Rendu évaluation de Classification

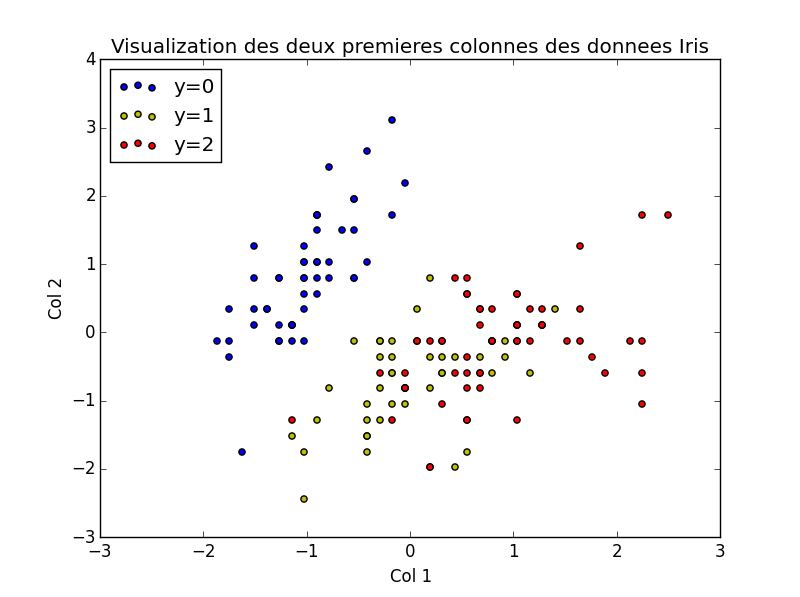
Aliréza BANAEI

1. Pour les méthodes mentionnées faire un tableau avec les renseignements suivants:

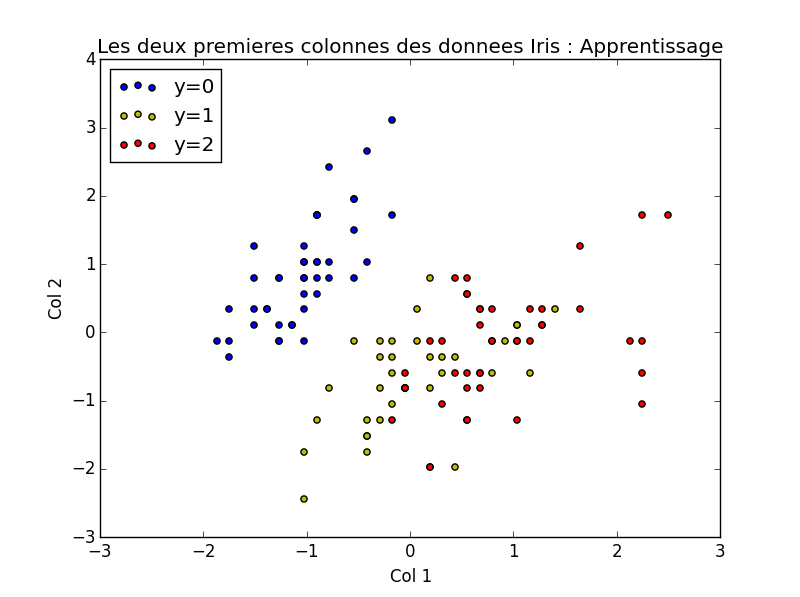
* temps de calcul en seconde pris par chaque méthode pour la partie apprentissage (pour l’entraînement sur les 80% des données)
* temps de calcul en seconde pris par chaque méthode pour la partie validation (sur les 20% restants)
* pourcentage d’erreurs de classification de chaque méthode

Dans un premier temps on explore les données en les visualisant :

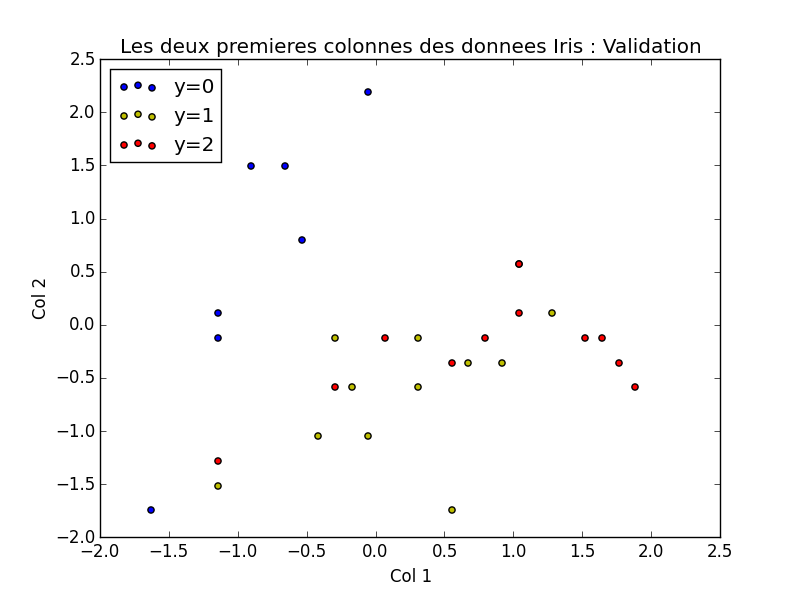
Toutes les données :



Es données d'apprentissage (80% des données sélectionnées aléatoirement) :



Et les 20% restant constituant les données de validation :



Et voici les résultats pour les différents classifieurs :

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Fit time** | **Validation Time** | **Errors** |
| **GaussianNB** | 0. | 0.00099993 | 0.26666667 |
| **LDA** | 0.00100017 | 0. | 0.26666667 |
| **Logistic regression** | 0.00099993 | 0. | 0.26666667 |
| **QDA** | 0.00099993 | 0.00099993 | 0.3 |
| **KNN** | 0. | 0.00100017 | 0.3 |
| **KNN with CV** | 0. | 0.00899982 | 0.26890119 |
| **SVM** | 0.00100017 | 0.00099993 | 0.3 |

2. On affichera les matrices de confusion associées : celles de la meilleure et de la pire des méthodes obtenues (au sens du nombre d’erreurs commises) parmi celles étudiées. Commentez vos résultats.

Selon le tableau ci-dessus les 3 méthodes Naive Bais, LDA et Logistic Regression se partagent les meilleurs scores (erros = 0.26) tandis que KNN, KNN avec cross-validation et SVM se partagent les pires (erros=0.3). Et QDA se situe au milieu. Il est donc difficile de trouver une meilleure et une pire. J'affiche donc toutes les matrices de confusion. Ces résultats sont calculés (avec l'apprentissage sur les 80%) sur les 20% restants.

mc\_gaussainNB

array([[ 6., 1., 0.],

[ 0., 8., 3.],

[ 0., 4., 8.]])

mc\_lda

array([[ 6., 1., 0.],

[ 0., 7., 4.],

[ 0., 3., 9.]])

mc\_lr

array([[ 6., 1., 0.],

[ 0., 7., 4.],

[ 0., 3., 9.]])

mc\_qda

array([[ 6., 1., 0.],

[ 0., 7., 4.],

[ 0., 4., 8.]])

mc\_knn

array([[ 6., 1., 0.],

[ 0., 8., 3.],

[ 0., 5., 7.]])

mc\_svm

array([[ 6., 1., 0.],

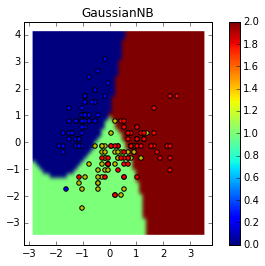
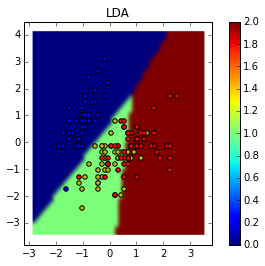
[ 0., 6., 5.],

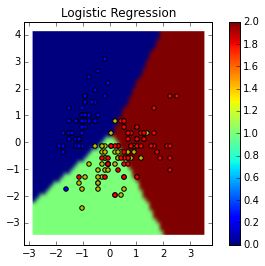
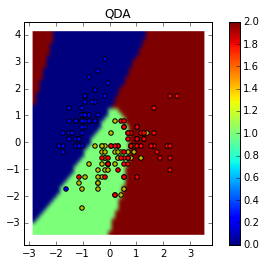
[ 0., 3., 9.]])

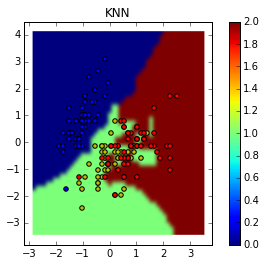
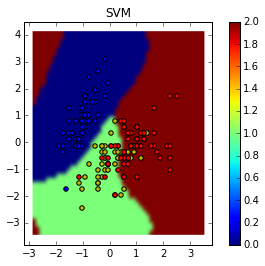
Les résultats des différents classifieurs sont assez proches ! Je ne sais comment intérpréter cela !

3. Proposer un (court) paragraphe synthétisant l’ensemble de vos expériences ci-dessus.

Les frontières de décision calculées par chacune des méthodes sont les suivantes :

Ces illustrations montrent des résultats (classifications) très différents en fonction des algorithmes utilisés, mais en termes de performances (erreurs), ils sont tous plus ou moins au même niveau. Cela provient probablement de la répartition des observations qui sont très imbriquées (surtout les rouges et les jaunes) et probablement aucune méthode ne pourrait classer ces observations mieux que ça. Cela montre les limites des algorithmes de classification face aux observations de la vie réelle.

4. Écrire une fonction qui affiche le graphe (tridimensionnel) d’une densité de gaussienne en dimension deux et qui prend en entrée un vecteur de moyenne …

Voici le code (voir le fichier .py joint) :

from numpy.linalg import inv

import math as math

from matplotlib.backends.backend\_pdf import PdfPages

# Calcul de la densite de probabilite de gaussienne

def calcul\_pd(x, y, mean, cov):

x\_mean = np.array([x-mean[0], y-mean[1]])

cov\_inv = inv(cov)

x\_mean\_T = x\_mean.T

r = math.exp(-0.5 \* np.dot(np.dot(x\_mean\_T, cov\_inv), x\_mean))

return r/math.sqrt(((2\*math.pi)\*\*2)\*np.linalg.det(cov))

# Trace la courbe en 3d et sauvegarde en pdf

def trace\_courbe\_save\_pdf(mean, cov, fileName):

if (cov == cov.T).all():

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

x = y = np.arange(-3.0, 3.0, 0.02)

Xs, Ys = np.meshgrid(x, y)

zs = np.array([calcul\_pd(xx,yy, mean, cov) for xx,yy in zip(np.ravel(Xs), np.ravel(Ys))])

Zs = zs.reshape(Xs.shape)

ax.plot\_surface(Xs, Ys, Zs)

ax.set\_xlabel('X1')

ax.set\_ylabel('X2')

ax.set\_zlabel('P')

ax.view\_init(elev=30., azim=120)

plt.show()

pp = PdfPages(fileName)

pp.savefig(fig)

pp.close()

else:

print 'Erreur : Matrice de covariance symetriques'

mean = np.array([0, 0])

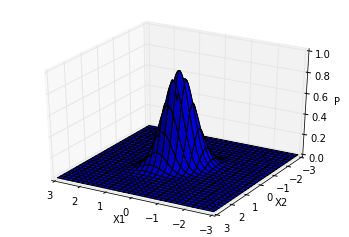
cov = np.array(([0.2, 0.1], [0.1, 0.2]))

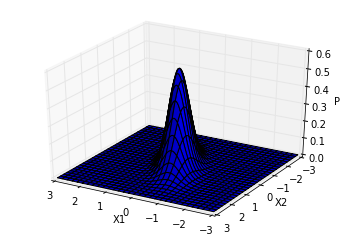
trace\_courbe\_save\_pdf(mean, cov, 'gausse\_1.pdf')

cov = np.array(([.2, -.2], [-.2, .6]))

trace\_courbe\_save\_pdf(mean, cov, 'gausse\_2.pdf')

Les résultats pour les deux matrices de covarience:





5. Enregistrer par lignes de commande les ﬁgures au format pdf. Veiller à ce que la taille de chaque image en pdf soit plus petit que 400ko. On ajoutera ces deux pdf au ﬁchier déposé.

Les fichiers pdf sont joints.