))
    c2=cl.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]: resultatsSVC = recuperation("/Users/NAIT/classification/pact35/modules/Clas
```

```
0.0023740000000005423
```

In [4]:
```
### LinearSVC:

from sklearn.svm import LinearSVC
from sklearn.datasets import make_classification
import random as rd
import pylab as pl



def entraineSVC(nX,nY,c):
    model = LinearSVC(C=c, class_weight=None, dual=False, fit_intercept=Tru
    model.fit(nX,nY)
    return model


#C = la valeur de la marge (réel positif)
#dual = False
#random_state has no impact
#tol donne l'erreur  10^-5 près
#verbose is for output



def testSVC(nX, nY, nXn, nYn,c):
    #nX et nY les parties d'entraînement
    #nXn et nYn les parties de test
    model = entraineSVC(nX,nY,c)
    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 efficSVC(X,ident,p0,p1,pas,iteration):  # C constant
    abs = np.linspace(p0,p1,pas)

    # On enlève les cas triviaux pathologiques 0 et 1
    if p0 == 0:
        abs = abs[1:]
```

```
    if p1 == 1:
        abs = abs[:(pas-2)]
    res = []

    for p in abs:
        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-c
    pl.plot(abs,res)
    pl.show()
    return abs,res


def efficSVCpara(Banque):    # Tracé de la surface de efficacité(p,c)
    p0 = 0
    p1 = 1
    pas = 11
    P = np.linspace(0,1,11)
    P = list(P)
    P = P[1:10]
    AbscisseP = []
    K = []


    for i in range(len(P)):              # Proportion prise dans la biblioh
        p = P[i]
        EnsemblePartitionP = Banque[i]
        AbscisseC = []


        for n in range(100,1,-1): # C variation
            c1 = cl.clock()
            cn = 1/n
            K.append(cn)
            Z1 = []
            for (nX, nY, nXn, nYn) in EnsemblePartitionP:
                zi1 = testSVC(nX, nY, nXn, nYn,cn)
                Z1.append(zi1)
            z1,incertitude1 = moyenne(Z1)
            AbscisseC.append((z1,incertitude1))
            c2 = cl.clock()
            print("Pour p = " + str(p) + " et k = " + str(cn) + " on a pris
```

```
            for n in range(1,100): # C variation
                c1 = cl.clock()
                cn = n
                K.append(cn)
                Z2 = []
                for (nX, nY, nXn, nYn) in EnsemblePartitionP:
                    zi2 = testSVC(nX, nY, nXn, nYn,cn)
                    Z2.append(zi2)
                z2,incertitude2 = moyenne(Z2)
                AbscisseC.append((z2,incertitude2))
                c2 = cl.clock()
                print("Pour p = " + str(p) + " et k = " + str(cn) + " on a pris
            AbscisseP.append(AbscisseC)


        return P,K,AbscisseP


    #P=[0.10000000000000001, 0.20000000000000001, 0.30000000000000004, 0.400000
    #C=[0.01, 0.010101010101010102, 0.01020408163265306, 0.010309278350515464,
    #Z = recuperation(resultatsSVC)

    # Format de Z: une liste de 9 listes de 200 couples de floats
```

In [5]:
```
    #SVCRes = efficSVCpara(resultatsSVC)

    SVCRes =[0.091716368455931063, 0.091815772034459897, 0.091915175612988731,
```

In [6]:
```
    import matplotlib as mpl
    from pylab import *
    from mpl_toolkits.mplot3d import Axes3D
    import numpy as np
    import matplotlib.pyplot as plt


    x = np.array([ 0.1]*198 + [ 0.2]*198 + [ 0.3]*198+ [ 0.4]*198+ [ 0.5]*198+
    print(len(x))
    y = np.array([0.01, 0.010101010101010102, 0.01020408163265306, 0.0103092783
    *9)
    print(len(y))
    z = np.array(SVCRes)
    print(len(z))

    fig = plt.figure()
    ax = fig.gca(projection='3d')

    plt.title("Taux de réussite r du classifieur LinearSVC en fonction \n de la
    ax.set_xlabel('proportion p')
```

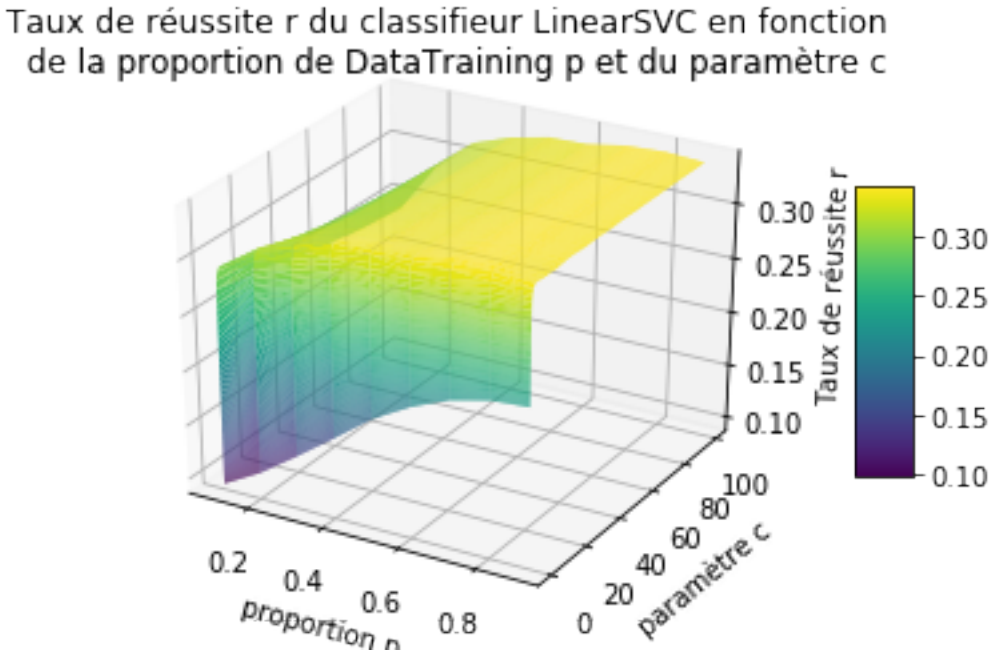
```
        ax.set_ylabel('paramètre c')
        ax.set_zlabel('Taux de réussite r')

        # to Add a color bar which maps values to colors.
        surf=ax.plot_trisurf(x, y, z, cmap=plt.cm.viridis, linewidth=0.2)
        fig.colorbar( surf, shrink=0.5, aspect=5)
        plt.savefig('CourbeSVCthiz.png')
        plt.show()

        # Rotate it
        ax.view_init(30, 45)
        plt.show()

        # Other palette
        ax.plot_trisurf(y, x, z, cmap=plt.cm.jet, linewidth=0.01)
        plt.show()
```

```
1782
1782
1782
```



0.1 Détermination du maximum d'efficacité du linear SVC et des paramètres associés

```
In [7]: zmax = 0
        imax = 0
```

```
        k = 0
        for zi in z:
            k+=1
            if zi>zmax:
                zmax = zi
                imax = k
        pmax = x[imax]
        kmax = y[imax]
        print ("Efficacité maximale du classifieur linearSVC = " + str(zmax*100) +
        print("obtenue pour un paramètre c = " + str(kmax))
        print("obtenue pour une proportion de DataTraining p = " + str(pmax))
```

```
Efficacité maximale du classifieur linearSVC = 34.2279411765%
obtenue pour un paramètre c = 90.0
obtenue pour une proportion de DataTraining p = 0.8
```

0.2 CONCLUSION : Éfficacité du classifieur linear SVC maximale, de maximum 34.22% de réussite avec un paramètre c = 90 et une proportion de DataTraining p = 0.8

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.