

الجمهورية الجزائرية الديمقراطية الشعبية  
وزارة التعليم العالي والبحث العلمي

UNIVERSITÉ BADJI MOKHTAR - ANNABA  
BADJI MOKHTAR – ANNABA UNIVERSITY



جامعة باجي مختار – عنابة

## Analyse des Méthodes de Classification pour la Prédiction de Types de Cancer à partir du Dataset HAM10000

**Présenté par :** Mekersi Ilyes  
Laraba Abd Raouf

Année Universitaire : 2023/2024

## Table de matière :

1) Analyse exploratoire des données du dataset HAM10000 :.....	3
2) Répartition des types de cancer dans l'ensemble de données HAM10000 : .....	5
3) Chargement des données du dataset en python : .....	6
4) Bayes naïf (BN) :.....	8
5) K plus proches voisins (KNN) : .....	10
6) Arbre de décision : .....	12
7) Machine à vecteurs de support (SVM) : .....	14
8) K-means Clustering : .....	16

## Introduction

Ce rapport présente une analyse des données dermatoscopiques utilisant des techniques d'apprentissage automatique. Les données sont issues de l'ensemble de données HAM10000, qui contient des images de lésions cutanées accompagnées de métadonnées cliniques. L'objectif principal de cette analyse est de développer des modèles de classification pour diagnostiquer les différents types de lésions cutanées.

Nous commençons par prétraiter les données, en extrayant les métadonnées et en préparant les images pour l'analyse. Ensuite, nous explorons plusieurs algorithmes d'apprentissage automatique, notamment les arbres de décision, les forêts aléatoires, et les méthodes de clustering comme K-means. Nous évaluons la performance de ces modèles à l'aide de différentes mesures telles que l'exactitude, la précision et le rappel.

## 1) Analyse exploratoire des données du dataset HAM10000 :

- Visualisation de la distribution des données :

```
# Label encoding to numeric values from text
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(skin_df['dx'])
skin_df['Label'] = le.transform(skin_df["dx"])

# Data distribution visualization
fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(221)
skin_df['dx'].value_counts().plot(kind='bar', ax=ax1)
ax1.set_ylabel('Count')
ax1.set_title('Cell Type')

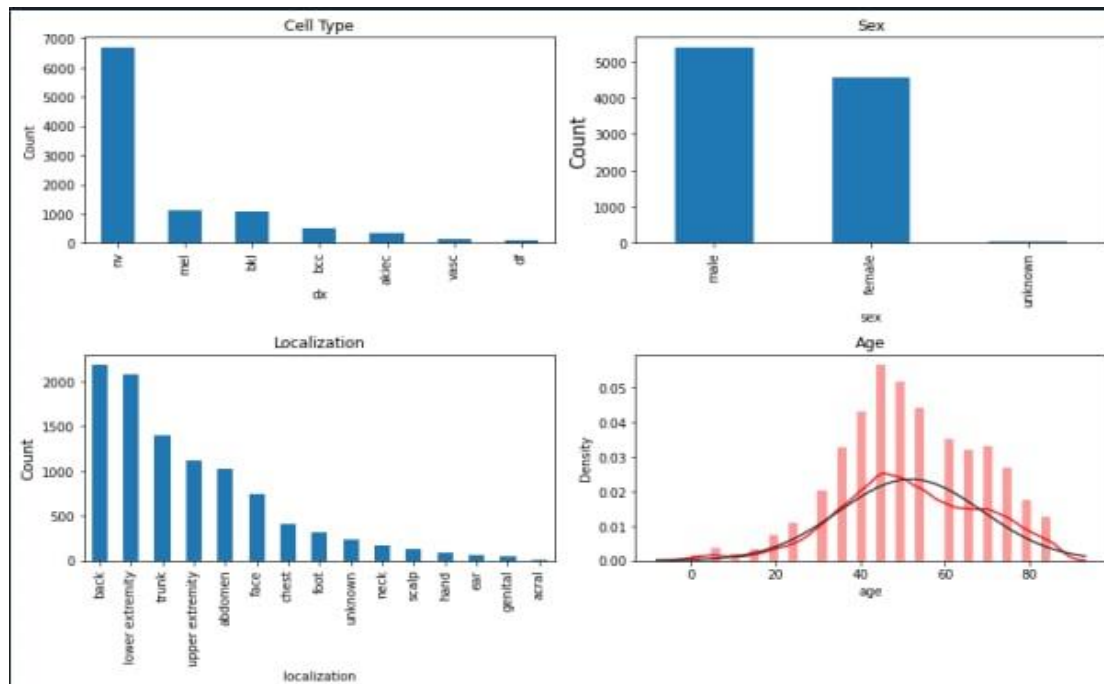
ax2 = fig.add_subplot(222)
skin_df['sex'].value_counts().plot(kind='bar', ax=ax2)
ax2.set_ylabel('Count', size=15)
ax2.set_title('Sex')

ax3 = fig.add_subplot(223)
skin_df['Localization'].value_counts().plot(kind='bar')
ax3.set_ylabel('Count', size=12)
ax3.set_title('Localization')

ax4 = fig.add_subplot(224)
sample_age = skin_df[pd.notnull(skin_df['age'])]
sns.distplot(sample_age['age'], fit=stats.norm, color='red')
ax4.set_title('Age')

plt.tight_layout()
plt.show()
```

Dans cette partie, nous visualisons la distribution des données en traçant des graphiques pour les types de cellules, le sexe, la localisation et l'âge des patients.

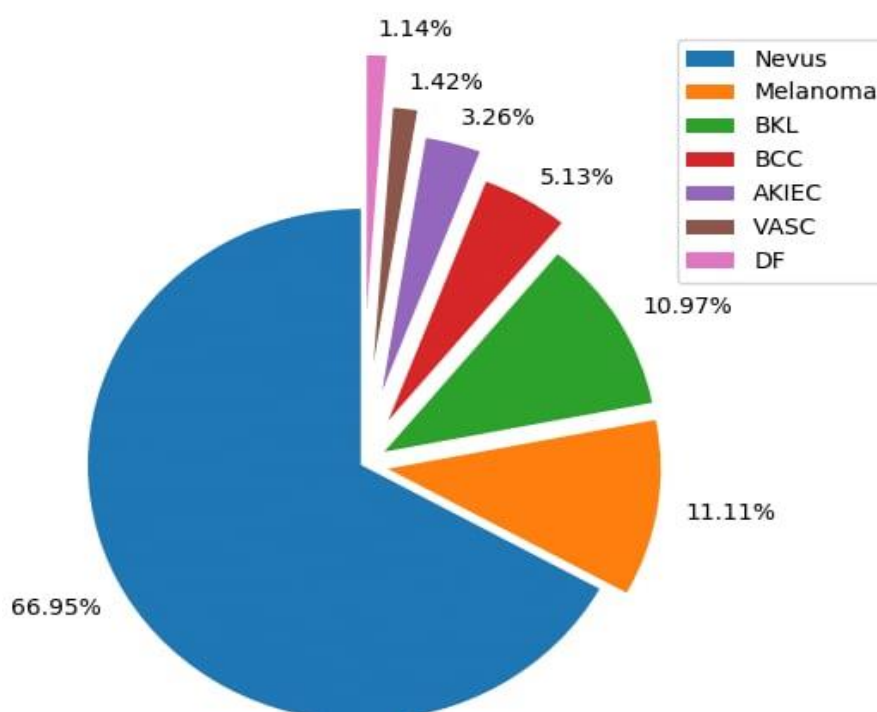


- ❖ Distribution des types de cellules (dx) : Ce subplot met en évidence la fréquence de chaque type de lésion cutanée dans l'échantillon, ce qui permet de comprendre la répartition relative des différentes affections dermatologiques étudiées.
- ❖ Répartition par sexe : En affichant la répartition des patients selon leur sexe, ce subplot permet d'évaluer s'il existe des différences significatives dans la prévalence des lésions cutanées entre les sexes.
- ❖ Répartition des localisations des lésions cutanées : Ce subplot offre un aperçu des zones du corps les plus couramment affectées par les lésions cutanées, ce qui peut être utile pour comprendre les schémas de localisation et identifier les zones à risque élevé.
- ❖ Distribution de l'âge des patients : En présentant la distribution de l'âge des patients, ce subplot permet d'évaluer la répartition d'âge dans l'échantillon et d'identifier toute tendance particulière, comme des pics d'incidence à certaines tranches d'âge.

En combinant ces différentes visualisations, le plot offre une compréhension approfondie de la composition démographique de l'échantillon et des caractéristiques des lésions cutanées étudiées. Cela peut être précieux pour orienter les analyses ultérieures et aider à formuler des hypothèses ou des recommandations dans le domaine de la dermatologie.

## 2) Répartition des types de cancer dans l'ensemble de données HAM10000 :

Skin Pathology	Number of images
Actinic keratosis (akiec)	327
Basal cell carcinoma (bcc)	514
Dermatofibroma (df)	115
Melanoma (mel)	1113
Nevus (nv)	6705
Pigmented benign keratosis (bkl)	1099
Vascular lesions (vasc)	142
Total	10015



Ce graphique à secteurs montre la répartition des différents types de cancer dans l'ensemble de données HAM10000. Cet ensemble de données contient des images de lésions cutanées, classées en sept catégories.

Le graphique montre que la majorité des lésions cutanées dans l'ensemble de données (66,95%) appartiennent à la catégorie DF, suivie des catégories AKIEC (10,97%) et VASC (11,11%). Les catégories Nevus, Melanoma, BKL et BCC sont moins représentées, avec des pourcentages respectifs de 1,14%, 1,42%, 3,26% et 5,13%.

Ce graphique à secteurs permet de visualiser rapidement la composition de l'ensemble de données ISIC2018/HAM10000 et de comprendre la répartition des différents types de cancer.

### 3) Chargement des données du dataset en python :

- **Les bibliothèques :**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import time
```

matplotlib.pyplot : Pour créer des visualisations, notamment des graphiques.

numpy : Pour le traitement des tableaux et des matrices.

pandas : Pour la manipulation des données tabulaires à partir du fichier CSV contenant les métadonnées des images.

os : Pour l'accès aux fonctionnalités du système d'exploitation, notamment la manipulation des chemins de fichiers.

glob : Pour trouver tous les chemins correspondants à un modèle de chemin spécifié.

PIL (Pillow) : Pour le chargement et la manipulation d'images.

sklearn.metrics.confusion\_matrix : Pour générer la matrice de confusion.

sklearn.metrics.classification\_report : Pour générer le rapport de classification.

sklearn.naive\_bayes.GaussianNB : Pour implémenter le modèle de Bayes naïf.

sklearn.model\_selection.train\_test\_split : Pour diviser les données en ensembles d'entraînement et de test.

sklearn.preprocessing.LabelEncoder : Pour encoder les étiquettes de texte en valeurs numériques.

time : Pour mesurer le temps d'exécution du code.

Ces bibliothèques sont essentielles pour charger, prétraiter, entraîner et évaluer le modèle de Bayes naïf sur l'ensemble de données HAM10000.

- **Prétraitement des données :**

```
# Read metadata
skin_df = pd.read_csv('C:/Users/WINDOWS/Desktop/ham/HAM10000_metadata.csv')

# Define image size
SIZE = 32

# Label encoding to numeric values from text
le = LabelEncoder()
le.fit(skin_df['dx'])
skin_df['label'] = le.transform(skin_df['dx'])
```

Dans cette partie, nous lisons les métadonnées à partir du fichier CSV contenant des informations sur les images dermatoscopiques. Ensuite, nous définissons une taille d'image de 32x32 pixels. Nous utilisons également LabelEncoder pour convertir les étiquettes de texte en valeurs numériques.

- **Chargement des images :**

```
# Read images based on image ID from the CSV file
image_path = {os.path.splitext(os.path.basename(x))[0]: x
               for x in glob(os.path.join('C:/Users/WINDOWS/Desktop/ham/', '*', '*.jpg'))}

# Define a function to load images with error handling
def load_image(file_path):
    try:
        img = Image.open(file_path)
        img = img.resize((SIZE, SIZE))
        return np.asarray(img)
    except Exception as e:
        return None

start_time = time.time()
# Define image paths and load images
skin_df['path'] = skin_df['image_id'].map(image_path.get)
skin_df['image'] = skin_df['path'].map(load_image)
```

Cette partie charge les images à partir des chemins spécifiés dans le fichier CSV en utilisant une fonction définie pour charger les images avec une gestion d'erreur intégrée. Les images sont redimensionnées à la taille spécifiée et converties en tableaux numpy.

- **Préparation des données pour le modèle :**

```
# Prepare data for modeling
X = np.asarray(df_balanced['image'].tolist()) / 255.0
Y = df_balanced['Label']
Y_cat = to_categorical(Y, num_classes=7)

# Filter out classes with very few samples
class_counts = df_balanced['Label'].value_counts()
valid_classes = class_counts[class_counts >= 100].index
df_balanced_filtered = df_balanced[df_balanced['Label'].isin(valid_classes)]

# Prepare data for modeling
X_filtered = np.asarray(df_balanced_filtered['image'].tolist()) / 255.0
Y_filtered = df_balanced_filtered['Label']
Y_cat_filtered = to_categorical(Y_filtered, num_classes=7)

# Split data into training and testing sets using stratified sampling
x_train, x_test, y_train, y_test = train_test_split(X_filtered, Y_cat_filtered, test_size=0.1, random_state=42, stratify=Y_cat_filtered)
```

Dans cette partie, nous préparons les données pour l'entraînement du modèle en convertissant les listes d'images en tableaux numpy et en divisant les données en ensembles d'entraînement et de test en utilisant une stratification pour maintenir la distribution des classes.

- **Méthode de validation :**

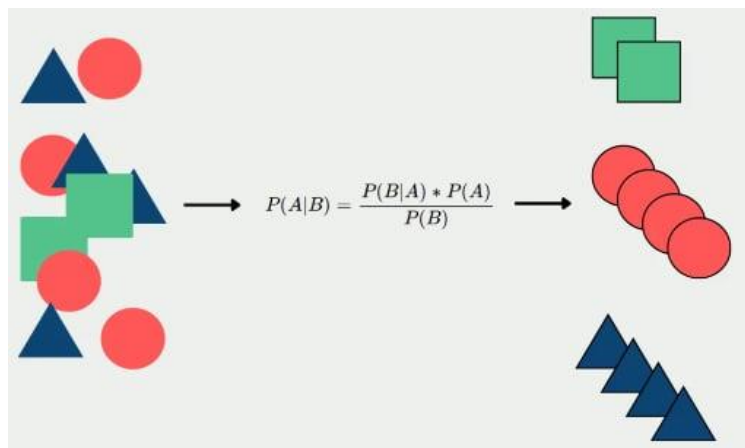
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=42, stratify=Y)
```

La méthode utilisée dans ce code est un "random split", également connu sous le nom de "holdout method", plutôt que la validation croisée. Dans la validation croisée, les données sont divisées en plusieurs sous-ensembles (plis) et le modèle est entraîné et évalué plusieurs fois en utilisant différents plis comme ensemble de test et d'entraînement. En revanche, dans la méthode de répartition aléatoire (random split), les données sont divisées une seule fois en deux ensembles, un pour l'entraînement et un pour les tests, généralement de manière aléatoire.

## 4) Bayes naïf (BN) :

### 1. Définition

Le classificateur Bayes naïf est un modèle probabiliste simple basé sur le théorème de Bayes avec une supposition de naïveté, c'est-à-dire qu'il suppose que les caractéristiques sont conditionnellement indépendantes entre elles. Malgré cette simplification, le classificateur Bayes naïf est souvent utilisé avec succès dans la classification de texte et d'autres tâches de classification.



### 2. Méthodologie

- Entraînement du modèle de Bayes naïf :



```
# Instantiate the Naive Bayes classifier
naive_bayes_model = GaussianNB()

# Train the model
naive_bayes_model.fit(x_train.reshape(x_train.shape[0], -1), y_train)
```

Nous instancions le modèle de Bayes naïf à l'aide de GaussianNB de scikit-learn et nous entraînons le modèle sur les données d'entraînement.



### 3. Résultats

	Resultat 1	Resultat 2
Description	<code>test_size=0.01</code> Dans ce premier test on a commencé par prendre 1% du dataset pour le test.	<code>test size=0.1</code> On a recommencé l'exécution mais cette fois on a pris 10% du dataset pour le test.
Accuracy	<code>Test accuracy: 0.3564356435643564</code> Exactitude sur les données de test : 35.64 %	<code>Test accuracy: 0.3962075848303393</code> Exactitude sur les données de test : 39.62 %
Temps d'exécution	<code>Execution time: 66.20 seconds</code> Temps d'exécution : 66.20 secondes	<code>Execution time: 68.00 seconds</code> Temps d'exécution : 68 secondes
Matrice de confusion	<pre>Confusion matrix: [[ 0  1  1  1  0  0  0]  [ 1  3  0  1  0  0  0]  [ 2  0  5  1  1  1  1]  [ 1  0  0  0  0  0  0]  [ 1  0  2  0  3  3  2]  [ 3  8 12  1  9 23 12]  [ 0  0  0  0  0  0  2]]</pre>	<pre>Confusion matrix: [[ 6  7  6  6  0  1  7]  [ 8 25  7  4  1  1  5]  [ 5 15 50  6 13  8 13]  [ 2  4  3  3  0  0  0]  [10 10 28  3 27 10 23]  [22 73 97 17 63 282 117]  [ 1  3  3  2  1  0  4]]</pre>
Rapport de classification	<pre>Classification report:               precision    recall  f1-score   support     akiec      0.00      0.00      0.00         3     bcc      0.25      0.60      0.35         5     bkl      0.25      0.45      0.32        11     df       0.00      0.00      0.00         1     mel      0.23      0.27      0.25        11     nv       0.85      0.34      0.48        68     vasc      0.12      1.00      0.21         2   accuracy      0.36      101   macro avg      0.24      0.38      0.23      101  weighted avg      0.64      0.36      0.41      101</pre>	<pre>Classification report:               precision    recall  f1-score   support     akiec      0.11      0.18      0.14        33     bcc      0.18      0.49      0.27        51     bkl      0.26      0.45      0.33       110     df       0.07      0.25      0.11        12     mel      0.26      0.24      0.25       111     nv       0.93      0.42      0.58       671     vasc      0.02      0.29      0.04        14   accuracy      0.40      1002   macro avg      0.26      0.33      0.25      1002  weighted avg      0.70      0.40      0.47      1002</pre>
		

- **Comparaison des résultats :** Pour la comparaison des deux résultats de Bayes naïf, le deuxième modèle a une légèrement meilleure exactitude sur les données de test (39.62% contre 35.64% pour le premier modèle) mais il prend aussi un peu plus de temps pour s'exécuter (68 secondes contre 66.20 secondes pour le premier modèle). Le choix dépendrait des besoins spécifiques : si

le temps d'exécution est critique, le premier modèle pourrait être préférable malgré une exactitude légèrement inférieure, tandis que si l'exactitude est prioritaire et que le temps d'exécution supplémentaire n'est pas un problème, le deuxième modèle pourrait être choisi.

## 5) K plus proches voisins (KNN) :

### 1. Définition

L'algorithme des k plus proches voisins (KNN) est une méthode d'apprentissage supervisé utilisée pour la classification et la régression. Dans KNN, l'étiquette d'un échantillon est prédite en examinant les k échantillons les plus proches dans l'espace des caractéristiques et en prenant une décision basée sur la majorité de leurs étiquettes.

### 2. Méthodologie

- Les bibliothèques :

```
from sklearn.neighbors import KNeighborsClassifier
```

KNeighborsClassifier est une classe de la bibliothèque scikit-learn qui implémente l'algorithme des k plus proches voisins (KNN) pour la classification supervisée. Dans l'algorithme KNN, lorsqu'une prédiction doit être faite pour une nouvelle instance, l'algorithme recherche les k instances les plus proches dans l'ensemble de données d'entraînement. La classe de l'instance majoritaire parmi ces voisins est ensuite attribuée à la nouvelle instance. En d'autres termes, la prédiction est basée sur la classe majoritaire des k voisins les plus proches. KNeighborsClassifier offre une flexibilité pour personnaliser différentes métriques de distance (euclidienne, de Manhattan, etc.) et les poids des voisins (uniforme ou pondéré).

- Apprentissage et évaluation du modèle KNN :

```
# Reshape data for KNN
x_train_flattened = x_train.reshape(x_train.shape[0], -1)
x_test_flattened = x_test.reshape(x_test.shape[0], -1)

# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3) # You can adjust the number of neighbors as needed
knn.fit(x_train_flattened, y_train)

# Evaluate the KNN model
accuracy = knn.score(x_test_flattened, y_test)
```

Dans cette partie, nous initialisons et entraînons un classificateur KNN sur les données d'entraînement, puis évaluons ses performances sur les données de test.

### 3. Résultats :

	Resultat 1	Resultat 2
Description	<code>n_neighbors=5</code> On a commencé par essayer 5 voisins pour ce premier test KNN.	<code>(n_neighbors=3)</code> On a recommencé par essayer 3 voisins cette fois.
Accuracy	<code>Test accuracy: 0.8174659985683608</code> Exactitude sur les données de test : 81.74%	<code>Test accuracy: 0.8883321403006442</code> Exactitude sur les données de test : 88.83%

Temps d'exécution	<pre>Precision: 0.8195912485744501 Recall: 0.8210450966356478 F1 Score: 0.8136492901582006 Error: 0.18253400143163923 Execution time: 96.90783929824829 sec</pre>	<pre>Precision: 0.8879267680408609 Recall: 0.8883321403006442 F1 Score: 0.8856953192767975 Error: 0.11166785969935578 Execution time: 99.01370644569397 sec</pre>
	Temps d'exécution : 96 secondes	Temps d'exécution : 99 secondes
Matrice de confusion		
Rapport de classification	<pre>Classification Report:               precision    recall  f1-score   support       0:       0.75        0.95        0.84        200      1:       0.87        0.88        0.87        200      2:       0.72        0.74        0.73        198      3:       0.93        1.00        0.97        200      4:       0.78        0.55        0.64        200      5:       0.79        0.64        0.71        199      6:       0.89        1.00        0.94        200   accuracy          0.82  macro avg         0.82        0.82        0.81        1397  weighted avg      0.82        0.82        0.81        1397</pre>	<pre>Classification Report:               precision    recall  f1-score   support       0:       0.90        0.98        0.94        200      1:       0.93        0.94        0.93        200      2:       0.76        0.85        0.80        198      3:       0.97        1.00        0.98        200      4:       0.85        0.72        0.78        200      5:       0.87        0.73        0.79        199      6:       0.93        1.00        0.97        200   accuracy          0.89  macro avg         0.89        0.89        0.89        1397  weighted avg      0.89        0.89        0.89        1397</pre>

- Comparaison des résultats :** Le deuxième résultat semble avoir de meilleures performances en termes d'exactitude, de précision, de rappel et de score F1, mais il nécessite également un peu plus de temps pour s'exécuter. Le choix entre les deux résultats dépendrait des priorités du projet : si l'accent est mis sur la précision et la performance du modèle, le deuxième résultat pourrait être préféré malgré le léger surcoût en temps d'exécution. Si le temps d'exécution est crucial et que les performances du modèle sont acceptables, le premier résultat pourrait être choisi.

## 6) Arbre de décision :

### 1. Définition

L'arbre de décision est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il divise l'ensemble de données en sous-ensembles plus petits en fonction des caractéristiques de l'ensemble de données. Ces divisions sont effectuées de manière récursive selon un critère tel que la pureté des classes dans chaque sous-ensemble. L'objectif est de construire un arbre qui prend des décisions en testant les valeurs des caractéristiques.

### 2. Méthodologie

- Prétraitement des données:

```
# Label encoding to numeric values from text
le = LabelEncoder()
le.fit(skin_df['dx'])
skin_df['Label'] = le.transform(skin_df["dx"])

# Balance data
df_balanced = skin_df.groupby('Label').apply(lambda x: x.sample(n=2000, replace=True, random_state=42)).reset_index(drop=True)

# Check for classes with fewer samples than the desired test size
classes_with_few_samples = df_balanced['Label'].value_counts()[df_balanced['Label'].value_counts() < 100]

# Remove these classes from the data
df_balanced = df_balanced[~df_balanced['Label'].isin(classes_with_few_samples.index)]
```

Cette partie du code effectue diverses opérations de prétraitement sur les données, telles que l'encodage des étiquettes en valeurs numériques, l'équilibrage des données en échantillonnant aléatoirement un nombre fixe d'exemples de chaque classe, et la suppression des classes avec un nombre d'échantillons inférieur à un seuil donné.

- Préparation des données pour la modélisation:

```
# Prepare data for modeling
X = np.asarray(df_balanced['image'].tolist())
Y = df_balanced['Label']

# Split data into training and testing sets using stratified sampling
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.17, random_state=42, stratify=Y)

# Scale the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

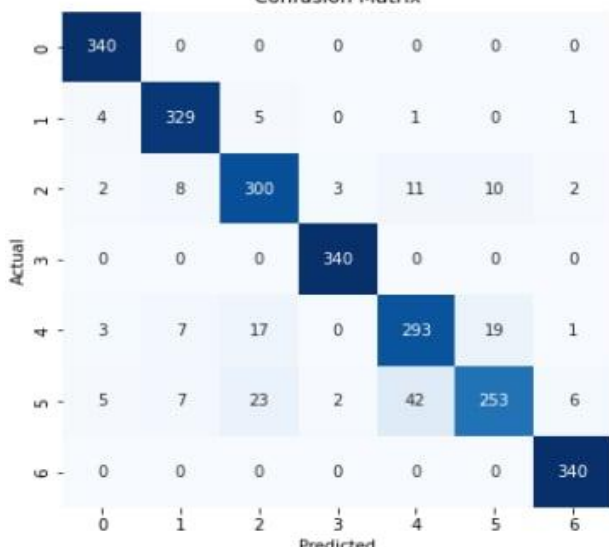
Cette partie du code prépare les données pour l'entraînement et le test du modèle. Les fonctionnalités et les étiquettes sont séparées, puis les données sont divisées en ensembles d'entraînement et de test. Enfin, les fonctionnalités sont mises à l'échelle pour une meilleure convergence du modèle.

- **Modélisation :**

```
# Use Decision Tree classifier
dt = DecisionTreeClassifier(random_state=42)
parameters = {'max_depth': [None, 10, 20, 50],
              'min_samples_split': [2, 5, 10]}
grid_search = GridSearchCV(dt, parameters, cv=3, n_jobs=-1)
start_time = time()
grid_search.fit(x_train, y_train)
best_dt = grid_search.best_estimator_
```

Cette partie du code crée un classificateur d'arbre de décision et utilise une recherche sur grille pour trouver les meilleurs hyperparamètres pour ce classificateur.

### 3. Résultats

	Resultat																																																							
Description :																																																								
Accuracy	<div>Test accuracy: 0.9245998315080034</div> <div>Exactitude sur les données de test : 92.45%</div>																																																							
Temps d'exécution	<div>Execution Time: 191.80520296096802 seconds</div> <div>Temps d'exécution : 191.805 secondes</div>																																																							
Matrice de confusion	<div>Confusion Matrix</div>  <p>Actual</p> <p>Predicted</p>																																																							
Rapport de classification	<div>Classification Report:</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>akiec</td><td>0.96</td><td>1.00</td><td>0.98</td><td>340</td></tr><tr><td>bcc</td><td>0.94</td><td>0.97</td><td>0.95</td><td>340</td></tr><tr><td>bkl</td><td>0.87</td><td>0.89</td><td>0.88</td><td>336</td></tr><tr><td>df</td><td>0.99</td><td>1.00</td><td>0.99</td><td>340</td></tr><tr><td>mel</td><td>0.84</td><td>0.86</td><td>0.85</td><td>340</td></tr><tr><td>nv</td><td>0.90</td><td>0.75</td><td>0.82</td><td>338</td></tr><tr><td>vasc</td><td>0.97</td><td>1.00</td><td>0.99</td><td>340</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.92</td><td>2374</td></tr><tr><td>macro avg</td><td>0.92</td><td>0.92</td><td>0.92</td><td>2374</td></tr><tr><td>weighted avg</td><td>0.92</td><td>0.92</td><td>0.92</td><td>2374</td></tr></table>		precision	recall	f1-score	support	akiec	0.96	1.00	0.98	340	bcc	0.94	0.97	0.95	340	bkl	0.87	0.89	0.88	336	df	0.99	1.00	0.99	340	mel	0.84	0.86	0.85	340	nv	0.90	0.75	0.82	338	vasc	0.97	1.00	0.99	340	accuracy			0.92	2374	macro avg	0.92	0.92	0.92	2374	weighted avg	0.92	0.92	0.92	2374
	precision	recall	f1-score	support																																																				
akiec	0.96	1.00	0.98	340																																																				
bcc	0.94	0.97	0.95	340																																																				
bkl	0.87	0.89	0.88	336																																																				
df	0.99	1.00	0.99	340																																																				
mel	0.84	0.86	0.85	340																																																				
nv	0.90	0.75	0.82	338																																																				
vasc	0.97	1.00	0.99	340																																																				
accuracy			0.92	2374																																																				
macro avg	0.92	0.92	0.92	2374																																																				
weighted avg	0.92	0.92	0.92	2374																																																				

## 7) Machine à vecteurs de support (SVM) :

### 1. Définition

La machine à vecteurs de support (SVM) est un algorithme de machine learning utilisé pour la classification et la régression. L'objectif de SVM est de trouver un hyperplan dans un espace de caractéristiques qui sépare de manière optimale les différentes classes d'observations. Il est particulièrement efficace dans les cas où les données sont linéairement séparables, mais peut également être étendu à des espaces de caractéristiques de plus grande dimension grâce à l'utilisation de noyaux.

### 2. Méthodologie

- Les bibliothèques :

```
from sklearn.svm import SVC
```

Cette bibliothèque fournit une implémentation de l'algorithme de machine à vecteurs de support (SVM) pour la classification et la régression. SVC (Support Vector Classifier) est utilisé pour entraîner un modèle SVM à l'aide de divers noyaux, tels que linéaire, polynomial, radial basis function (RBF), etc. Il permet d'ajuster les paramètres pour optimiser la performance du modèle.

- Entraînement du modèle SVM :

```
# Define and train the SVM model
svm_model = SVC(kernel='linear', random_state=42) # You can try different kernels, e.g., 'rbf'
svm_model.fit(x_train, y_train)
```

Dans cette partie, nous définissons un modèle SVM avec un noyau linéaire et l'entraînons sur les données d'entraînement.

- Évaluation du modèle :

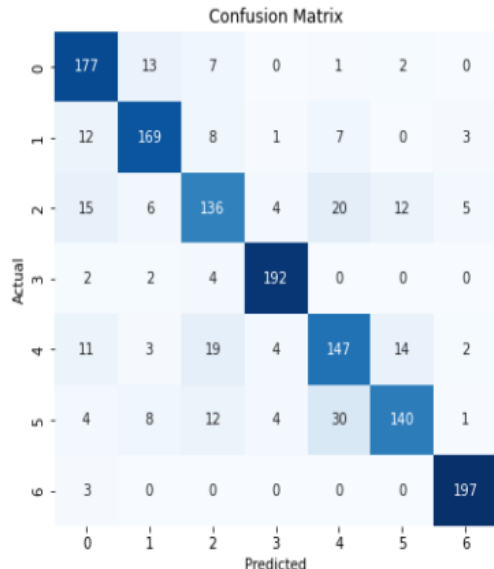
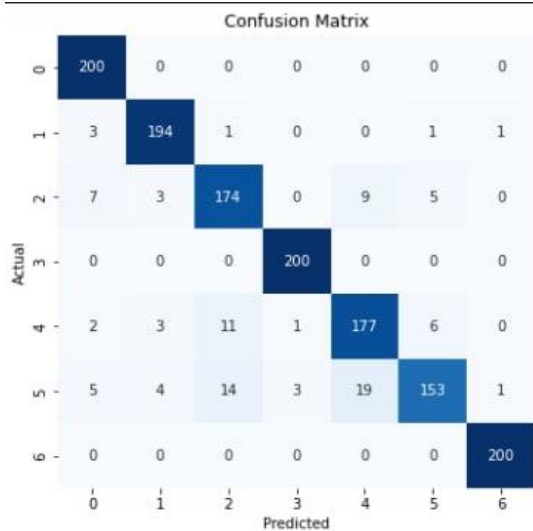


```
# Predict on the test data
y_pred = svm_model.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

Dans cette partie, nous utilisons le modèle entraîné pour prédire les étiquettes des données de test, puis nous évaluons les performances du modèle en termes d'exactitude, de précision, de rappel et de score F1 sur les données de test.

### 3. Résultats

	Resultat 1	Resultat 2
Description	<pre>kernel='rbf'</pre> On a fait un test avec kernel "rbf" (Radial Basis Function)	<pre>kernel='linear'</pre> Dans ce résultat on a changé le kernel vers linear

Accuracy	<div>Test accuracy: 0.8289191123836793 Precision: 0.8281726767270469 Recall: 0.8289191123836793 F1 Score: 0.8271921034267908 Execution time: 309.309583902359 seconds</div> <div>Exactitude sur les données de test :82.89 Précision : 82.81% Rappel :82.91 % Score F1 : 82.71%</div>	<div>Test accuracy: 0.9291338582677166 Precision: 0.929202797835135 Recall: 0.9291338582677166 F1 Score: 0.9278326124222405 Execution time: 240.5082724094391 seconds</div> <div>Exactitude sur les données de test :92.91 % Précision : 92.92% Rappel :92.91 % Score F1 : 92.78%</div>																																																																																																														
Temps d'exécution	Temps d'exécution : 309.30 secondes	Temps d'exécution : 240.50 secondes																																																																																																														
Matrice de confusion	<div>Confusion Matrix</div>  <div>Actual</div> <div>Predicted</div>	<div>Confusion Matrix</div>  <div>Actual</div> <div>Predicted</div>																																																																																																														
Rapport de classification	<table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>akiec</td><td>0.79</td><td>0.89</td><td>0.83</td><td>200</td></tr><tr><td>bcc</td><td>0.84</td><td>0.84</td><td>0.84</td><td>200</td></tr><tr><td>bk1</td><td>0.73</td><td>0.69</td><td>0.71</td><td>198</td></tr><tr><td>df</td><td>0.94</td><td>0.96</td><td>0.95</td><td>200</td></tr><tr><td>mel</td><td>0.72</td><td>0.73</td><td>0.73</td><td>200</td></tr><tr><td>nv</td><td>0.83</td><td>0.70</td><td>0.76</td><td>199</td></tr><tr><td>vasc</td><td>0.95</td><td>0.98</td><td>0.97</td><td>200</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.83</td><td>1397</td></tr><tr><td>macro avg</td><td>0.83</td><td>0.83</td><td>0.83</td><td>1397</td></tr><tr><td>weighted avg</td><td>0.83</td><td>0.83</td><td>0.83</td><td>1397</td></tr></tbody></table>		precision	recall	f1-score	support	akiec	0.79	0.89	0.83	200	bcc	0.84	0.84	0.84	200	bk1	0.73	0.69	0.71	198	df	0.94	0.96	0.95	200	mel	0.72	0.73	0.73	200	nv	0.83	0.70	0.76	199	vasc	0.95	0.98	0.97	200	accuracy			0.83	1397	macro avg	0.83	0.83	0.83	1397	weighted avg	0.83	0.83	0.83	1397	<table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>akiec</td><td>0.92</td><td>1.00</td><td>0.96</td><td>200</td></tr><tr><td>bcc</td><td>0.95</td><td>0.97</td><td>0.96</td><td>200</td></tr><tr><td>bk1</td><td>0.87</td><td>0.88</td><td>0.87</td><td>198</td></tr><tr><td>df</td><td>0.98</td><td>1.00</td><td>0.99</td><td>200</td></tr><tr><td>mel</td><td>0.86</td><td>0.89</td><td>0.87</td><td>200</td></tr><tr><td>nv</td><td>0.93</td><td>0.77</td><td>0.84</td><td>199</td></tr><tr><td>vasc</td><td>0.99</td><td>1.00</td><td>1.00</td><td>200</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>1397</td></tr><tr><td>macro avg</td><td>0.93</td><td>0.93</td><td>0.93</td><td>1397</td></tr><tr><td>weighted avg</td><td>0.93</td><td>0.93</td><td>0.93</td><td>1397</td></tr></tbody></table>		precision	recall	f1-score	support	akiec	0.92	1.00	0.96	200	bcc	0.95	0.97	0.96	200	bk1	0.87	0.88	0.87	198	df	0.98	1.00	0.99	200	mel	0.86	0.89	0.87	200	nv	0.93	0.77	0.84	199	vasc	0.99	1.00	1.00	200	accuracy			0.93	1397	macro avg	0.93	0.93	0.93	1397	weighted avg	0.93	0.93	0.93	1397
	precision	recall	f1-score	support																																																																																																												
akiec	0.79	0.89	0.83	200																																																																																																												
bcc	0.84	0.84	0.84	200																																																																																																												
bk1	0.73	0.69	0.71	198																																																																																																												
df	0.94	0.96	0.95	200																																																																																																												
mel	0.72	0.73	0.73	200																																																																																																												
nv	0.83	0.70	0.76	199																																																																																																												
vasc	0.95	0.98	0.97	200																																																																																																												
accuracy			0.83	1397																																																																																																												
macro avg	0.83	0.83	0.83	1397																																																																																																												
weighted avg	0.83	0.83	0.83	1397																																																																																																												
	precision	recall	f1-score	support																																																																																																												
akiec	0.92	1.00	0.96	200																																																																																																												
bcc	0.95	0.97	0.96	200																																																																																																												
bk1	0.87	0.88	0.87	198																																																																																																												
df	0.98	1.00	0.99	200																																																																																																												
mel	0.86	0.89	0.87	200																																																																																																												
nv	0.93	0.77	0.84	199																																																																																																												
vasc	0.99	1.00	1.00	200																																																																																																												
accuracy			0.93	1397																																																																																																												
macro avg	0.93	0.93	0.93	1397																																																																																																												
weighted avg	0.93	0.93	0.93	1397																																																																																																												
																																																																																																																



- **Comparaison des résultats** : Le deuxième résultat semble avoir de meilleures performances en termes d'exactitude, de précision, de rappel et de score F1, mais il nécessite également un peu plus de temps pour s'exécuter. Le choix entre les deux résultats dépendrait des priorités du projet : si l'accent est mis sur la précision et la performance du modèle, le deuxième résultat pourrait être préféré malgré le léger surcoût en temps d'exécution. Si le temps d'exécution est crucial et que les performances du modèle sont acceptables, le premier résultat pourrait être choisi.

## 8) K-means Clustering :

### 1. Définition

L'algorithme K-means est une méthode de clustering non supervisée utilisée pour regrouper des données similaires en k clusters. L'objectif principal de K-means est de partitionner les données en clusters de manière à minimiser la variance intra-cluster, c'est-à-dire que les observations à l'intérieur d'un cluster sont les plus similaires possibles les unes aux autres.

### 2. Méthodologie

- Extraction de caractéristiques avec KMeans :

```
# Use KMeans for feature extraction
n_clusters = 200 # You can experiment with different values
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
x_train_features = kmeans.fit_transform(x_train)
x_test_features = kmeans.transform(x_test)
```

Cette partie utilise l'algorithme KMeans pour extraire des caractéristiques à partir des données d'image. Les données sont transformées en utilisant les centroids des clusters comme nouvelles caractéristiques.

- Entraînement du modèle et évaluation :

```
# Use Random Forest classifier instead of Logistic Regression
rf = RandomForestClassifier(random_state=42)
parameters = {'n_estimators': [50, 100, 200],
              'max_depth': [None, 10, 20],
              'min_samples_split': [2, 5, 10]}
grid_search = GridSearchCV(rf, parameters, cv=3, n_jobs=-1)
start_time = time.time()
grid_search.fit(x_train_features, y_train)
best_rf = grid_search.best_estimator_

# Predict on the test data
y_pred = best_rf.predict(x_test_features)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print('Test accuracy:', accuracy)
```

Cette partie utilise un modèle Random Forest pour classer les caractéristiques extraites par KMeans. Le modèle est entraîné et évalué sur les données de test.



### 3. Résultats

	Resultat 1	Resultat 2
Description	<code>n_clusters = 5</code> On essayer un nombre de clusters symbolique ou minimum (5)	<code>n_clusters = 100</code> Cette fois on essayer un nombre grand de clusters (100)
Accuracy	<code>Test accuracy: 0.9241231209735147</code> Exactitude sur les données de test :92.41 %	<code>Test accuracy: 0.9377236936292055</code> Exactitude sur les données de test :93.77 %
Temps d'exécution	<code>Execution Time: 117.60134553909302 seconds</code> Temps d'exécution : 117.60 secondes	<code>Execution Time: 169.87408256530762 seconds</code> Temps d'exécution : 169.87 secondes
Matrice de confusion	<p>Confusion Matrix</p> <p>Actual</p> <p>Predicted</p>	<p>Confusion Matrix</p> <p>Actual</p> <p>Predicted</p>
Rapport de classification	<pre> Classification Report:               precision    recall  f1-score   support       0       0.94        1.00        0.97        200      1       0.89        0.97        0.93        200      2       0.87        0.89        0.88        198      3       0.97        1.00        0.99        200      4       0.88        0.87        0.87        200      5       0.92        0.73        0.82        199      6       0.99        1.00        1.00        200   accuracy          0.92        1397  macro avg         0.92        0.92        0.92        1397  weighted avg      0.92        0.92        0.92        1397 </pre>	<pre> Classification Report:               precision    recall  f1-score   support       0       0.95        1.00        0.97        200      1       0.92        0.98        0.95        200      2       0.90        0.89        0.90        198      3       1.00        1.00        1.00        200      4       0.86        0.89        0.87        200      5       0.92        0.80        0.86        199      6       1.00        1.00        1.00        200   accuracy          0.94        1397  macro avg         0.94        0.94        0.94        1397  weighted avg      0.94        0.94        0.94        1397 </pre>

- **Comparaison des résultats :** Dans les résultats de K-Means, le premier modèle a une exactitude de 92.41% avec un temps d'exécution de 117.60 secondes, tandis que le deuxième modèle a une exactitude légèrement meilleure de 93.77% mais prend plus de temps à s'exécuter, avec 169.87 secondes. La différence entre les deux exactitudes n'est pas aussi significative que dans les résultats précédents, mais le deuxième modèle offre tout de même une amélioration. Cependant, le temps d'exécution plus long du deuxième modèle pourrait être considéré comme une déficience, surtout si les performances supplémentaires ne sont pas considérées comme cruciales pour l'application en question. Ainsi, le premier modèle pourrait être choisi pour son compromis entre exactitude et temps d'exécution.

**Remarque :** Le temps d'exécution inclut le chargement et le prétraitement des images.

## Conclusion

L'analyse des données dermatoscopiques à l'aide de techniques d'apprentissage automatique offre un potentiel considérable pour le diagnostic des lésions cutanées. Les modèles développés dans ce rapport montrent des performances prometteuses dans la classification des différentes classes de lésions cutanées. Cependant, il reste des défis à relever, notamment l'interprétabilité des modèles et la généralisation à de nouveaux ensembles de données. En continuant à explorer et à affiner ces techniques, nous pouvons contribuer à améliorer le diagnostic précoce et le traitement des maladies de la peau.