Rendu évaluation de Classification

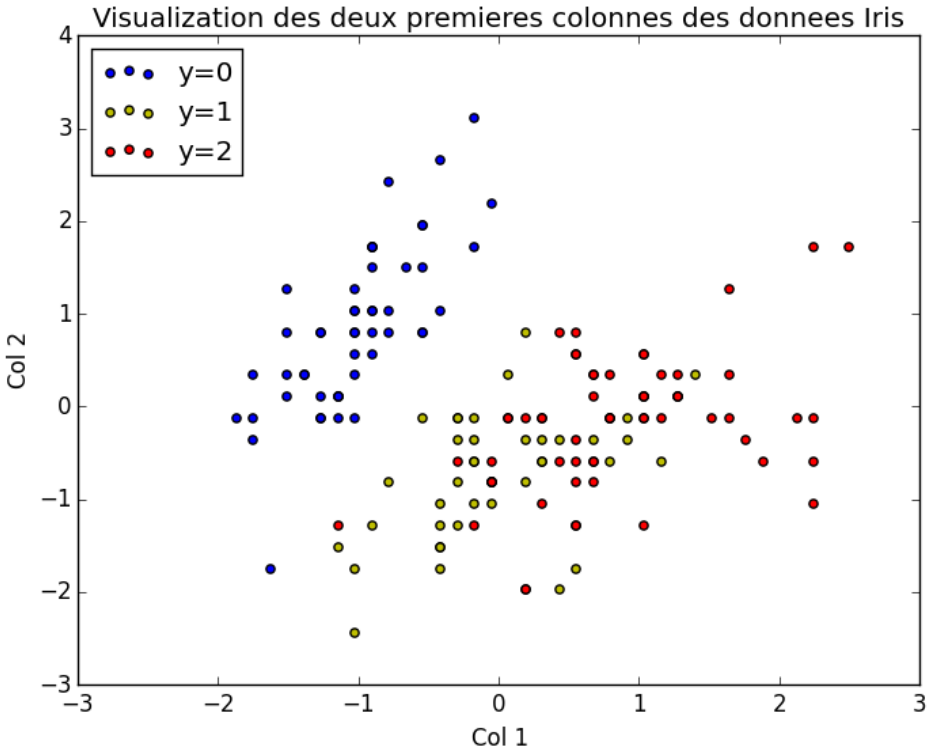
Aliréza BANAEI

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

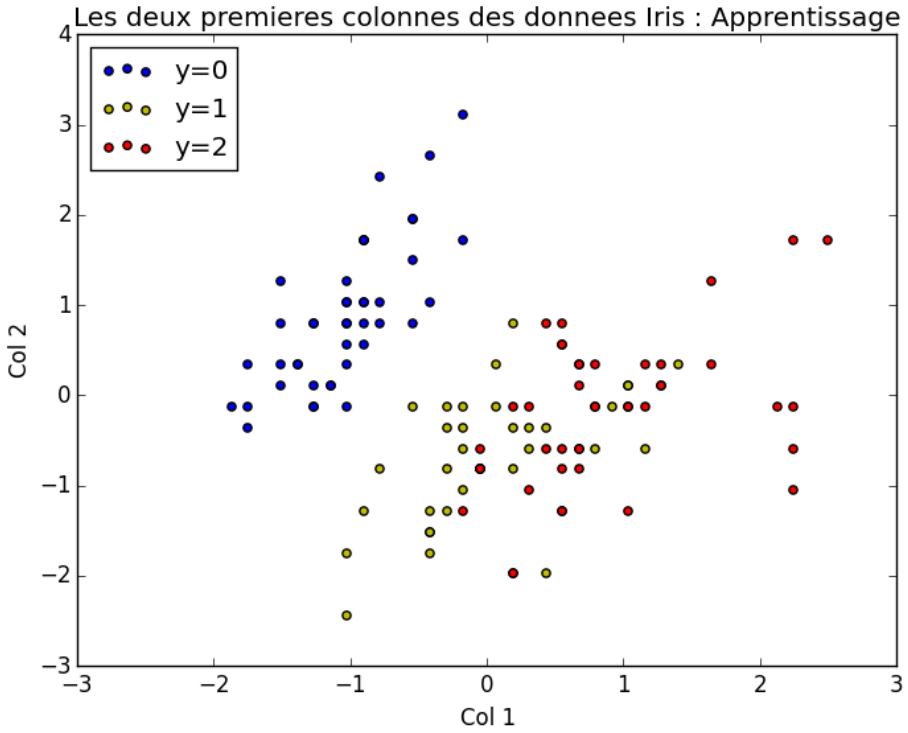
- **temps de calcul en seconde pris par chaque méthode pour la partie apprentissage (pour l'entraînement sur les 80% des données)**
- **temps de calcul en seconde pris par chaque méthode pour la partie validation (sur les 20% restants)**
- **pourcentage d'erreurs de classification de chaque méthode**

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

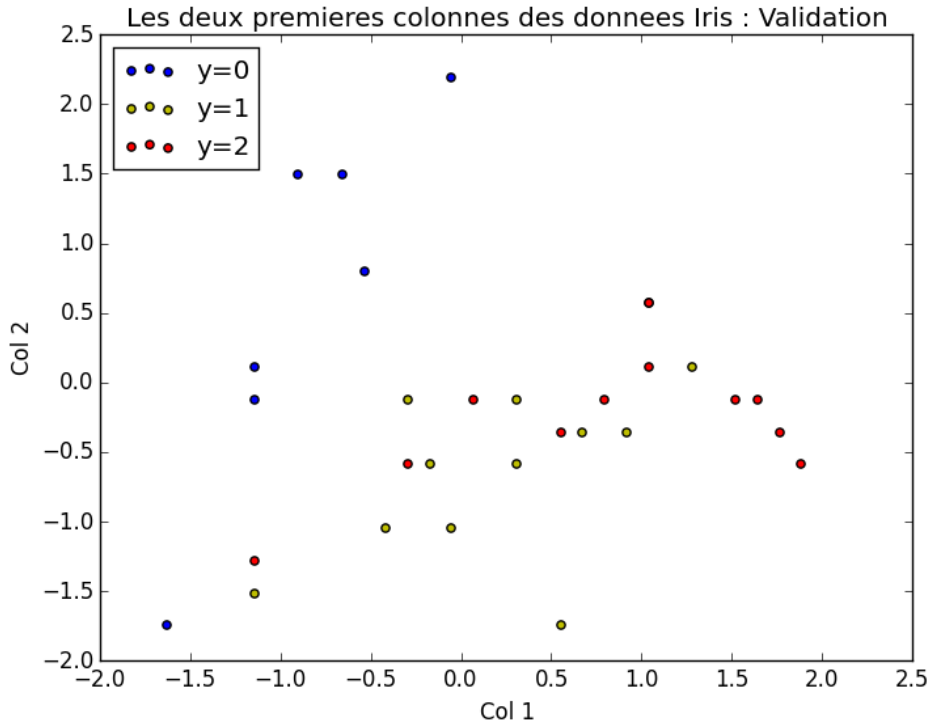
Toutes les données :



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.

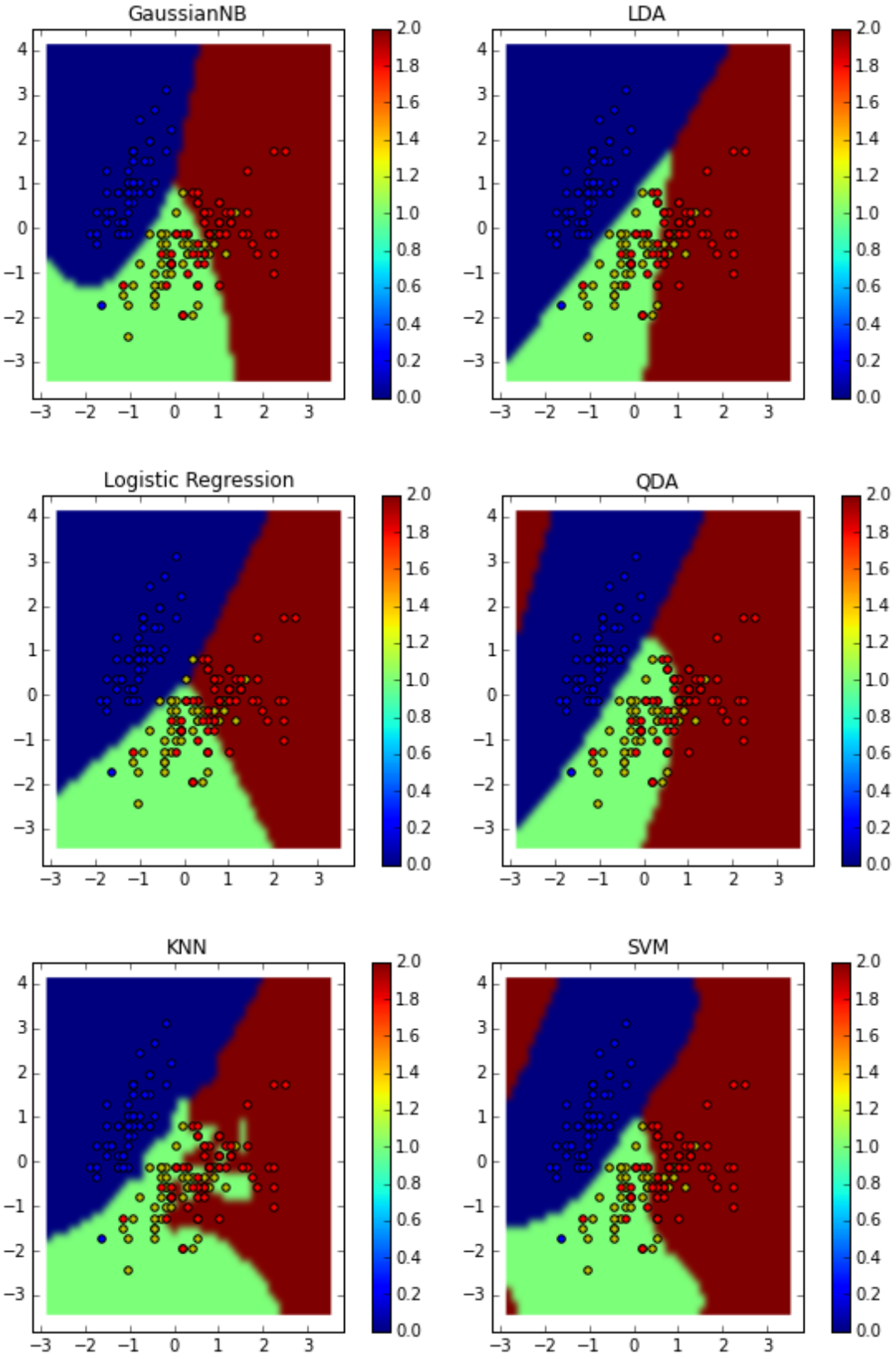
Selon le tableau ci-dessus les 3 méthodes Naive Bais, LDA et Logistic Regression se partagent les meilleurs scores (erros = 0.26) tandis que KNN, KNN avec cross-validation et SVM se partagent les pires (erros=0.3). Et QDA se situe au milieu. Il est donc difficile de trouver une meilleure et une pire. 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 :



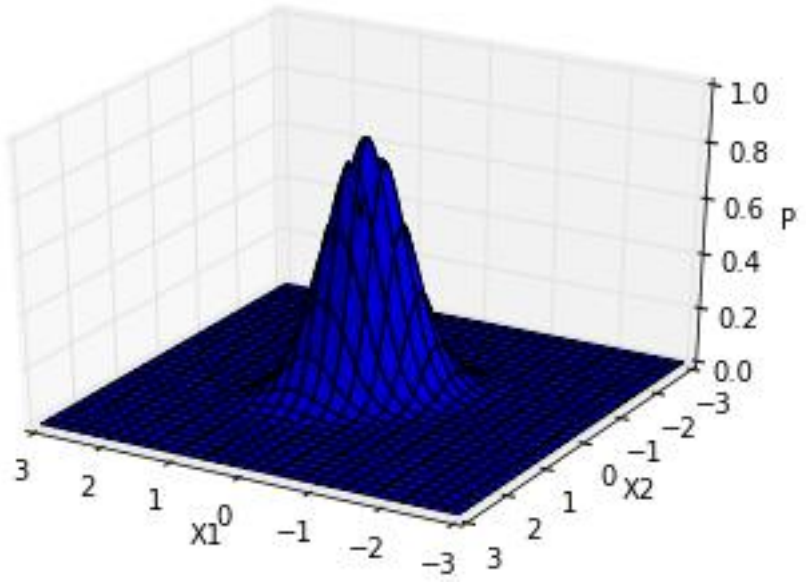
Ces illustrations montrent des résultats (classifications) très différents en fonction des algorithmes utilisés, mais en termes de performances (erreurs), ils sont tous plus ou moins au même niveau. Cela provient probablement de la répartition des observations qui sont très imbriquées (surtout les rouges 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.

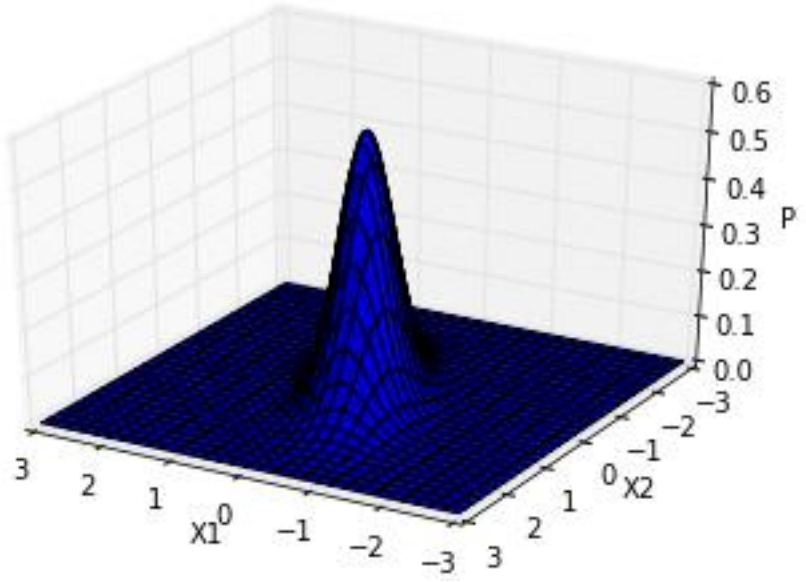
4. Écrire une fonction qui affiche le graphe (tridimensionnel) d'une densité de gaussienne en dimension deux et qui prend en entrée un vecteur de moyenne …

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.