OPEN ACCESS

International Journal of Advances in Artificial Intelligence and Machine Learning Vol 1 No 1 (2024) https://doi.org/10.58723/ijaaiml.v1i1.296

Check for updates

Research Article

Scalability and Efficiency: A Comparative Study of Face Recognition Technologies

Received: Sept 09, 2024	Revised: Oct 30, 2024	Accepted: Nov 05, 2024	Publish: Nov 07, 2024

Mohd Zaki Zakari*, Misinem, Nyimas Sopiah, Lusiana Efrizoni

Abstract:

This article addresses the challenge of selecting the most effective machine learning algorithm for face recognition tasks, a common problem in academic research and practical applications. To tackle this issue, we conducted a comparative analysis of five widely used algorithms: Linear Discriminant Analysis (LDA), Logistic Regression, Naive Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). The study involved implementing each algorithm on a standardized dataset, followed by a rigorous evaluation of their performance based on accuracy metrics. The results revealed that LDA, Logistic Regression, and SVM significantly outperformed the other models, each achieving an impressive accuracy of 97%. This high accuracy indicates that these algorithms are well-suited for handling datasets with linearly separable classes. Naive Bayes also showed a strong performance with 90% accuracy, proving effective under the feature independence assumption. However, KNN lagged, with an accuracy of 70%, highlighting its sensitivity to data scale and local structure, which affects its applicability in larger datasets or real-time scenarios. The findings suggest that while LDA, Logistic Regression, and SVM are optimal for datasets with clear class distinctions, the choice of an algorithm should still be guided by specific data characteristics and computational constraints. This study underscores the necessity for carefully considering each algorithm's strengths and limitations, ensuring that the selected model aligns with the unique demands of the application. Future work could explore ensemble methods and advanced parameter tuning further to enhance the performance and robustness of these models.

Keywords: Comparison Algorithms, Face Recognition, Machine Learning, Olivetti dataset

1. INTRODUCTION

In the rapidly evolving field of biometric security, face recognition technology has emerged as a cornerstone of countless applications, ranging from law enforcement to personal device security. Despite its widespread adoption, the scalability and efficiency of face recognition systems remain pressing concerns, particularly as the volume of data and the number of users continues to grow exponentially (Gill et al., 2024). These challenges necessitate rigorous comparative analyses to determine which algorithms perform best under various operational constraints.

Face recognition technologies rely heavily on the underlying algorithms' ability to accurately and efficiently process and match facial data (Adjabi et al., 2020). Common metrics for evaluating these algorithms include accuracy, precision, recall, and F1 score (Vakili et al., 2020). This study aims to conduct a comprehensive comparison of several prominent algorithms: Linear Discriminant Analysis (LDA), Logistic Regression, Naïve Bayes, K-Nearest Neighbours (KNN), and Support Vector Machine (SVM). These algorithms were chosen due to their prevalent use in the field, providing a broad spectrum of techniques that range from statistical approaches to geometric and template-based methods (Fatima et al., 2020).

The selection of algorithms for this comparative study on face recognition technologies is guided by their distinct characteristics and demonstrated efficacy across varying contexts. LDA is renowned for its effective dimensionality reduction and ability to maintain class separability, making it ideal for time-sensitive applications (Reddy et al., 2020). Logistic Regression offers a robust probabilistic framework, ideal for environments that demand quick, probabilistic decision-making (Dumitrescu et al., 2022). On the other hand, Naive Bayes simplifies computations by assuming feature independence, enhancing scalability for large-scale applications (Alizadeh et al., 2021).

KNN, a non-parametric method, is adaptable to different data types and proves effective in datasets with high variability, which is typical in dynamic face

recognition settings (Huang et al., 2020). Meanwhile, an SVM is adept at handling high-dimensional data, which is crucial for detailed and complex face recognition tasks (Hussain, 2019).

These algorithms were chosen to provide a comprehensive overview of the scalability and efficiency of face recognition systems. They span from statistical methods to geometric and template-based approaches, each bringing unique strengths to tackle the challenges of modern biometric security. This broad spectrum ensures that the study addresses various aspects of face recognition technology, from simplicity and speed to accuracy and computational efficiency (Kortli et al., 2020).

Previous research, such as the works by Zhao et al. (2020) and Sharmila et al. (2019), highlights the varying performance of these algorithms under different conditions. These studies suggest that while SVM and LDA perform well in controlled environments, their efficiency may diminish in more dynamic settings. Conversely, KNN has shown robustness in outdoor scenarios. Building upon these insights, this study will compare these algorithms in different environments and their scalability for large-scale deployments.

This comparative analysis will identify optimal algorithms that balance accuracy with efficiency, addressing the critical needs of expanding data volumes and computational demands in face recognition systems. The findings are expected to guide future advancements in biometric technology, ensuring more secure and efficient implementations.

2. MATERIAL AND METHOD

This research systematically evaluates six prominent machine learning algorithms for face recognition: LDA, Logistic Regression, Naïve Bayes, KNN, and SVM. The evaluation focuses on the mathematical models underlying each algorithm, their similarities and differences, and their advantages and disadvantages.

Linear Discriminant Analysis (LDA)

LDA is a statistical technique used in pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events (Zhao et al., 2024). The goal of LDA is to project the features in the dataset onto a lower-dimensional space with good class separability to avoid overfitting (the "curse of dimensionality") and reduce computational costs.

LDA specifically seeks to reduce dimensions by maximizing the between-class variance and minimizing the within-class variance. Mathematically, this involves calculating two scatter matrices: within-class scatter matrix (S_W) and between-class scatter matrix (S_B).

Within-Class Scatter Matrix (S_W)

This matrix is a measure of the amount of variance within each class. It is computed by summing up the scatter matrices for each class:

$$S_W = \sum_{I=1}^{K} S_i = \sum_{i}^{k} {}_{=1} \sum_{x \epsilon D_i} (x - \mu_i)\ (x - \mu_i)^T \qquad (1)$$

where μ_i is the mean vector of class *i*, D_i is the set of observations from class *i*, and *x* is the data vector in D_i.

Between-Class Scatter Matrix (S_B)

This matrix represents the separation between the different classes. It is computed based on the differences between the mean of each class and the overall mean of all samples:

$$S_B = \sum_{I=1}^{K} N_i\ (\mu_i - \mu)(\mu_i - \mu)^T \qquad (2)$$

where μ is the overall mean of all samples, μ_i is the mean vector of class iii, and N_i is the number of samples in class *i*.

The objective of LDA is to maximize the ratio of the determinant of S_B S_W. This is typically done by solving the generalized eigenvalue problem for $S_w^{-1} S_B$ and select the top eigenvectors (corresponding to the largest eigenvalues), which form the transformation matrix to project the data onto a lower-dimensional space.

Advantages and Disadvantages of LDA

LDA is valued for its simplicity and efficiency. Implementing it is relatively straightforward and demands fewer computational resources than more complex classifiers such as SVM. This makes LDA a practical choice for projects where computational efficiency is a priority (Jelodar et al., 2019). Additionally, LDA exhibits strong performance in scenarios involving multiple classes. Its method of maximizing the distance between class means and minimizing variance within each class makes it effective for distinguishing between multiple groups. Another significant advantage of LDA is its robustness; it tends to be less prone to overfitting,

particularly when the dataset has more samples than features. This characteristic ensures that LDA can perform reliably even with extensive data.

LDA comes with several limitations. The algorithm assumes that all input data follows a Gaussian distribution, an assumption that may not hold in many real-world scenarios. This can lead to suboptimal performance when the data distribution deviates from normality. LDA also presupposes that all classes share identical covariance matrices, meaning it might not perform well when this uniformity does not exist, potentially reducing the classifier's effectiveness. Furthermore, LDA's reliance on mean vectors and covariance matrices to construct the classifier makes it sensitive to outliers. Outliers can significantly distort the mean and variance, leading to misleading representations of classes and negatively affecting the classifier's ability to make accurate predictions.

While LDA is a useful tool for pattern recognition and classification tasks due to its straightforward implementation and efficiency, its effectiveness is bounded by certain data distribution and covariance assumptions. Its sensitivity to outliers further necessitates careful data preprocessing to ensure optimal performance.

Logistic Regression

Logistic Regression is a statistical method for predicting binary outcomes (Schober & Vetter, 2021). Unlike linear regression, which predicts continuous outcomes, logistic regression estimates the probabilities of binary outcomes, typically classifying data into one of two categories. The core of logistic regression is the logistic function, also known as the sigmoid function, which maps any real-valued number into the (0, 1) interval, making it suitable for probability estimation.

The logistic function is given by:

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad (3)$$

where z is the linear combination of input features (x) and weights (w), expressed as $z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$ Here, w_0 is the intercept, and $w_1, w_2, \cdots, w_n$ the coefficients the model learns during training.

The model uses the logit (log of odds) as the link function. The odds of the dependent variable equaling a particular value (usually "success" or "1") is the exponential of the linear combination of the inputs:

$$Odds = \frac{p}{1-p} = e^z \quad (4)$$

where p is the probability of the outcome. The logistic regression model thus estimates the log odds of the dependent variable being 1, which can be transformed back into a probability with the sigmoid function.

Advantages and Disadvantages of Logistic Regression

Logistic Regression is widely appreciated for its probabilistic approach to classification problems (Salehi et al., 2019). Unlike models that only predict a label, it provides the probabilities associated with these predictions, offering a more nuanced view of the results. This feature is particularly valuable in decision-making processes where understanding the likelihood of an outcome is as crucial as the outcome itself. Additionally, logistic regression exhibits robustness against noise, making it a reliable choice for applications that involve large datasets prone to inconsistencies. Its computational efficiency is another significant advantage. Unlike more complex models like neural networks, logistic regression requires less computational power, making it ideal for limited resources or rapid response times.

However, logistic regression does have limitations that can affect its performance in certain scenarios. The model inherently assumes a linear relationship between the independent variables and the log odds of the dependent variable. This assumption can lead to underfitting when the decision boundary is complex or non-linear. While beneficial for computation, the model's simplicity might not capture more complex patterns without adequate modification. Furthermore, standard logistic regression is designed for binary classification and requires adjustments, such as shifting to a multinomial logistic regression, to manage multi-class problems. This adaptation can complicate the model, making it more challenging to implement and interpret.

Logistic Regression is a powerful tool for binary classification with large datasets; its application is best suited to scenarios where the relationship between variables is approximately linear. In more complex datasets, particularly those with non-linear relationships or multiple classes, the limitations of logistic regression become apparent, necessitating more sophisticated approaches or significant modifications to the standard model.

Naive Bayes

Naive Bayes classifiers are a family of probabilistic algorithms based on applying Bayes' theorem, with the "naive" assumption of conditional independence between every pair of features given the value of the class variable (Khajenezhad et al., 2021). This

simplification makes Naive Bayes easy to implement and highly efficient. The formula for Bayes' theorem in this context is:

$$P(C|X) = \frac{P(X|C)\ x\ P(C)}{P(X)} \quad (5)$$

where:

- *P(C/X)* is the posterior probability of the class (*C*, target) given predictors (*X*, features).
- *P(C)* is the prior probability of the class.
- *P(X/C)* is the likelihood, which is the probability of the predictor given the class.
- *P(X)* is the prior probability of the predictor.

The classifier combines this model with a decision rule, typically choosing the most probable hypothesis to make predictions. During training, the algorithm counts the occurrences of categories and features, calculates the corresponding probabilities, and stores these as its model. Prediction applies these probabilities to calculate the most likely category for a given instance.

Naive Bayes is a highly efficient and fast classification algorithm suitable for large datasets and real-time analysis. However, its reliance on the independence assumption can lead to decreased performance where feature interdependencies are crucial. Therefore, while Naive Bayes is excellent for baseline models and initial insights into data, more complex algorithms might be necessary for higher accuracy in interconnected feature environments.

Advantages and Disadvantages of Naive Bayes Classifiers

Naive Bayes classifiers are celebrated for their efficiency; they require only a modest amount of data to accurately estimate the parameters needed for classification (Zaw et al., 2019). This efficiency makes them particularly advantageous for handling large datasets. Another key strength of Naive Bayes is its speed. The simplicity of its calculations, primarily due to the independence assumption, allows these classifiers to perform extremely quickly, making them suitable for applications that require real-time predictions. Furthermore, the straightforward nature of Naive Bayes makes it easy to implement and understand. This ease of implementation ensures that the model can be quickly deployed and maintained, which benefits projects with tight development timelines or limited computational resources.

Naive Bayes classifiers also come with some significant limitations. The core feature independence assumption simplifies computation but is rarely true in practice. Most real-world features exhibit some degree of correlation, and the naïve assumption of their independence can lead to biased estimates, ultimately affecting the model's accuracy. In scenarios involving multiple classes, the algorithm also faces challenges with data scarcity. If a feature-class combination is not observed during training, the likelihood estimation for that combination defaults to zero, which can skew the prediction probabilities. This problem is typically mitigated with smoothing techniques like Laplace estimation, but these can add complexity to the model. Lastly, Naive Bayes generally underperforms when dealing with complex features with significant relationships, such as in text or image data where interaction terms or high pixel correlation are common. This limitation makes it less suitable for more complex classification tasks where the relationships between features significantly influence the outcome.

Naive Bayes classifiers are an excellent choice for quick, preliminary analysis and large-scale applications due to their computational efficiency and simplicity; their practical application can be limited by their basic assumptions and difficulty handling complex feature interactions. For more nuanced or interconnected datasets, other more sophisticated machine learning algorithms might be necessary to achieve higher accuracy.

K-Nearest Neighbors (KNN)

KNN is a straightforward, instance-based learning algorithm where the function is only approximated locally, and all computation is deferred until classification (Halder et al., 2024). The KNN algorithm does not build a model in the traditional sense. Instead, it classifies new data points based on the majority vote of the 'k' nearest data points, thus the name K-Nearest Neighbors.

The primary steps involved in KNN are:

1. **Select the number 'k' of neighbors**: This is a user-defined constant and a critical parameter that directly influences the prediction.
2. **Calculate the distance**: For each point in the data, compute the distance between the point in question and the training examples to determine the 'k' closest neighbors. The Euclidean distance is the most common metric for calculating this distance, though other metrics, such as Manhattan or Minkowski, can also be used. The Euclidean distance between two points *x* and *y* with *N* dimensions is calculated as:

$$d(x,y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2} \quad (6)$$

3. **Majority Voting**: Once the 'k' nearest neighbors are identified, the algorithm aggregates their

classes, and the new point is assigned the class most common among its nearest neighbors.

KNN is a highly flexible and easy-to-implement option for classification and regression tasks. It is particularly well-suited for applications where the decision boundary is irregular. However, its reliance on a complete dataset for operation and sensitivity to the scale and relevance of features can limit its efficiency and accuracy, especially as the size of the data increases.

Advantages and Disadvantages of K-Nearest Neighbors (KNN)

KNN is renowned for its simplicity and versatility, making it an accessible algorithm for those new to machine learning (Abu Alfeilat et al., 2019). It can handle classification and regression tasks adeptly, offering a straightforward implementation path. The non-parametric nature of KNN is another significant benefit; it does not assume any specific form of data distribution. This capability allows KNN to adapt to the actual complexities of the data, potentially capturing intricate patterns missed by parametric methods. Additionally, KNN is highly interpretable. It classifies new examples based on the majority vote from the nearest neighbors, providing clear and understandable reasoning for its predictions. This enhances trust and ease of validation in its applications.

KNN comes with several disadvantages, particularly concerning scalability and performance. As the size of the dataset increases, KNN's efficiency significantly decreases since each query involves recalculating the distance to all training instances, which can be computationally expensive. This characteristic makes KNN less suitable for applications with large datasets or where real-time performance is crucial. The algorithm also requires that all training data be in memory to make these calculations, leading to high memory usage, which can be a limitation in memory-constrained environments. Furthermore, KNN's performance is highly sensitive to irrelevant features and the scale of the data. Since it relies on distance calculations, disparate scales among features or noise introduced by irrelevant features can skew the results, necessitating careful preprocessing steps like feature scaling and selection to ensure the model's accuracy and reliability.

KNN offers ease of use and excellent adaptability to complex data relationships; its practical application is often hampered by its high computational and memory demands and its sensitivity to data quality and structure. These factors must be carefully managed to leverage KNN's capabilities in machine learning projects fully.

Support Vector Machine (SVM)

SVM is a powerful and versatile machine-learning model capable of performing linear and nonlinear classification, regression, and even outlier detection (Mustafa & Mohsin, 2021). SVM is fundamentally a classifier method that finds a hyperplane or a set of hyperplanes in a high-dimensional space that can distinctly classify the data points.

The optimal hyperplane is chosen to have the largest possible margin between support vectors, which are the data points closest to the hyperplane. The margin is the distance between the hyperplane and the nearest points from each class. A larger margin is associated with a lower generalization error of the classifier.

Mathematically, suppose the training data is linearly separable. In that case, you can select two parallel hyperplanes that separate the two data classes to make the distance between them as large as possible. The region bounded by these two hyperplanes is called the "margin," and the maximum-margin hyperplane is the hyperplane that lies halfway between them. These concepts are formalized as follows:

1. **Linear SVM**: In the case of linear SVM, the hyperplane is defined as the set of points xxx satisfying:
   $$w \,.\, x + b = 0 \quad (7)$$
   where *w* is normal to the hyperplane, and *b* is the bias (intercept).

   The objective is to minimize $\|w\|$ to maximize the margin under the constraint that all data points xix_ixi are classified correctly, formulated as:

   $$y_i(w.\ x_i + b) \geq 1 \quad (8)$$

   for all *i*, where y_i are the class labels.

2. **Nonlinear SVM**: For nonlinear classification, SVM uses a kernel function to transform the data into a higher-dimensional space where a linear hyperplane can separate data points. Common kernels include the polynomial, radial basis function (RBF), and sigmoid.

SVM is a powerful tool in the machine learning arsenal, capable of delivering high accuracy and robustness, especially in complex and high-dimensional datasets. However, its performance heavily depends on the correct choice of kernel and parameter tuning, which can require significant computational resources and expertise in machine learning.

Advantages and Disadvantages of Support Vector Machines (SVM)

SVM are well-regarded for their effectiveness in high-dimensional spaces, making them particularly suitable for datasets where the number of features exceeds the number of samples (Cervantes, et al., 2020). This capability stems from SVM's reliance on support vectors (critical training points) rather than dimensions, allowing it to manage complex datasets efficiently. Additionally, SVMs exhibit a high degree of versatility due to the range of kernel functions available. These kernels can be selected or tailored specifically to fit the unique structure of the data, whether the relationship among data points is linear or nonlinear. For instance, linear, polynomial, and radial basis function (RBF) kernels provide different ways of handling data separability, enhancing the model's adaptability. Another notable advantage is SVM's memory efficiency. The model primarily relies on a subset of training data (the support vectors) to make predictions, significantly reducing the memory load during the testing phase and enhancing computational efficiency.

Despite these strengths, SVMs also present certain challenges. The selection of an appropriate kernel function is crucial and can be complex. The right choice of kernel and its parameters significantly impacts the model's performance, with a poor choice potentially leading to overfitting, especially in cases where the data does not inherently support the assumptions of the chosen kernel. Parameter tuning is another critical aspect; SVM requires careful adjustment of parameters such as the regularization parameter (C) and specific kernel parameters. This tuning process, often requiring methods like cross-validation and grid search, can be both time-consuming and computationally intensive. Moreover, SVMs are known for their scalability issues—while efficient during testing, the training process has a computational complexity ranging from $O(n^2)$ to $O(n^3)$. This aspect makes SVM less suitable for large datasets, where training can be slow.

SVMs are powerful tools that deliver high accuracy in complex and high-dimensional datasets. However, their practical application can be limited by the need for meticulous kernel selection and parameter tuning and their computational demands during the training phase. These factors must be carefully managed to leverage SVMs' capabilities in machine learning projects fully.

Comparison Analysis of Algorithms for Face Recognition Machine Learning Approach

When evaluating the suitability of various machine learning algorithms for face recognition, it is important to consider their strengths and weaknesses in the specific demands of face recognition tasks. Algorithms such as LDA, Logistic Regression, Naive Bayes, KNN, and SVM each offer unique advantages and face particular limitations (Cavazos et al., 2021).

LDA is adept at dimensionality reduction and effectively maintaining class discriminatory information, which is beneficial in multi-class settings typical of face recognition. However, LDA assumes all classes share the same covariance matrix, which might not hold in real-world scenarios where facial expressions and lighting conditions vary widely. Logistic Regression provides probabilistic outputs beneficial for binary classification, like distinguishing whether a face belongs to a specific individual. However, it struggles with common non-linear decision boundaries due to the complex nature of human faces.

Naive Bayes is known for its computational efficiency and quick handling of high-dimensional data, making it suitable for real-time applications. Nonetheless, its feature independence assumption can undermine its effectiveness, as facial features are typically interrelated. KNN excels in flexibility and simplicity, classifying images based on the most similar examples in the training set without assuming any specific data distribution. However, KNN can become computationally burdensome as data volumes increase, and its performance heavily depends on the selection of the number of neighbors and the distance metric used.

SVM shines in high-dimensional spaces and efficiently handles the complexity of face recognition through kernel tricks that tackle non-linear data separations. The downside is that SVM requires meticulous parameter tuning and kernel selection, which can be time-consuming and computationally expensive, especially with large datasets.

The choice of a machine learning algorithm for face recognition depends significantly on specific application requirements, including the need for speed, accuracy, and the ability to handle large, complex datasets. While LDA and SVM are suitable for high-accuracy needs in controlled environments, Naive Bayes and KNN offer quicker but potentially less accurate alternatives. Logistic Regression, while typically less powerful alone, can be valuable in composite or ensemble methods that leverage its strengths while mitigating its weaknesses.

3. RESULT AND DISCUSSION

The Olivetti Faces dataset provides a robust framework for effectively evaluating the performance of machine learning algorithms for face recognition. This dataset, known for its collection of faces, offers a standardized way to assess and compare the efficacy of different algorithms under controlled conditions.

Data Preparation

The Olivetti dataset can be obtained from the Kaggle website at https://www.kaggle.com/datasets/imrandude/olivetti (Sheik, 2017). The Olivetti dataset consists of 400 grayscale images of 40 distinct subjects, each representing 10 images. The initial step involves preprocessing the data to ensure it suits the machine learning models. This includes normalizing the images to have zero mean and unit variance, resizing them to a common scale, and applying image enhancement techniques such as histogram equalization to improve contrast. Given the grayscale nature and uniformity of image sizes (64x64), this dataset allows for straightforward preprocessing without extensive manipulation. The dataset displays 40 faces from different people, as shown in Figure 1.

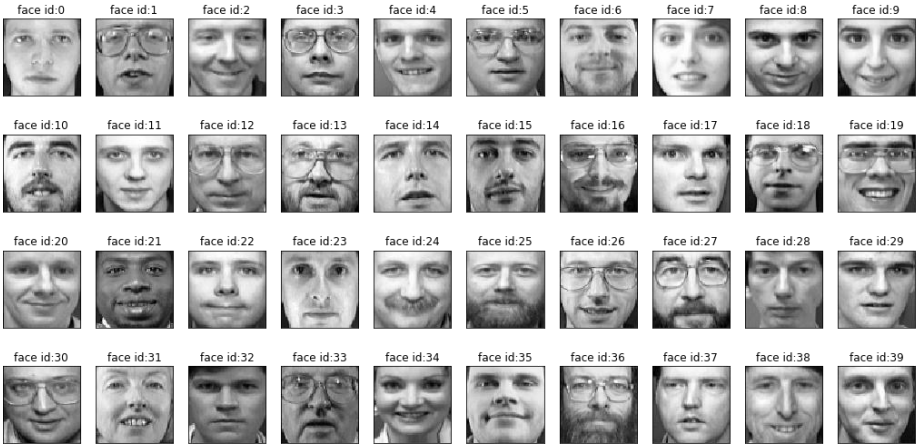

Figure 1. 40 distinct faces of people in the dataset.

We have 10 different images for each face belonging to the same person. As shown in Figure 2, five random people with all 10 images are displayed.

Figure 2. Five random people with all 10 image

Training and Testing Split

The dataset should be divided into training and testing sets for a valid comparison. A common approach is to use a 70-30 split, where 70% of the data is used for training the models, and the remaining 30% is used for testing, as shown in Figure 3 and Figure 4, which show Python code and the results.

```python
from sklearn.model_selection import train_test_split

# Assuming you have X (features) and target (labels)

# Split the data into training and temporary sets (70% training, 30% temp)
X_train_temp, X_test_val, y_train_temp, y_test_val = train_test_split(X, target, test_size=0.3, stra
tify=target, random_state=0)

# Further split the temporary set into testing and validation sets (50% testing, 50% validation)
X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5, stratify=y_te
st_val, random_state=0)

print("X_train shape:", X_train_temp.shape)
print("y_train shape:", y_train_temp.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
```

Figure 3. Python code to split the dataset

```
X_train shape: (280, 4096)
y_train shape: (280,)
X_test shape: (60, 4096)
y_test shape: (60,)
X_val shape: (60, 4096)
y_val shape: (60,)
```

Figure 4. The results of splitting the dataset

Initially, the dataset is divided into temporary training and testing subsets. Specifically, 30% of the data is allocated to the test set with the test_size=0.3 parameter. The stratify=y parameter ensures that the training and testing datasets have the same proportion of class labels as the original dataset, maintaining a balanced distribution of classes. This is important for models that are sensitive to unbalanced training data. The random_state=1 ensures that the splits are reproducible, meaning the same split will occur every time the code is run.

Further splitting of the temporary training data creates the final and validation sets. The validation set comprises 20% of the temporary training data specified by test_size=0.2. This secondary split is crucial for validating the model during training, allowing for hyperparameter tuning without compromising the test set, which is reserved strictly for final evaluation. The code's output confirms the structure and size of each data subset. The training set includes 280 samples with 4096 features, making up approximately 56% of the initial data when accounting for both splits. The testing and validation sets contain 60 and 68 samples, respectively, with 4096 features each. These splits help ensure that the model can be trained on a substantial amount of data, validated to avoid overfitting, and tested to assess its performance on unseen data. This structured approach to data splitting supports effective machine learning workflows by allowing distinct datasets for different stages of model training and evaluation. Thus, it promotes better generalization and performance in predictive tasks.

Model Implementation and Results

First, we tried using LDA algorithms to create the model and evaluate the dataset by testing and validating it. Then, other algorithms performed similar processing to compare all the results.

The LDA Model Creation and the Analysis Results

The model was created using the Python code, and then the training dataset was used for the training process, as shown in Figure 5.

```
[ ]  clf = LinearDiscriminatAnalysis()
     clf.fit(X_train_temp, y_train_temp)
     y_pred = clf.predit(X_temp)
     print("accuracy score:{:.2f}".format(metrixs.accuracy_score(y_test, y_pred)))

accuracy score:0.97
```

Figure 5. The created model, training, and evaluation are based on accuracy.

Next, Figures 6 and 7 show the confusion matrix and evaluate classification metrics such as precision, recall, and F1-score.

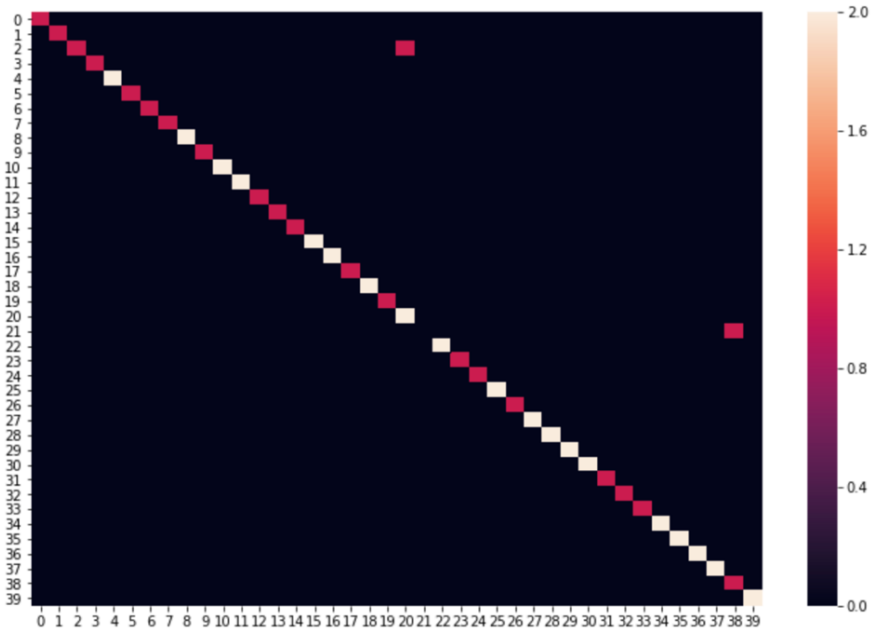

Figure 6. The confusion matrix of the LDA model

The image presented is a classification report generated from evaluating a machine learning model on a multi-class classification task. This report is instrumental in assessing the model's performance across several classes, which, in this case, appear to range from 0 to 39, totaling 40 distinct classes. The report breaks down four key performance metrics: precision, recall, F1-score, support for each class, and overall averages.

Precision is the ratio of correctly predicted positive observations to the total predicted positives. High precision across most classes (many having a perfect score of 1.00) indicates that the model is accurate in its predictions, rarely labeling a negative sample as positive. Recall, or sensitivity, measures the model's ability to find all positives. Like precision, many classes have a perfect recall score, demonstrating the model's effectiveness in identifying all samples from those classes.

F1-score is the harmonic mean of precision and recall and is particularly useful in comparing the test accuracy of different algorithms, especially in cases where class imbalances might exist. Most classes show a perfect F1 score 1.00, suggesting an excellent balance between precision and recall. However, some classes show lower scores, indicating areas where the model may struggle due to a lack of data or inherent difficulty distinguishing that particular class. The support metric indicates the number of actual occurrences of each class in the dataset, which helps understand its weight in the evaluation.

The report also includes three types of averages: micro, macro, and weighted averages. The micro average considers the aggregate contributions of all classes to compute the average metric, resulting in a score of 0.97 across precision, recall, and F1-score, reflecting high overall accuracy. The macro average computes the unweighted mean of the metrics, providing a big-picture view but not accounting for class imbalance. The weighted average compensates for class imbalance by weighting each class's contribution to the average by size, showing similar high performance.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	1
2	1.00	0.50	0.67	2
3	1.00	1.00	1.00	1
4	1.00	1.00	1.00	2
5	1.00	1.00	1.00	1
6	1.00	1.00	1.00	1
7	1.00	1.00	1.00	1
8	1.00	1.00	1.00	2
9	1.00	1.00	1.00	1
10	1.00	1.00	1.00	2
11	1.00	1.00	1.00	2
12	1.00	1.00	1.00	1
13	1.00	1.00	1.00	1
14	1.00	1.00	1.00	1
15	1.00	1.00	1.00	2
16	1.00	1.00	1.00	2
17	1.00	1.00	1.00	1
18	1.00	1.00	1.00	2
19	1.00	1.00	1.00	1
20	0.67	1.00	0.80	2
21	0.00	0.00	0.00	1
22	1.00	1.00	1.00	2
23	1.00	1.00	1.00	1
24	1.00	1.00	1.00	1
25	1.00	1.00	1.00	2
26	1.00	1.00	1.00	1
27	1.00	1.00	1.00	2
28	1.00	1.00	1.00	2
29	1.00	1.00	1.00	2
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	1
32	1.00	1.00	1.00	1
33	1.00	1.00	1.00	1
34	1.00	1.00	1.00	2
35	1.00	1.00	1.00	2
36	1.00	1.00	1.00	2
37	1.00	1.00	1.00	2
38	0.50	1.00	0.67	1
39	1.00	1.00	1.00	2
micro avg	0.97	0.97	0.97	60
macro avg	0.95	0.96	0.95	60
weighted avg	0.96	0.97	0.96	60

Figure 7. Precision, recall, and F1-score of the LDA model.

This classification report reveals that the model performs exceptionally well across most classes with only a few exceptions, showcasing high accuracy and reliability in its predictive capabilities. This detailed assessment helps pinpoint specific classes where improvements may be needed, either through additional data collection, enhanced feature engineering, or tuning model parameters.

Comparison and Results Analysis

The LDA creation model is then replicated for other algorithms under the same experimental conditions, like the same spilling dataset, to ensure fairness and consistency in the comparison. After all algorithms are tested, the results are recorded and organized into a table, as shown in Table 1.

Table 1. The comparison of accuracy results

No	Algorithms	Accuracy (%)
1	Linear Discriminant Analysis (LDA)	97
2	Logistic Regression	97
3	Naïve Bayes	90
4	K-Nearest Neighbours (KNN)	70
5	Support Vector Machine (SVM)	97

Table 1 presents accuracy results from a comparative analysis of five machine learning algorithms: LDA, Logistic Regression, Naive Bayes, KNN, and SVM. Each algorithm was evaluated in the context of a classification task.

LDA and Logistic Regression both report an impressive accuracy of 97%. This high level of accuracy suggests that the dataset likely features linearly separable classes, which both algorithms are particularly adept at handling. LDA's success can be attributed to its ability to maximize the ratio of between-class variance to within-class variance, effectively separating different classes in a dataset. Similarly, Logistic Regression's performance indicates that the decision boundary between classes is nearly linear, making it suitable for datasets where binary or dichotomous features prevail. However, both algorithms assume linearity in data relationships, which may not hold for more complex datasets, potentially leading to misclassifications if applied inappropriately.

With an accuracy of 90%, Naive Bayes performs slightly less than LDA and Logistic Regression but still shows strong predictive power. This could be due to its assumption that all predictors are independent of each other, an assumption that is rarely met in real-world data scenarios. Naive Bayes is known for its effectiveness in large datasets and performs well under the conditional independence assumption. However, its performance might degrade with data where feature interdependencies are significant, making it less reliable than algorithms that can accommodate such complexities.

KNN significantly underperforms relative to other algorithms with an accuracy of only 70%. This lower performance may result from the algorithm's sensitivity to the local data structure and the scale of the data features. KNN's efficacy highly depends on the choice of 'k' (the number of neighbors considered) and the distance metric used. An inappropriate choice of 'k' or not scaling the features correctly can greatly impair the model's ability to make accurate predictions. Moreover, KNN is computationally intensive as it involves calculations over the entire dataset for each prediction, making it less suitable for large datasets or real-time applications.

SVM matches the high accuracy of LDA and Logistic Regression, showcasing its robustness with a 97% accuracy rate. SVM's effectiveness stems from its ability to construct a hyperplane or set of hyperplanes in a high-dimensional space, effectively handling linear and non-linear boundaries depending on the kernel used. While SVM is powerful, its performance heavily relies on the correct choice of kernel and the tuning of its parameters, which can be complex and time-consuming. Incorrect parameter choices or kernel misfits can lead to poor generalization on unseen data.

While LDA, Logistic Regression, and SVM offer high accuracy and are suitable for datasets with distinct class boundaries, they require careful consideration of the underlying data structure to avoid misapplication. Despite its slightly lower accuracy, Naive Bayes offers a fast and effective solution for large, simpler datasets. KNN, though versatile, demands careful parameter tuning and consideration of computational efficiency, making it less favorable for larger datasets or applications requiring quick decision-making. Each algorithm has its strengths and limitations, and the choice of algorithm should be tailored to the specific characteristics of the dataset and the requirements of the task at hand.

4. CONCLUSION

The comparative analysis of five prominent machine learning algorithms—LDA, Logistic Regression, Naive Bayes, KNN, and SVM—on a classification task offers insightful reflections on each algorithm's suitability and performance across different data characteristics. LDA, Logistic Regression, and SVM emerged as top performers with a 97% accuracy rate, demonstrating their robust capability to handle linearly separable data and efficiently classify with high precision. Their performance, however, is contingent upon properly aligning the algorithm's assumptions with the underlying data distribution.

Despite its feature independence assumption, Naive Bayes also displayed commendable performance, proving its utility in scenarios requiring rapid, probabilistic decision-making. Its slightly lower accuracy of 90% underscores potential challenges in datasets with feature dependencies. Conversely, KNN's lower performance at 70% accuracy highlights its sensitivity to the local data structure, the scale of data features, and its computational demands, which can limit its applicability in larger datasets or scenarios requiring real-time prediction.

This analysis underscores the importance of choosing the right algorithm based on specific data traits and the problem context. It also highlights the necessity for careful parameter tuning and consideration of each algorithm's computational efficiency and scalability. Leveraging a combination of these algorithms or advancing their configuration through ensemble methods and hyperparameter optimization could further enhance predictive performance and adaptability in diverse application domains.

AUTHOR INFORMATION

Corresponding Authors

Mohd Zaki Zakaria, University Technology Mara, Malaysia.

Email: zaki@tmsk.uitm.edu.my

Authors

Misinem, Universitas Bina Darma.

Nyimas Sopiah, Universitas Bina Darma, Palembang, Indonesia

Lusiana Efrizoni, Universitas Sains dan Teknologi Indonesia, Pekanbaru, Indonesia.

REFFERENCE

Abu Alfeilat, H. A., Hassanat, A. B. A., Lasassmeh, O., Tarawneh, A. S., Alhasanat, M. B., Eyal Salman, H. S., & Prasath, V. B. S. (2019). Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. *Big Data*, *7*(4), 221–248. https://doi.org/10.1089/big.2018.0175

Adjabi, I., Ouahabi, A., Benzaoui, A., & Taleb-Ahmed, A. (2020). Past, Present, and Future of Face Recognition: A Review. *Electronics*, *9*(8). https://doi.org/10.3390/electronics9081188

Alizadeh, S. H., Hediehloo, A., & Harzevili, N. S. (2021). Multi independent latent component extension of naive Bayes classifier. *Knowledge-Based Systems*, *213*, 106646. https://doi.org/https://doi.org/10.1016/j.knosys.2020.106646

Cavazos, J. G., Phillips, P. J., Castillo, C. D., & O'Toole, A. J. (2021). Accuracy Comparison Across Face Recognition Algorithms: Where Are We on Measuring Race Bias? *IEEE Transactions on Biometrics, Behavior, and Identity Science*, *3*(1), 101–111. https://doi.org/10.1109/TBIOM.2020.3027269

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, *408*, 189–215. https://doi.org/https://doi.org/10.1016/j.neucom.2019. 10.118

Dumitrescu, E., Hué, S., Hurlin, C., & Tokpavi, S. (2022). Machine learning for credit scoring: Improving logistic regression with non-linear decision-tree effects. *European Journal of Operational Research*, *297*(3), 1178–1192. https://doi.org/https://doi.org/10.1016/j.ejor.2021.06.053

Fatima, A., Shahid, A. R., Raza, B., Madni, T. M., & Janjua, U. I. (2020). State-of-the-Art Traditional to the Machine- and Deep-Learning-Based Skull Stripping Techniques, Models, and Algorithms. *Journal of Digital Imaging*, *33*(6), 1443–1464. https://doi.org/10.1007/s10278-020-00367-5

Gill, S. S., Wu, H., Patros, P., Ottaviani, C., Arora, P., Pujol, V. C., Haunschild, D., Parlikad, A. K., Cetinkaya, O., Lutfiyya, H., Stankovski, V., Li, R., Ding, Y., Qadir, J., Abraham, A., Ghosh, S. K., Song, H. H., Sakellariou, R., Rana, O., … Buyya, R. (2024). Modern computing: Vision and challenges. *Telematics and Informatics Reports*, *13*, 100116. https://doi.org/https://doi.org/10.1016/j.teler.2024.100116

Halder, R. K., Uddin, M. N., Uddin, Md. A., Aryal, S., & Khraisat, A. (2024). Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. *Journal of Big Data*, *11*(1), 113. https://doi.org/10.1186/s40537-024-00973-y

Huang, C., Li, Y., Loy, C. C., & Tang, X. (2020). Deep Imbalanced Learning for Face Recognition and Attribute Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(11), 2781–2794. https://doi.org/10.1109/TPAMI.2019.2914680

Hussain, S. F. (2019). A novel robust kernel for classifying high-dimensional data using Support Vector Machines. *Expert Systems with Applications*, *131*, 116–131. https://doi.org/https://doi.org/10.1016/j.eswa.2019.04.037

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., & Zhao, L. (2019). Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications*, *78*(11), 15169–15211. https://doi.org/10.1007/s11042-018-6894-4

Khajenezhad, A., Bashiri, M. A., & Beigy, H. (2021). A distributed density estimation algorithm and its application to naive Bayes classification. *Applied Soft Computing*, *98*, 106837. https://doi.org/https://doi.org/10.1016/j.asoc.2020.106837

Kortli, Y., Jridi, M., al Falou, A., & Atri, M. (2020). Face Recognition Systems: A Survey. *Sensors*, *20*(2). https://doi.org/10.3390/s20020342

Mustafa A. D., & Mohsin A. A. . (2021). Machine Learning Applications based on SVM Classification A Review. *Qubahan Academic Journal*, *1*(2), 81–90. https://doi.org/10.48161/qaj.v1n2a50

Reddy, G. T., Reddy, M. P. K., Lakshmanna, K., Kaluri, R., Rajput, D. S., Srivastava, G., & Baker, T. (2020). Analysis of Dimensionality Reduction Techniques on Big Data. *IEEE Access*, *8*, 54776–54788. https://doi.org/10.1109/ACCESS.2020.2980942

Salehi, F., Abbasi, E., & Hassibi, B. (2019). The Impact of Regularization on High-dimensional Logistic Regression. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 32). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/ab49ef78e2877bfd2c2bfa738e459bf0-Paper.pdf

Schober, P., & Vetter, T. R. (2021). Logistic Regression in Medical Research. *Anesthesia & Analgesia*, *132*(2). https://journals.lww.com/anesthesia-analgesia/fulltext/2021/02000/logistic_regression_in_medical_research.12.aspx

Sharmila, Sharma, R., Kumar, D., Puranik, V., & Gautham, K. (2019). Performance Analysis of Human Face Recognition Techniques. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 1–4. https://doi.org/10.1109/IoT-SIU.2019.8777610

Sheik M. I. (2017). *olivetti*. https://www.kaggle.com/datasets/imrandude/olivetti

Zhao, F., Li, J., Zhang, L., Li, Z., & Na, S.-G. (2020). Multi-view face recognition using deep neural networks. *Future Generation Computer Systems*, *111*. https://doi.org/10.1016/j.future.2020.05.002

Zhao, S., Zhang, B., Yang, J., Zhou, J., & Xu, Y. (2024). Linear discriminant analysis. *Nature Reviews Methods Primers*, *4*(1), 70. https://doi.org/10.1038/s43586-024-00346-y

Zaw, H. T., Maneerat, N., & Win, K. Y. (2019). Brain tumor detection based on Naïve Bayes Classification. *2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, 1–4. https://doi.org/10.1109/ICEAST.2019.8802562