Introduction to Machine Learning - Project

Talla DIAGNE

12 novembre 2024

Table des matières

Introduction to Machine Learning Assignment

Medical Image Diagnosis Enhancement Using PCAA and KNN

Talla DIAGNE

Nous attestons que ce travail est original, que nous citons en référence toutes les sources utilisées et qu'il ne comporte pas de plagiat.

Introduction

Medical image analysis plays a crucial role in diagnosing diseases and disorders. In this report, we present an analysis of medical images using Principal Component Analysis (PCA) for dimensionality reduction and K-Nearest Neighbors (KNN) for classification. The goal is to develop a model that can accurately classify medical images into different categories, aiding in diagnosis and treatment planning.

1 Dataset exploration and preprocessing

Understanding the dataset is the first step in any machine learning task. Our dataset consists of medical images(MRI brain scans). We associate each image with a label indicating the corresponding medical condition or anomaly. We explored the dataset to gain insights into its structure and characteristics. This exploration involved determining the number of images, image dimensions, and the distribution of labels across the dataset. The dataset contains images from various medical conditions, including glioma, meningioma, pituitary tumors, and normal scans.

```
Train dataset:
Number of images: 5712
Image dimensions: (512, 512, 3)
Label distribution: {'glioma': 1321, 'meningioma': 1339, 'notumor': 1595, 'pituitary': 1457}
```

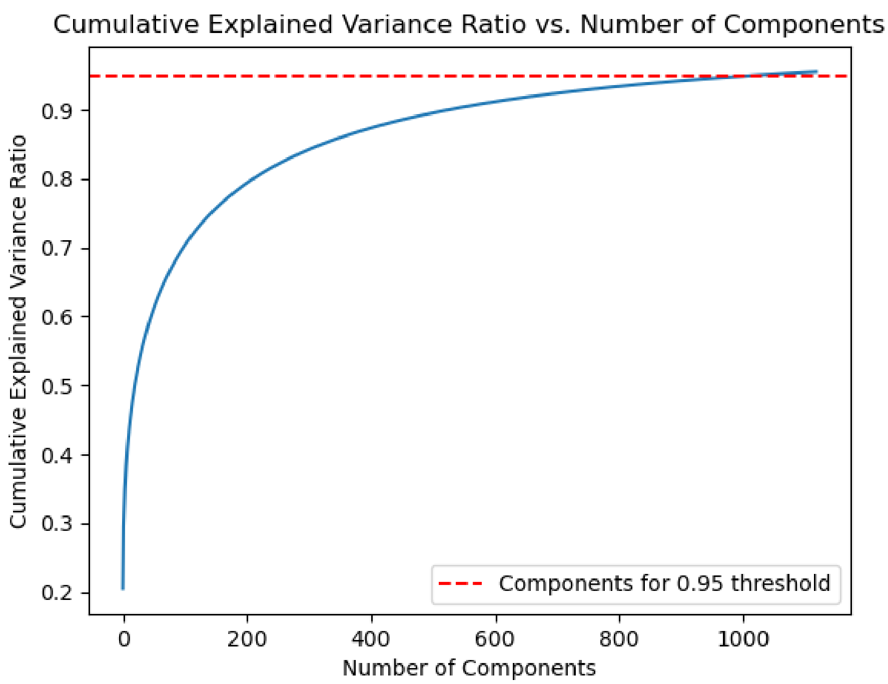
Before applying PCA and KNN, we preprocess the images to ensure they are suitable for analysis. The preprocessing steps include :

- Loading the images from the provided folders and subfolders
- Extracting the labels from the subfolder names
- Converting the images to grayscale to simplify the analysis

— Resizing the images to a consistent size of 100x100 pixels to standardize all the images and flatenning them to prepare it for PCA
— Normalizing the pixel values to be within the range [0, 1] to facilitate computation

2 Principal Component Analysis (PCA)

PCA (Principal Component Analysis) is a technique used to reduce the dimensionality of high-dimensional data while preserving its important features. In our analysis, we applied PCA to preprocess the image data, aiming to decrease its dimensionality while retaining as much variance as possible. To determine the optimal number of principal components, we relied on the explained variance ratio. This ratio indicates the proportion of variance in the data explained by each principal component. By setting a threshold, typically at 95%, we aimed to retain enough variance to accurately represent the data. After analysis, we found that retaining 1119 principal components met our threshold of capturing 95% of the variance. This selection strikes a balance between reducing dimensionality and preserving essential information necessary for subsequent tasks like classification. Including more components may better preserve the informations in the data but increases complexity ; selecting too few components may lead to significant information loss.
However, this result can seem a bit strange when we know that, later in this documents, we will plot the decision boundaries of the KNN classifier but only by using the first 2 parameters.



In order to evaluate the effectiveness of PCA in retaining essential image features, we visually compared original medical images with their reconstructed counterparts. This comparison helps assess how well PCA preserves crucial information during dimensionality reduction, guiding the determination of optimal principal components for accurate classification

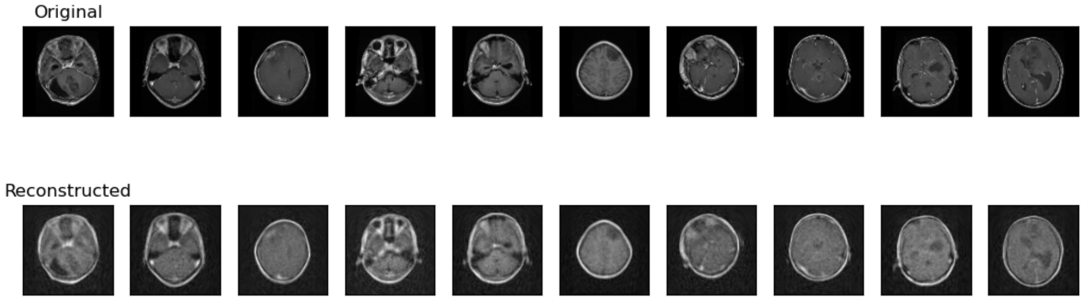


FIGURE 1 – Comparison between original and reconstructed medical images

3 Training and evaluation

With the preprocessed data and the reduced feature space obtained from PCA, we train a KNN classifier. KNN is an effective algorithm for classification tasks, particularly when dealing with labeled datasets.To optimize the performance of the KNN classifier, we explore the impact of different values of the hyperparameter k, which represents the number of neighbors used in the classification. We use a technique called cross-validation to find the optimal value of k that maximizes the classifier's performance. One way to visualize the relationship between the value of k and the classifier's performance is to plot the accuracy against the number of neighbors.

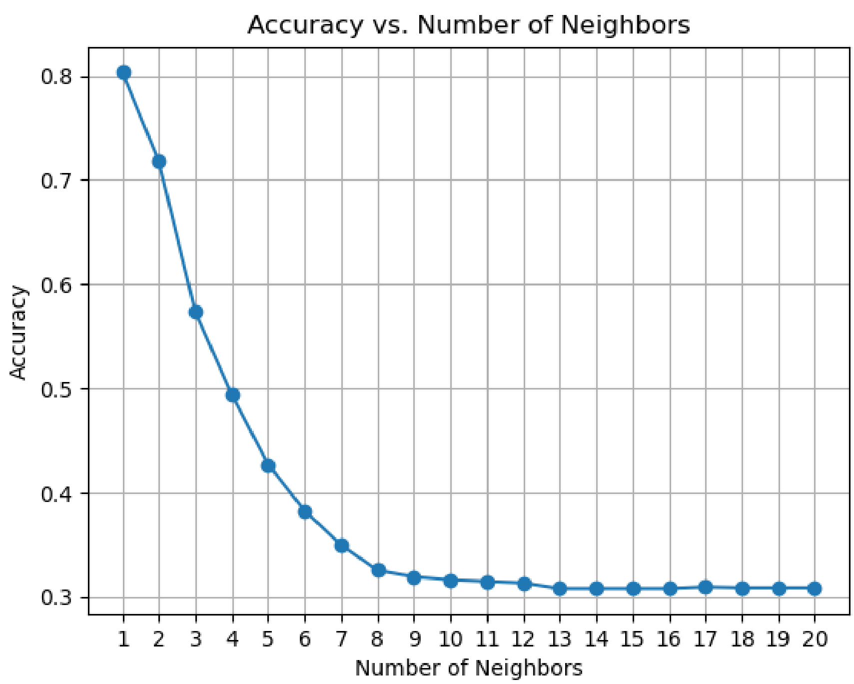


FIGURE 2 – Program performance depending on the number of neighbors

This plot [2] helps us understand how the choice of k affects the accuracy of the classifier. By analyzing this curve, we can determine the optimal value of k that yields the highest accuracy on the test data. We also decided to keep $k = 2$, because the accuracy is very good.

To visualize the performance of the KNN classifier, decision boundaries were additionally plotted for various values of k. Decision boundaries delineate the classification boundaries between different classes within the feature space. By plotting these boundaries for different k values, the model's complexity can be observed as it varies with the number of neighbors considered for classification. For each k value, data points were colored according to their actual class. Subsequently, the feature space was divided into regions colored based on the class predicted by the KNN classifier. This visualization provides insight into how decision boundaries change with varying k values and their impact on the model's generalization to unseen data.

The uniformity of decision boundaries in the plot can vary with different values of 'k' in the k-nearest neighbors (KNN) algorithm. A smaller 'k' tends to result in more irregular and fragmented decision boundaries, as the classifier relies on a smaller number of nearest neighbors for prediction, making it more sensitive to noise in the data. Conversely, larger values of 'k' lead to smoother and more uniform decision boundaries, as the classifier considers a larger number of neighbors, resulting in a more generalized representation of the feature space. This variation in decision boundary uniformity highlights the relation between bias and variance in the KNN algorithm and emphasizes the importance of selecting an appropriate value of 'k' based on the complexity of the dataset and desired model behavior. However, the result here seems a bit strange to us because with the optimal value found before (k=2), the image is not the smoothest one.

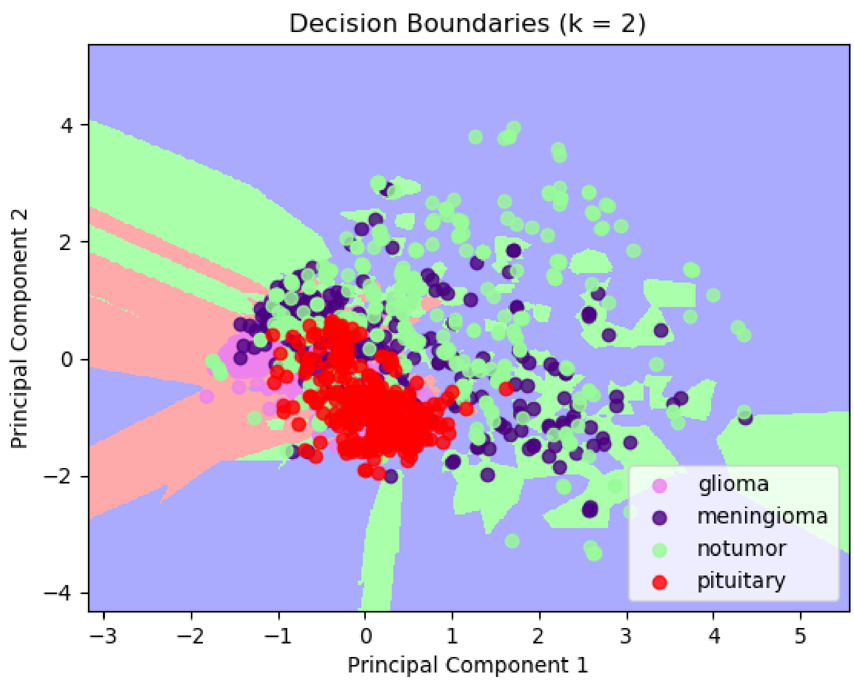


FIGURE 3 – Accuracy : 0.7208237986270023

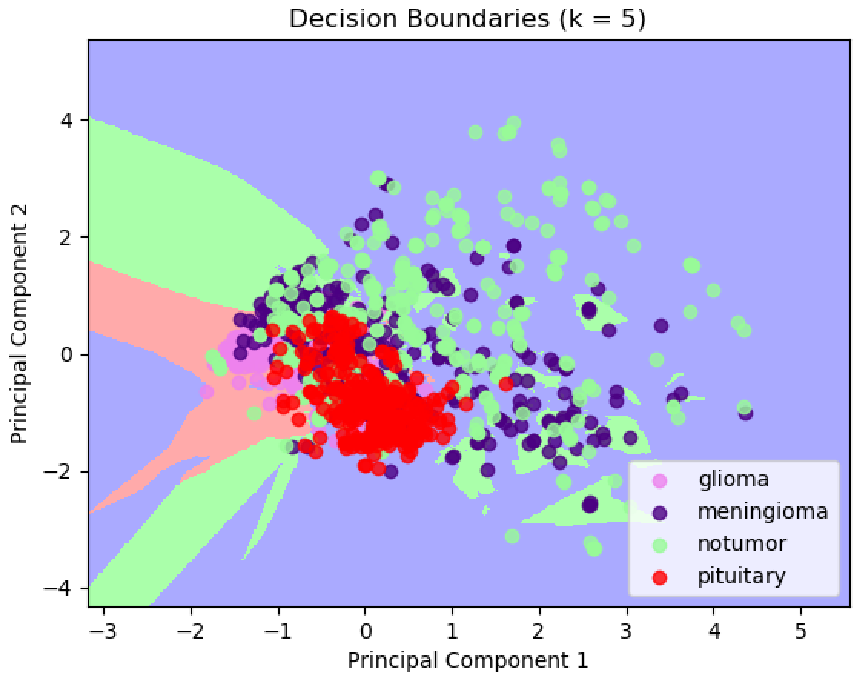


FIGURE 4 – Accuracy : 0.6071700991609459

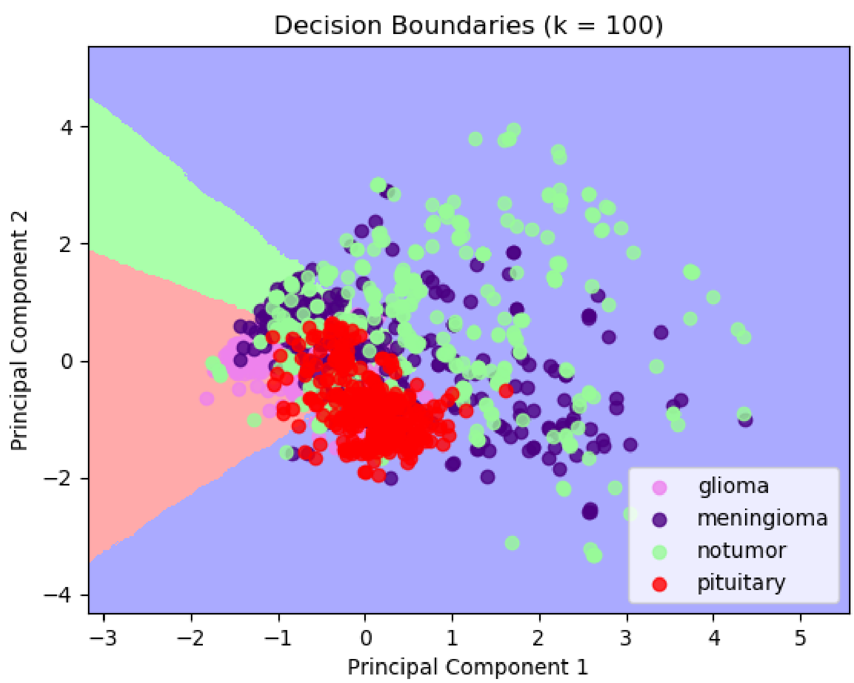


FIGURE 5 – Accuracy : 0.6071700991609459

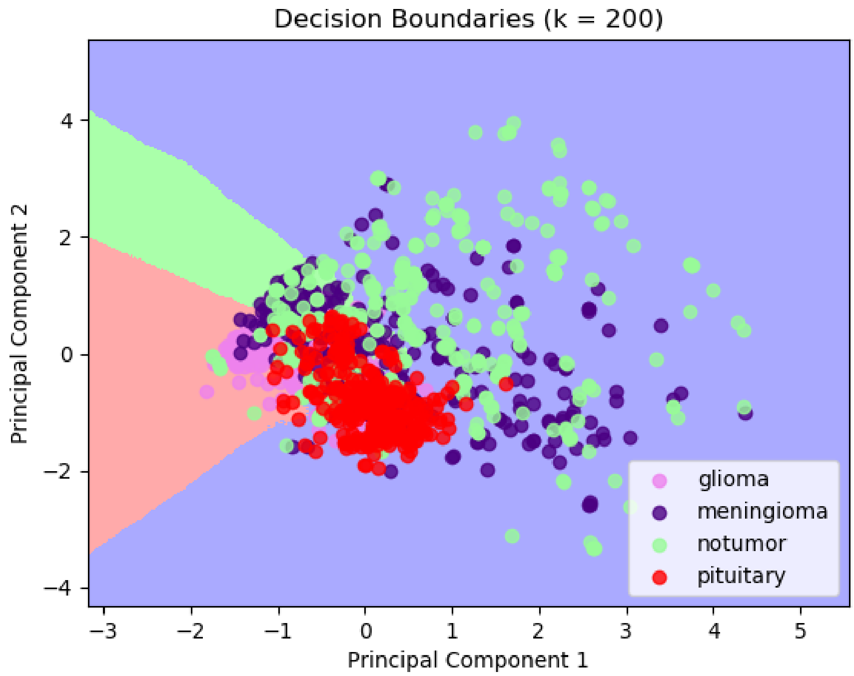


FIGURE 6 – Accuracy : 0.6048817696414951

Analyzing the decision boundary plots for different k values enables the identification of regions where the classifier may be uncertain and prone to classification errors. This aids in understanding the model's limitations and facilitates informed decision-making regarding the selection of the optimal k value to enhance classifier performance.

Results

Task 1 : Dataset Exploration Number of images : 5712 Image dimensions : (512, 512, 3) Label distribution : 'glioma' : 1321, 'meningioma' : 1339, 'notumor' : 1595, 'pituitary' : 1457

K = 2 :

- Accuracy : 0.72082379862700023
- Precision : 0.7905195335685307
- Recall : 0.71472158657513355
- F1-score : 0.7155498319658

Sure, here are the tables rewritten with the provided values :

TABLE 1 – Classification kNN Train Report

	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	1321
1	1.00	1.00	1.00	1339
2	1.00	1.00	1.00	1595
3	1.00	1.00	1.00	1457
Accuracy : 1.00, Macro avg : 1.00, Weighted avg : 1.00, Total support : 5712				

Table 2 – Classification kNN Test Report

	Precision	Recall	F1-score	Support
0	0.68	0.56	0.62	300
1	0.66	0.74	0.70	306
2	0.89	0.97	0.93	405
3	0.76	0.69	0.73	300
Accuracy : 0.76, Macro avg : 0.75, Weighted avg : 0.76, Total support : 1311				

Conclusion

In conclusion, the application of PCA and KNN for medical image analysis showcases a promising way to improve diagnostic accuracy and treatment planning. By effectively preprocessing the images, reducing dimensionality with PCA, and training a KNN classifier, we achieve notable results in classifying medical images into different conditions. However, further optimization and fine-tuning of parameters could enhance the classification performance even more. Exploring alternative algorithms and techniques, such as deep learning approaches, may provide valuable insights for future research in medical image analysis. Moreover, the visualizations of decision boundaries shed light on the model's behavior and highlight the importance of selecting an appropriate value for the hyperparameter k. Understanding the linkss between bias and variance in the KNN algorithm is crucial for achieving optimal performance and generalization to unseen data.
In summary, while PCA and KNN offer a solid foundation for medical image analysis, ongoing research and experimentation are essential for pushing the boundaries of diagnostic accuracy and contributing to advancements in healthcare technology.

For the full code implementation and detailed analysis, please refer to the attached source code and documentation.

Link for the code on Google Colab :
https://colab.research.google.com/drive/1tKHOB46Nh3cwUqRWU6YFNKZbfSw04ONe?usp=sharing

```
import os
import numpy as np
import cv2
import matplotlib
# matplotlib.style.use('ggplot')
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import torch
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsRegressor
import pandas

# Task 1: Dataset Exploration
def explore_dataset(dataset_folder):
    labels = os.listdir(dataset_folder)
    num_images = 0
    dimensions = None
    label_distribution = {}

    for label in labels:
        label_folder = os.path.join(dataset_folder, label)
        images = os.listdir(label_folder)
        num_images += len(images)
        label_distribution[label] = len(images)
        if dimensions is None:
            image_path = os.path.join(label_folder, images[0])
            image = cv2.imread(image_path)
            dimensions = image.shape

    print("Number of images:", num_images)
    print("Image dimensions:", dimensions)
    print("Label distribution:", label_distribution)


# Task 2: Data Preprocessing
def preprocess_images(dataset_folder, image_size=(100, 100)):
    images = []
    labels = []
    label_dict = {'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
    for label in os.listdir(dataset_folder):
        label_folder = os.path.join(dataset_folder, label)
        for image_file in os.listdir(label_folder):
            image_path = os.path.join(label_folder, image_file)
            image = cv2.imread(image_path)
            image = cv2.resize(image, image_size)
            image2 = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            # image = image.flatten()
            image = cv2.resize(image2, (100, 100))
            image = image / 255.0
            images.append(image)
            labels.append(label_dict[label])
    # images = np.array(images).view(images.shape[0], -1)
```

```python
    images = np.array(images).flatten().reshape(len(images), 1 * 100 * 100)
    return images, np.array(labels)


# Task 3: Principal Component Analysis (PCA)
def apply_pca(X_train, X_test, n_components):
    pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    return pca, X_train_pca, X_test_pca

def determine_components_variance_ratio(pca, threshold=0.95):
    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
    num_components = np.argmax(cumulative_variance_ratio >= threshold) + 1
    plt.plot(cumulative_variance_ratio)
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance Ratio')
    plt.title('Cumulative Explained Variance Ratio vs. Number of Components')
    plt.axhline(y=threshold, color='r', linestyle='--', label=f'Components for {threshold}
   threshold')
    plt.legend()
    plt.show()
    return num_components

def plot_gallery(images, h, w, n_row=3, n_col=4):
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.xticks(())
        plt.yticks(())


# Task 4: Training and Evaluation
def train_and_evaluate(X_train, X_test, y_train, y_test, n_neighbors):
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    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')
    # Afficher les métriques d'évaluation
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    return knn


# Task 5: Visualization
def visualize_decision_boundaries(X_train, y_train, X_test, y_test, n_neighbors, knn):
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
    label_dict = {'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
    inverse_label_dict = {v: k for k, v in label_dict.items()}
```

```python
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
    colors = {0: 'violet', 1: 'indigo', 2: 'palegreen', 3:'red'}
    ## plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=cmap_bold)
    # plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test, cmap=cmap_bold, marker='x')
    for label in np.unique(y_test):
        indices = np.where(y_test == label)
        plt.scatter(X_test[indices, 0], X_test[indices, 1], c=colors[label], alpha=0.8,
                    label='{}'.format(inverse_label_dict[label]))
    plt.legend(loc='lower right')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("Decision Boundaries (k = %i)" % n_neighbors)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

def original_and_reconstructed(images, images_init, pca, knn, labels):
    print(images.shape)
    h, w = 100, 100
    reconstructed_images = pca.inverse_transform(images)
    n = 10
    plt.figure(figsize=(20, 4))
    for i in range(n):
        ax = plt.subplot(2, n, i + 1)
        plt.imshow((images_init[i]).reshape((h, w)), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if i == 0:
            ax.set_title("Original")
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow((reconstructed_images[i]).reshape((h, w)), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if i == 0:
            ax.set_title("Reconstructed")
    plt.show()

def classification_report_function(X_train, y_train, X_test, y_test, n_neighbors):
    model = KNeighborsClassifier(algorithm="auto")
    parameters = {"n_neighbors": [n_neighbors],
                  "weights": ["distance"]}
    model_optim = GridSearchCV(model, parameters, cv=5, scoring="accuracy")
    model_optim.fit(X_train, y_train)
    model_optim.best_estimator_
    for (i,x,y) in zip(["Train","Test"],[X_train,X_test],[y_train,y_test]):
        print("Classification kNN",i," report:\n",classification_report(y,model_optim.predict
    (x)))


def plot_accuracy_vs_neighbors(X_train_pca, y_train, X_test_pca, y_test, max_neighbors):
    accuracies = []
    neighbors_range = range(1, max_neighbors + 1)
    for n_neighbors in neighbors_range:
        knn_classifier = train_and_evaluate(X_train, X_test, y_train, y_test, n_neighbors)
        y_pred = knn_classifier.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)
```

```
    plt.plot(neighbors_range, accuracies, marker='o')
    plt.title('Accuracy vs. Number of Neighbors')
    plt.xlabel('Number of Neighbors')
    plt.ylabel('Accuracy')
    plt.xticks(neighbors_range)
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    training_folder = r'D:\2A\Machine Learning\Medical Diagnosis\Training'
    testing_folder = r'D:\2A\Machine Learning\Medical Diagnosis\Testing'
    n_neighbors = 2

    # Task 1: Dataset Exploration
    print("Task 1: Dataset Exploration")
    explore_dataset(training_folder)

    # Task 2: Data Preprocessing
    print("\nTask 2: Data Preprocessing")
    X_train, y_train = preprocess_images(training_folder)
    X_test, y_test = preprocess_images(testing_folder)

    # Task 3: PCA
    print("\nTask 3: Principal Component Analysis (PCA)")
    n_components = 1119 # found with determine_components_variance_ratio
    pca, X_train_pca, X_test_pca = apply_pca(X_train, X_test, n_components)
    print("Dimension de X_train_pca:", X_train_pca.shape)
    print("Dimension de X_test_pca:", X_test_pca.shape)
    print("Proportion de variance expliquée par chaque composante principale:")
    explained_variance_ratio = pca.explained_variance_ratio_
    print(explained_variance_ratio)
    print("Dimension de X_train_pca:", X_train_pca.shape)
    nb_of_components = determine_components_variance_ratio(pca, threshold=0.95)
    print(nb_of_components)
    X_test_pca = pca.transform(X_test)

    # Task 4: Training and Evaluation
    print("\nTask 4: Training and Evaluation")
    print(X_test_pca.shape)
    knn_classifier = train_and_evaluate(X_train_pca, X_test_pca, y_train, y_test, n_neighbors
)

    # Task 5: Visualization
    print("\nTask 5: Visualization")
    original_and_reconstructed(X_test_pca, X_test, pca, knn_classifier, y_test)
    visualize_decision_boundaries(X_train_pca[:, :2], y_train, X_test_pca[:, :2], y_test,
n_neighbors, knn_classifier)

    # Find optimal K
    classification_report_function(X_train_pca[:, :2], y_train, X_test_pca[:, :2], y_test,
n_neighbors)

    # Plot_accuracy_vs_neighbors
    max_neighbors = 20
    plot_accuracy_vs_neighbors(X_train_pca, y_train, X_test_pca, y_test, max_neighbors)
```