M1 Informatique - UE ARF.
Rapport des TME.

Treü Marc, Karmim Yannis
Spécialité DAC.

31 mars 2019

Table des matières

Pour une meilleure lisibilité tout nos morceaux de codes pertinents se trouvent en annexe.

1 TME 1 : Arbres de décision, sélection de modèles.

Sur ce TME on a travaillé sur les arbres de décision et la classification à partir de la IMDB.
On a dû étudier comment fonctionne ces arbres, en particulier la façon dont on les génère et on choisi nos variables de décisions à l'aide de l'entropie. Puis on a étudié les impacts des hyper-paramètres sur le taux d'erreur en apprentissage et en test pour évaluer le sur et sous apprentissage.
Enfin on a implémenté de la validation croisé pour calibrer ce sur-sous apprentissage.

1.1 Quelques expériences préliminaires.

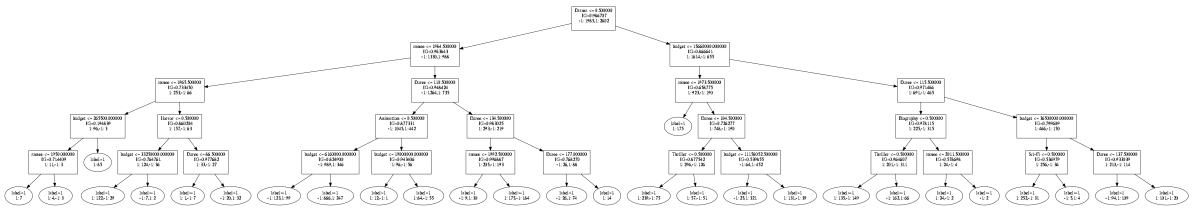

FIGURE 1 – Exemple d'arbre généré à l'aide du code fourni et de pydot, de profondeur 5 et de score 73% de bonne classification en apprentissage

Plus on descend dans un arbre de décision moins il y a d'exemples à séparer dans les noeuds . Ce qui est normal puisque à chaque noeud de notre arbre, on divise notre base d'exemples à l'aide des critères de décision des noeuds supérieurs.
Si l'on test uniquement sur nos données d'apprentissage, on remarque que lorsqu'on augmente la profondeur, les scores de bonnes classifications augmentent.
$profondeur = 5$: 73.6% de bonne classification.
$profondeur = 15$: 88.2% de bonne classification.
$profondeur = 20$: 89.8% de bonne classification.

C'est normal que ces scores augmentent puisque notre arbre de décision tend à s'adapter parfaitement à nos données d'apprentissage. Mais il faut être vigilant au sur-apprentissage. Donc le score sur les données d'apprentissage n'est pas un indicateur fiable du comportement.
Il faut alors tester notre modèle sur des données que l'on a pas encore vu, on

peut par exemple partitionner nos données en apprentissage et en test.

1.2 Sur et sous apprentissage.

Nous avons crée une fonction de partitionnement en test et apprentissage, et avons testé ce partitionnement sur plusieurs profondeurs d'arbres.
Voici plusieurs graphiques montrant nos taux d'erreurs en apprentissage et en test en fonction de la profondeur de l'arbre et chaque graphique représente un partitionnement différent.
Les courbes en pointillés représentent les erreurs en test.
Les courbes lisse représentent les erreurs en apprentissage.

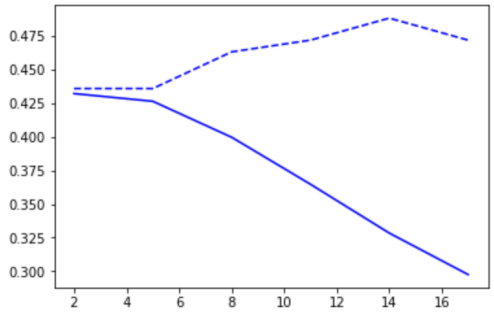

FIGURE 2 – Taux d'erreur en fonction de la profondeur pour un partitionnement avec 80% des données en apprentissage

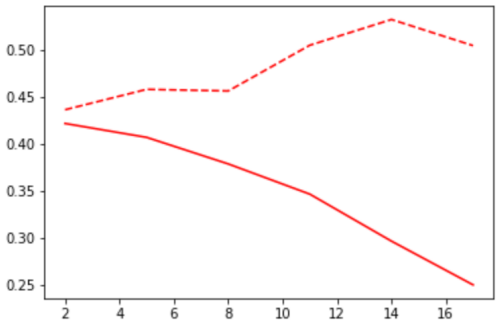

FIGURE 3 – Taux d'erreur en fonction de la profondeur pour un partitionnement avec 50% des données en apprentissage

On remarque que plus la profondeur est grande, moins l'on a d'erreur en apprentissage, comme on l'a expliqué c'est tout a fait normal puisque notre arbre va tendre à s'adapter parfaitement aux données apprises.

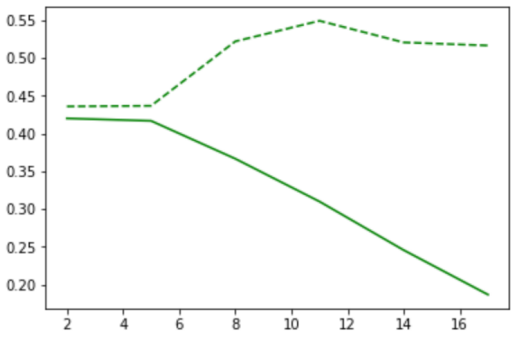

FIGURE 4 – Taux d'erreur en fonction de la profondeur pour un partitionnement avec 20% des données en apprentissage

Cependant on remarque que cette tendance est inversé pour l'erreur en test, qui elle augmente plus la profondeur est grande. C'est dû au sur-apprentissage des données qui a conduit notre arbre à avoir un modèle uniquement adapté aux données apprises au détriment d'un modèle plus général.
De même pour un partitionnement avec peu de données en apprentissage, notre modèle sous apprend ce qui conduit à plus d'erreurs dans le test.

1.3 Validation croisée : sélection de modèle.

La problématique que l'on rencontre maintenant est comment calibrer notre classifieur et nos paramètres pour éviter le sur et sous apprentissage afin d'avoir les meilleurs scores possibles.
On peut utiliser la méthode de la validation croisée ou *cross-validation* qui consiste à diviser nos données en K blocs de données. Ces blocs servent à entrainer notre modèle, c'est à dire à apprendre sur nos données et les différents paramètres comme la profondeur de l'arbre par exemple.
Puis l'on garde un de ces blocs pour tester notre modèle.
À la fin, on sélectionne le meilleur modèle, c'est à dire celui qui a produit le moins d'erreur en test.
Ci dessous notre code implémenté pour la validation croisée.

2 TME 2 : Estimation de denisté.

Les données ici sont des points d'intérêts de la région parisienne, par exemple bar, restaurants, distributeurs etc donnés à une certaine localisation.
L'objectif de nos algorithmes était d'éstimer la densité des ces POI à différentes localisation.
Différentes méthodes et algorithmes ont été utilisés, comme la méthode des histogrammes et la méthode des noyaux.

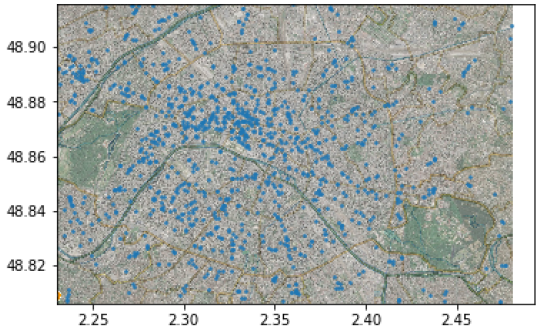

FIGURE 5 – Localisation de tout les ATM à Paris et aux alentours.

2.1 Méthode des histogrammes.

Notre premier modèle est la méthode des histogrammes, qui consiste à discrétiser notre espace puis à compter les localisations qui tombent dans chaque partis de l'espace.

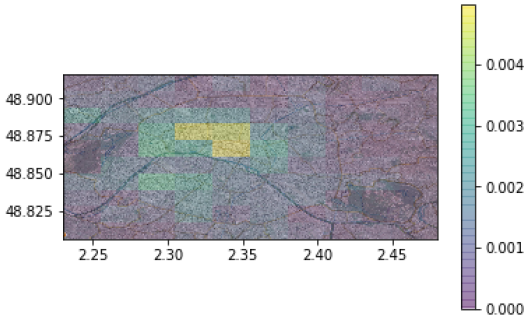

FIGURE 6 – Estimation de densité par histogramme sur les ATM de la Figure 5 avec un pas de discrétisation de 5.

Désormais, en donnant un nouveau point on peut prédire la densité d'ATM aux alentours.

```
In [9]: H.predict((48.850,2.32))
The density at this point  (48.85, 2.32)  is  0.001176470588235294
Out[9]: 0.001176470588235294
```

FIGURE 7 – Prédiction de densité sur un nouveau point avec la méthode des histogrammes.

Les limites de la méthode des histogrammes sont que si l'on définit des secteurs trop grand on risque de biaiser notre estimation en attribuant la même densité à des points éloignés, ce qui n'est pas représentatif de la réalité.
Au contraire si on définit des secteurs trop petit on aura beaucoup de zones où la densité sera nulle puisque aucun point n'est tombé dedans.
Pour le choix du meilleur pas de discretisation on peut également faire une $cross - validation$ sur les données.

2.2 Méthode des noyaux.

La méthode des noyaux permets de rectifier les problèmes de la méthode des histogrammes.
On a implémenté un modèle avec des noyaux de Parzen. Les paramètres du modèles sont la taille de l'hypercube h.

On a implémenté une fonction 3D pour mieux se représenter la courbe renvoyer par les noyaux de Parzen. Lorsque notre fenêtre est trop petite, notre fonction va ressembler de plus en plus à des pics de Dirac, tandis que si la fenêtre est trop grande on va avoir une fonction assez uniforme.
Comme pour la méthode des histogrammes on peut utiliser de la *crossvalidation* pour déterminer la meilleure fenêtre possible de notre modèle.

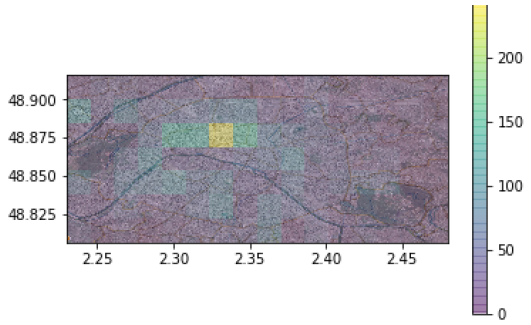

FIGURE 8 – Estimation de densité par noyaux de Pazen sur les ATM de la Figure 5 avec $h = 0.015$

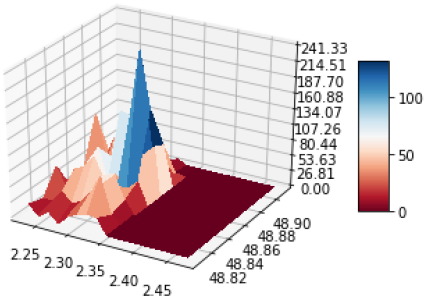

FIGURE 9 – Représentation 3D sur les noyaux de Pazen sur les ATM de la Figure 5 avec $h = 0.015$

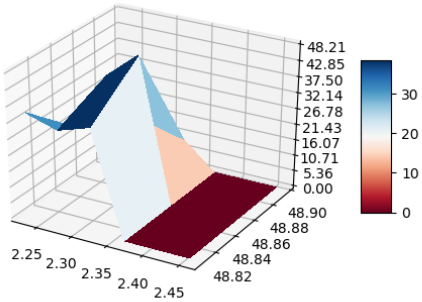

FIGURE 10 – Représentation 3D plutôt uniforme sur les noyaux de Pazen sur les ATM de la Figure 5 avec $h = 0.045$

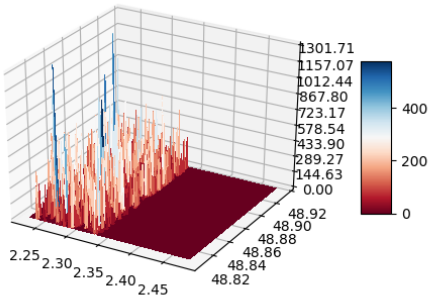

FIGURE 11 – Représentation 3D pic de Dirac sur les noyaux de Pazen sur les ATM de la Figure 5 avec $h = 0.001$

3 TME 3 : Descente de gradient.

Dans ce TME nous nous intéressons à la descente de gradient pour optimiser des fonctions simple pour commencer, puis des fonctions qui ne sont plus optimisable analytiquement, comme dans le cas dans une regression logistique.

3.1 Optimisation de fonctions

Pour débuter avec des fonctions simple nous avons codé la fonction $f(x) = x\cos x$ et sa dérivé $f'(x) = \cos x - x\sin x$.
On voie bien sur la figure suivante que que le gradient en bleu tend vers 0 a mesure que f converge vers un minimum local.
On constate la même chose avec la fonction $f(x) = -\log x + x^2$ qui a pour

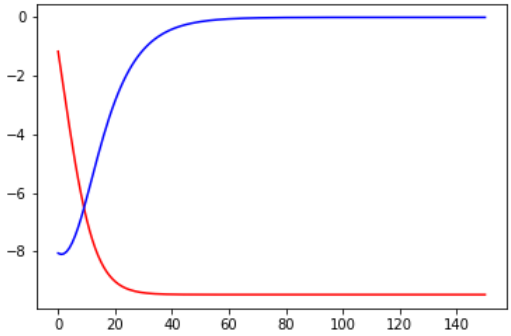

FIGURE 12 – valeurs de f en rouge, et du gradient de f en bleu, avec 150 iteration.

dérivé $f'(x) = -\frac{1}{x} + 2x$ En ce qui conserne la fonction de Rosenbrock, on peux essayer de visualiser directement grace aux isocourbes ou bien à l'aide d'une projection en 3D de l'espace d'optimisation.

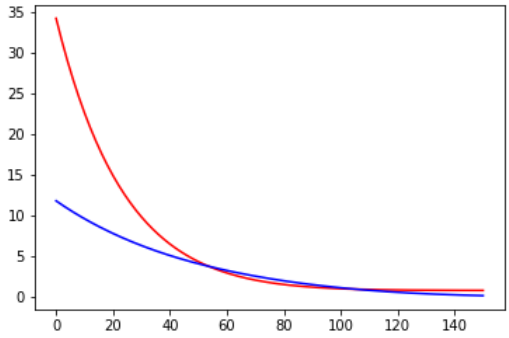

FIGURE 13 – valeurs de f en rouge, et du gradient de f en bleu, avec 150 iteration.

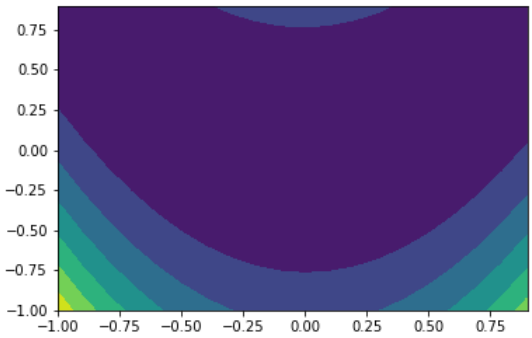

FIGURE 14 – Isocourbe de la fonction de coût de Rosenbock.

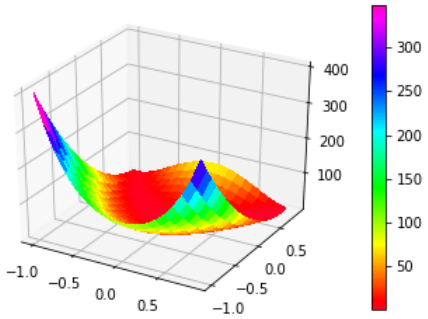

FIGURE 15 – Représentation 3D de l'espace de la fonction de Rosenbock.

3.2 Régression logistique

On va teste la classe 1 contre la classe 5 du jeu de données USPS pour la suite du TME. Aprés avoir entrainer le modèle, on observe un taux de 94,6% de bonne classification sur notre jeu de données. Ce qui est plus faible qu'avec un classificateur bayésien naïf (celui de scikit learn), avec lequel on obtient 99% de bonne classification. Toute fois il ne faut pas perdre a l'esprit que l'on a teste le score des modèles sur les mêmes données qui nous ont servie a les entraîner.
Pour ce que est des valeurs de w, elles sont extrement proche de 0, puisqu'elles sont compris dans un intervales allant de 0.004 à -0.002.

4 TME 4 : Perceptron.

Dans ce TME on étudie plusieurs fonctions de coût qu'on adapte à l'algorithme du Perceptron. En particulier la fonction de coût des moindres carrés et *Hing_loss*.
Pour minimiser l'erreur on utilise la descente de gradient comme méthode d'optimisation qu'on implémente pour chaque fonction de coût.

4.1 Données générées artificellement

Les données sont générées artificiellement selon deux ou quatres gaussiennes. Nos modèles sont évalués sur un ensemble de train et test.
Comme la fonction Hinge Loss est une fonction convexe, l'algorithme de des-

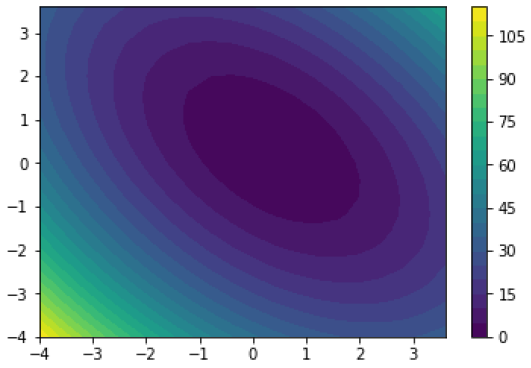

FIGURE 16 – Isocourbe de la fonction de coût des moindres carrés.

cente de gradient adapté à ce coût convergera forcément si l'on prend un pas *epsilon* pas trop grand. En effet si le pas *epsilon* est trop grand notre descente de gradient risque d'oscillé autour du minimum et de jamais l'atteindre.
Au contraire si notre pas *epsilon* est trop petit notre algorithme convergera,

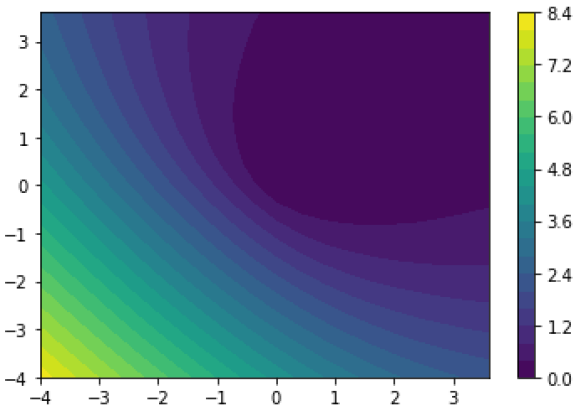

FIGURE 17 – Isocourbe de la fonction de coût Hinge Loss.

mais lentement.
On remarque que les scores en apprentissage et en test des deux fonctions sont sensiblement les mêmes.
En rajoutant une dimension à X (nos données) qui vaut toujours 1 et maintenant dans W la dernière dimension sera notre biais.

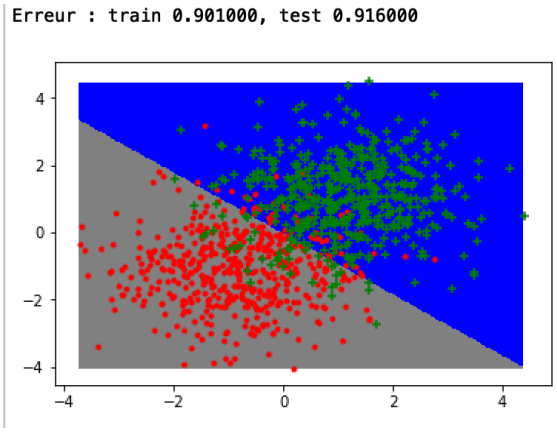

FIGURE 18 – Score (et non l'erreur) du perceptron et sa frontière en utilisant Hinge Loss comme fonction de coût.

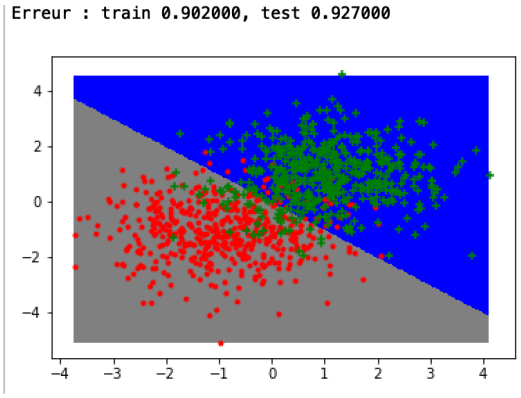

FIGURE 19 – Score (et non l'erreur) du perceptron et sa frontière en utilisant les moindres carrés comme fonction de coût.

4.2 Données USPS

On peut utiliser l'algorithme du Perceptron pour comparer des classes entre elles, par exemple avec le fichier USPS qui contient des images scanné de chiffres écrit à la main.
On a implémenté une fonction qui sélectionne deux classes, c'est à dire deux nombres par exemple 1 et 7 qui sont souvent confondus, et applique l'algorithme du perceptron sur ces données pour séparer ces chiffres.
Comme la dimension des images est de taille 256, on ne peut pas visualiser la frontière de séparation.

```
Compare deux classes représentant les nombres 1 et 7 en
utilisant un perceptron avec la fonction de coût hinge_loss

Score : train 0.609091, test 0.642336
```

FIGURE 20 – Score du perceptron pour différencier les chiffres 1 et 7.

Les scores en test dépassent rarement 70% ce qui n'est pas énorme, cela peut s'expliquer puisque la frontière du perceptron en apprentissage n'est pas forcément optimale. On verra qu'avec des SVM on atteint de bien meilleur score.

5 TME 6 : SVM et Noyaux.

Dans ce TME nous avons étudié les Support Vector Machine. Les SVM contrairement au Perceptron nous donnent une frontière optimale en maximisant la marge entourant la frontière de décision.
À l'aide de différent noyaux que nous avons expérimenté, les SVM permettent également de séparer de manière non linéaire des données.

5.1 Données artificielles et utilisation de plusieurs noyaux.

Sur les données générées artificiellement on a testé plusieurs noyaux pour les SVM. On remarque bien que certaines des frontières notamment avec un noyaux RBF ne sont pas des frontières linéaires.
Pour ces données nous n'obtenons pas de meilleures scores qu'avec le perceptron, c'est à dire entre 90 et 95%. Ce n'est pas surprenant puisque ces données sont facilement séparable linéairement.

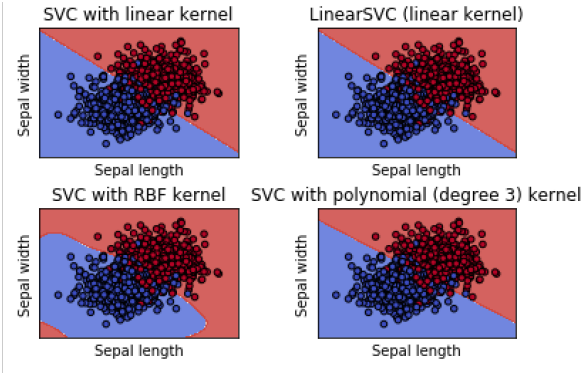

FIGURE 21 – Graphique montrant les SVM avec plusieurs noyaux appliqué sur les données générées artificiellement.

```
score:  SVC with linear kernel  =  0.918
score:  LinearSVC (linear kernel)  =  0.917
score:  SVC with RBF kernel  =  0.918
score:  SVC with polynomial (degree 3) kernel  =  0.909
```

FIGURE 22 – Score des différents SVM avec plusieurs noyaux appliqué sur les données générées artificiellement.

5.2 Comparaison de plusieurs classes sur les données USPS.

5.2.1 One versus One.

Comme pour le TME sur le Perceptron, on a essayé de comparé les classes de chiffres entre elles, et voir si avec les SVM on obtenait de meilleurs résultats.

```
In [138]: t = compare_classe(1,7,trainX,trainY)

In [139]: print(predict_resultat(1,7,testX,testY,t))
0.9902676399026764

In [140]: t = compare_classe(2,8,trainX,trainY)

In [141]: print(predict_resultat(2,8,testX,testY,t))
0.9560439560439561
```

FIGURE 23 – Score pour la différentiation de deux classes avec des SVM de noyau linéaire sur les données USPS.

A Code TME ARBRES DE DÉCISION

```
# coding: utf-8

# # Quelques exemples préliminaires

# In[3]:


get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
import  pickle
import  numpy as np
import pydot
# data : tableau (films ,features), id2titles : dictionnaire  id -> titre ,
# fields : id  feature  -> nom
[data , id2titles , fields ]= pickle.load(open("imdb_extrait.pkl","rb"))
# la  derniere  colonne  est le vote
datax=data [: ,:32]
datay=np.array ([1 if x[33] >6.5  else  -1 for x in data])



from  decisiontree  import  DecisionTree
dt = DecisionTree ()
dt.max_depth = 5
#on fixe la  taille  de l'arbre a 5
dt.min_samples_split = 2
#nombre  minimum d'exemples  pour  spliter  un noeud
dt.fit(datax ,datay)
dt.predict(datax [:5 ,:])
print(dt.score(datax ,datay))
# dessine l'arbre  dans un  fichier  pdf   si pydot  est  installe.
dt.to_pdf("test_tree.pdf",fields)
# sinon  utiliser  http :// www.webgraphviz.com/
dt.to_dot(fields)
#ou dans la  console
print(dt.print_tree(fields ))


# On a une profondeur de 5 et un score de 0.736 de bonne classification pour cet arbre.

# In[4]:
```

```
#dt.max_depth = 15 Comme profondeur on a Out[6]: 0.8820579899716591 de précision


# In[5]:


#dt.max_depth = 20 Comme profondeur on a Out[8]: 0.8984085458905603 de précision


# Plus notre profondeur augmente plus on a de précisions. Mais on doit faire attention au su

# # Sur et sous apprentissage

# On défini une fonction pour partitionné nos data en ensemble data d'entrainements et data

# In[6]:


def partitionnement_test(datax,datay,rp,rdm,couleur): #rp la proportion qui sera dans l'appr

    dt = DecisionTree()
    dt.min_samples_split = 2
    if rdm:
        rp = random.uniform(0,1)
    #inceap nos indices dans datax qui vont servir pour notre apprentissage, et indicet pour
    #On tire indiceap aléatoirement avec la proportion rp dans datax, et on effectue des tir
    indiceap = np.random.choice(np.arange(len(datax)), int(rp*len(datax)), replace = False)
    indicet = []
    for i in range(0,len(datax)):
        if i not in indiceap:
            indicet.append(i)
    testy = np.zeros((len(indicet)), int)
    apprentissagey = np.zeros((len(indiceap)),int)




    testx = np.delete(datax,indiceap,axis=0)

    apprentissagex = np.delete(datax,indicet,axis=0)

    for i in range(0,len(indiceap)):
        apprentissagey[i] = datay[indiceap[i]]
    for i in range(0,len(indicet)):
        testy[i] = datay[indicet[i]]
```

```
    l_scoretest = []
    l_scoreapprentissage = []
    #On test différentes profondeurs d'arbres avec comme pas de 3 pour éviter un trop long t
    for i in range(2,20,3):
        dt.max_depth = i
        dt.fit(apprentissagex ,apprentissagey)
        dt.predict(apprentissagex[:5 ,:])
        l_scoretest.append(1 - dt.score(testx,testy))
        l_scoreapprentissage.append(1 - dt.score(apprentissagex,apprentissagey))
    plt.plot(range(2,20,3),l_scoretest,couleur+'--',range(2,20,3),l_scoreapprentissage,coule
    plt.show()


partitionnement_test(datax,datay,0.8,False,'b')


# BLEU : 0.8 en APPRENTISSAGE,  0.2 en TEST
# ROUGE : 0.5 en APPRENTISSAGE et en TEST
# VERT : 0.2 en APPRENTISSAGE , 0.8 en TEST

# In[7]:


partitionnement_test(datax,datay,0.5,False,'r')


# In[8]:


partitionnement_test(datax,datay,0.2,False,'g')


# In[9]:


#L'erreur en apprentissage est en trai continue tandis que l'erreur en test est en pointillé


# ## Intérpretation

# On remarque que pour un ensemble d'apprentissage assez réduit l'erreur pour le test est pl

# # Validation croisée: selection de modèle
```

B Code TME ESTIMATION DE DENSITÉ

On a crée un fichier Prediction.py avec nos classes pour chaque modèles.
Prediction.py

```
"""Fichier des modèles de prediction pour le TME2.


- Modèle des histogrammes
- Modèle des noyaux de Parzen
- Modèle des noyaux Gaussiens.



"""
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
import operator
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter



class histogramme:

    def __init__(self,data):
        self.data = data
        self.mat = []
    def fit(self,poi,X,Y,pas):
        self.pas = pas
        self.xmin = X[0]
        self.xmax = X[1]
        self.ymin = Y[0]
        self.ymax = Y[1]
        self.xinter = (X[1]-X[0])/pas
        self.yinter = (Y[1]-Y[0])/pas
        N = len(self.data[poi]) # Le nombre de point d'intéret poi en région parisienne
        self.mat = np.zeros((pas,pas)) # création d'une matrice de taille pas x pas qui va c


        for clef in self.data[poi]:
            coord = self.data[poi][clef][0]

            y= int((coord[0]-Y[0])/self.yinter)
```

```
            x = int((coord[1]-X[0])/self.xinter)
            self.mat[y][x] += 1



        self.mat = np.divide(self.mat,(N*pas)) # on divise par N? N*pas ? N*pas**2 ?

        return self.mat




    def predict(self,coord):
        if coord[0] < self.ymin or coord[1] < self.xmin or coord[0] > self.ymax or coord[1]
            raise ValueError('coords must be between xmin xmax and ymin ymax')

        else:
             y= int((coord[0]-self.ymin)/self.yinter)
             x = int((coord[1]-self.xmin)/self.xinter)
             density = self.mat[y][x]

             print( "The density at this point ",coord," is ",density)
             return density

    def affichage_3D(self):
        """marche pas """
        print(self.ymin)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        xpos = [self.xmin+ self.xinter for i in range(self.pas) ]
        ypos = [self.ymin+ self.yinter for i in range(self.pas) ]
        zpos = 0
        print(self.mat)
        dx = np.ones_like(xpos)
        dy = np.ones_like(ypos)
        dz = self.mat.ravel()
        ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average')


        plt.show()
        plt.savefig('hist3D.png')


class noyau_parzen:
```

```
def __init__(self,data,X,Y,poi):
    self.data = data
    self.xmin = X[0]
    self.xmax = X[1]
    self.ymin = Y[0]
    self.ymax = Y[1]
    self.poi = poi
    self.N = len(self.data[self.poi])

def indicatrice_phi(self,coord):
    phi = np.sqrt(sum([i**2 for i in coord]))
    #print(phi)
    if phi <=0.5:
        return 1
    else:
        return 0


def densite(self,h,yx):
    """
    On donne en paramètre le point d'intéret
    ainsi que h la longueur de l'hypercube.

    Renvoie l'estimation de densité au point yx.
    """


    k = 0
    for clef in self.data[self.poi]:
        coord = self.data[self.poi][clef][0]
        d = tuple(np.divide(tuple(map(operator.sub, yx, coord)),h)) # Pour chaque coordo

        k+= (self.indicatrice_phi(d))/h**2


    return k/(self.N)


def estimation_parzen(self,h):
    self.h = h #LONGUEUR HYPERCUBE.
    nbcubex = int((self.xmax -self.xmin)/self.h)   # Nombre d'hypercube de longueur h qu
    nbcubey = int((self.ymax - self.ymin)/self.h)  # Nombre d'hypercube de longueur h qu



    self.mat = np.zeros((nbcubey,nbcubex)) # Va contenir les estimations de densités cen
```

```
    for j in range(0,nbcubey):
        for i in range(0,nbcubex):
            x = self.xmin + i*self.h
            y = self.ymin + j*self.h
            point = (y+(self.h/2),x+(self.h/2)) # Le point est le centre de l'hypercube
            d = self.densite(self.h,point)
            self.mat[j][i] =d
    return self.mat

def predict(self,coord):
    if coord[0] < self.ymin or coord[1] < self.xmin or coord[0] > self.ymax or coord[1]
        raise ValueError('coords must be between xmin xmax and ymin ymax')

    else:
         density = self.densite(self.h,coord)

         print( "The density at this point ",coord," is ",density)
         return density



def affichage_3D(self):




    # Make data.
    X = np.arange(self.xmin, self.xmax, self.h)
    Y = np.arange(self.ymin, self.ymax, self.h)

    Z = np.zeros((len(Y),len(X)))

    X, Y = np.meshgrid(X, Y)

    for j in range(len(Y)):
        for i in range(len(X)):
            x = self.xmin + i*self.h
            y = self.ymin + j*self.h
            point = (y+(self.h/2),x+(self.h/2)) # Le point est le centre de l'hypercube
            d = self.densite(self.h,point)
            Z[j][i] = d
```

```
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,cmap=cm.RdBu,linewidth=0, antia

        ax.zaxis.set_major_locator(LinearLocator(10))
        ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

        fig.colorbar(surf, shrink=0.5, aspect=5)

        fig1 = plt.gcf()

        plt.show()
        fig1.savefig('parzen3Dh='+str(self.h)+'.png',dpi=100)
        plt.close()
```

tme2-poi.py

```
from Prediction import histogramme , noyau_parzen
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pickle




plt.ion()
parismap = mpimg.imread('data/paris-48.806-2.23--48.916-2.48.jpg')

## coordonnees GPS de la carte
```

```
xmin,xmax = 2.23,2.48   ## coord_x min et max
ymin,ymax = 48.806,48.916 ## coord_y min et max

def show_map():
    plt.imshow(parismap,extent=[xmin,xmax,ymin,ymax],aspect=1.5)
    ## extent pour controler l'echelle du plan

poidata = pickle.load(open("data/poi-paris.pkl","rb"))
## liste des types de point of interest (poi)
print("Liste des types de POI" , ", ".join(poidata.keys()))

## Choix d'un poi
typepoi = "atm"

## Creation de la matrice des coordonnees des POI
geo_mat = np.zeros((len(poidata[typepoi]),2))
for i,(k,v) in enumerate(poidata[typepoi].items()):
    geo_mat[i,:]=v[0]

## Affichage brut des poi
show_map()
## alpha permet de regler la transparence, s la taille
plt.scatter(geo_mat[:,1],geo_mat[:,0],alpha=0.8,s=3)


######################################################

# discretisation pour l'affichage des modeles d'estimation de densite
steps = 10
xx,yy = np.meshgrid(np.linspace(xmin,xmax,steps),np.linspace(ymin,ymax,steps))
grid = np.c_[xx.ravel(),yy.ravel()]

# A remplacer par res = monModele.predict(grid).reshape(steps,steps)
#res = histogramme(poidata).fit("restaurant",[2.23,2.48],[48.806,48.916]).reshape(steps,step
#res = np.random.random((steps,steps))
#H = histogramme(poidata)
#res = H.fit(typepoi,[xmin,xmax],[ymin,ymax],steps)
#H.predict((48.850,2.32))
parzen = noyau_parzen(poidata,[xmin,xmax],[ymin,ymax],typepoi)
res = parzen.estimation_parzen(0.015)


plt.figure()
show_map()
plt.imshow(res,extent=[xmin,xmax,ymin,ymax],interpolation='none',alpha=0.3,origin = "lower")
plt.colorbar()
```

```
plt.scatter(geo_mat[:,0],geo_mat[:,1],extent=[xmin,xmax,ymin,ymax],alpha=0.3)
```

C Code TME3

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

def make_grid(xmin=-5,xmax=5,ymin=-5,ymax=5,step=20,data=None):
    """ Cree une grille sous forme de matrice 2d de la liste des points
    :return: une matrice 2d contenant les points de la grille, la liste x, la liste y
    """
    if data is not None:
        xmax,xmin,ymax,ymin = np.max(data[:,0]),np.min(data[:,0]),\
                              np.max(data[:,1]),np.min(data[:,1])
    x,y = np.meshgrid(np.arange(xmin,xmax,(xmax-xmin)*1./step),
                      np.arange(ymin,ymax,(ymax-ymin)*1./step))
    grid=np.c_[x.ravel(),y.ravel()]
    return grid, x, y



def load_usps(filename):
    with open(filename,"r") as f:
        f.readline()
        data =[ [float(x) for x in l.split()] for l in f if len(l.split())>2]
    tmp = np.array(data)
    return tmp[:,1:],tmp[:,0].astype(int)


def optimise(fonc,dfonc,xinit, eps,max_iter):
    """ Implémentation de l'algorithme de descente de gradient
    fonc -> la fonction à optimiser.
    dfonc -> le gradient de cette fonction
    xinit -> le point initial
    eps -> Le pas de gradient
    max_iter -> Le nombre d'itérations.
    return : triplet (x_histo,f_histo,grad_histo) respectivement la liste des points xt, f(x
    """

    X = [xinit]
    fX = [fonc(xinit)]
    dfX = [dfonc(xinit)]
    for i in range(max_iter):
        xinit = xinit - eps*dfonc(xinit)
        X.append(xinit)
```

```
        fX.append(fonc(xinit))
        dfX.append(dfonc(xinit))

    return np.array(X),np.array(fX),np.array(dfX)


def xcosx(x):
    return x*np.cos(x)

def dxcosx(x):
    return np.cos(x) - x*np.sin(x)

def logx(x):
    return -np.log(x) + x**2

def dlogx(x):
    return (-1/x) + 2*x

def rosenBrock(x1,x2):
    return 100*(x2-x1**2)**2 +(1 - x1)**2

def drosenBrock(x1,x2):
    return (-400*x1*(x2-x1**2)-2*(1-x1),200*(x2-x1**2))

"""           AFFICHAGES             """

def affiche2d(fonc,dfonc,xinit,eps,max_iter,figname="default.png"):
    """Affiche en fonction du nombre d'itérations les valeurs de fX et du gradient de f."""

    absi = [i for i in range(max_iter+1)]
    X,fX,dfX = optimise(fonc,dfonc,xinit,eps,max_iter)
    plt.clf()
    plt.plot(absi,fX,'r',absi,dfX,'b')
    plt.savefig(figname)



affiche2d(xcosx,dxcosx,random.randint(10),0.01,150,"xcox.png")
affiche2d(logx,dlogx,random.randint(1,10),0.01,150,"logx.png")

## Grille de discretisation
grid , xx , yy = make_grid(xmin=-1,xmax=1,ymin=-1,ymax=1)
t = np.array([rosenBrock(c[0],c[1]) for c in grid]).reshape(xx.shape)
## Affichage 2D
plt.contourf(xx,yy,t)
plt.savefig('rosenBrock2D.png')
```

```
fig = plt.figure()
## construction d'un referentiel 3d
ax = fig.gca( projection='3d')
surf = ax.plot_surface(xx,yy,t,rstride=1,cstride=1,\
cmap=cm.gist_rainbow,linewidth=0,antialiased=False)
fig.colorbar(surf)
#x_histo = np.array([drosenBrock(c[0],c[1]) for c in grid])
#ax.plot(x_histo[:,0],x_histo[:,1],f_histo.ravel(),color='black')
#plt.show()
plt.savefig('rosenBrock3D.png')



class Regression_Logistique:

    def __init__(self,eps,max_iter):
        self.eps = eps
        self.max_iter = max_iter
        self.w = None

    def fit(self,datax,datay):

        # initialisation des parametres
        self.w = np.random.rand(datax.shape[1])

        for iteration in range(self.max_iter):

            erreur = datay - self.activation(np.dot(datax,self.w))

            gradient = np.dot(datax.T,erreur)

            self.w -= self.w - self.eps * gradient

    def activation(self,score):
        """
        La fonction sigmoid de la régression logistique
        """
        return 1 / (1 + np.exp(-score))


    def predict(self,x):
        return 1 if self.activation(np.dot(x.T,self.w))>0.5 else 0


    def score(self,testX,testY):
        nb_bonne_classification = sum([1 for i in range(len(testX)) if self.predict(testX[i]
```

```
        return nb_bonne_classification/len(testX)


X,Y = load_usps("./USPS_train.txt")     # On load les données USPS

trainX = X[np.where(np.in1d(Y, [1,5]))] # On ne garde que les classe
trainY = Y[np.where(np.in1d(Y, [1,5]))] # 1 et 5

trainY[trainY == 5.0] = 0               # On transforme notre probleme
                                        # Pour que la classe 5 soit egale a 0

regression = Regression_Logistique(0.000005,10000) # On crée notre régression
regression.fit(trainX,trainY)                 # que l'on fit avec nos données

print(regression.score(trainX,trainY))
print(max(regression.w))
print(min(regression.w))


from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(trainX, trainY).predict(trainX)

print(sum([1 for i in range(len(trainX)) if y_pred[i]==trainY[i]])/len(trainX))
```

D Code TME PERCEPTRON

```
from arftools import *
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm


def mse(datax,datay,w):
    """ retourne la moyenne de l'erreur aux moindres carres """
    resultat = 0
    taille = len(datax)
    for i in range(taille):
        resultat += (np.matmul(datax[i],w)-datay[i])**2
    return resultat/taille

def mse_g(datax,datay,w):
    """ retourne le gradient moyen de l'erreur au moindres carres """
    resultat = 0
    taille = len(datax)
    D = datax.shape[1]
    for i in range(taille):

        resultat += 2*np.matmul(np.reshape(datax[i],(D,1)),w)-2*sum(datay[i]*datax[i])
    return resultat[0]/taille

def hinge(datax,datay,w):
    """ retourn la moyenne de l'erreur hinge """

    #return np.maximum(np.zeros((testy.shape)),np.dot(testy,np.dot(testx,w.T)))

    resultat = 0
    taille = len(datax)
    for i in range(taille):
        resultat+=(max(0,-datay[i]*np.matmul(datax[i],w.T)))
    return resultat/taille

def hinge_g(datax,datay,w):
    """ retourne le gradient moyen de l'erreur hinge """
    resultat = np.zeros((w.shape))
    taille = len(datax)
    for i in range(taille):
        if -datay[i]*np.dot(datax[i],w.T) < 0:
            resultat += -datay[i]*datax[i]
    return resultat/taille
```

```
class Lineaire(object):

    def __init__(self,loss=hinge,loss_g=hinge_g,max_iter=1000,eps=0.01,biais=False):
        """ :loss: fonction de cout
            :loss_g: gradient de la fonction de cout
            :max_iter: nombre d'iterations
            :eps: pas de gradient
        """
        self.max_iter, self.eps = max_iter,eps
        self.loss, self.loss_g = loss, loss_g
        self.biais = biais

    def fit(self,datax,datay,testx=None,testy=None):
        """ :datax: donnees de train
            :datay: label de train
            :testx: donnees de test
            :testy: label de test
        """
        # on transforme datay en vecteur colonne
        datay = datay.reshape(-1,1)
        N = len(datay)
        datax = datax.reshape(N,-1)
        D = datax.shape[1]

        """
        if self.biais:
            D+=1
            b = np.ones((N,D))
            b[:,:-1] = datax
            datax = b"""
        self.w = np.random.random((1,D))
        for i in range(self.max_iter):

            self.w = self.w - self.eps*self.loss_g(datax,datay,self.w)



    def predict(self,datax):

        if len(datax.shape)==1:
            datax = datax.reshape(1,-1)

        return np.sign(np.matmul(datax,self.w.T))

    def score(self,datax,datay):
        taille = len(datax)
```

```
        """if self.biais:
            D+=1
            b = np.ones((taille,D))
            b[:,:-1] = datax
            datax = b
        print("SHAPE=",np.shape(datax))"""
        score = 0

        for i in range(taille):
            if self.predict(datax[i]) == datay[i]:
                score += 1
        return score/taille


def load_usps(filename):
    with open(filename,"r") as f:
        f.readline()
        data =[ [float(x) for x in l.split()] for l in f if len(l.split())>2]
    tmp = np.array(data)
    return tmp[:,1:],tmp[:,0].astype(int)


def compare_classe(n1,n2,trainx,trainy,testx,testy,loss=mse,loss_g=mse_g,max_iter=1000,eps=0

    print("Compare deux classes représentant les nombres", n1 ,"et", n2, "en utilisant un pe

    trainX = trainx[np.where(np.in1d(trainy, [n1,n2]))]

    trainY = trainy[np.where(np.in1d(trainy, [n1,n2]))]

    testX = testx[np.where(np.in1d(testy, [n1,n2]))]
    testY = testy[np.where(np.in1d(testy, [n1,n2]))]

    perceptron = Lineaire(loss,loss_g,max_iter=max_iter,eps=eps)
    perceptron.fit(trainX,trainY)

    print("Score : train %f, test %f"% (perceptron.score(trainX,trainY),perceptron.score(tes


def show_usps(data):
    plt.imshow(data.reshape((16,16)),interpolation="nearest",cmap="gray")



def plot_error(datax,datay,f,step=10):
    grid,x1list,x2list=make_grid(xmin=-4,xmax=4,ymin=-4,ymax=4)
```

```
    plt.contourf(x1list,x2list,np.array([f(datax,datay,w) for w in grid]).reshape(x1list.sha
    plt.colorbar()
    plt.show()



if __name__=="__main__":
    """ Tracer des isocourbes de l'erreur """


    """
    #DONNÉES GÉNERÉS ARTIFICIELLEMENT
    plt.ion()
    trainx,trainy =  gen_arti(nbex=1000,data_type=0,epsilon=1)
    testx,testy =  gen_arti(nbex=1000,data_type=0,epsilon=1)
    plt.figure()
    plot_error(trainx,trainy,mse)
    plt.figure()
    plot_error(trainx,trainy,hinge)


    #Choix biais
    biais = True
    D = np.shape(trainx)[1]
    tailleTrain = len(trainx)
    tailleTest = len(testx)
    if biais:
        D+=1
        b = np.ones((tailleTrain,D))
        b[:,:-1] = trainx
        trainx = b
        c = np.ones((tailleTest,D))
        c[:,:-1] = testx
        testx = c




    perceptron = Lineaire(mse,mse_g,max_iter=1000,eps=0.1)
    perceptron.fit(trainx,trainy)
    print(np.shape(trainx))
    #plt.figure()
    #plot_frontiere(trainx,perceptron.predict,200)
    #plot_data(trainx,trainy)
    print("Score : train %f, test %f"% (perceptron.score(trainx,trainy),perceptron.score(tes
```

```
"""
train = load_usps('data/USPS_train.txt')
trainx, trainy = train
test = load_usps('data/USPS_test.txt')
testx, testy = test
compare_classe(4,5,trainx,trainy,testx,testy,loss=hinge,loss_g=hinge_g,max_iter=1000,eps
#compare_classe(6,9,trainx,trainy,testx,testy,loss=hinge,loss_g=hinge_g,max_iter=1000,ep
```

E Code TME SVM

```
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt


def load_usps(filename):
    with open(filename,"r") as f:
        f.readline()
        data =[ [float(x) for x in l.split()] for l in f if len(l.split())>2]
    tmp = np.array(data)
    return tmp[:,1:],tmp[:,0].astype(int)


def compare_classe(n1,n2,dataX,dataY,typeSVM ='linear'):

    """Compare deux classes représentant les nombres n1 et n2 en utilisant un SVM de type ty

    X = dataX[np.where(np.in1d(dataY, [n1,n2]))]
    Y = dataY[np.where(np.in1d(dataY, [n1,n2]))]

    svm_lineaire = svm.SVC(kernel =typeSVM)
    svm_lineaire.fit(X,Y)

    return svm_lineaire

def predict_resultat(n1,n2,testX,testY,model):
    masque = np.where(np.in1d(testY, [n1,n2]))
    resultat = model.predict(testX[masque]) == testY[masque]
    return sum(resultat)/len(masque[0])

def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
```

```
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

def gen_arti(centerx=1,centery=1,sigma=0.1,nbex=1000,data_type=0,epsilon=0.02):
    """ Generateur de donnees,
        :param centerx: centre des gaussiennes
        :param centery:
        :param sigma: des gaussiennes
        :param nbex: nombre d'exemples
        :param data_type: 0: melange 2 gaussiennes, 1: melange 4 gaussiennes, 2:echequier
        :param epsilon: bruit dans les donnees
        :return: data matrice 2d des donnnes,y etiquette des donnnees
    """
    if data_type==0:
         #melange de 2 gaussiennes
         xpos=np.random.multivariate_normal([centerx,centerx],np.diag([sigma,sigma]),nbex//2
         xneg=np.random.multivariate_normal([-centerx,-centerx],np.diag([sigma,sigma]),nbex/
         data=np.vstack((xpos,xneg))
         y=np.hstack((np.ones(nbex//2),-np.ones(nbex//2)))
    if data_type==1:
        #melange de 4 gaussiennes
        xpos=np.vstack((np.random.multivariate_normal([centerx,centerx],np.diag([sigma,sigma
        xneg=np.vstack((np.random.multivariate_normal([-centerx,centerx],np.diag([sigma,sigm
        data=np.vstack((xpos,xneg))
        y=np.hstack((np.ones(nbex/2),-np.ones(nbex//2)))

    if data_type==2:
```

```
        #echiquier
        data=np.reshape(np.random.uniform(-4,4,2*nbex),(nbex,2))
        y=np.ceil(data[:,0])+np.ceil(data[:,1])
        y=2*(y % 2)-1
    # un peu de bruit
    data[:,0]+=np.random.normal(0,epsilon,nbex)
    data[:,1]+=np.random.normal(0,epsilon,nbex)
    # on mélange les données
    idx = np.random.permutation((range(y.size)))
    data=data[idx,:]
    y=y[idx]
    return data,y

if __name__ == "__main__":


    """

    SUR LES DONNÉES USPS "

    """
    train = load_usps('data/USPS_train.txt')
    trainX, trainY = train
    test = load_usps('data/USPS_test.txt')
    testX, testY = test

    t =compare_classe(4,5,trainX,trainY)
    print(predict_resultat(4,5,testX,testY,t))


    t =compare_classe(4,5,trainX,trainY,'poly')
    print(predict_resultat(4,5,testX,testY,t))


    t =compare_classe(4,5,trainX,trainY,'rbf')
    print(predict_resultat(4,5,testX,testY,t))

    """""""""""""""""""



    """

    SUR LES DONNÉES GÉNÉRÉES ARTIFICIELLEMENT

    """
```

```
X,y=  gen_arti(nbex=1000,data_type=0,epsilon=1)
#X,testy =  gen_arti(nbex=1000,data_type=0,epsilon=1)

C = 1.0  # SVM regularization parameter
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, C=C))
models = (clf.fit(X, y) for clf in models)

# title for the plots
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel')


# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    print("score: ",title, " = ",clf.score(X,y))

plt.show()
```