face_recognize

April 22, 2025

1

[143]:
```python
import os
import numpy as np
from PIL import Image, ImageOps, ImageEnhance, ImageFilter
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import (
    confusion_matrix, accuracy_score, roc_curve, auc, f1_score,
    precision_score, recall_score, cohen_kappa_score
)
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
```

1.1

[144]:
```python
#
train_dir = "split_data/train"
test_dir = "split_data/test"
test_out_dir = "split_data/test_out"
```

1.2

[145]:
```python
#
def preprocess_image(image_path, target_size=(64, 64), is_train=False):
    img = Image.open(image_path).convert("L")  #
    width, height = img.size

    if is_train:
        # 1.
        scale = np.random.uniform(0.8, 1.2)  #
        new_size = (int(target_size[0] * scale), int(target_size[1] * scale))
        img = img.resize(new_size)

        # 2.
        x = np.random.randint(0, new_size[0] - target_size[0] + 1)
        y = np.random.randint(0, new_size[1] - target_size[1] + 1)
        img = img.crop((x, y, x + target_size[0], y + target_size[1]))

        # 3.
        if np.random.random() > 0.5:
            img = img.rotate(np.random.randint(-15, 15))  #
        if np.random.random() > 0.5:
            img = img.transpose(Image.FLIP_LEFT_RIGHT)
        enhancer = ImageEnhance.Brightness(img)
        img = enhancer.enhance(np.random.uniform(0.7, 1.3))  #
        enhancer = ImageEnhance.Contrast(img)
        img = enhancer.enhance(np.random.uniform(0.8, 1.2))  #
        if np.random.random() > 0.3:  #
            img = img.filter(ImageFilter.GaussianBlur(radius=np.random.
↪uniform(0, 1)))

    else:
        #
        x = (width - target_size[0]) // 2
        y = (height - target_size[1]) // 2
        img = img.crop((x, y, x + target_size[0], y + target_size[1]))

    #
    img = ImageOps.equalize(img)
    img = np.array(img) / 255.0
    return img.flatten()  #
```

1.3

```
[146]: #
#
def load_dataset(data_dir, target_size=(64, 64), max_images_per_person=None):
    # print(f"   {data_dir}   ...")
    data = []
    labels = []
    filenames = []  #
    for subject_id in os.listdir(data_dir):
        subject_path = os.path.join(data_dir, subject_id)
        if os.path.isdir(subject_path):
            images = os.listdir(subject_path)
            if max_images_per_person:
                images = images[:max_images_per_person]
            for img_name in images:
                img_path = os.path.join(subject_path, img_name)
                img = preprocess_image(img_path, target_size)
                data.append(img)
                labels.append(subject_id)
                filenames.append(f"{subject_id}/{img_name}")  #
    print(f"{data_dir}      ")
    return np.array(data), np.array(labels), filenames
```

1.4 PCA

[147]:
```
# PCA
def pca_reduction(train_data, test_data, test_out_data, n_components):
    print(f"   PCA     {n_components}   ...")
    train_data_flat = train_data.reshape(train_data.shape[0], -1)
    test_data_flat = test_data.reshape(test_data.shape[0], -1)
    test_out_data_flat = test_out_data.reshape(test_out_data.shape[0], -1)

    mean_face = np.mean(train_data_flat, axis=0)

    train_data_centered = train_data_flat - mean_face
    test_data_centered = test_data_flat - mean_face
    test_out_data_centered = test_out_data_flat - mean_face

    #    n_components
    max_components = min(train_data_centered.shape[0], train_data_centered.
 ↪shape[1])
    if n_components > max_components:
        print(f" : n_components={n_components}      {max_components}")
        n_components = max_components

    pca = PCA(n_components=n_components)
    train_data_pca = pca.fit_transform(train_data_centered)
    test_data_pca = pca.transform(test_data_centered)
    test_out_data_pca = pca.transform(test_out_data_centered)
    print("PCA    ")
    return train_data_pca, test_data_pca, test_out_data_pca
```

1.5   SVM

[148]:
```
from sklearn.model_selection import StratifiedKFold

def train_svm_classifiers(train_data, train_labels):
    print("   SVM   ...")
    unique_labels = np.unique(train_labels)
    classifiers = {}
    # for label in unique_labels:
    #     binary_labels = (train_labels == label).astype(int)
    #     param_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01], 'kernel':␣
 ↪['rbf']}
    #     #
    #     cv = StratifiedKFold(n_splits=3)
    #     grid_search = GridSearchCV(SVC(probability=True), param_grid, cv=cv)
    #     grid_search.fit(train_data, binary_labels)
    #     best_svm = grid_search.best_estimator_
    #     classifiers[label] = best_svm
    for label in unique_labels:
        binary_labels = (train_labels == label).astype(int)
        #
        svm = SVC(probability=True, C=0.01, gamma=0.001, kernel='rbf')
        svm.fit(train_data, binary_labels)
        classifiers[label] = svm
    print("SVM     ")
    return classifiers
```

1.6

[149]:
```
#
def vote_predict(classifiers, data, threshold):
    # print("    ...")
    predictions = []
    all_scores = []
    for sample in data:
        scores = []
        for label, classifier in classifiers.items():
            score = classifier.predict_proba([sample])[0][1]
            scores.append(score)
        max_score = np.max(scores)
        all_scores.append(max_score)
        if max_score < threshold:
            predictions.append("OUT")
        else:
            predictions.append("IN")
    # print("    ")
    return np.array(predictions), np.array(all_scores)
```

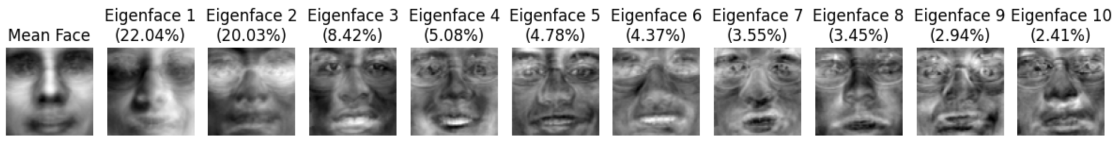
1.7

[150]:
```
#
def predict_person(classifiers, data):
    predictions = []
    for sample in data:
        scores = []
        labels = []
        for label, classifier in classifiers.items():
            score = classifier.predict_proba([sample])[0][1]
            scores.append(score)
            labels.append(label)
        predicted_label = labels[np.argmax(scores)]
        predictions.append(predicted_label)
    return np.array(predictions)
```

1.8

[151]:
```
#     10
def plot_mean_and_top_eigenfaces(train_data, n_components=50, top_n=10):
    print("    10   ...")
    train_data_flat = train_data.reshape(train_data.shape[0], -1)
    mean_face = np.mean(train_data_flat, axis=0)

    # PCA
    pca = PCA(n_components=n_components)
    pca.fit(train_data_flat - mean_face)
    eigenfaces = pca.components_.reshape((n_components, 64, 64))
    explained_variance_ratio = pca.explained_variance_ratio_

    #
    plt.figure(figsize=(12, 6))
    plt.subplot(1, top_n + 1, 1)
    plt.imshow(mean_face.reshape(64, 64), cmap='gray')
    plt.title("Mean Face")
    plt.axis("off")

    #   10
    for i in range(top_n):
        plt.subplot(1, top_n + 1, i + 2)
        plt.imshow(eigenfaces[i], cmap='gray')
        plt.title(f"Eigenface {i + 1}\n({explained_variance_ratio[i]:.2%})")
        plt.axis("off")
    plt.tight_layout()
    plt.show()
    print("   ")

#             Excel   =50
import pandas as pd


def save_pca_to_excel_with_filenames(train_data, filenames, n_components=50,␣
 ↪output_file="pca_results.xlsx"):
    print("  PCA     Excel   ...")
    train_data_flat = train_data.reshape(train_data.shape[0], -1)
    mean_face = np.mean(train_data_flat, axis=0)

    # PCA
    pca = PCA(n_components=n_components)
    train_data_pca = pca.fit_transform(train_data_flat - mean_face)
    explained_variance_ratio = pca.explained_variance_ratio_
    #
    df_pca = pd.DataFrame(train_data_pca, columns=[f"PC{i + 1}" for i in␣
 ↪range(n_components)])
    df_pca.insert(0, "Filename", filenames)  #
    df_pca.to_excel(output_file, index=False, sheet_name="PCA Results")

    #
    df_variance = pd.DataFrame({
        "Principal Component": [f"PC{i + 1}" for i in range(n_components)],
        "Explained Variance (%)": explained_variance_ratio * 100
    })
    with pd.ExcelWriter(output_file, mode="a", engine="openpyxl") as writer:
        df_variance.to_excel(writer, index=False, sheet_name="Explained␣
 ↪Variance")

    print(f"PCA     {output_file} ")
#
n = 6  #
train_data, train_labels, filenames = load_dataset(train_dir,␣
 ↪max_images_per_person=n)

#    10
plot_mean_and_top_eigenfaces(train_data, n_components=40, top_n=10)

#   PCA    Excel
save_pca_to_excel_with_filenames(train_data, filenames, n_components=40,␣
 ↪output_file="pca_results.xlsx")
```

```
split_data/train
    10  …
```



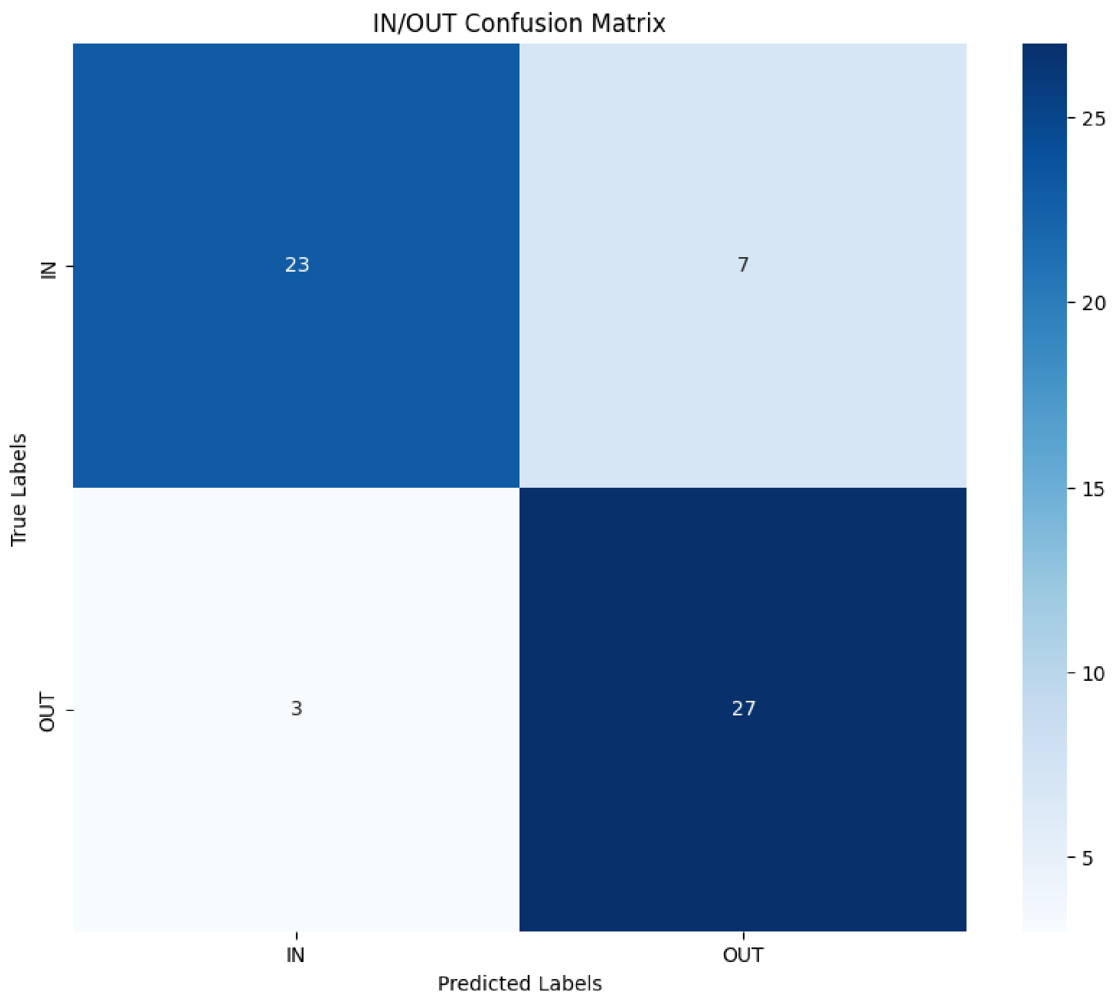
```
 PCA     Excel  …
PCA     pca_results.xlsx
```

1.9

```
[155]: #
n = 6  #
train_data, train_labels, filenames = load_dataset(train_dir,␣
 ↪max_images_per_person=n)
test_data, test_labels, filenames = load_dataset(test_dir,␣
 ↪max_images_per_person=n)
test_out_data, test_out_labels, filenames = load_dataset(test_out_dir,␣
 ↪max_images_per_person=n)

print("    ")
print(f"   : {train_data.shape},    : {test_data.shape},      : {test_out_data.
 ↪shape}")

# PCA
n_components = 50
train_data_pca, test_data_pca, test_out_data_pca = pca_reduction(train_data,␣
 ↪test_data, test_out_data, n_components)

#
classifiers = train_svm_classifiers(train_data_pca, train_labels)

#
# print("    ...")
all_predictions, all_scores = vote_predict(classifiers, np.
 ↪concatenate([test_data_pca, test_out_data_pca]), 0.5)
true_labels = np.concatenate([np.ones(len(test_labels)), np.
 ↪zeros(len(test_out_labels))])
fpr, tpr, thresholds = roc_curve(true_labels, all_scores)
roc_auc = auc(fpr, tpr)
optimal_threshold = thresholds[np.argmax(tpr - fpr)]
print(f"  : {optimal_threshold:.3f}")

#
test_predictions, _ = vote_predict(classifiers, test_data_pca,␣
 ↪optimal_threshold)
test_out_predictions, _ = vote_predict(classifiers, test_out_data_pca,␣
 ↪optimal_threshold)
all_predictions = np.concatenate([test_predictions, test_out_predictions])
all_true_labels = np.concatenate([["IN"] * len(test_labels), ["OUT"] *␣
 ↪len(test_out_labels)])

# ======================      ======================
#
cm = confusion_matrix(all_true_labels, all_predictions, labels=["IN", "OUT"])
tp, fp, fn, tn = cm[0,0], cm[1,0], cm[0,1], cm[1,1]
```

```
#
in_out_accuracy = accuracy_score(all_true_labels, all_predictions)
in_out_precision = precision_score(all_true_labels, all_predictions,␣
↪pos_label="IN")
in_out_recall = recall_score(all_true_labels, all_predictions, pos_label="IN")
in_out_f1 = f1_score(all_true_labels, all_predictions, pos_label="IN")
in_out_specificity = tn / (tn + fp) if (tn + fp) != 0 else 0.0
in_out_kappa = cohen_kappa_score(all_true_labels, all_predictions)

#
print("\n====       ====")
print(f"  : {in_out_accuracy:.2f}")
print(f"  : {in_out_precision:.2f}")
print(f"  : {in_out_recall:.2f}")
print(f"  : {in_out_specificity:.2f}")
print(f"F1 : {in_out_f1:.2f}")
print(f"Kappa : {in_out_kappa:.2f}")
print(f"AUC : {roc_auc:.2f}")

#
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["IN", "OUT"], yticklabels=["IN", "OUT"])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('IN/OUT Confusion Matrix')
plt.show()
```

```
split_data/train
split_data/test
split_data/test_out

   : (60, 4096),    : (30, 4096),      : (30, 4096)
   PCA     50    …
PCA
   SVM   …
SVM
  : 0.737

====       ====
  : 0.83
  : 0.88
  : 0.77
  : 0.90
F1 : 0.82
Kappa : 0.67
AUC : 0.86
```

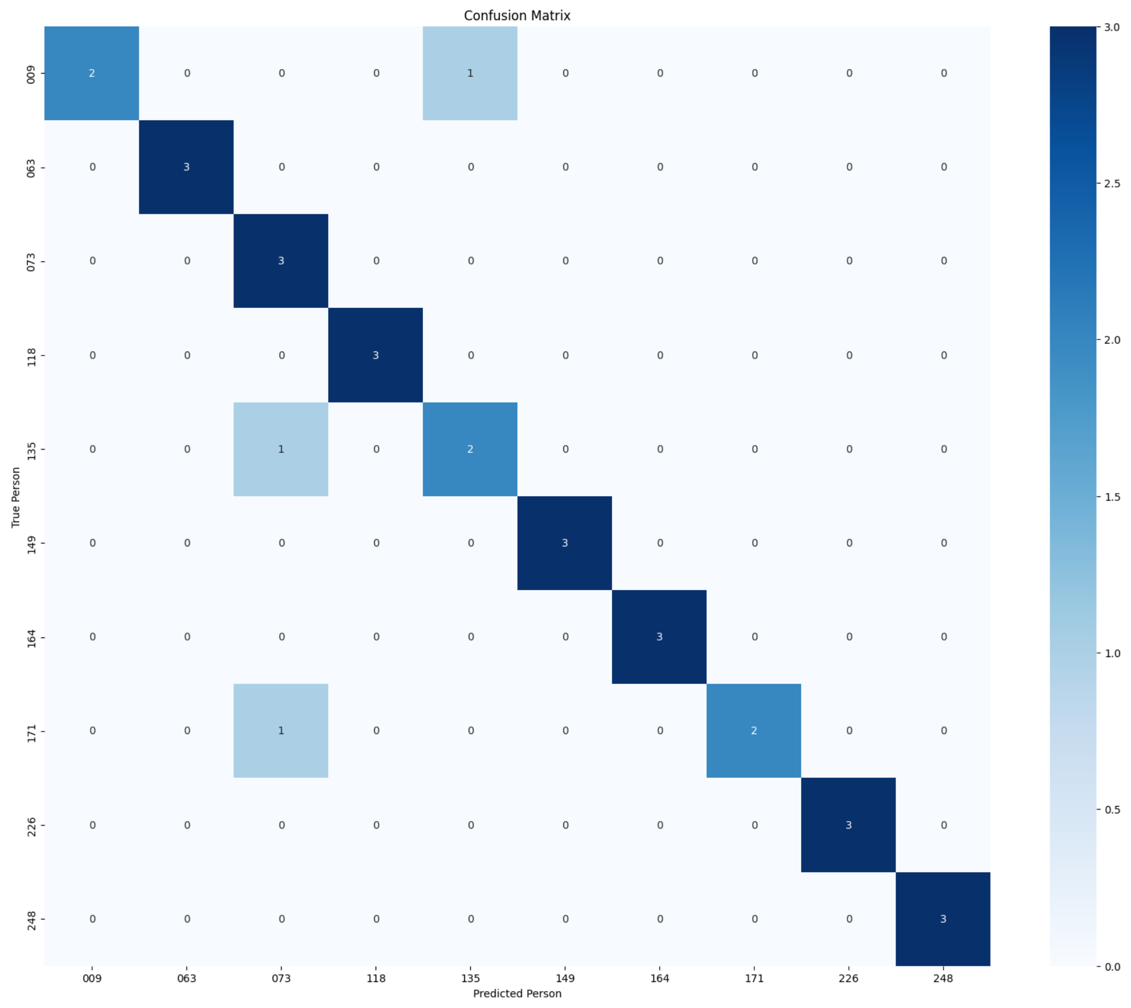


1.10

```
[156]: #
person_predictions = predict_person(classifiers, test_data_pca)

#
person_cm = confusion_matrix(test_labels, person_predictions)
person_labels = sorted(np.unique(train_labels))

#
person_accuracy = accuracy_score(test_labels, person_predictions)
person_f1_weighted = f1_score(test_labels, person_predictions,␣
  ↪average='weighted')
person_f1_macro = f1_score(test_labels, person_predictions, average='macro')
person_precision_macro = precision_score(test_labels, person_predictions,␣
  ↪average='macro')
```

```
person_recall_macro = recall_score(test_labels, person_predictions,␣
    ↪average='macro')
person_kappa = cohen_kappa_score(test_labels, person_predictions)

#
print("\n====      ====")
print(f"  : {person_accuracy:.2f}")
print(f" F1  : {person_f1_weighted:.2f}")
print(f"  F1  : {person_f1_macro:.2f}")
print(f"    : {person_precision_macro:.2f}")
print(f"    : {person_recall_macro:.2f}")
print(f"Kappa  : {person_kappa:.2f}")

#
plt.figure(figsize=(20, 16))
sns.heatmap(person_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=person_labels, yticklabels=person_labels)
plt.xlabel('Predicted Person')
plt.ylabel('True Person')
plt.title('Confusion Matrix')
plt.show()
```

```
====      ====
  : 0.90
 F1  : 0.90
  F1  : 0.90
    : 0.93
    : 0.90
Kappa  : 0.89
```



1.11

```
[154]: #       n
def evaluate_different_n(train_dir, test_data, test_labels, test_out_data,␣
 ↪test_out_labels, n_components, n_values, num_repeats=10):
    print("       n         ...")
    results = []
    for n in n_values:
        # print(f"\n   n = {n}")
        accuracies = []
        for repeat in range(num_repeats):
            # print(f"    {repeat + 1}/{num_repeats}   ...")
            #
            train_data, train_labels ,_= load_dataset(train_dir,␣
 ↪max_images_per_person=n)
```

```
            train_data_pca, test_data_pca, test_out_data_pca =␣
↪pca_reduction(train_data, test_data, test_out_data, n_components)

            #
            classifiers = train_svm_classifiers(train_data_pca, train_labels)

            #
            all_predictions, all_scores = vote_predict(classifiers, np.
↪concatenate([test_data_pca, test_out_data_pca]), 0.5)
            true_labels = np.concatenate([np.ones(len(test_labels)), np.
↪zeros(len(test_out_labels))])
            fpr, tpr, thresholds = roc_curve(true_labels, all_scores)
            optimal_threshold = thresholds[np.argmax(tpr - fpr)]

            #
            test_predictions, _ = vote_predict(classifiers, test_data_pca,␣
↪optimal_threshold)
            test_out_predictions, _ = vote_predict(classifiers,␣
↪test_out_data_pca, optimal_threshold)
            all_predictions = np.concatenate([test_predictions,␣
↪test_out_predictions])
            all_true_labels = np.concatenate([["IN"] * len(test_labels),␣
↪["OUT"] * len(test_out_labels)])

            #
            in_out_accuracy = accuracy_score(all_true_labels, all_predictions)
            accuracies.append(in_out_accuracy)

        #
        mean_accuracy = np.mean(accuracies)
        std_accuracy = np.std(accuracies)
        print(f"n = {n},    : {mean_accuracy:.2f},   : {std_accuracy:.2f}")
        results.append((n, mean_accuracy, std_accuracy))

    #
    n_values, mean_accuracies, std_accuracies = zip(*results)
    plt.figure(figsize=(8, 6))
    plt.errorbar(n_values, mean_accuracies, yerr=std_accuracies, fmt='o-',␣
↪color='b', ecolor='r', capsize=5)
    plt.xlabel("Number of Images per Person (n)")
    plt.ylabel("Accuracy")
    plt.title("Accuracy vs Number of Images per Person")
    plt.grid()
    plt.show()
    print("    n    ")
```

```
#       n_components
def evaluate_different_components(train_data, train_labels, test_data,␣
 ↪test_labels, test_out_data, test_out_labels, n_components_values,␣
 ↪num_repeats=10):
    print("       n_components        ...")
    results = []
    for n_components in n_components_values:
        # print(f"\n   n_components = {n_components}")
        accuracies = []
        for repeat in range(num_repeats):
            # print(f"    {repeat + 1}/{num_repeats}   ...")
            train_data_pca, test_data_pca, test_out_data_pca =␣
 ↪pca_reduction(train_data, test_data, test_out_data, n_components)

            #
            classifiers = train_svm_classifiers(train_data_pca, train_labels)

            #
            all_predictions, all_scores = vote_predict(classifiers, np.
 ↪concatenate([test_data_pca, test_out_data_pca]), 0.5)
            true_labels = np.concatenate([np.ones(len(test_labels)), np.
 ↪zeros(len(test_out_labels))])
            fpr, tpr, thresholds = roc_curve(true_labels, all_scores)
            optimal_threshold = thresholds[np.argmax(tpr - fpr)]

            #
            test_predictions, _ = vote_predict(classifiers, test_data_pca,␣
 ↪optimal_threshold)
            test_out_predictions, _ = vote_predict(classifiers,␣
 ↪test_out_data_pca, optimal_threshold)
            all_predictions = np.concatenate([test_predictions,␣
 ↪test_out_predictions])
            all_true_labels = np.concatenate([["IN"] * len(test_labels),␣
 ↪["OUT"] * len(test_out_labels)])

            #
            in_out_accuracy = accuracy_score(all_true_labels, all_predictions)
            accuracies.append(in_out_accuracy)

        #
        mean_accuracy = np.mean(accuracies)
        std_accuracy = np.std(accuracies)
        print(f"n_components = {n_components},    : {mean_accuracy:.2f},   :␣
 ↪{std_accuracy:.2f}")
        results.append((n_components, mean_accuracy, std_accuracy))
    #
    n_components_values, mean_accuracies, std_accuracies = zip(*results)
    plt.figure(figsize=(8, 6))
    plt.errorbar(n_components_values, mean_accuracies, yerr=std_accuracies,␣
  ↪fmt='o-', color='g', ecolor='r', capsize=5)
    plt.xlabel("Number of Principal Components (n_components)")
    plt.ylabel("Accuracy")
    plt.title("Accuracy vs Number of Principal Components")
    plt.grid()
    plt.show()
    print("    n_components    ")

#       n   n_components
if __name__ == "__main__":
    #
    test_data, test_labels, _ = load_dataset(test_dir, max_images_per_person=n)
    test_out_data, test_out_labels , _ = load_dataset(test_out_dir,␣
  ↪max_images_per_person=n)

    #       n
    n_values = [1,2,3,4,5,6]   #
    evaluate_different_n(train_dir, test_data, test_labels, test_out_data,␣
  ↪test_out_labels, n_components=50, n_values=n_values)

    #       n = 6
    train_data, train_labels, _ = load_dataset(train_dir,␣
  ↪max_images_per_person=6)

    #       n_components
    n_components_values = [10, 20, 30, 40, 50]
    evaluate_different_components(train_data, train_labels, test_data,␣
  ↪test_labels, test_out_data, test_out_labels, n_components_values)
```

```
split_data/test
split_data/test_out
      n        …
split_data/train
   PCA     50    …
 : n_components=50      10
PCA
   SVM   …
SVM
split_data/train
   PCA     50    …
 : n_components=50      10
PCA
```

```
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       10
PCA
```

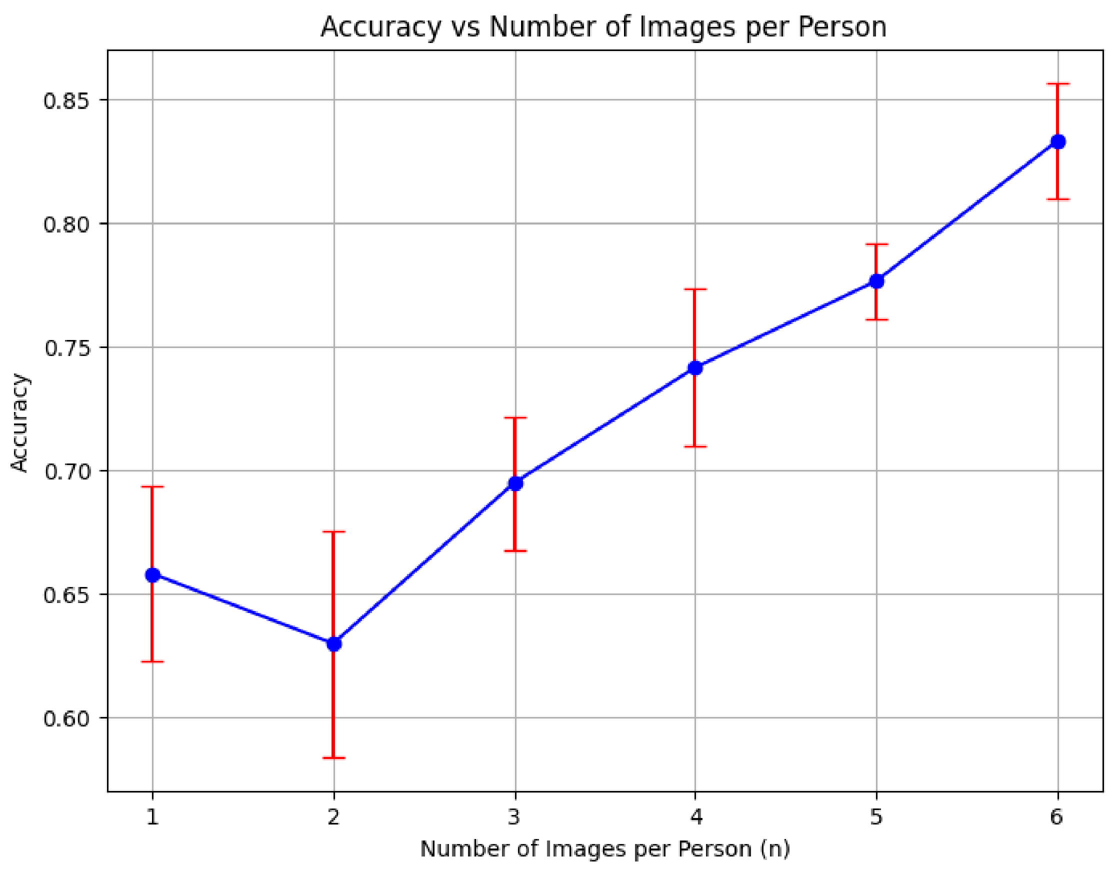
```
   SVM   …
SVM
n = 1,    : 0.66,   : 0.04
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
```

```
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       20
PCA
   SVM   …
SVM
n = 2,    : 0.63,   : 0.05
split_data/train
   PCA     50   …
 : n_components=50       30
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       30
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       30
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       30
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       30
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50      30
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      30
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      30
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      30
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      30
PCA
   SVM  …
SVM
n = 3,    : 0.70,   : 0.03
split_data/train
   PCA    50   …
 : n_components=50      40
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      40
PCA
   SVM  …
SVM
split_data/train
   PCA    50   …
 : n_components=50      40
PCA
   SVM  …
SVM
split_data/train
```

```
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
 : n_components=50       40
PCA
   SVM   …
SVM
n = 4,    : 0.74,   : 0.03
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
```

```
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
n = 5,    : 0.78,   : 0.02
split_data/train
   PCA     50   …
PCA
```

```
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
split_data/train
   PCA     50   …
PCA
   SVM   …
SVM
n = 6,    : 0.83,   : 0.02
```



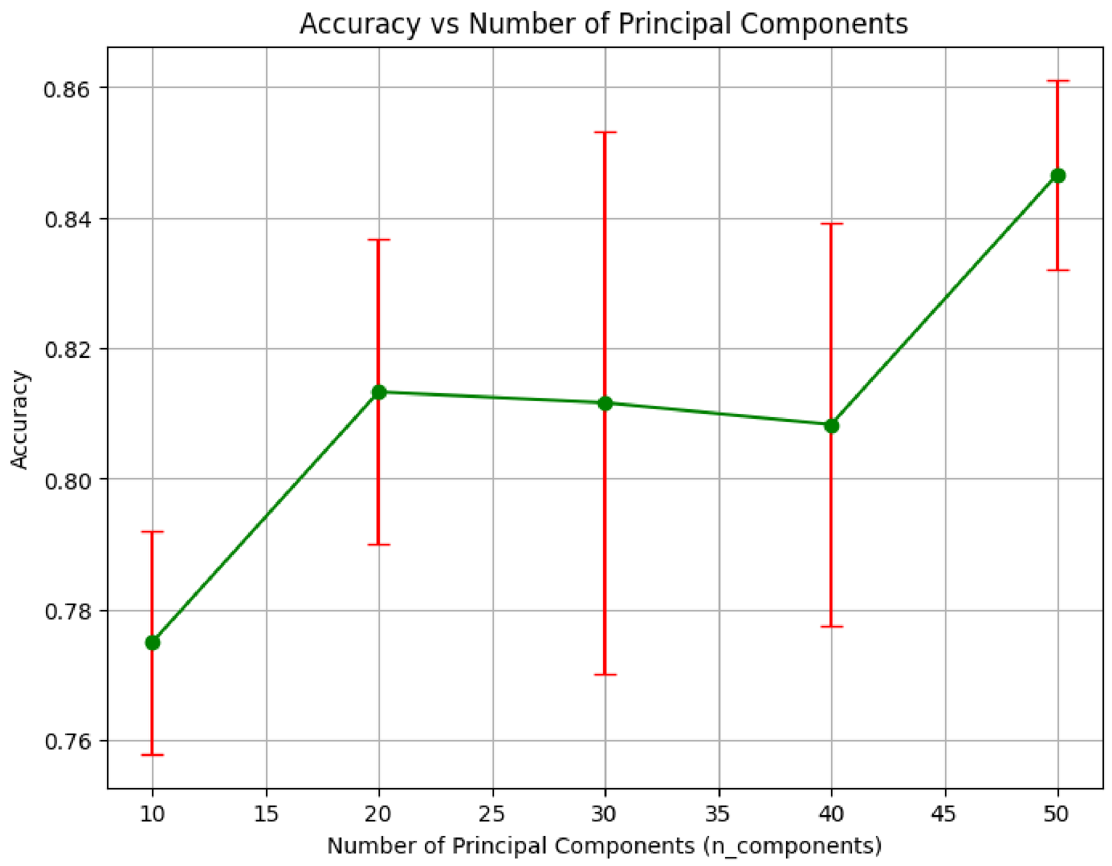
n
split_data/train
n_components …
PCA 10 …
PCA
SVM …
SVM
PCA 10 …
PCA
SVM …
SVM
PCA 10 …
PCA
SVM …
SVM
PCA 10 …
PCA
SVM …
SVM
PCA 10 …

```
PCA
   SVM  …
SVM
   PCA    10   …
PCA
   SVM  …
SVM
   PCA    10   …
PCA
   SVM  …
SVM
   PCA    10   …
PCA
   SVM  …
SVM
   PCA    10   …
PCA
   SVM  …
SVM
   PCA    10   …
PCA
   SVM  …
SVM
n_components = 10,    : 0.78,   : 0.02
   PCA    20   …
PCA
   SVM  …
SVM
   PCA    20   …
PCA
   SVM  …
SVM
   PCA    20   …
PCA
   SVM  …
SVM
   PCA    20   …
PCA
   SVM  …
SVM
   PCA    20   …
PCA
   SVM  …
SVM
   PCA    20   …
PCA
   SVM  …
SVM
```

```
   PCA     20    …
PCA
   SVM   …
SVM
   PCA     20    …
PCA
   SVM   …
SVM
   PCA     20    …
PCA
   SVM   …
SVM
   PCA     20    …
PCA
   SVM   …
SVM
n_components = 20,    : 0.81,   : 0.02
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
```

```
SVM
   PCA     30    …
PCA
   SVM   …
SVM
   PCA     30    …
PCA
   SVM   …
SVM
n_components = 30,    : 0.81,   : 0.04
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
   SVM   …
SVM
   PCA     40    …
PCA
```

```
   SVM   …
SVM
n_components = 40,    : 0.81,   : 0.03
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
   PCA     50    …
PCA
   SVM   …
SVM
n_components = 50,    : 0.85,   : 0.01
```



`n_components`