naive-bayes-ananlysis

April 25, 2025

Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data set.

The Olivetti Face Dataset is a collection of images of faces, used primarily for face recognition tasks. The dataset contains 400 images of 40 different individuals, with 10 images per person. The dataset was created for research in machine learning and pattern recognition, especially in the context of facial recognition.

The Olivetti dataset provides the following key features:

*400 Images: Each image is a grayscale photo of a person's face.

40 People: The dataset contains 40 different individuals, and each individual Has 10 different images.

*Image Size: Each image is 64x64 pixels, resulting in 4096 features (flattened vector) per image.

*Target Labels: Each image is associated with a label representing the individual (0 to 39).

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
from sklearn.datasets import fetch_olivetti_faces
data = fetch_olivetti_faces()
```

[3]:
```
data.keys()
```

[3]:
```
dict_keys(['data', 'images', 'target', 'DESCR'])
```

[4]:
```
print("Data Shape:", data.data.shape)
print("Target Shape:", data.target.shape)
print("There are {} unique persons in the dataset".format(len(np.unique(data.
 ↪target))))
print("Size of each image is {}x{}".format(data.images.shape[1],data.images.
 ↪shape[1]))
```

```
Data Shape: (400, 4096)
Target Shape: (400,)
```

```
There are 40 unique persons in the dataset
Size of each image is 64x64
```

[5]:
```python
def print_faces(images, target, top_n):
    # Ensure the number of images does not exceed available data
    top_n = min(top_n, len(images))

    # Set up figure size based on the number of images
    grid_size = int(np.ceil(np.sqrt(top_n)))
    fig, axes = plt.subplots(grid_size, grid_size, figsize=(15, 15))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.2, wspace=0.
 ↪2)

    for i, ax in enumerate(axes.ravel()):
        if i < top_n:
            ax.imshow(images[i], cmap='bone')
            ax.axis('off')
            ax.text(2, 12, str(target[i]), fontsize=9, color='red')
            ax.text(2, 55, f"face: {i}", fontsize=9, color='blue')
        else:
            ax.axis('off')

    plt.show()
```

[6]:
```python
print_faces(data.images,data.target,400)
```

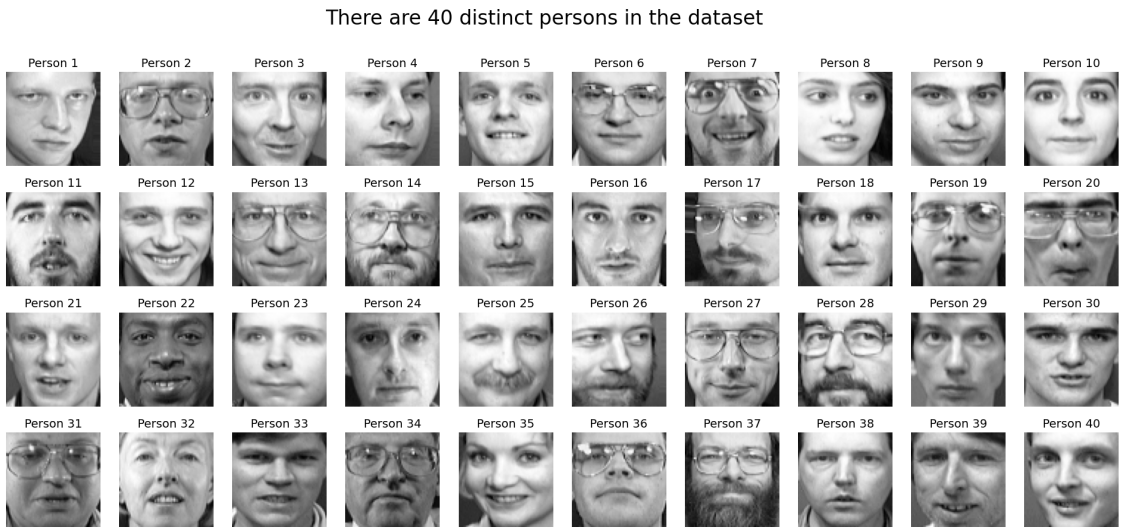
[7]:
```
#let us extract unique charaters present in dataset
def display_unique_faces(pics):
    fig = plt.figure(figsize=(24, 10))  # Set figure size
    columns, rows = 10, 4  # Define grid dimensions

    # Loop through grid positions and plot each image
    for i in range(1, columns * rows + 1):
        img_index = 10 * i - 1  # Calculate the image index
        if img_index < pics.shape[0]:  # Check for valid image index
            img = pics[img_index, :, :]
            ax = fig.add_subplot(rows, columns, i)
            ax.imshow(img, cmap='gray')
            ax.set_title(f"Person {i}", fontsize=14)
```

```
            ax.axis('off')

    plt.suptitle("There are 40 distinct persons in the dataset", fontsize=24)
    plt.show()
```

[8]:
```
display_unique_faces(data.images)
```



[11]:
```
from sklearn.model_selection import train_test_split
X = data.data
Y = data.target
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,␣
 ↪random_state=46)

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
```

```
x_train:  (280, 4096)
x_test:  (120, 4096)
```

[12]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

# Train the model
nb = GaussianNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)
```

```
# Calculate accuracy
nb_accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)

# Display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Display accuracy result
print(f"Naive Bayes Accuracy: {nb_accuracy}%")
```

```
Confusion Matrix:
[[3 0 0 … 0 0 0]
 [0 1 0 … 0 0 0]
 [0 0 1 … 0 0 0]
 …
 [0 0 0 … 2 0 0]
 [0 0 0 … 0 3 0]
 [1 0 0 … 0 0 1]]
Naive Bayes Accuracy: 73.33%
```

[13]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report

# Initialize and fit Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)

# Calculate accuracy
accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(f"Multinomial Naive Bayes Accuracy: {accuracy}%")
```

```
Multinomial Naive Bayes Accuracy: 85.83%
```

[14]:
```
# Calculate the number of misclassified images
misclassified_idx = np.where(y_pred != y_test)[0]
num_misclassified = len(misclassified_idx)

# Print the number of misclassified images and accuracy
print(f"Number of misclassified images: {num_misclassified}")
print(f"Total images in test set: {len(y_test)}")
print(f"Accuracy: {round((1 - num_misclassified / len(y_test)) * 100, 2)}%")
```

```python
# Show all misclassified images
n_misclassified_to_show = num_misclassified
plt.figure(figsize=(25, 5))  # Adjust width based on how many you want to show

for i in range(n_misclassified_to_show):
    idx = misclassified_idx[i]
    plt.subplot(1, n_misclassified_to_show, i + 1)
    plt.imshow(x_test[idx].reshape(64, 64), cmap='gray')
    plt.title(f"True: {y_test[idx]}\nPred: {y_pred[idx]}")
    plt.axis('off')

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

```
Number of misclassified images: 17
Total images in test set: 120
Accuracy: 85.83%
```

[ ]:

```python

```

[ ]:

```python

```