Evaluating the Accuracy of Machine Learning Models on Live Face Camera Images

Nickolaus A. White

Department of Computational Mathematics, Michigan State University

CMSE890: Applied Machine Learning

Dr. Michael Murillo

April 27, 2023

Evaluating the Accuracy of Machine Learning Models on Live Face Camera Images

The purpose of this project is to combine highly customizable machine learning models from scikit-learn with OpenCV (cv2) live face camera images. This is an educational report which highlights the process of discovering the most suitable machine learning models for facial recognition using the Python coding language.

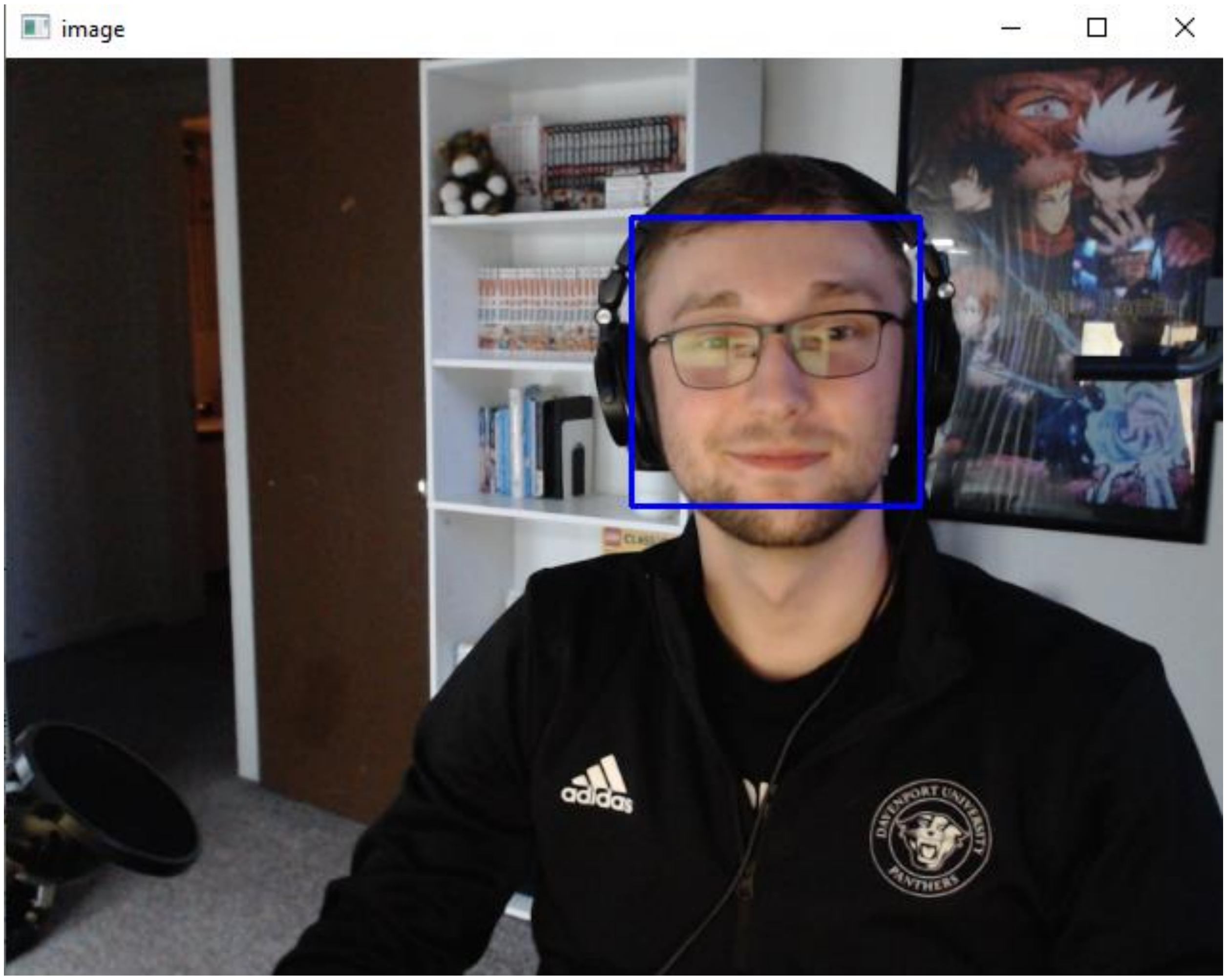
Facial recognition is a technology that has been around for a few decades but has recently gained more attention due to its use in security, law enforcement, and social media. Live facial recognition is a more recent development that involves capturing and analyzing a person's face in real-time. In this project, the OpenCV Python library is used to develop a live facial recognition system. 50 images from each volunteer will be captured and stored in a file system for further analysis. Using various machine learning algorithms to train the system, models are run and tuned to improve classification accuracy on the different users' images. This project is a combination of both OpenCV and scikit-learn, a mix between real-time facial recognition and highly customizable machine learning models. The main goal of this project is to achieve high accuracy models on image data captured from a computer webcam.

Data Collection

The first step in developing a live facial recognition system is to capture images of people's faces. The application does this by using the OpenCV library to access the webcam of a computer and capture images in real-time. The code is setup to capture 50 photos of someone in rapid succession following the user's input of an eight integer long User ID, and the execution of the Enter key on the keyboard to verify the user is ready. Capturing will only take place if a rectangular box is formed around the user's face to verify that a face is detected within the bounds of the webcam. See Figure 1 below for an example of the application being run.

Figure 1

Live face capture

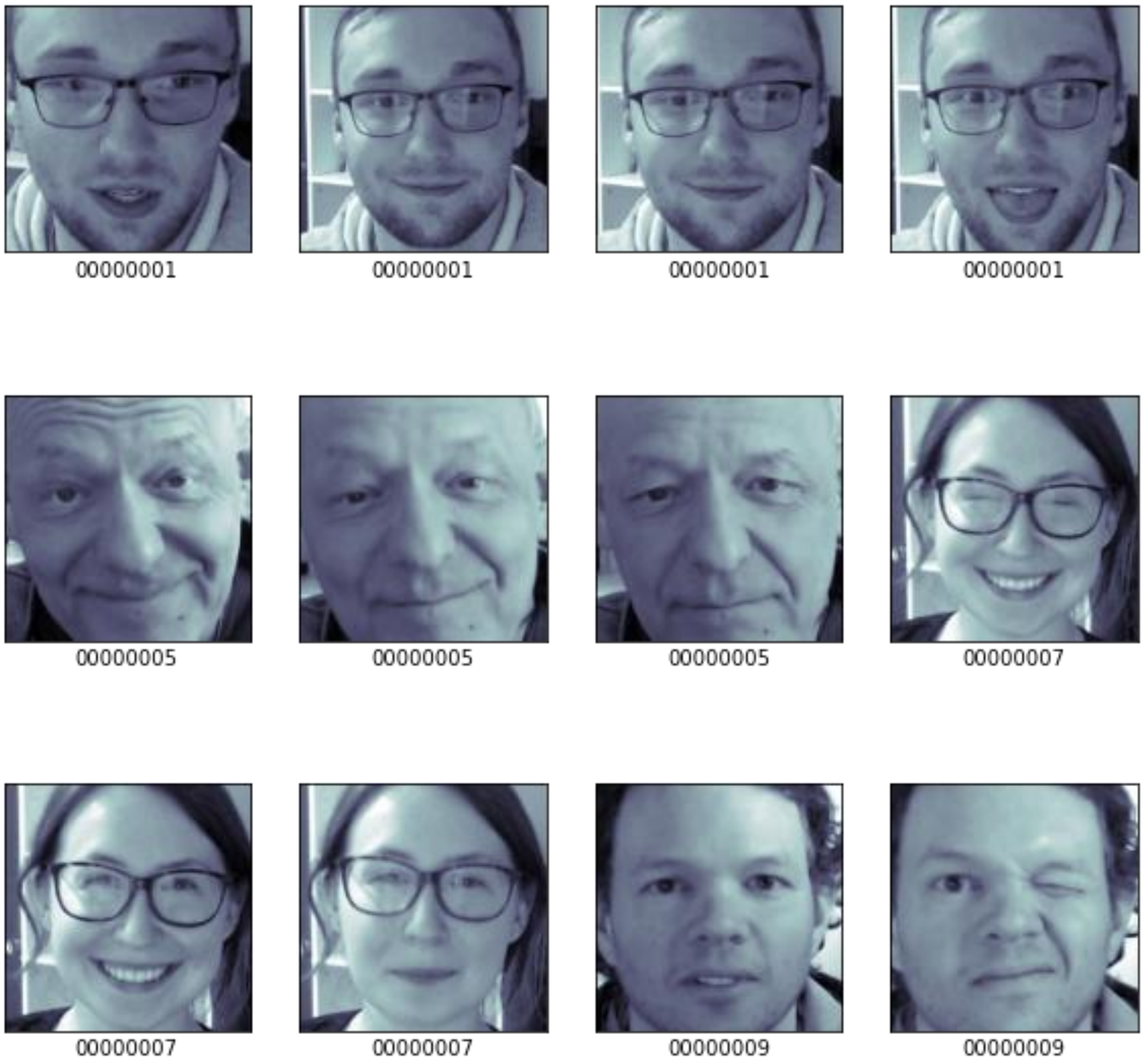


Note: The above figure demonstrates the user's point of view when the system captures 50 photos in succession to store within the file system for future classification.

Once the images are captured, they are stored in a folder for further processing, each photo is tagged with the User ID and photo number.

Figure 2

Preview of images used in machine learning models



Note. The above shows the sample images within the file system and their associated user IDs.

It is important to capture images of people with different facial expressions, lighting conditions, and backgrounds to ensure that the system is robust and can handle different scenarios. Because of this, the data collection for the model represented in this report was carefully executed. Varying environments were used to capture the volunteers’ faces; different angles, and different facial expressions were also considered.

In this trial of the facial recognition system, myself, and three volunteers were used to create the dataset. The dataset consisted of 200 total images, 50 per user. Given that this is a lower number of differing classes for the machine learning models, it is assumed that the prediction accuracy will be quite high. As mentioned before, results will vary based on the image quality, but it will also differ based on the size of the data.

Data Preprocessing

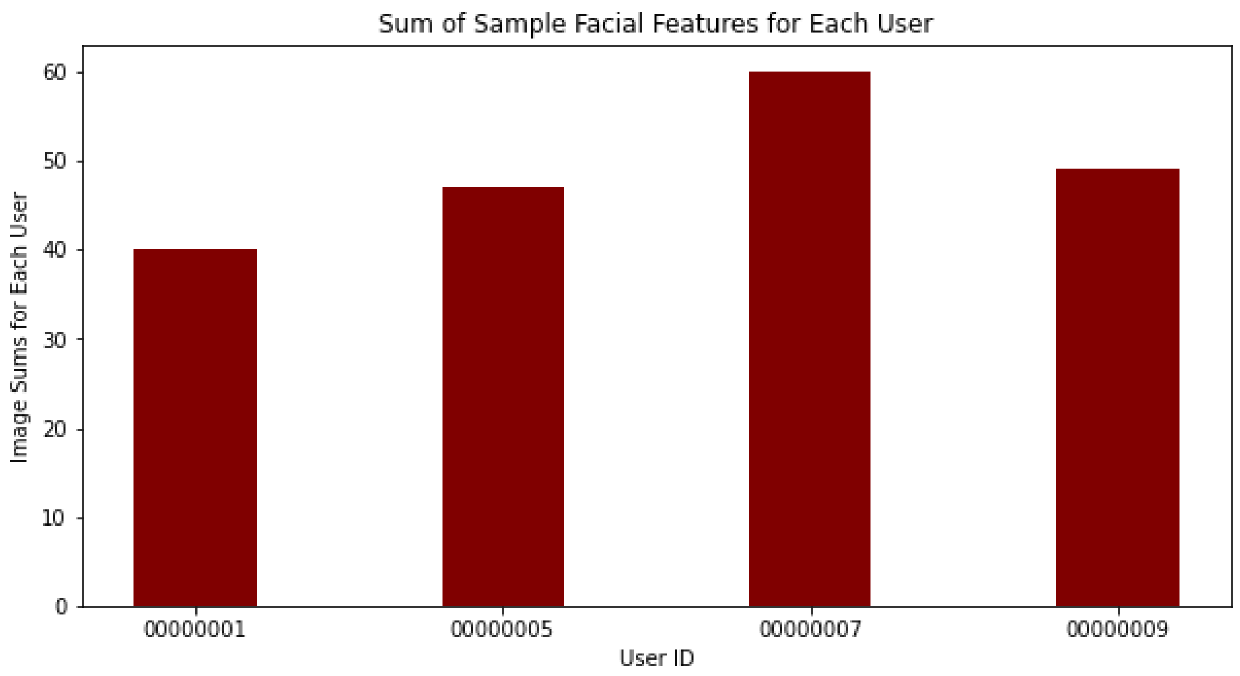
Before the captured images are usable for training the facial recognition system, they need to be preprocessed. This involves resizing the images to a standard size, converting them to grayscale, and transforming each pixel value to float32, dividing each pixel by 255. Respectively, the numpy.array() and numpy.resize() functions were used to carry out these actions.

Facial features also need to be extracted from the images. There are many algorithms for facial feature extraction, but the best fit for this application was OpenCV's detectMultiScale(). This function detects objects of different sizes in the input image, then returns the detected objects as a list of rectangles. This helps in the classification of different users when the images are fed into the varying models.

Once all image arrays are modified for better use in the models, the last preprocessing step is carried out: verifying the adequate number of images per user. Using the matplotlib library, a bar graph is presented to show the sum of samples for each class.

Figure 3

Sum of samples for each class



Note. The above figure shows the sums of extracted facial feature images, grouped by User ID.

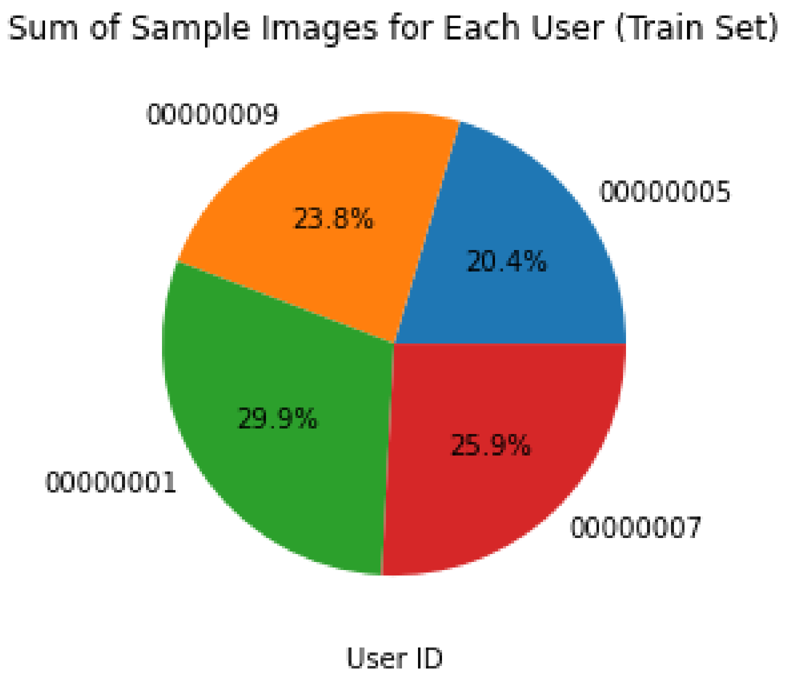
Based on the results shown in Figure 1, the dataset is altered to include only a select number of images per user. Thus, preventing the model from being over trained on only one dominant user.

Once preprocessing is finished, the images are extracted and used to train the facial recognition system. The scikit-learn library is heavily used in this portion of the project to implement various machine learning algorithms.

To be able to implement the many machine learning algorithms, a train-test split is needed, preferably a 70/30 ratio split. In order to represent the entire dataset within the training dataset, scikit-learn' s model_selection.train_test_split() function is utilized to accurately shuffle and separate the images and labels into train and test sets. To verify that the input for the models is usable, pie charts are provided for dispersion of both the training and testing set, see Figure 4 and Figure 5 below.

Figure 4

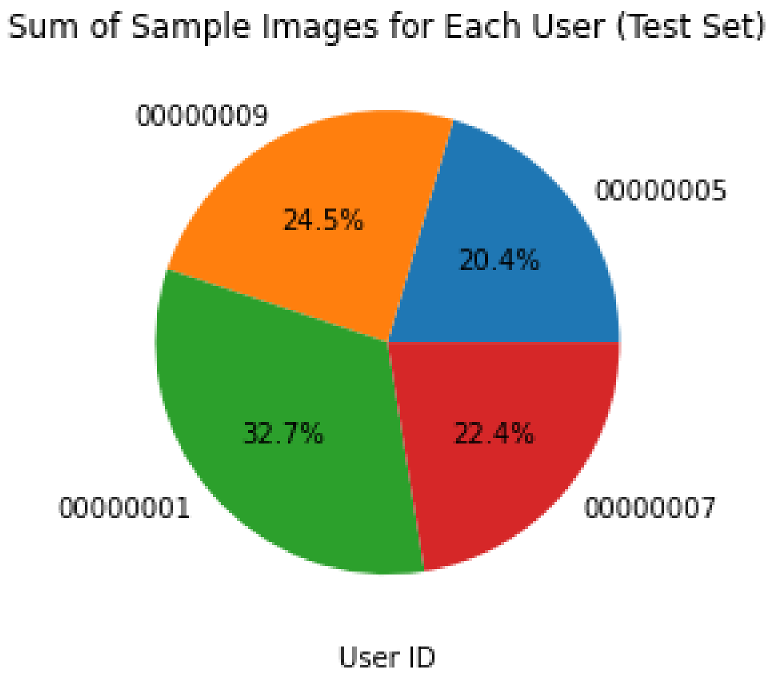
Unique images for each class within the training data set



Note. The above shows the sums of images within the training set, grouped by User ID.

Figure 5

Unique images for each class within the testing data set



Note. The above shows the sums of images within the testing set, grouped by User ID.

Machine Learning Algorithms

SVM

The first algorithm used is the Support Vector Machine (SVM) algorithm. SVM is a powerful machine learning algorithm that can be used for classification and regression tasks. The SVM algorithm is used in this scenario to train our system to recognize different faces based on their facial features. To find the best hyperparameters for this model, GridSearchCV() is utilized, varying the input parameters to find the best possible model. Varying the C, gamma, and kernel hyperparameters, the best possible model is found and used for model predictions.

SVMs are based on the idea of finding the hyperplane that best separates two classes of data in a high-dimensional space. In the case of facial recognition, SVMs can be used as a classifier to distinguish between different faces. This is achieved by training the SVM with labeled data, where each face is associated with a particular label or class. The SVM then learns to distinguish between these classes by finding the hyperplane that maximizes the margin between the two classes of data.

The math behind SVM involves finding the optimal hyperplane that separates the data into two classes with the maximum margin. “The goal is to have the largest possible margin between the decision boundary that separates the two classes and the training instances” (Geron, 2019). This is done by solving an optimization problem that involves maximizing the distance between the hyperplane and the closest data points from each class, subject to some constraints. “The linear SVM classifier model predicts the class of a new instance x by simply computing the decision function $w^T x + b = w_1 x_1 + \cdots + w_n x_n + b$”, 0 if $w^Tx + b < 0$ and 1 if $w^Tx + b \geq 0$ (Geron, 2019). In simpler terms, the math behind the SVM involves the following steps:

1. Data Transformation: The input data is transformed into a higher-dimensional space using a kernel function, such as a radial basis function (RBF) or polynomial kernel, to allow for a linear separation of the data in the transformed space.
2. Hyperplane Definition: The SVM algorithm defines a hyperplane that separates the two classes with the maximum margin between them. The margin is the distance between the hyperplane and the closest data points from each class.
3. Optimization: The SVM algorithm solves an optimization problem to find the hyperplane that maximizes the margin and minimizes the classification error. This involves solving a quadratic programming (QP) problem to find the weights and biases that define the hyperplane.
4. Classification: During classification, the SVM algorithm assigns new data points to the class that the hyperplane separates them into based on their position relative to the hyperplane.

SVM is used for facial recognition because it can effectively handle non-linear data and high-dimensional data, making it suitable for processing images and facial features. SVM can also handle unbalanced datasets, where some classes may have fewer examples than others, making it suitable for facial recognition tasks where the number of samples in different classes can vary significantly. Additionally, SVM has a regularization parameter that can be tuned to control the tradeoff between model complexity and accuracy, making it robust to overfitting and suitable for handling noisy data. Overall, SVM is a versatile and widely used algorithm in machine learning.

There are different types of SVMs, such as linear SVMs, kernel SVMs, and multiclass SVMs, each with their own mathematical formulations and algorithms for solving them. The application used in this project utilizes both a linear and radial basis function (rbf) kernel SVM. Further research is planned to incorporate the different types of SVM's and how their results compare to each other.

Random Forest

Random Forest is a machine learning algorithm that is used for classification, regression, and other tasks. It is an ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model. "The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features" (Geron, 2019). This results in greater tree diversity, which generally yields a better overall model.

An important thing to understand is the math behind Random Forest which involves the following steps:

1. Randomly select a subset of the training data and a subset of the features for each tree in the forest.
2. For each tree, grow it using the selected data and features by recursively splitting the data based on the selected features until a stopping criterion is met (e.g., maximum depth or minimum number of samples per leaf).
3. During the splitting process, the algorithm calculates the information gain or the decrease in impurity (e.g., Gini impurity or entropy) at each split to determine the best split.
4. Once all the trees are grown, the algorithm predicts the label of a new data point by aggregating the predictions of all the trees using either majority vote (for classification) or average (for regression).

Random Forest is used within this facial recognition system because it is effective at handling high-dimensional data, such as facial images, and can handle noisy and missing data. Random Forest can also handle nonlinear relationships between the features and the labels and can be trained quickly on large datasets. Furthermore, the use of multiple trees in the forest reduces the risk of overfitting and improves the generalization performance of the model.

Multilayer Perceptron

Multilayer Perceptron (MLP) is a type of feedforward neural network that is used for supervised learning tasks, including facial recognition. The main idea behind MLP is to use multiple layers of connected neurons to learn a nonlinear mapping between the input and output data. The math behind MLP involves the following steps:

1. Input Layer: The first layer in an MLP is the input layer, which receives the input data, such as facial images, as a vector of features.
2. Hidden Layers: The input layer is followed by one or more hidden layers, each consisting of a set of neurons that perform a weighted sum of the inputs, followed by an activation function, such as sigmoid or ReLU, to introduce nonlinearity into the model.
3. Output Layer: The final layer of the MLP is the output layer, which produces a vector of outputs, such as the probabilities of different classes in a classification task.
4. Training: During training, the weights of the connections between the neurons in the MLP are updated using a backpropagation algorithm, which uses the gradient of the loss function with respect to the weights to adjust the weights in the direction that minimizes the loss.

MLP is used in this facial recognition because it can learn complex nonlinear relationships between the input features and the output labels, making it effective for distinguishing between different faces. MLP can also handle high-dimensional data, such as facial images, and can learn features automatically from the data, reducing the need for manual feature engineering. Finally, the ability of MLP to model complex functions with multiple hidden layers makes it a powerful tool for many machine learning tasks, including facial recognition.

K-nearest Neighbors

The K-nearest Neighbors (KNN) algorithm is a non-parametric machine learning algorithm that is used for classification and regression tasks. The main idea behind KNN is to classify a new data point by finding the K nearest data points in the training set and using their labels to make a prediction. The math behind KNN involves the following steps:

1. Distance Metric: The first step in KNN is to choose a distance metric, such as Euclidean distance or cosine similarity, to measure the similarity between data points.
2. Training: During training, KNN stores the entire training dataset.
3. Prediction: During prediction, KNN finds the K nearest data points in the training set to the new data point based on the chosen distance metric. It then assigns the label of the new data point based on the majority vote of the K nearest neighbors.

KNN is used because it can classify a new facial image by comparing it to a database of known faces. The distance metric used in KNN can be based on facial features, such as landmarks, textures, or appearance, allowing the algorithm to identify similar faces in the database. KNN can also handle nonlinear relationships between the features and the labels and can be trained quickly on large datasets. Finally, KNN is a simple and intuitive algorithm that requires little parameter tuning.

Gaussian Process Classification

Gaussian process classification (GPC) is a probabilistic machine learning algorithm that is used for classification tasks. The main idea behind GPC is to model the probability distribution over functions that map the input data to the output labels, using a Gaussian process prior over the function space. The math behind GPC with Laplace approximation involves the following steps:

1. Gaussian Process Prior: The first step in GPC is to define a Gaussian process prior over the function space, which specifies a mean function and a covariance function that describe the properties of the functions that are consistent with the prior knowledge.
2. Likelihood Function: The likelihood function specifies the probability of observing the output labels given the input data and the function values. In GPC, the likelihood function is often assumed to be a Bernoulli distribution for binary classification tasks.
3. Posterior Distribution: The posterior distribution over the function space is computed using Bayes' rule, which combines the prior distribution with the likelihood function. However, the posterior distribution is generally intractable due to the high dimensionality of the function space.
4. Laplace Approximation: To compute the posterior distribution, a Laplace approximation is often used, which approximates the posterior distribution with a Gaussian distribution that is centered at the mode of the posterior distribution and has a covariance matrix equal to the negative Hessian of the log posterior density.
5. Prediction: During prediction, the Laplace approximation is used to compute the mean and variance of the predictive distribution over the output labels, given the input data. The mean and variance can be used to make a probabilistic classification decision.

GPC is used for facial recognition because it can model complex nonlinear relationships between the input features and the output labels, making it effective for distinguishing between different faces. GPC can also handle uncertainty in the predictions and can be used to quantify the confidence in the predicted labels, making it useful for applications where the accuracy of the predictions is critical, such as this facial recognition system. Finally, GPC can be used to model complex dependencies between the input and output variables, allowing it to capture subtle relationships between facial features and identities, making it a powerful tool for many machine learning tasks.

Decision Tree

A decision tree is a non-parametric machine learning algorithm that is used for classification and regression tasks. The main idea behind a decision tree is to recursively split the input space into regions based on the values of the input features and assign a label to each region based on the majority vote of the training examples that belong to that region. The math behind a decision tree involves the following steps:

1. Splitting Criterion: The first step in building a decision tree is to choose a splitting criterion, such as information gain or Gini impurity, that measures the quality of a split based on the entropy or purity of the resulting regions.
2. Recursive Splitting: During training, the decision tree algorithm recursively splits the input space based on the chosen splitting criterion, creating a tree structure with internal nodes that correspond to the splitting rules and leaf nodes that correspond to the output labels.

3. Prediction: During prediction, the decision tree algorithm traverses the tree from the root to a leaf node, following the splitting rules that match the input features, and assigns the label of the corresponding leaf node to the input data point.

A decision tree was used for this facial recognition system because it can be used to learn complex non-linear decision boundaries that separate different faces based on their features, such as shape, color, and texture. Decision trees can also handle both categorical and continuous input features, making them versatile for handling different types of facial features. Additionally, decision trees can be easily interpreted and visualized, allowing human experts to understand and debug the decision-making process. Finally, decision trees can be trained quickly on large datasets, making them suitable for real-time applications, such as live facial recognition.

Adaptive Boosting

Adaptive Boosting, or rather AdaBoost, is an ensemble machine learning algorithm that is used for classification tasks. The main idea behind AdaBoost is to combine multiple weak classifiers into a strong classifier that can make accurate predictions on the input data. The math behind AdaBoost involves the following steps:

1. Weighted Data: During training, the AdaBoost algorithm assigns a weight to each training example, which reflects the difficulty of classifying that example based on the previously weak classifiers.
2. Weak Classifier: The AdaBoost algorithm selects a weak classifier, such as a decision tree or a simple linear classifier, that performs slightly better than random guessing on the weighted data.

3. Error Calculation: The AdaBoost algorithm calculates the error of the weak classifier based on the weighted data and updates the weights of the training examples to give more weight to the examples that were misclassified by the weak classifier.
4. Classifier Weight: The AdaBoost algorithm computes the weight of the weak classifier based on its classification error and adds the weak classifier to the ensemble with a weight that reflects its accuracy.
5. Boosting: The AdaBoost algorithm repeats steps 2-4 multiple times, selecting a new weak classifier for each round and updating the weights of the training examples based on the previous classifiers, until a predefined number of rounds or a desired accuracy is reached.
6. Prediction: During prediction, the AdaBoost algorithm combines the output of the weak classifiers in the ensemble, weighted by their accuracy, to make a final prediction on the input data.

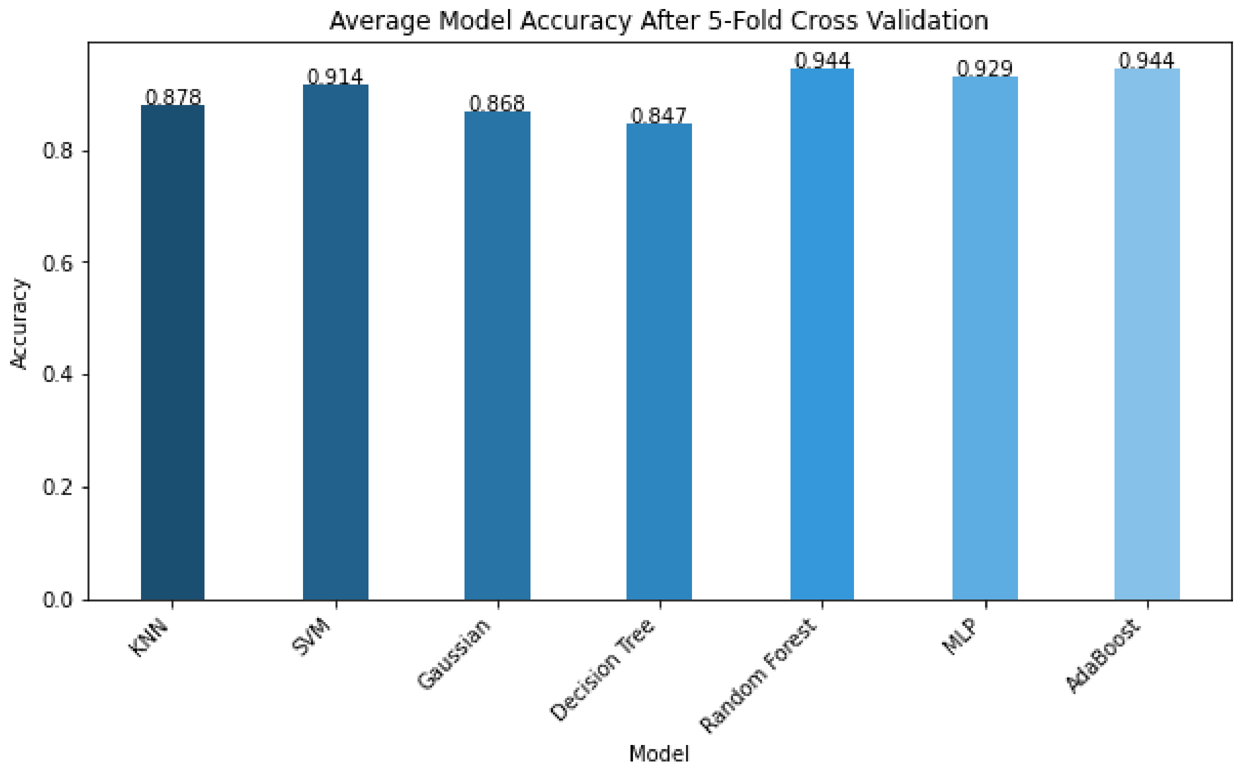
AdaBoost is used for facial recognition because it can effectively combine multiple weak classifiers to create a strong classifier that is robust to noise and variations in the input data, making it effective for handling complex facial features and conditions, such as changes in lighting, pose, and expression. AdaBoost can also handle unbalanced and noisy datasets, where some classes may have fewer examples or more noise than others, making it suitable for real-world facial recognition applications. Finally, AdaBoost can be easily parallelized and scaled to handle large datasets and complex models, making it a popular choice for many machines learning tasks.

Evaluation Metrics

To evaluate the performance of the facial recognition system, a multi-step evaluation metric is used. This metric is accuracy, or rather how well the system can correctly classify the images. However, this is done in a more complex manner. Using the mean of results from a 5-fold cross validation, an accuracy is output when the input contains a test set of images and their respected labels. The classifiers are run in succession, one after another. Thus, scores from each iteration in the stratified k-fold are added together and divided by their count. This results in the mean, or average accuracy score across all folds in the cross validation. See Figure 4 below, this shows the results from the differing models when using their best set of hyperparameters.

Figure 6

All model results from 5-fold cross validation



Note. The above shows the *mean accuracies for each model after performing 5-fold cross validation, standard scaling, and hyperparameter tuning.*

This metric was chosen due to its accounting for different types of scenarios. It provides an estimate of the generalization performance of the model on unseen data, as it evaluates the performance of the model on different subsets of the data and gives an idea of the stability of the model's performance. As shown above, the best performing models were Random Forest and AdaBoost. This is most likely due to AdaBoosts handling of unbalanced and noisy datasets, and Random Forests effectiveness at handling high-dimensional data, which is apparent in the facial images. Overall, all the models scored quite high in accuracy, each of them reaching above 84%.

The addition of other evaluation metrics was considered but they were ultimately left out due to their high time complexity, mainly caused by the multiple machine learning models being ran in succession. Evaluation metrics such as receiver operating characteristic curves (ROC), precision and recall, and confusion matrices were all considered and attempted. These metrics are critical tools in assessing the performance of machine learning models. They provide valuable insights into the model's ability to correctly identify and classify data points and help identify areas for improvement. Plans are underway for adding these evaluation metrics to the project without the high cost of computation.

Conclusion

In this project, a live facial recognition system was developed using Pythons OpenCV and scikit-learn libraries. Images of people's faces are captured through the webcam, stored in a folder, preprocessed, and run through various models. The outcomes of these models performed better than expected, proving that live camera footage, when in actionable environments, produces the capability of training many different machine learning models.

To evaluate the performance of the system, evaluation metrics, such as the mean model accuracy, are utilized. By using this metric, the effectiveness of the system is measured and the ability to identify areas for improvement is plausible. Looking over the results from the differing machine learning models, the classifications performed quite well, reaching above the original 80% accuracy goal.

One limitation of the system is that it relies on the quality of the images captured by a webcam. If the lighting conditions are poor, or the person's face is partially obscured, the system may not be able to accurately recognize the user. There could also be signs of user error; for example, a person capturing 50 photos of their wall when starting the live face capture then walking away, or, moving quickly which blurs the photos. As of right now, there is no option to delete photos from the dataset, but this function is planned for implementation in the future.

Another limitation is the potential for bias in the system. If the images used to train the system are not representative of the entire population, the system may be biased towards certain groups of people. This is an important consideration, especially if the system is used in law enforcement or other sensitive applications. As of right now, a limitation of photos is set per user to prevent this from happening, but verifying the training and test split data is essential for obtaining the most accurate machine learning models.

Lastly, the project lacked something very crucial, a large dataset. The accuracy and effectiveness of the models depend heavily on the quality and quantity of data they are trained on. In this case, the models were not able to be fully tested due to the lack of data. Without enough data, it becomes difficult to validate the model's performance, evaluate its accuracy, and assess its ability to generalize to new, unseen data. This has potentially limited the model's usefulness in real-world applications. To overcome this challenge, more volunteers will be needed, which is the first action to take in future work.

In conclusion, live facial recognition is a powerful technology that has many potential applications. By using Python’s cv2 and scikit-learn libraries, accurate and efficient facial recognition systems are easy to engineer. However, it is important to be aware of the limitations of these systems and to continually evaluate and improve their performance. As stated previously, future work is planned for this facial recognition system, involving a user interface, more flexible code, and a much larger data set. If time allows, these additions will be exciting to pursue.

References

Geron, A. (2019). *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools and techniques to build Intelligent Systems* (2nd ed.). O'Reilly.