Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.98549765

# PERFORMANCE EVALUATION OF CLASSIFICATION ALGORITHMS FOR HANDWRITTEN DIGIT RECOGNITION: INSIGHTS FROM LOAD_DIGITS AND CUSTOM DATASET

**Rajinder Kaur[1], Anantdeep[2]**
Department of Computer Science and Engineering, Punjabi University, Patiala, India
[1]rkkhinda1999@gmail.com, [2]boparai.dolly@gmail.com

**ABSTRACT**

***Purpose:*** The purpose of this research is to evaluate six classification algorithms for handwritten digit recognition. The study aims to assess their accuracy and effectiveness using established and custom datasets. It provides valuable insights for algorithm selection, guiding researchers and practitioners in handwriting recognition.

***Methodology:*** The study utilises a comparative analysis approach to evaluate the performance of the classification algorithms. Two datasets are employed: the established load_digits dataset and a custom dataset of handwritten digits. Confusion matrices are used to measure accuracy, and the algorithms are assessed based on their performance on the respective datasets.

***Findings:*** The findings of this study reveal that KNN exhibits the highest accuracy in recognising handwritten digits, achieving remarkable accuracy scores of 0.99 on the custom dataset and 0.95 on the load_digits dataset. SVM closely follows with accuracies of 0.98 on the custom dataset and 0.96 on the load_digits dataset. Additionally, SGD demonstrates strong performance with accuracies of 0.96 and 0.90 on the respective datasets. GNB, DT, and RF show comparable performance levels, with DT recording the lowest accuracy.

***Originality/Value:*** This research provides valuable insights and originality in evaluating multiple classification algorithms for handwritten digit recognition. It aids researchers and practitioners in algorithm selection for optimal performance, utilising both established and custom datasets. The findings contribute to the existing knowledge base in handwriting recognition and suggest future directions for algorithm refinement and exploration of ensemble methods or deep learning approaches.

**Keywords:** *SVM, KNN,Gaussian Naive Bayes, Random forests, Stochastic Gradient Descent*
**Article Type:** Research paper

## 1. INTRODUCTION

The current era of technology is marked by tremendous advancements in the field of machine learning, which has enabled efficient automation of arduous tasks that were once considered time-consuming and laborious. One such task is the automated recognition of handwritten digits, which has been made possible by the application of machine learning algorithms. It is possible to identify and classify handwritten digits with a higher degree of accuracy, which is particularly important in applications such as automated check processing or security. In this research paper, we present a systematic study of the efficacy of such algorithms in recognising handwritten digits. As part of our study, we analyse the performance of the algorithms on the

existing sklearn dataset of handwritten digits, as well as a new dataset created by us, which consists of handwritten digits drawn on MS Paint.

The use of machine learning algorithms in recognising handwritten digits has been extensively studied and documented in the past, with various approaches and techniques employed for obtaining satisfactory performance. However, the complexity of the task remains, with the variation in penmanship, writing style, and individual handwriting making it difficult to develop a generalisable algorithm. In this regard, the use of existing datasets such as the load_digits dataset from sklearn that consists of 1,797 8x8 images of handwritten digits, where each image is a grayscale representation of a single digit. The images have been preprocessed to have the same dimensions, so that all the digits are of equal size and shape. Additionally, the dataset includes labels for each image so that the machine learning algorithms can be trained to accurately recognize the digits. Furthermore, the development of our own dataset, consisting of handwritten digits drawn on MS Paint, will allow us to incorporate a variety of writing styles and individual handwriting into our study.

Our study is divided into two parts: first, we analyse the performance of the machine learning algorithms on the existing sklearn dataset of handwritten digits, and second, we employ the same algorithms to our newly created dataset of handwritten digits drawn on MS Paint. In order to evaluate the performance of the algorithms, we employ the wellknown metrics of accuracy, precision, and recall. Additionally, we compare the performance of the algorithms on the two datasets, and analyse the differences in the results. By doing so, we aim to identify the best algorithms for recognising handwritten digits and to provide valuable insights into the task of automated recognition of handwritten digits.

## 2. LITERATURE SURVEY

[1] In this paper, HCR systems have been developed to help individuals and organizations to convert Handwriting notes into electronic copies which can be communicated and stored electronically in a more efficient and robust manner. Research in HCR has focused on the development of Artificial Neural Networks (ANNs) as a viable approach for recognising Handwriting characters. In particular, convolutional neural networks (CNNs) were used to process and recognize handwritten characters and digits. Architecture of CNN is presented that will be used in OCR. The architecture consists of several layers, beginning with the input layer and ending with the output layer. In between the input and output layers are convolutional layers, pooling layers, and additional convolutional layers. Each layer has its own specific purpose, helping to interpret the input data and generate an output. CNNs, which are a part of the field of Artificial Intelligence, are modelled on the human visual system, allowing for the OCR system to be sensitive to different features of objects.

The methodology and design of OCR systems has been widely studied, with a focus on the phases of handwriting recognition such as acquisition, digitisation, segmentation, feature extraction, and recognition. In the image acquisition and digitisation phase, an image of the text is captured and converted into a digital signal. The preprocessing phase involves cleaning up the image to make it easier to process. This includes noise reduction, contrast enhancement, and de-skewing. The segmentation phase involves dividing the image into meaningful segments

such as lines, words, and characters. The feature extraction phase involves extracting the features from the image, such as edges, curves, and shapes. Lastly, the recognition phase involves comparing the extracted features with known characters and assigning labels to the characters. In this phase, the neural network is utilised for character classification and recognition from the image.
Various testing methods have been used to evaluate the accuracy of the systems, including unit testing, integration testing, GUI testing, and validation testing.
This dataset contains 21,000 training and 21,000 testing gray-scale images of handwritten digits (from 0 to 9), each of which is of size 28 x 28 pixels. Every image is standardised and centred, and each pixel represents a single structure of the digit. Decision tree classification model is used for training purpose. Accuracy of 83.4% is achieved in which digit 1 got the higher accuracy of 93.73% and digit 0 have least accuracy of 83.56%.
The aim of the research is to develop a Handwriting character recognition system that can be used for Handwriting notes, and to demonstrate the effectiveness of neural networks for Handwriting character recognition.

[2] In this paper, an efficient system for recognising medieval handwritten Gurmukhi characters has been proposed. Presegmented characters of 150 documents of Gurmukhi manuscripts from different books were scanned to create a database consisting of 1140 samples of 43-classess (25 samples of each class). For recognition, feature extraction techniques such as zoning, discrete cosine transformations, and gradient features have been used. In addition, four classifiers (k-NN, SVM, Decision Tree, Random Forest) and combinations of these four classifiers with voting scheme have been employed for classification. Adaptive boosting and bagging were explored to improve the recognition results, and a new state of the art for the recognition of medieval handwritten Gurmukhi manuscripts was achieved. Using the proposed framework, a maximum recognition accuracy of 95.91% was attained using adaptive boosting and a combination of the four different classifiers considered in this paper.

[3] The objective of this paper is to find an effective software that can precisely recognise characters. Training and testing data consists of pre-processed gray-scale images of hand-written digits. The images are 28 pixels by 28 pixels, totalling 784 pixels. Each pixel has a value ranging from 0 to 255. The images have been processed with MATLAB and converted into Comma-separated values (CSV) files. Classification of the dataset was performed using WEKA. A comparison of Hoeffding Tree, Decision tree and Random forest methodologies is conducted on a set of benchmarks to determine which is most effective for pattern recognition. Decision tree performs poorly in terms of accuracy, precision, recall, F-measure and Kappa statistic as compared to Random Forest and Hoeffding tree. Random Forest has the highest accuracy, precision, recall, F-measure and Kappa statistic values, while Hoeffding tree has a slightly lower accuracy but is the fastest to build. Therefore, if time is a constraint, then Hoeffding tree is the most effective with an accuracy of 73%.

[4] This paper proposes a hybrid model of CNN and SVM for the classification of handwritten digits from the MNIST dataset. The proposed system combines the best features from both SVM and CNN classifiers. A 5x5 kernel/filter is used to extract the most distinctive features from the raw input images. In this model, the SVM is used as a binary classifier to

replace the softmax layer of the CNN. The proposed system was evaluated for the MNIST dataset and achieved a classification accuracy of 99.28%.

[5] This paper explored the usage of decision tree classification to recognize handwritten digits from the standard Kaggle digits dataset. The model was evaluated against each digit from 0 to 9, with 42000 data sets being split into half for training and 21000 for testing. The classifier was trained to accurately identify the class of a given handwritten digit. Results showed that the model had an accuracy of 83.4% with the digit 1 being recognised with the best accuracy, while the digit 8 was recognised with the least accuracy, as it resembled many other digits.

[6] In this study, handwriting digit recognition process was conducted using different algorithms, namely Support Vector Machine (SVM), Decision Tree, Random Forest, Artificial Neural Networks (ANN), K-Nearest Neighbor (KNN) and K-Means Algorithm. The MNIST dataset was used for the training and testing of the system. All tests were performed on PyCharm and the accuracy of the various algorithms were compared. The accuracy of SVM was found to be 90%, KNN 98%, and the Decision Tree algorithm had the lowest accuracy at 87%.

[7] This paper presents an Optical Character Recognition (OCR) system for the recognition of Printed text with noise. The system consists of four main steps: pre-processing of the text image, segmentation of each character, feature extraction using zoning-based technique and classification. The system is implemented using MATLAB 2016a and is currently under development. The experimental datasets used in this work are CHARS74K and ICDAR2003. CHARS74K contains a total of over 74K images, while ICDAR2003 contains 249 images with 5370 characters and 1106 words. The system is evaluated using k-Nearest Neighbor algorithm, and Euclidean Distance is used for the comparison of similarity. The proposed character recognition system achieves a high recognition rate.

[8] This paper proposes a method for efficient hand-written digit detection that is off-line in the python language platform. The main goal of this study is to ensure reliable and precise digit recognition. Various machine learning algorithms have been used for the identification of digits using MNIST including Support Vector Machine (SVM), Multilayer Perceptron, Decision Tree, Naïve Bayes, K-Nearest Neighbor (KNN) and Random Forest classifiers. The Decision Tree approach relies on the Hunt's algorithm to create multiple decision trees based on the C4.5 simulation algorithm of the existing DT. The SVM algorithm creates an optimal algorithm by a 'training' process in which training data is used to create an algorithm capable of distinguishing between classes. The Random Forest classifiers predict by collecting the ensemble's predictions through voting for classification by dominance. The Naïve Bayes model is based on two major simplifying assumptions that predictive attributes are conditionally self-reliant given the class. The KNN model is trained on training samples, then similar samples are identified in test data. The SVM recognition method has achieved the highest accuracy of 95.88%. This was an initial effort to make it simpler to identify hand-written digits without using any common methods for classification.

[9] This paper explores the use of three classification models to recognize handwritten digits from the MNIST dataset. Python, OpenCV, and sklearn were used to run the classification and read the dataset. The Histogram of Oriented Gradients (HOG) feature detection was used to extract the features from the MNIST dataset. The SVM and MLP Neural Network classifiers were tested using the performRecognition.py script. An image was loaded and preprocessed to be tested by the classifier by converting it to grayscale and applying Gaussian filtering. The image was then thresholded and contours were found in the image. Rectangles were then obtained to contain each contour. The KNN classifier showed the highest accuracy of 99.26% when the value of k was set to 15. The results showed that both the SVM and KNN classifiers were able to correctly predict the dataset, while the MLP Neural Network made some mistakes in predicting the number 9. This could be because the SVM and KNN classifiers were able to predict directly from the feature extraction while the MLP Neural Network required additional processing.

[10] This paper has analysed various approaches for recognition of handwritten Devanagari characters. For this purpose, approximately 80 papers have been collected and 42 papers have been taken for technical analysis. The selected papers belong to different kinds of journals such as Elsevier, Springer, IEEE, Conference, and others. The papers have been categorised into eight classes such as Convolutional Neural Network (CNN), Support Vector Machine (SVM), Fuzzy Model, K-Nearest Neighbor (KNN), Bayes Theorem, Principal Component Analysis (PCA), Gaussian Distance, and Others. The paper also discusses various preprocessing techniques, classifiers used, and recognition techniques for handwritten Devanagari character recognition. Deep learning techniques are commonly used for character recognition due to their high tolerance and low error rate. This survey paper helps researchers and developers to understand various techniques and the way they are implemented for recognition. The image is scanned first, and then the data is preprocessed. The preprocessing involves techniques such as Binarization, removal of noise and Normalisation. After that, features are extracted from the preprocessed data so that the relevant data is further used to train the model. Various classification techniques are applied, and the best approach is considered based on accuracy. Multilayer perceptron models are used to compare the input image with the trained set, to get higher accuracy. This paper has focused on various approaches, algorithms and techniques. The study concludes that SVM and CNN classifiers both provide better results with an accuracy of 99.6% and 98.47% respectively.

## 3. BACKGROUND KNOWLEDGE OF LOAD_DIGIT DATASET AND CLASSIFICATION ALGORITHMS

The *load_digits* dataset from sklearn is an open source dataset of handwritten digits that is used to train and test algorithms for handwritten digit recognition. The dataset is composed of 1797 samples and each sample consists of 8x8 grayscale images, representing a single digit from 0 to 9. The images are grayscale, with each pixel having a value between 0 and 16, representing different shades of grey. Each image is labeled with the corresponding digit that it represents, making it easy to use the dataset for supervised learning tasks. For handwriting recognition, the data from the load_digits dataset can be used to train a machine learning model. The model can learn the patterns of each digit from the images. It can also learn the differences between each

digit. Once the model is trained, it can then be used to classify an unknown image as one of the ten digits.

Let's plot the sixth image of the handwritten digits from the images using matplotlib to see how it looks.

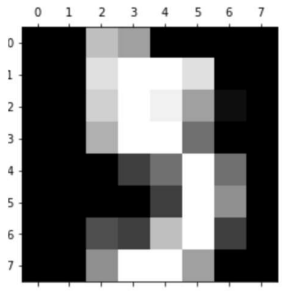



*Fig 1: Image from load_digits dataset using matplotlib*

*(Source : Author's own work)*

Dataset is split into a training and testing dataset. The training dataset consists of 50% of the images in the dataset, which is used to fit the machine learning model. The testing dataset consists of the remaining 50% of the images, which is used to evaluate the performance of the model.

The evaluation of the model is done using confusion matrix.

*Confusion Matrix:* A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of correct and incorrect predictions made by the model. The matrix is made up of four quadrants: true positives, false positives, true negatives, and false negatives. The true positives are the number of correctly predicted labels. The false positives are the number of incorrectly predicted labels. The true negatives are the number of correctly predicted labels that should have been predicted correctly. The false negatives are the number of incorrectly predicted labels that should have been predicted correctly.

### 3.1 Support Vector machine (SVM)

The Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks[5]. It is based on learning techniques that analyse data and recognise patterns in order to make predictions[1]. SVM is an effective tool for solving complex problems and has been successfully applied to a wide range of tasks such as image classification, text categorisation, and bioinformatics. SVMs are particularly popular in applications where the data is non-linearly separable, such as pattern recognition and handwriting recognition. SVMs are able to classify data by constructing a hyperplane in the feature space that best separates the data, allowing for efficient classification[1,4,10]. In handwriting recognition, images of handwritten digits are used as input for the SVM. These images are then pre-processed in order to extract their features, such as the stroke width, the shape of the strokes, and the size of the letter. The features are then used to train the SVM. This training process consists of finding the optimal hyperplane that separates the data into different classes. The hyperplane is determined by maximising the margin, which is the distance between the closest points in the training set and the hyperplane.

Once the SVM has been trained, it can be used to classify new data points. When a new data point is presented, such as a digit 5, the SVM will first pre-process the image in order to extract its features. The features are then compared to the training set of data points and the hyperplane constructed during the training process. By calculating the distance between the data point and the hyperplane, as well as the distance to the closest point in the training set, the SVM is able to determine which class the data point belongs to. If the distance to the hyperplane is greater than a certain threshold, the data point is assigned to the class of the hyperplane. Otherwise, the data point is classified as belonging to the class of the closest point in the training set. In this case, if the distance to the hyperplane is greater than the threshold, the data point will be classified as a 5.

### 3.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a popular optimisation algorithm used in machine learning. It is used to efficiently optimise a cost function by taking small steps in the direction of the gradient with respect to the cost function. SGD has been applied to a wide range of problems, including classification and regression, and is commonly used in the training of neural networks. In particular, SGD has been used for handwritten digits recognition tasks, where it can be used to train a deep neural network to classify handwritten digits.

In the handwritten digits recognition task, SGD is used to update the weights of the neural network in order to minimise the cost function. The cost function is typically a cross-entropy cost function, which measures the difference between the predicted output of the neural network and the desired output. At each iteration, SGD takes a step in the direction of the gradient with respect to the cost function, which is calculated using the backpropagation algorithm. The step size is determined by a learning rate, which is a parameter that controls the size of the step taken by SGD. By taking small steps in the direction of the gradient, SGD is able to efficiently find the minimum of the cost function, thereby optimising the neural network for the handwritten digits recognition task.

By taking small steps in the direction of the gradient with respect to the cost function, SGD is able to efficiently find a minimum of the cost function and optimise the neural network for the task.

### 3.3 Gaussian Naive Bayes

Gaussian Naive Bayes is a classification algorithm that is based on the Bayes Theorem and the assumption of independence between the features. It is widely used in many applications, such as spam filtering and text classification. In this paper, we will discuss its application in handwritten digits recognition.

The traditional approach to this task is to use a Support Vector Machine (SVM) or a Neural Network. However, these methods require large amounts of data and are computationally expensive. Gaussian Naive Bayes is an alternative approach to this problem. It is a simple yet powerful tool that can be used for handwritten digits recognition. The basic idea behind it is that it assumes that the data points are independent of each other. This means that the classifier does not need to learn the correlations between the features, but instead can directly assign class labels based on the feature values.

The Gaussian Naive Bayes classifier works by calculating the probability distribution of the data points. It then computes the probability of a given data point belonging to each class. The class with the highest probability is then assigned as the label for that data point. In the context of handwritten digits recognition, the Gaussian Naive Bayes classifier can be used to identify which digit is present in an image. It can be used to classify the digits in an image by making use of the features such as the size, shape and orientation of the digits.

For example, when trying to classify a 6, the classifier will calculate the probability of a given image belonging to the class of 6s. It will then compare this probability to the probability of the image belonging to other classes, such as 5s, 7s, and so on. The class with the highest probability is then assigned as the label for that data point. In this case, if the probability of the image belonging to the 6s class is higher than the probabilities of other classes, then the classifier will assign the label of 6 to that image.

It is a simple yet powerful tool that can be used to classify patterns in digital images.

### 3.4 Decision trees

Decision trees are a type of supervised learning algorithm used for classification and regression tasks. In the context of handwritten digits recognition, decision trees can be used to classify an input image of a digit into one of the 10 classes (0-9). The decision tree works by breaking down a problem into smaller and smaller subproblems until a solution is found[2]. The decision tree starts at the root node which contains the entire set of data, and then splits the data based on the most predictive feature. It then recursively splits the data further until it reaches a point where the data can no longer be split. The end result is a tree of nodes with each leaf node representing a class label[3].

To illustrate this process, consider an example of a decision tree for recognising the digit 1. The root node of the tree would contain the entire dataset of images of the digit 1. The tree would then split the data based on a certain criteria, such as the presence of an outline. If the data contains an outline, the tree would split the data into two branches, one branch containing images with an outline and the other branch containing images without an outline. The tree would then continue to split the data further based on other criteria such as the presence of a line connecting the top and bottom of the image. The tree would continue to split the data until it reaches a point where the data can no longer be split. The end result would be a tree of nodes and each leaf node would represent a class label of the digit 1.

### 3.5 Random Forest

Random forest is an ensemble machine learning algorithm that can be used for both classification and regression tasks. It is a supervised learning method which uses decision trees as its base estimators. The random forest algorithm creates multiple decision trees from randomly selected subsets of the training data, then averages their predictions to form an overall prediction[3]. This approach reduces variance in the estimation process, leading to more accurate results than a single decision tree alone.

In handwritten digits recognition task, Random Forest works by building a set of Decision Trees using different subsets of the training data and different parameters such as maximum depth or number of features per node. Each Decision Tree will make predictions on unseen test data points based on what it has learned from the training dataset. The final prediction given by

Random Forest is calculated by taking an average over all these individual decisions made by each tree in the forest.

For example, when recognising digit 7 in hand-written images: firstly, we need to extract features like stroke widths and angles between strokes from each image; Secondly we use this feature vector along with labels (0 - 9) indicating which digit it belongs to as input for our Random Forest classifier; Finally after some iterations through all trees in our model we get a probability score for every label (0 - 9). In this case if 7 has got highest probability score among other numbers then our model predicts that particular image contains 7 digit.

### 3.6 K-Nearest Neighbour (KNN)

K-Nearest Neighbour (KNN) is a supervised machine learning algorithm used for classification and regression problems. It works by finding the closest training examples in its feature space to the new data point or query, and then labelling it with the most frequent class among those neighbours[9]. KNN has been widely used in handwritten digits recognition due to its simple nature and good performance.

In handwritten digits recognition, each digit is represented as an image which can be divided into several features such as pixel intensity values of individual pixels, size of the object etc. When a new input image arrives at the system, these features are extracted from it and compared with all other images present in its database using some distance metric such as Euclidean Distance or Manhattan Distance[7]. The ‘k’ nearest neighbours of this input image are identified based on their distance from it; here ‘k’ represents number of neighbours taken into consideration for prediction purpose[6,8]. Finally, majority voting takes place among them to assign label/classification to that particular input image i.e., if more than half of k nearest neighbour belongs to same class then that particular class will be assigned as label otherwise some special measures needs to be taken depending upon problem statement like assigning random labels etc.

For example: Consider 8 images belonging to different classes stored in database along with their respective feature vectors [x1 , x2....xn] . Now when we provide one more sample vector say[y1 , y2.....yn], firstly we calculate euclidean distances between this vector and each vectors stored in our dataset(if any other distance metric you want you can use). After calculation suppose 1st 3 smallest distances come out from our 8th sample data points so now majority voting take place amongst these 3 points hence result will come out 8 because 2 out of three belong same class i.e.,8th sample data point have maximum probability being classified as 8.

In conclusion, KNN is a simple yet powerful algorithm that can be used for handwritten digits recognition due to its high accuracy and low computational complexity. It uses majority voting approach among the k neighbours of given data point to assign appropriate label/classification to it.

## 4. METHODOLOGY

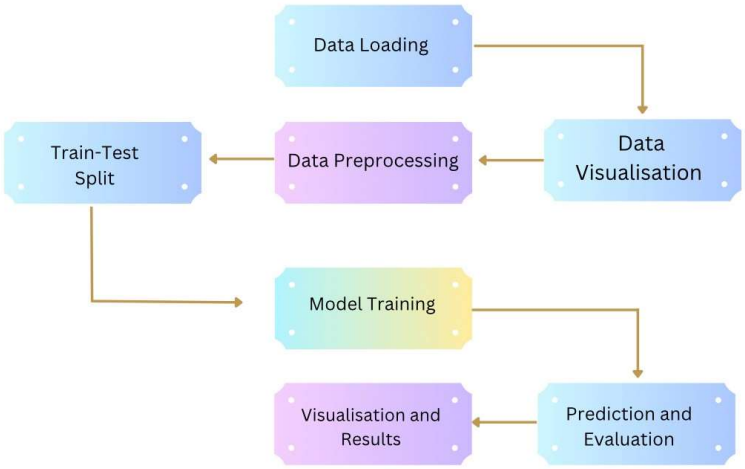



*Fig 2: Flowchart of handwritten digits recognition (Source: Author's own work)*

In the research paper, the methodology employed includes the usage of multiple classifiers and two datasets, one of which is a custom dataset. Here's an overview of the methodology used:

*Data Loading:* The code begins by importing the necessary libraries and loading the digit dataset. The dataset contains grayscale images of hand-drawn digits (0-9).

*Data Visualisation:* The code uses 'matplotlib' to display an example image from the dataset using 'plt.matshow()'. This allows visual inspection of the input data.

*Data Preprocessing:* The dataset is then preprocessed by reshaping the images using 'digits.images.reshape ( )' into a 2D array, where each row represents an image. This transformation allows the images to be used as input features for the SVM classifier.

*Train-Test Split:* The data is split into training and testing sets using train_test_split ()' from 'sklearn.model_selection'. This function randomly divides the data, with a test size of 50%, into 'X_train and X_test' for input features, and 'y_train and y_test' for target labels.

*Model Training:* Multiple classifiers are selected and trained on the training sets of both datasets. SVM, Stochastic Gradient Descent (SGD), Gaussian Naive Bayes, K-Nearest Neighbour (KNN), Decision trees, Random forests, . Each classifier is initialised, trained using the respective training set, and tuned with appropriate hyperparameters.

*Prediction and Evaluation:* The trained classifiers are used to make predictions on the respective test sets. The predictions are then evaluated using performance metrics such as accuracy, precision, recall, F1 score, or confusion matrix.The evaluation metrics for each classifier are calculated using 'metrics.classification_report()'.

*Visualisation of Results:* The code utilises matplotlib to visualise the results. It can display sample images from the training set alongside the corresponding predictions made by each classifier. This allows for a visual understanding of the classifiers' performance and can aid in explaining the results in the research paper.

By utilising multiple classifiers and employing different datasets, the research paper showcases a broader exploration of classification techniques and evaluates their performance on distinct datasets. It provides a comprehensive analysis of various classifiers, their effectiveness, and their suitability for different types of data.

## 5. RESULTS AND DISCUSSIONS

This research paper explored the performance of six algorithms, Support Vector Machines (SVM), Stochastic Gradient Descent (SGD), Gaussian Naive Bayes, Decision trees, Random Forest and K Nearest Neighbors (KNN), on two datasets: the existing load digits dataset and a custom dataset of handwritten digits created using MS Paint. We created the dataset using a Python script that captures screenshots of handwritten digits and processes them using the OpenCV library. The script captures 1000 screenshots of handwritten digits (100 for each digit) and saves them to a folder. Each screenshot is then processed by converting it to grayscale, applying a Gaussian blur, and resizing it to 28×28 pixels using the OpenCV library. The resulting images are then flattened into binary arrays by iterating over each pixel and adding it to the array as a binary value (O or 1). The dataset is saved as a CS file with 785 columns, where the first column represents the label (0-9) and the remaining columns represent the flattened image data. Let's visualise specific image at index 641.

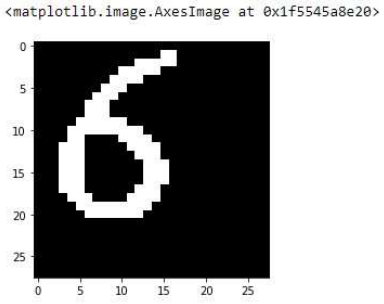



*Fig 3: Image from dataset (Source: Author's own work)*

I. Support vector machine (SVM)

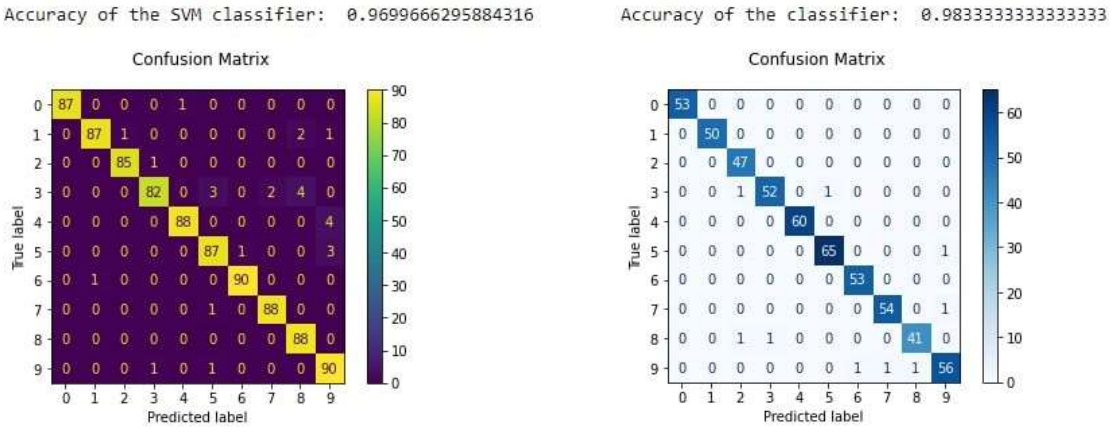
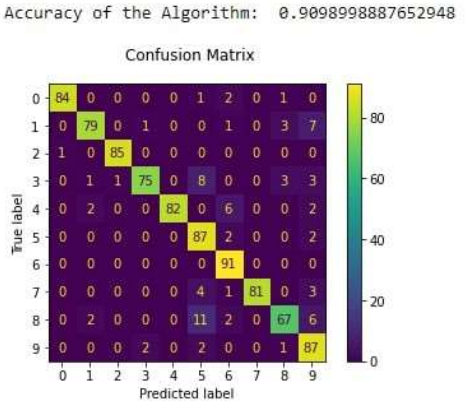


*Fig 4: load_digits dataset*
*(Source : Author's own work)*

*Fig 5: Custom dataset*
*(Source: Author's own work)*

The first algorithm used the load_digits dataset from the sklearn library, while the second algorithm was developed on a custom dataset of 1008 digits. The first algorithm utilised a radial basis function (RBF) kernel with a regularisation parameter of C=1 and gamma=0.001, while the second algorithm used a polynomial kernel with degree three and a regularisation parameter of C=10. The second algorithm achieved a higher accuracy of 0.983, compared to the accuracy of 0.966 achieved by the first algorithm.

## II. Stochastic Gradient Descent (SGD)



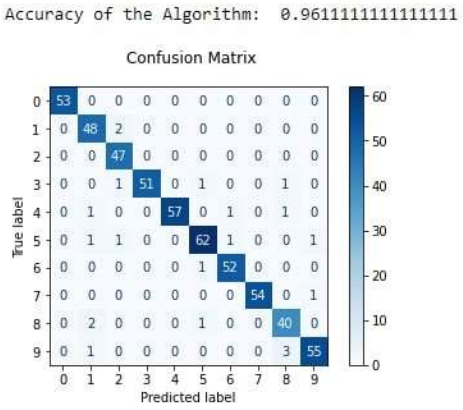




*Fig 6: load_digits dataset*
*(Source : Author's own work)*

*Fig 7: Custom dataset*
*(Source : Author's own work)*

Both algorithms used the same loss function ("hinge") and penalty ("l2") with a maximum of 29 iterations, but they used different datasets. Our results show that Algorithm 2 outperformed Algorithm 1, achieving an accuracy of 0.96 compared to 0.90.

## III. Gaussian Naive Bayes (GNB)

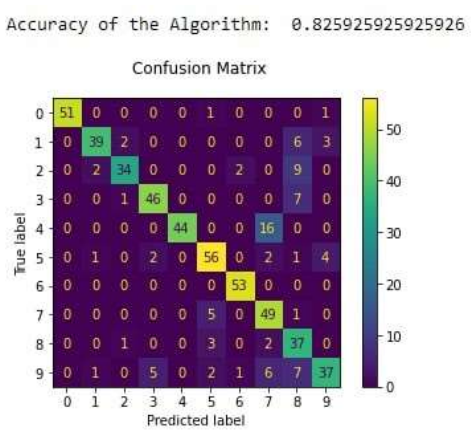



*Fig 8: load_digits dataset (Source : Author's own work).*

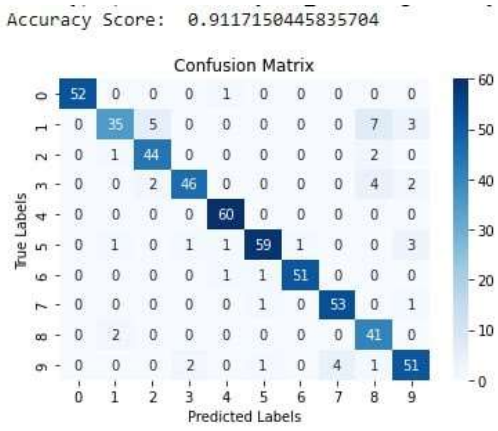



*Fig 9: Custom dataset (Source : Author's own work)*

Both algorithms were tuned using grid search to find the best hyperparameters. The results indicate that the second algorithm achieved higher accuracy (91.1%) compared to the first algorithm (82.5%).

## IV. Decision trees (DT)

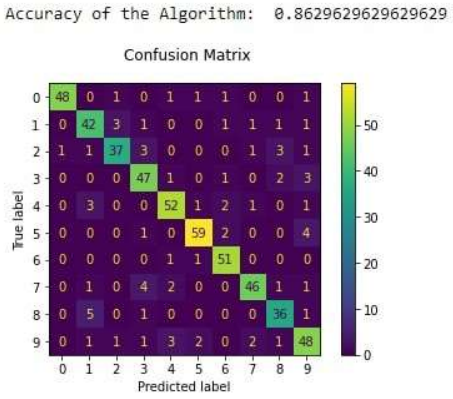



*Fig 10: load_digits dataset (Source : Author's own work).*

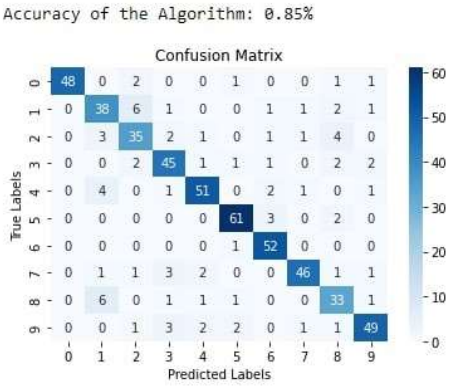



*Fig 11: Custom dataset (Source : Author's own work)*

The decision tree algorithm works by splitting the data into subsets based on the features of the dataset. It uses an information gain measure to determine the most informative features and their thresholds for splitting the data. The algorithm continues to recursively split the subsets until a stopping criterion is met. The accuracy of the algorithm is determined by comparing the predicted labels to the true labels of the test set.

## V. Random Forest (RF)

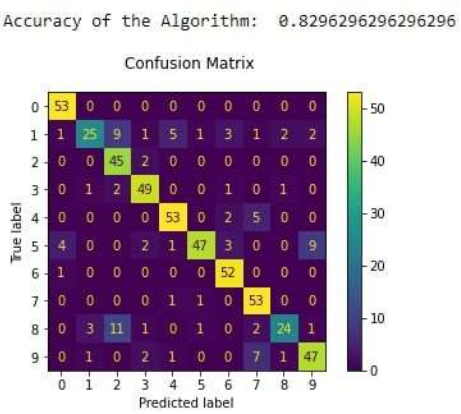



*Fig 12: load_digits dataset. (Source : Author's own work)*

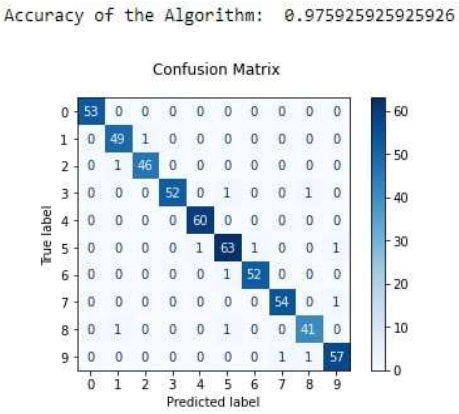
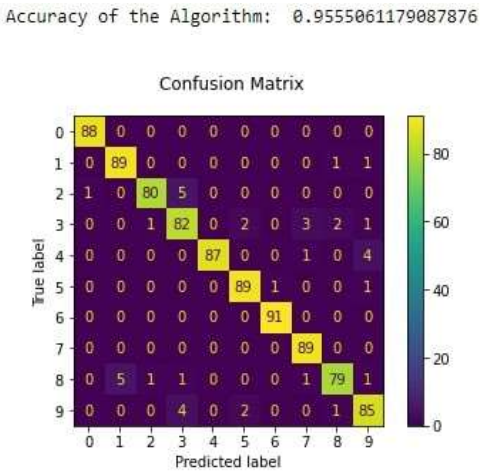



*Fig 13: Custom dataset (Source : Author's own work)*

The first implementation employs default hyperparameters, providing a baseline performance for the algorithm. It focuses on simplicity and ease of implementation, utilising a maximum depth of 2.

In contrast, the second implementation explores alternative hyperparameter settings to enhance the classifier's accuracy. By increasing the maximum depth to 260, utilising 300 decision trees (n_estimators), and employing the "entropy" criterion for splitting, this implementation aims to improve the model's ability to capture complex relationships within the dataset and make more accurate predictions.

## VI. K-Nearest Neighbour (KNN)



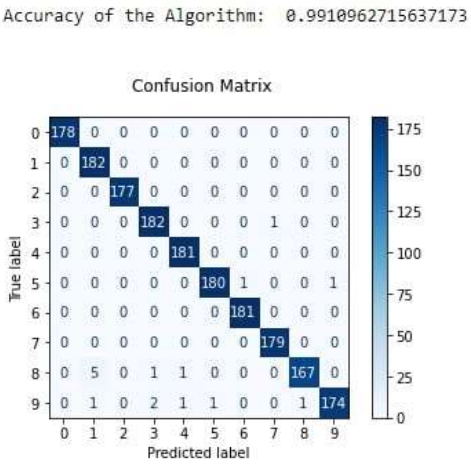



*Fig 14: load_digits dataset (Source : Author's own work)*

*Fig 15: Custom dataset (Source : Author's own work)*

The first implementation uses a basic KNN classifier without preprocessing or hyperparameter tuning, providing predictions and evaluation metrics for digit classification. On the other hand, the second implementation applies preprocessing techniques like scaling and dimensionality reduction with PCA, along with hyperparameter tuning using GridSearchCV, resulting in improved performance. It selects the best KNN classifier, achieving higher accuracy and offering comprehensive evaluation metrics.

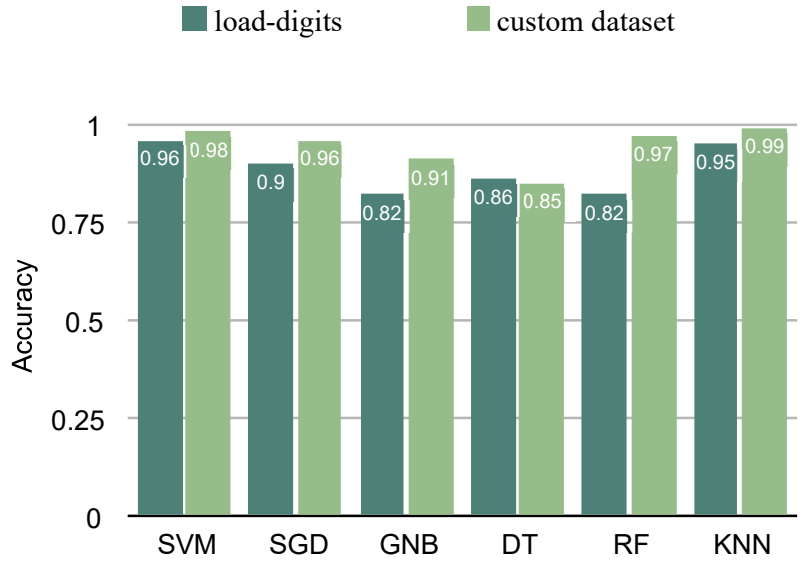



*Fig 16 : The graph shows that the custom dataset yields higher accuracy values for most of the machine learning algorithms tested.*

*(Source : Results are based on the author's own analysis)*

**FUTURE SCOPE**

The potential of custom datasets to yield higher accuracy values for most machine learning algorithms tested against the load_digits dataset for handwriting recognition is vast. With increased access to data, more sophisticated feature engineering and better algorithms, accuracy levels can be pushed higher. Firstly, custom datasets should have a greater number of samples than the load_digits dataset. This will allow for more complex models to be trained and increase the overall accuracy of the model. Also, more valuable features can be identified and exploited, such as the shape and size of the characters, the angle at which they are written, and the pressure used to write them. Furthermore, the use of ensemble methods such as boosting and bagging can be employed to increase the accuracy of the model. These methods enable multiple weak

base classifiers to be combined to form a powerful ensemble model, which is often more accurate than any of the individual models. Finally, the use of deep learning models such as convolutional neural networks can be employed to further increase the accuracy of the model. Deep learning models are capable of extracting more complex features from the data, such as the texture of the characters, which can further increase the accuracy of the model.

## CONCLUSION

This research paper has provided an analysis of six popular classification algorithms on two datasets: the load_digits dataset and a custom dataset created on Microsoft Paint with 1008 digits for all digits. Based on the results, k-Nearest Neighbours (KNN) was found to be one among top performing algorithms for classifying handwritten digits, with an accuracy of 0.955 on the load_digits dataset and 0.99 on the custom dataset. Support Vector Machines (SVM), Stochastic Gradient Descent (SGD) also had relatively similar high accuracy levels, with SVM having an accuracy of 0.96 on the load_digits dataset and 0.98 on the custom dataset while SGD had an accuracy of 0.90 on the load_digits dataset and 0.96 on the custom dataset. Gaussian Naive Bayes (GNB), Decision Trees (DT), Random Forests (RF) all had relatively lower accuracy levels, with DT having the lowest accuracy of 0.86 on the load_digits dataset and 0.85 on the custom dataset. In conclusion, this research paper has demonstrated that KNN is considered as one among top performing algorithms for classifying handwritten digits alongside SVM & SGD.

## Funding statement

The authors did not receive any financial assistance for the research, authorship or publication of this work.

## ETHICS STATEMENT

The researcher is committed to conducting research ethically and with integrity, adhering to all relevant ethical guidelines and protecting the confidentiality of any information collected. Honesty has been maintained in all scientific communications, and all data, results, methods and procedures, and publication status have been accurately reported without any fabrication, falsification, or misrepresentation.

## REFERENCES

1. Aqab, S. and Tariq, M.U., 2020. Handwriting recognition using artificial intelligence neural network and image processing. *International Journal of Advanced Computer and Application (IJACSA)*, *11*(7), pp.137-146.
2. Kumar, M., Jindal, S.R., Jindal, M.K. and Lehal, G.S., 2019. Improved recognition results of medieval handwritten Gurmukhi manuscripts using boosting and bagging methodologies. *Neural Processing Letters*, *50*, pp.43-56.
3. Lavanya, K., Bajaj, S., Tank, P. and Jain, S., 2017, June. Handwritten digit recognition using hoeffding tree, decision tree and random forests—A comparative approach. In *2017 international conference on computational intelligence in data science (ICCIDS)* (pp. 1-6). IEEE.

4. Ahlawat, S. and Choudhary, A., 2020. Hybrid CNN-SVM classifier for handwritten digit recognition. *Procedia Computer Science*, *167*, pp.2554-2560.
5. Assegie, T.A. and Nair, P.S., 2019. Handwritten digits recognition with decision tree classification: a machine learning approach. *International journal of electrical and computer engineering (IJECE)*, *9*(5), pp.4446-4451.
6. KARAKAYA, R. and KAZAN, S., 2021. Handwritten digit recognition using machine learning. *Sakarya university journal of science*, *25*(1), pp.65-71.
7. Barnouti, N.H., Abomaali, M. and Al-Mayyahi, M.H.N., 2018. An efficient character recognition technique using K-nearest neighbor classifier. *International Journal of Engineering & Technology*, *7*(4), pp.3148-3153.
8. Gope, B., Pande, S., Karale, N., Dharmale, S. and Umekar, P., 2021. Handwritten digits identification using MNIST database via machine learning models. In *IOP Conference Series: Materials Science and Engineering* (Vol. 1022, No. 1, p. 012108). IOP Publishing.
9. Hamid, N.A. and Sjarif, N.N.A., 2017. Handwritten recognition using SVM, KNN and neural network. *arXiv preprint arXiv:1702.00723*.
10. Agrawal, M., Chauhan, B. and Agrawal, T., 2022. Machine Learning Algorithms for Handwritten Devanagari Character Recognition: A Systematic Review. *vol*, *7*, pp.1-16.