UNIVERSITÉ SIDI MOHAMED BEN ABDLLAH

FACULTÉ DES SCIENCES DHAR EL MAHRAZ DE FÈS

DÉPARTEMENT D'INFORMATIQUE

IMAGE MINING

ATELIER 2

Présenter par :

ABIBOU SOUKAYNA

Année universitaire : 2020/2021

Filière MIDVI

Plan:

1. Introduction :

L'objectif principal de cet atelier est de créer un système de classification d'images à base des algorithmes de Machine Learning et Data Mining. La classification est le processus de prédiction de la classe d'une image donnée. Les classes sont parfois appelées étiquettes ou catégories. Comme pour le CBIR, la classification sera effectuée en se basant sur le vecteur descripteur de l'image au lieu d'utiliser l'image.

La classification peut être supervisée ou non supervisée ; pour la classification supervisée on dispose d'éléments déjà classés avec leurs étiquètes en avance alors que pour le cas de classification non supervisée les éléments ne sont pas classées et on cherche à les regrouper en classes.

Dans cet atelier, notre objectif est de développer un système de classification supervisée.

Dans ce système, deux parties (ou 3) seront élaborées :

1- Apprentissage : Créer un modèle en se basant sur les descripteurs de la base d'images. Cette phase consiste en l'approximation d'une fonction de mappage entre la matrice de caractéristiques (variables d'entrées) et les étiquettes (variables de sorties)

2- Classification (prédiction) : prédire l'étiquette d'une nouvelle entrée (image) par le modèle élaboré.

2. Base de Données :

Dans cet atelier de classification, nous allons utiliser seulement deux groupes d'images (deux classes).

La première classe va contenir les images de type (classe) « voiture » et la deuxième classe va contenir les images de types « bateau ».

Chaque classe sera mise dans un dossier portant successivement les noms « obj_car » pour les voitures et « obj_ship » pour les bateaux. La classe « obj_car » contient 400 images et la classe « obj_ship » contient 90 images.

3. Bon modèle :

L'objectif de la classification supervisée est de pouvoir prédire correctement la classe d'une (nouvelle) image à partir des connaissances.

La validation et l'évaluation d'un système de classification est effectuée en divisant la base d'images d'apprentissage en deux parties : partie d'apprentissage ou d'entrainement (training) et base de tests ou de validation. Sachant que les images sont étiquetées au préalable, dans la base d'apprentissage les images et les étiquettes seront utilisées pour élaborer le modèle de classification alors que les images de la base de tests seront utilisées pour prédire les étiquettes de chaque image. Les étiquettes prédites seront comparées avec les étiquettes originelles pour mesurer la performance du modèle élaboré. Une multitude d'algorithmes de classification supervisée ont été présentés dans la littérature. Ci-dessous une liste non exhaustive des méthodes les plus utilisées :

- Machine à vecteurs de support (SVM ; Support Vector Machine).
- Classification naïve bayésienne.
- Méthode des k plus proches voisins (KNN ; K-Nearest Neighbors).
- Arbre de décision.
- Réseau de neurones.

3.2 Evaluation du modèle de classification :

Dans ce type de système, il est commode pour évaluer la performance de prédiction du modèle de classification établi de diviser la base d'images en deux parties ; une partie pour l'apprentissage (training) et une autre partie pour le test et la validation du modèle.

Les mesures les plus utilisées pour l'évaluation d'un modèle de classification sont Sensibilité (taux de TP), Spécificité (taux de TN) et de précision (accuracy).

Recall or Sensitivity = TP/ (TP + FN)

Accuracy= (TP+TN)/ (TP+TN+FP+FN)

Precision = TP / (TP+FP)

F-score or F-mesure = 2*(Presion*Recall) / (Precision+ Recall)

Avec :

TP: nombre de « True Positives ». C'est le nombre d'images classées par le modèle dans la classe « obj_car » qui appartiennent effectivement à cette classe

TN: nombre de « True Negatives. C'est le nombre d'images classées par le modèle dans la classe « obj_ship » qui appartiennent effectivement à cette classe

FP: nombre de « False Positives » : C'est le nombre d'images qui sont classées par le modèle dans la classe « obj_car » alors et qu'elles appartiennent à la classe

« obj_ship »

FN: nombre de « False Negatives ». C'est le nombre d'images qui sont classées par le modèle dans la classe « obj_ship » alors et qu'elles appartiennent à la classe

« obj_car ».

Ces paramètres nous permettant de comparer des déférents modèles en eux et aussi de comparer des versions de même modèle avec des paramètres déférents comme dans la méthode K-fold.

4. Extraction des caractéristiques :

La première étape du processus de classification est l'extraction de caractéristiques.
Pour extraire les caractéristiques des images de notre Data-set on vas utiliser les méthodes d'extraction d'atelier 1.

4.2 Méthodes nécessaires pour le features extraction :

Entrée [3]:

```
def hsvHistogramFeatures(image):
    rows,cols,dd = image.shape
    image = cv.cvtColor(image, cv.COLOR_RGB2HSV)

    h = image[...,0]
    s = image[...,1]
    v = image[...,2]

    numberOfLevelsForH = 8
    numberOfLevelsForS = 2
    numberOfLevelsForV = 2

    maxValueForH = np.max(h)
    maxValueForS = np.max(s)
    maxValueForV = np.max(v)

    hsvColorHisto = np.zeros((8,2,2))

    quantizedValueForH = (h*numberOfLevelsForH/maxValueForH)
    quantizedValueForS = (s*numberOfLevelsForS/maxValueForS)
    quantizedValueForV = (v*numberOfLevelsForV/maxValueForV)

    index = np.zeros((rows*cols,3))
    index[:,0] = quantizedValueForH.flatten()
    index[:,1] = quantizedValueForS.flatten()
    index[:,2] = quantizedValueForV.flatten()

    for i in range(len(index[:,0])):
        if(index[i,0]==0 or index[i,1]==0 or index[i,2]==0):
            continue
        hsvColorHisto[int(index[i,0]),int(index[i,1]),int(index[i,2])] +=1
    hsvColorHisto = hsvColorHisto.flatten()
    hsvColorHisto /= np.sum(hsvColorHisto)
    return hsvColorHisto.reshape(-1)
```

```
def  textureFeatures(img):
    features_texture = np.zeros(4)
    im = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
    glcm = greycomatrix(im, [1,2,3], [0],levels=256, symmetric=True, normed=True)
    features_texture[0] = np.mean(greycoprops(glcm, 'correlation'))
    features_texture[1] = np.mean(greycoprops(glcm, 'contrast'))
    features_texture[2] = np.mean(greycoprops(glcm, 'homogeneity'))
    features_texture[3] = np.mean(greycoprops(glcm, 'energy'))

    return features_texture

def shapeFeatures(img):
    im = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
    shapeFeat = cv.HuMoments(cv.moments(im)).flatten()
    shapeFeat /= np.mean(shapeFeat)
    return shapeFeat

def getFeatures(img, fsize):
    features = np.zeros(fsize)
    if fsize == 6:
        features[0:fsize] = color_Moments(img)
    elif fsize == 38:
        features[0:6] = color_Moments(img)
        features[6:fsize] = hsvHistogramFeatures(img)
    elif fsize == 42:
        features[0:6] = color_Moments(img)
        features[6:38] = hsvHistogramFeatures(img)
        features[38:fsize] = textureFeatures(img)
    elif fsize == 49:
        features[0:6] = color_Moments(img)
        features[6:38] = hsvHistogramFeatures(img)
        features[38:42] = textureFeatures(img)
        features[42:fsize] = shapeFeatures(img)
    else:
        print("error: no such size")
        exit(0)
    return features
```

```
def Indexation(fsize):
    leng_car = len(loaded_images_car)
    features_car = np.zeros((leng_car,fsize+1))
    for i in range(leng_car):
        features_car[i][0:fsize] = getFeatures(loaded_images_car[i],fsize)
        features_car[i][fsize] = 0

    leng_ship = len(loaded_images_ship)
    features_ship = np.zeros((leng_ship,fsize+1))
    for i in range(leng_ship):
        features_ship[i][0:fsize] = getFeatures(loaded_images_ship[i],fsize)
        features_ship[i][fsize] = 1


    features_total = np.zeros((leng_car+leng_ship,fsize+1))
    features_total[0:leng_car] = features_car
    features_total[leng_car:leng_car+leng_ship]=features_ship

    headers = [(lambda x: "features_"+str(x))(x) for x in range(1,50)]
    headers.append("class")

    df = pd.DataFrame(data=features_total.astype(float))
    df.to_csv('Base_Indexe_'+ str(fsize) +'.csv', sep=',', header=headers, float_format='%.4f', index=False)

    return features_total
Indexation(49)

print("c'est terminé")
```

```
c'est terminé
```

4.1Vecteurs caractéristiques de notre Data-set :

Entrée [6]:

```
loaded_images_test = list()
img_tst_name = list()
path_3='DataToPredict'
for filename in listdir(path_3):
    img_tst = image.imread(path_3+'/' + filename)
    loaded_images_test.append(img_tst)
    img_tst_name.append(filename)

def indexation_test(fsize):
    leng_test = len(loaded_images_test)
    features_test = np.zeros((leng_test,fsize))
    for i in range(leng_test):
        features_test[i][0:fsize] = getFeatures(loaded_images_test[i],fsize)

    headers = [(lambda x: "features_"+str(x))(x) for x in range(1,50)]

    df = pd.DataFrame(data=features_test.astype(float))
    df.to_csv('test_Base_Indexe_'+ str(fsize) +'.csv', sep=',', header=headers, float_format='%.4f', index=False)

    return features_test

indexation_test(49)
print("done")
```

```
done
```

5. Apprentissage :

Les algorithmes d'apprentissage automatique ont des hyperparamètres qui vous permettent d'adapter le comportement de l'algorithme à votre ensemble de données spécifique

Les hyperparamètres sont différents des paramètres, qui sont les coefficients ou poids internes d'un modèle trouvé par l'algorithme d'apprentissage. Contrairement aux paramètres, les hyperparamètres sont spécifiés par le praticien lors de la configuration du modèle.

En règle générale, il est difficile de savoir quelles valeurs utiliser pour les hyperparamètres d'un algorithme donné sur un ensemble de données donné, il est donc courant d'utiliser des stratégies de recherche aléatoire ou de grille pour différentes valeurs d'hyperparamètres.

Dans cette étape nous allons implémenter quelques classifieurs, comme :

1. K-Nearest Neighbors (KNN)
2. Support Vector Machine (SVM)
3. Decision Tree

5.1 Support Vector Machine (SVM) :

L'algorithme SVM, comme l'amplification de gradient, est très populaire, très efficace et fournit un grand nombre d'hyperparamètres à régler.

Le premier paramètre important est peut-être le choix du noyau qui contrôlera la manière dont les variables d'entrée seront projetées. Il y a beaucoup de choix, mais linéaire, polynomial et RBF sont les plus courants, peut-être juste linéaire et RBF dans la pratique.

• noyaux dans ['linéaire', 'poly', 'rbf', 'sigmoïde'] Si le noyau polynomial fonctionne, alors c'est une bonne idée de plonger dans l'hyperparamètre de degré.

• C Un autre paramètre critique est la pénalité (C) qui peut prendre une gamme de valeurs et a un effet dramatique sur la forme des régions résultantes pour chaque classe. Une échelle logarithmique peut être un bon point de départ.

C dans [100, 10, 1,0, 0,1, 0,001].

5.1.1 Recherche des meilleurs paramètres :

Entrée [11]:
```
predict_svm = []

for i in range(len(loaded_images_test)):
    fig = plt.figure()
    predict_svm.append(modele_svm.predict([features.iloc[i]]))
    fig.suptitle(Etiq[int(predict_svm[i])])
    plt.imshow(loaded_images_test[i])

df1 = pd.DataFrame(data=img_tst_name)
predict_svm_New = []
for i in predict_svm:
    if i == 0:
        predict_svm_New.append('obj_car')
    else:
        predict_svm_New.append('obj_shape')
df2 = pd.DataFrame(data=predict_svm_New)
svm_rapport = pd.concat([df1,df2],axis=1)
svm_rapport.to_csv('prediction_Par_SVM_soukayna.csv',sep=',',header=['name','predict'],index=False)
```

5.1.2 Trainer, tester et sauvegarder le modèle :

5.2 K-Nearest Neighbors (KNN) :

L'hyperparamètre le plus important pour KNN est le nombre de voisins (n_neighbors).

5.2.1 Recherche des meilleurs paramètres :

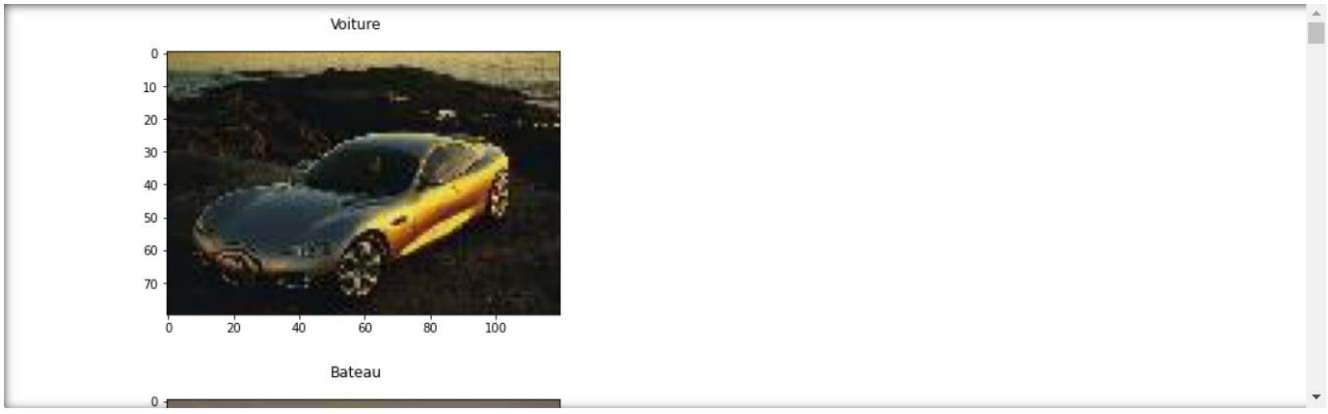
```
Entrée [12]: from sklearn.neighbors import KNeighborsClassifier
             modele_knn = KNeighborsClassifier(n_neighbors=1)
             modele_knn.fit(balanced_fe, balanced_tr)

Out[12]: KNeighborsClassifier(n_neighbors=1)
```

```
Entrée [19]: predict_knn=[]
             for i in range(len(loaded_images_test)):
                 fig = plt.figure()
                 predict_knn.append(modele_knn.predict([features.iloc[i]]))
                 fig.suptitle(Etiq[int(predict_knn[i])])
                 plt.imshow(loaded_images_test[i])

             df2 = pd.DataFrame(data=predict_knn)
             svm_rapport = pd.concat([df1,df2],axis=1)
             svm_rapport.to_csv('prediction_Par_KNN.csv',sep=',',header=['name','predict'],index=False)
```

5.2.1 Trainer, tester et sauvegarder le modèle :



5.3 Decision Tree Classifier :

5.3.1 Recherche des meilleurs paramètres :

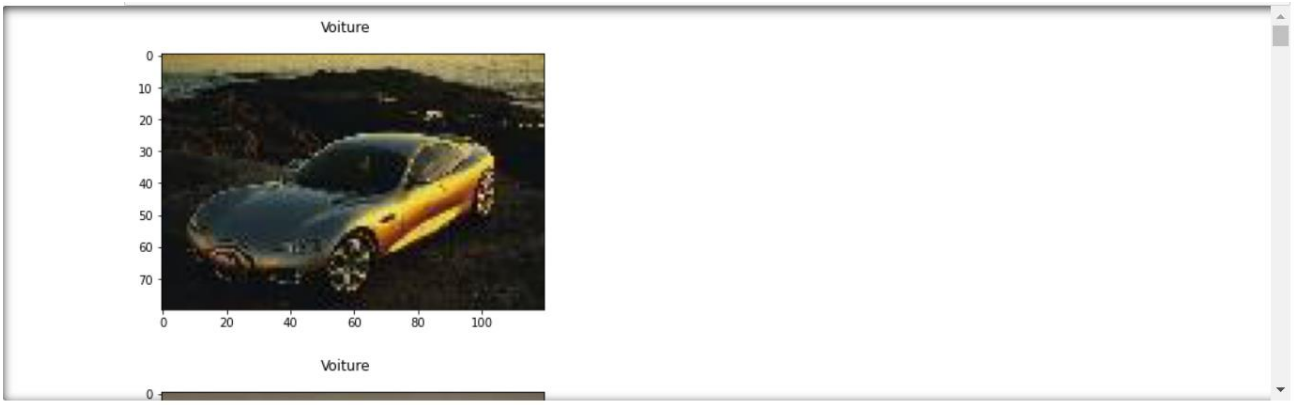
Entrée [14]:
```
from sklearn.tree import DecisionTreeClassifier

modele_arb = DecisionTreeClassifier()
modele_arb.fit(balanced_fe,balanced_tr)
```

Out[14]:
```
DecisionTreeClassifier()
```

Entrée [15]:
```
predict_arb=[]
for i in range(len(loaded_images_test)):
    fig = plt.figure()
    predict_arb.append(modele_arb.predict([features.iloc[i]]))
    fig.suptitle(Etiq[int(predict_arb[i])])
    plt.imshow(loaded_images_test[i])
predict_arb_New = []
for i in predict_svm:
    if i == 0:
        predict_arb_New.append('obj_car')
    else:
        predict_arb_New.append('obj_shape')
df2 = pd.DataFrame(data=predict_arb_New)
svm_rapport = pd.concat([df1,df2],axis=1)
svm_rapport.to_csv('prediction_Par_ARBRE_soukayna.csv',sep=',',header=['name','predict'],index=False)
```

5.3.2 Trainer, tester et sauvegarder le modèle :



6. Prédiction :

6.1 Matrice de confusion :

6.1.1 Support Vector Machine (SVM) :

Entrée [23]:
```
from sklearn.metrics import classification_report, confusion_matrix

y_true = [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]
print("La matrice de confusion en utilisant le modèle SVM")
print(confusion_matrix(y_true, predict_svm))

print(classification_report(y_true, predict_svm))
```

```
La matrice de confusion en utilisant le modèle SVM
[[10  0]
 [ 1  9]]
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        10
           1       1.00      0.90      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

6.1.2 K-Nearest Neighbors (KNN) :

Entrée [16]:
```
print("La matrice de confusion en utilisant le modèle KNN")
print(confusion_matrix(y_true, predict_knn))

print(classification_report(y_true, predict_knn))
```

```
La matrice de confusion en utilisant le modèle KNN
[[ 6  4]
 [ 0 10]]
              precision    recall  f1-score   support

           0       1.00      0.60      0.75        10
           1       0.71      1.00      0.83        10

    accuracy                           0.80        20
   macro avg       0.86      0.80      0.79        20
weighted avg       0.86      0.80      0.79        20
```

6.1.3 Decision Tree Classifier :

Entrée [15]:
```
print("La matrice de confusion en utilisant le modèle Tree_Decision")
print(confusion_matrix(y_true, predict_arb))

print(classification_report(y_true, predict_arb))
```

```
La matrice de confusion en utilisant le modèle Tree_Decision
[[ 9  1]
 [ 0 10]]
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        10
           1       0.91      1.00      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

7. Conclusion :

Dans cet atelier on a essayé de classifier des images, passant par l'extraction des caractéristiques à la recherche des meilleures valeurs des paramètres des modèles, à l'entrainement des modèles, en arrivants finalement à prédire les classes des images non classifier et comparer les résultats donnés par ces modèles.