

KNN35

February 1, 2019

```
In [3]: ### 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 [4]: KNNRes = recuperation("/Users/NAIT/classification/pact35/modules/Classifica

```

```

0.00203299999999999515

```

```
In [5]: from sklearn.neighbors import KNeighborsClassifier
import random as rd
import pylab as pl
```

```
def entraineKNN(nX,nY,n):
    model = KNeighborsClassifier(n_neighbors=n)
    model.fit(nX,nY)
    return model
```

```
def testKNN(nX, nY, nXn, nYn, k):
    #nX et nY les parties d'entraînement
    #nXn et nYn les parties de testKNN
    # k est le nombre de proches voisins
    model = entraineKNN(nX,nY,k)
    n = len(nXn)
    if n == 0:
        return -1
    goal = 0
    failure = 0

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

```
def effickNNpara(Banque):
    p0 = 0
    p1 = 1
    pas = 11
    P = np.linspace(0,1,11)
    P = list(P)
    P = P[1:10]
    K = []
    AbscisseP = []
```

```
for k in range(22):
    n_components = 2*k+1
    K.append(n_components)
```

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

```

    for k in range(22): # n_components variation
        c1 = cl.clock()
        n_components = 2*k + 1
        Z = []
        for (nX, nY, nXn, nYn) in EnsemblePartitionP:
            zi = testKNN(nX, nY, nXn, nYn, n_components)
            Z.append(zi)
        z,incertitude = moyenne(Z)
        AbscisseNComponents.append((z,incertitude))
        c2 = cl.clock()
        print("Pour n_components = " + str(n_components) + " et p = " + str(p))
        AbscisseP.append(AbscisseNComponents)

    return AbscisseP,P,K

#####

def efficKNN(X,ident,p0,p1,pas,iteration):
    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(testKNN(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(abs,ResP)
    pl.plot(abs,ResM)
    pl.show()
    return abs,res

```

```
#P=[0.10000000000000000001, 0.20000000000000000001, 0.30000000000000000004, 0.40000000000000000007, 0.5000000000000000001, 0.60000000000000000015, 0.7000000000000000002, 0.80000000000000000025, 0.9000000000000000003, 1.00000000000000000035, 1.1000000000000000004, 1.20000000000000000045, 1.3000000000000000005, 1.40000000000000000055, 1.5000000000000000006, 1.60000000000000000065, 1.7000000000000000007, 1.80000000000000000075, 1.9000000000000000008, 2.00000000000000000085, 2.1000000000000000009, 2.20000000000000000095, 2.300000000000000001, 2.40000000000000000105, 2.5000000000000000011, 2.60000000000000000115, 2.7000000000000000012, 2.80000000000000000125, 2.9000000000000000013, 3.00000000000000000135, 3.1000000000000000014, 3.20000000000000000145, 3.3000000000000000015, 3.40000000000000000155, 3.5000000000000000016, 3.60000000000000000165, 3.7000000000000000017, 3.80000000000000000175, 3.9000000000000000018, 4.00000000000000000185, 4.1000000000000000019, 4.20000000000000000195, 4.300000000000000002, 4.40000000000000000205, 4.5000000000000000021, 4.60000000000000000215, 4.7000000000000000022, 4.80000000000000000225, 4.9000000000000000023, 5.00000000000000000235, 5.1000000000000000024, 5.20000000000000000245, 5.3000000000000000025, 5.40000000000000000255, 5.5000000000000000026, 5.60000000000000000265, 5.7000000000000000027, 5.80000000000000000275, 5.9000000000000000028, 6.00000000000000000285, 6.1000000000000000029, 6.20000000000000000295, 6.300000000000000003, 6.40000000000000000305, 6.5000000000000000031, 6.60000000000000000315, 6.7000000000000000032, 6.80000000000000000325, 6.9000000000000000033, 7.00000000000000000335, 7.1000000000000000034, 7.20000000000000000345, 7.3000000000000000035, 7.40000000000000000355, 7.5000000000000000036, 7.60000000000000000365, 7.7000000000000000037, 7.80000000000000000375, 7.9000000000000000038, 8.00000000000000000385, 8.1000000000000000039, 8.20000000000000000395, 8.300000000000000004, 8.40000000000000000405, 8.5000000000000000041, 8.60000000000000000415, 8.7000000000000000042, 8.80000000000000000425, 8.9000000000000000043, 9.00000000000000000435, 9.1000000000000000044, 9.20000000000000000445, 9.3000000000000000045, 9.40000000000000000455, 9.5000000000000000046, 9.60000000000000000465, 9.7000000000000000047, 9.80000000000000000475, 9.9000000000000000048, 10.00000000000000000485, 10.1000000000000000049, 10.20000000000000000495, 10.300000000000000005, 10.40000000000000000505, 10.5000000000000000051, 10.60000000000000000515, 10.7000000000000000052, 10.80000000000000000525, 10.9000000000000000053, 11.00000000000000000535, 11.1000000000000000054, 11.20000000000000000545, 11.3000000000000000055, 11.40000000000000000555, 11.5000000000000000056, 11.60000000000000000565, 11.7000000000000000057, 11.80000000000000000575, 11.9000000000000000058, 12.00000000000000000585, 12.1000000000000000059, 12.20000000000000000595, 12.300000000000000006, 12.40000000000000000605, 12.5000000000000000061, 12.60000000000000000615, 12.7000000000000000062, 12.80000000000000000625, 12.9000000000000000063, 13.00000000000000000635, 13.1000000000000000064, 13.20000000000000000645, 13.3000000000000000065, 13.40000000000000000655, 13.5000000000000000066, 13.60000000000000000665, 13.7000000000000000067, 13.80000000000000000675, 13.9000000000000000068, 14.00000000000000000685, 14.1000000000000000069, 14.20000000000000000695, 14.300000000000000007, 14.40000000000000000705, 14.5000000000000000071, 14.60000000000000000715, 14.7000000000000000072, 14.80000000000000000725, 14.9000000000000000073, 15.00000000000000000735, 15.1000000000000000074, 15.20000000000000000745, 15.3000000000000000075, 15.40000000000000000755, 15.5000000000000000076, 15.60000000000000000765, 15.7000000000000000077, 15.80000000000000000775, 15.9000000000000000078, 16.00000000000000000785, 16.1000000000000000079, 16.20000000000000000795, 16.300000000000000008, 16.40000000000000000805, 16.5000000000000000081, 16.60000000000000000815, 16.7000000000000000082, 16.80000000000000000825, 16.9000000000000000083, 17.00000000000000000835, 17.1000000000000000084, 17.20000
```

0.0.1 La fonction retourne une liste, notée **KNNRes**, qui est une liste de listes de couples comportant le taux de réussite et un calcul d'incertitude; ce pour chaque valeur de paramètre de voisins **k**; et ce pour chaque proportion **p** du **DataTraining**. À noter que ce classifieur est donc testé non seulement selon la proportion de **DataTraining** utilisé mais aussi du paramètre d'entrée du classifieur.

```
In [2]: KNNRes = [(0.19713386348575213, 8.133905069254739e-06), (0.183167660702451
KNNResu = []
for i in range(len(KNNRes)):
    for j in range(len(KNNRes[0])):
        KNNResu.append(KNNRes[i][j][0])
print(KNNResu)

[0.19713386348575213, 0.18316766070245197, 0.20226971504307492, 0.21194499668654734
```

```
In [3]: 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]*22 + [ 0.2]*22 + [ 0.3]*22+ [ 0.4]*22+ [ 0.5]*22+ [ 0.6]*22)
print(x)
print(len(x))
y = np.array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45])
print(y)
print(len(y))
z = np.array(KNNResu)
print(z)
print(len(z))

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

plt.title("Taux de réussite r du classifieur KNN en fonction de \n la proportion p et le paramètre k")
ax.set_xlabel('proportion p')
ax.set_ylabel('paramètre k')
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('CourbeKNNthis.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()

```

```

[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.3
  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3
  0.3  0.3  0.3  0.3  0.3  0.3  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4
  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.5  0.5
  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5
  0.5  0.5  0.5  0.5  0.5  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6
  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.7  0.7  0.7
  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7
  0.7  0.7  0.7  0.7  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8
  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.9  0.9  0.9  0.9
  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9
  0.9  0.9  0.9]

```

198

```

[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43  1  3  5
  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43  1  3  5  7  9 11
 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43  1  3  5  7  9 11 13 15 17
 19 21 23 25 27 29 31 33 35 37 39 41 43  1  3  5  7  9 11 13 15 17 19 21 23
 25 27 29 31 33 35 37 39 41 43  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29
 31 33 35 37 39 41 43  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35
 37 39 41 43  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41
 43  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43]

```

198

```

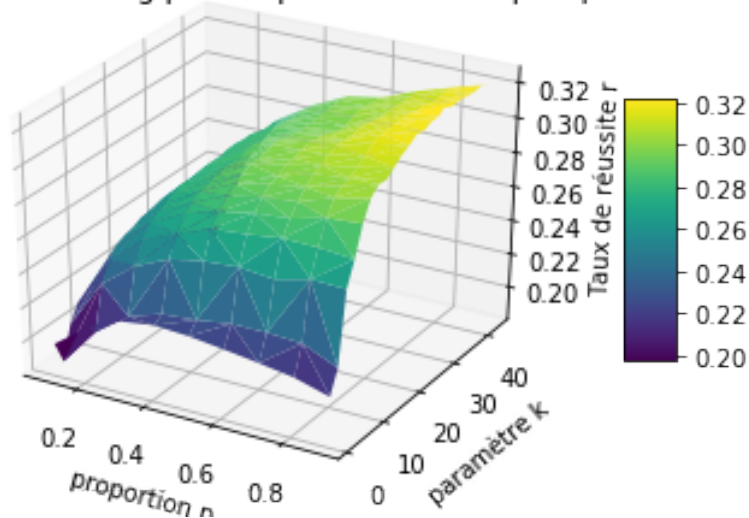
[ 0.19713386  0.18316766  0.20226972  0.211945    0.21494367  0.21491054
  0.2140159   0.2147283   0.21449636  0.21245858  0.21164679  0.21174619
  0.20997349  0.20758781  0.20611332  0.20463883  0.20311465  0.20213718
  0.19971836  0.19754805  0.19597416  0.19441683  0.21181073  0.20586811
  0.23137109  0.24267884  0.24793219  0.25014903  0.2514158   0.25385618
  0.25480626  0.25584948  0.25700447  0.25611028  0.25413562  0.25326006
  0.25298063  0.25230999  0.25255216  0.25240313  0.25137854  0.25
  0.24806259  0.24662817  0.22065079  0.22033177  0.24629945  0.25867716
  0.2653339   0.26875798  0.27071459  0.27320289  0.27518077  0.27447895]

```

0.27501063	0.27639302	0.27649936	0.27645683	0.27764781	0.27792429
0.27728626	0.27749894	0.2758826	0.27513824	0.27558486	0.27522331
0.22353671	0.22673611	0.25533234	0.26654266	0.27490079	0.27842262
0.28157242	0.28323413	0.2843998	0.28546627	0.28576389	0.28504464
0.28777282	0.28829365	0.28735119	0.2875744	0.28888889	0.28874008
0.28819444	0.28851687	0.28831845	0.28764881	0.22221228	0.22829457
0.25903399	0.27429934	0.28085868	0.28664281	0.2905486	0.29284436
0.29555754	0.29534884	0.29680978	0.29800239	0.29749553	0.29603459
0.29570662	0.29522958	0.29600477	0.29612403	0.2966607	0.29686941
0.29758497	0.29698867	0.21973294	0.23126855	0.26042285	0.27585312
0.28468101	0.29050445	0.29469585	0.29617953	0.29962908	0.30029674
0.30196588	0.30263353	0.30192878	0.30170623	0.29966617	0.29821958
0.29833086	0.3009273	0.30207715	0.30126113	0.30152077	0.3004822
0.21691249	0.23121927	0.25835792	0.27841691	0.28618486	0.29262537
0.29768928	0.30211406	0.30373648	0.30717797	0.30889872	0.30899705
0.30860374	0.30968535	0.30968535	0.30875123	0.30806293	0.30791544
0.30816126	0.30757129	0.30806293	0.30835792	0.21375	0.23014706
0.26330882	0.27794118	0.29139706	0.29948529	0.30294118	0.30426471
0.30727941	0.31029412	0.31448529	0.31661765	0.31683824	0.31705882
0.31669118	0.31713235	0.31544118	0.31617647	0.31411765	0.31426471
0.31507353	0.31463235	0.20785714	0.22114286	0.26257143	0.27614286
0.29271429	0.30357143	0.30828571	0.30757143	0.31128571	0.31528571
0.31771429	0.321	0.32	0.32228571	0.32228571	0.326
0.32257143	0.32314286	0.32228571	0.32185714	0.32271429	0.32057143]

198

Taux de réussite r du classifieur KNN en fonction de la proportion de DataTraining p et du paramètre k de plus proches voisins



0.1 Détermination du maximum d'efficacité du KNN et des paramètres associés

```
In [8]: 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 par KNN = " + str(zmax*100) + "%")
        print ("obtenue pour un paramètre de plus proches voisins k = " + str(kmax))
        print ("obtenue pour une proportion de DataTraining p = " + str(pmax))
```

```
Efficacité maximale du classifieur par KNN = 32.6%
obtenue pour un paramètre de plus proches voisins k = 33
obtenue pour une proportion de DataTraining p = 0.9
```

0.2 CONCLUSION : Efficacité du classifieur KNN maximale, de maximum 32.6% de réussite avec un paramètre de plus proches voisins k = 33 et 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.