Traditional ML Face Recognition System

Prepared by:

DIYA' ASHRAF JAMAL AL. QAISSE

Table of Contents

Machine Learning Project Documentation: Face Recognition

Project Overview

This project focuses on developing a facial recognition system capable of classifying celebrity images as either "approved" or "denied." The classification is based on metadata provided in an external Excel file (dataset.xlsx), which assigns a binary status to each celebrity. A key objective of this project is to demonstrate that while machine learning techniques can be applied to such tasks, their performance can degrade significantly when faced with a large and complex dataset, as evidenced by the evaluation metrics. This highlights the challenges and limitations of traditional machine learning approaches in scenarios that might typically call for more advanced techniques.

Pins Face Recognition Dataset Description

1.Introduction

The Pins Face Recognition dataset is a specialized collection of facial images designed for tasks related to face recognition and machine learning applications. It was compiled by Burak and is hosted on Kaggle [1]. This dataset is particularly useful for training and evaluating models that aim to identify individuals based on their facial features.

2.Content Overview

The dataset comprises a substantial collection of 17,534 cropped facial images [1]. These images were sourced from Pinterest, indicating a diverse range of poses, lighting conditions, and expressions that are typical of real-world scenarios. A key characteristic of this dataset is its organization around 105 distinct celebrities, effectively serving as classes for classification or identification tasks [1].

3.Data Structure

The dataset is structured hierarchically, with a main directory named 105_classes_pins_dataset. Within this directory, there are 105 subdirectories, each corresponding to a unique celebrity. Each subdirectory contains the cropped facial images associated with that specific celebrity. For instance, pins_Adriana Lima contains 213 files, pins_Alex Lawther contains 152 files, and so on [1]. This clear organization facilitates easy loading and processing for machine learning workflows.

4.Usage and Applications

Given its structure and content, the Pins Face Recognition dataset is highly recommended for face recognition projects [1]. It can be utilized for various machine learning tasks, including:

- Facial Classification: Training models to classify an input face as either "approved" or "denied" based on the dataset.xlsx metadata.
- Face Verification: Developing systems to determine if two faces belong to the same person.
- Feature Learning: Extracting robust facial features for downstream tasks.

5.Metadata and Licensing

- License: The dataset is released under the CC0: Public Domain license, allowing for free use, modification, and distribution [1].
- Expected Update Frequency: The dataset is expected to be updated annually [1].
- Tags: The dataset is categorized under Classification, Computer Science, Online Communities, and Deep Learning [1].

6.Summary Table

Feature	**Description**
Dataset Name	Pins Face Recognition
Source	Pinterest (cropped images)
Total Images	17,534
Number of Classes	105 (celebrities)
License	CC0: Public Domain
Update Frequency	Annually

Feature	**Description**
Primary Application	Face Recognition, Facial Classification

Preprocessing and Training Pipeline for Face Recognition

This section details the preprocessing steps applied to the Pins Face Recognition dataset and the subsequent training of a Support Vector Machine (SVM) model for facial classification. The pipeline leverages Haar Cascade classifiers for initial face and eye detection, Discrete Wavelet Transform (DWT) for feature extraction, and a GridSearchCV-tuned SVM for classification.

1. Project Initialization and Library Imports

The project begins by importing essential Python libraries for various tasks, including file system operations, data manipulation, computer vision, and machine learning. Key libraries include:

- **os**, **shutil**, **pathlib.Path**: For interacting with the operating system, managing files and directories.
- **re**, **pandas**: For regular expressions and data handling, particularly for processing metadata from the dataset.xlsx file.
- **cv2** (OpenCV), **numpy**: For image loading, manipulation, and numerical operations.
- **pywt** (PyWavelets): For performing Discrete Wavelet Transforms.
- **sklearn.model_selection**: For splitting data (train_test_split) and hyperparameter tuning (GridSearchCV, StratifiedKFold).
- **sklearn.preprocessing.StandardScaler**: For feature scaling.
- **sklearn.svm.SVC**: For the Support Vector Classifier model.
- **sklearn.pipeline.Pipeline**: To chain preprocessing and modeling steps.
- **sklearn.metrics**: For evaluating model performance (classification_report, confusion_matrix).

2. Face and Eye Detection using Haar Cascade Classifiers

Before feature extraction, images undergo a crucial preprocessing step to ensure that only valid facial regions containing eyes are considered. This is achieved using pre-trained Haar Cascade classifiers:

- **haarcascade_frontalface_default.xml**: Detects frontal faces.

- **haarcascade_eye.xml**: Detects eyes within detected facial regions.

These XML files are loaded using cv2.CascadeClassifier. The system includes validation checks to ensure these classifiers are loaded successfully, raising a FileNotFoundError if the paths are incorrect or the files are empty.

3. Cropping Images with get_cropped_image_if_2_eyes

The get_cropped_image_if_2_eyes function is central to the image preprocessing. It performs the following steps for each input image:

1. Image Loading: Reads the image using cv2.imread.
2. Grayscale Conversion: Converts the image to grayscale, which is often preferred for Haar Cascade detection due to computational efficiency and robustness.
3. Face Detection: Applies the face_cascade.detectMultiScale method to identify potential face regions.
4. Region of Interest (ROI) Definition: For each detected face, it defines a grayscale ROI for eye detection and a color ROI for potential cropping.
5. Eye Detection: Within each face ROI, eye_cascade.detectMultiScale is used to find eyes.
6. Validation and Cropping: A face is considered valid only if it contains a minimum of two eyes. If this condition is met, the color ROI of the face is returned as a cropped image; otherwise, None is returned.

This function ensures that only high-quality face crops, suitable for recognition tasks, proceed to the next stages.

4. Data Organization and Cropped Image Generation

The notebook sets up a structured directory for storing processed images:

7. Path Configuration: Defines path_to_data (raw images) and path_to_cropped_data (output for cropped images).
8. Directory Management: If path_to_cropped_data exists, it is removed to ensure a clean start, and then recreated.
9. Celebrity Folder Discovery: It identifies all subfolders within path_to_data, each representing a celebrity class.
10. Image Processing Loop: For each celebrity directory:
    - A corresponding subdirectory is created in path_to_cropped_data.
    - Each image in the celebrity's raw folder is passed to get_cropped_image_if_2_eyes.

- If a valid cropped face is returned, it is saved to the respective cropped_data subdirectory with a sequential filename (e.g., pins_Adriana_Lima_0001.jpg).
- A dictionary celebrity_file_names is populated, mapping cleaned celebrity names to a list of paths of their successfully cropped images.

5. Metadata Integration and Label Mapping

The dataset.xlsx file, provided by the user, is integrated into the pipeline to assign labels and filter celebrities:

11 Excel Loading: The dataset.xlsx file is loaded into a pandas DataFrame.
12 Name Normalization: A norm_name function is used to standardize celebrity names (lowercase, remove underscores, collapse spaces) for consistent matching between folder names and Excel entries.
13 Status to Label Mapping: The 'Status' column from the Excel file (e.g., 'approved', 'denied') is normalized and mapped to numerical labels (1 for 'approved', 0 for 'denied').
14 Label Validation: Checks are performed to ensure all 'Status' values in the Excel file are valid and mapped correctly.
15 Alignment and Filtering: The celebrity_file_names dictionary is filtered to include only those celebrities present in the dataset.xlsx file with a defined label. This ensures that only relevant data, as specified by the metadata, is used for training.

6. Feature Extraction with Wavelet Transform (w2d function)

Feature vectors (X) and corresponding labels (y) are constructed using a combination of raw pixel data and Discrete Wavelet Transform (DWT) features:

16 **w2d** Function: This function performs a 2D Wavelet Transform on an image:
   - Converts the image to grayscale and normalizes pixel values.
   - Applies pywt.wavedec2 (Daubechies 'db1' wavelet at level 5) to decompose the image into wavelet coefficients.
   - Crucially, it zeros out the 'Approximation' (low-frequency) component, retaining only the high-frequency detail coefficients (edges, textures).
   - Reconstructs the image from these detail coefficients, highlighting textural information.

17 Feature Vector Assembly: For each cropped image:
   - Raw Feature: The image is resized to 32x32 pixels, maintaining 3 color channels (RGB), and flattened into a vector.

  - Wavelet Feature: The w2d function is applied, and the resulting wavelet-transformed image is resized to 32x32 pixels (1 channel) and flattened.
  - Feature Stacking: The raw pixel feature vector and the wavelet feature vector are vertically stacked and then flattened into a single 1D feature vector. This combined vector captures both color information and textural details.
  - These feature vectors form the X matrix, and their corresponding labels (0 or 1 from the Excel file) form the y vector.

7. Data Splitting and Feature Scaling

To prepare the data for model training and evaluation, the following steps are performed:

18 Train-Test Split: The dataset (X, y) is divided into training and testing sets using train_test_split from sklearn.model_selection. An 80/20 split is used, with stratify=y to ensure that the proportion of 'approved' and 'denied' labels is maintained in both the training and testing sets. shuffle=True and random_state=6 ensure reproducibility.

19 Feature Scaling: StandardScaler from sklearn.preprocessing is applied to normalize the features:
  - The scaler is fit_transformed on the training data (X_train) to calculate the mean and standard deviation for each feature and then scale the training data.
  - The test data (X_test) is then transformed using the parameters learned from the training data. This prevents data leakage from the test set into the training process, ensuring an unbiased evaluation of the model's performance on unseen data.

8. Model Selection, Hyperparameter Tuning, and Evaluation

A Support Vector Machine (SVM) is chosen as the classification model, and its hyperparameters are optimized using GridSearchCV.

20 Pipeline Construction: A Pipeline is created to encapsulate the StandardScaler and the SVC model. This prevents data leakage during cross-validation by ensuring that scaling is applied independently within each fold.
  - StandardScaler(): Normalizes features.
  - SVC(kernel='rbf', class_weight='balanced'): A Support Vector Classifier with a Radial Basis Function (RBF) kernel. class_weight='balanced' is used to handle potential class imbalance between 'approved' and 'denied' labels.

21 Hyperparameter Tuning (**GridSearchCV**): GridSearchCV is employed to systematically search for the best combination of hyperparameters for the SVM model:

   - param_grid: Defines the range of C (regularization parameter) and gamma (kernel coefficient) values to explore. These parameters control the trade-off between model complexity and error, and the influence of individual training examples, respectively.
   - cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42): Specifies 5-fold stratified cross-validation, maintaining class proportions within each fold.
   - scoring='f1': The F1-score is chosen as the optimization metric, balancing precision and recall, which is particularly useful in classification tasks where both false positives and false negatives are important.
   - n_jobs=-1: Utilizes all available CPU cores for faster computation.

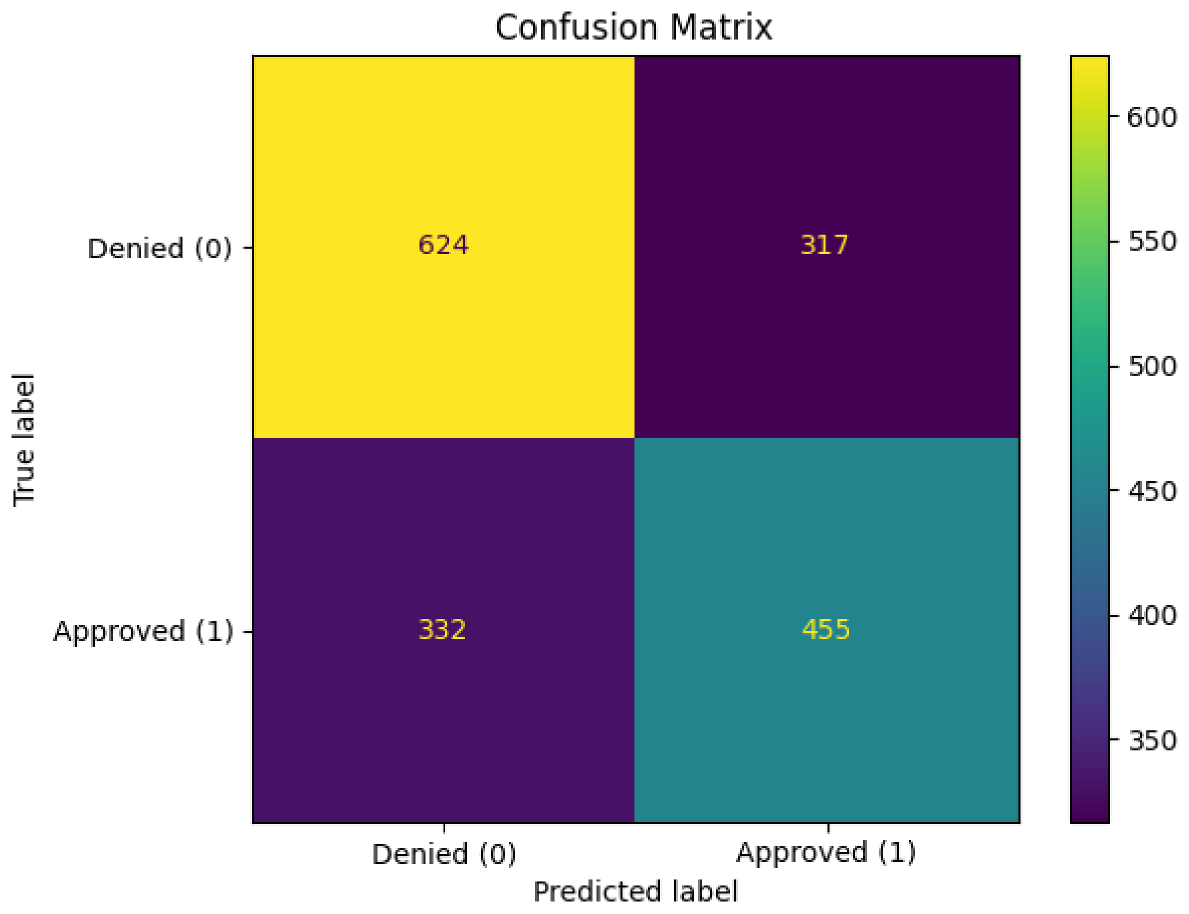
22 Training and Optimization: The grid.fit(X_train, y_train) method trains the pipeline across all hyperparameter combinations and cross-validation folds, identifying the best model.

23 Results Display: The best parameters (grid.best_params_) and the corresponding best cross-validation F1-score (grid.best_score_) are printed.

24 Final Evaluation: The best-performing model (best_model = grid.best_estimator_) is used to predict labels on the unseen test set (X_test). The model's performance is then comprehensively evaluated using:

   - Confusion Matrix: A table showing the counts of true positive, true negative, false positive, and false negative predictions.
   - Classification Report: Provides precision, recall, F1-score, and support for each class, offering a detailed breakdown of the model's performance.

Confusion Matrix Results



Classification Report Results

```
              precision    recall  f1-score   support

           0     0.6527    0.6631    0.6579       941
           1     0.5894    0.5781    0.5837       787

    accuracy                         0.6244      1728
   macro avg     0.6210    0.6206    0.6208      1728
weighted avg     0.6239    0.6244    0.6241      1728
```

References

[1] Pins Face Recognition | Kaggle