Journal of Science Engineering Technology and Management Science
Volume 02, Issue 06, June 2025 *ISSN: 3049-0952*
www.jsetms.com *DOI:10.63590/jsetms.2025.v02.i06.pp148-163*

EMBARKING ON THE MNIST HANDWRITTEN DIGIT CLASSIFICATION CHALLENGE: A BEGINNER'S JOURNEY INTO IMAGE DATA ANALYSIS

Dr SK Mahboob Basha[1], Sri Harsha Dixit[2] , SV.Swathi[2], N.Shashank[2], K.Prashanth[2]
[1]Professor, [2]UG Student, [1,2] Department of Computer Science and Engineering(DS)
Sree Dattha Group of Institutions, Sheriguda, Hyderabad, Telangana

To Cite this Article
Dr SK Mahboob Basha, Sri Harsha Dixit, SV.Swathi, N.Shashank, K.Prashanth, "Embarking On The Mnist Handwritten Digit Classification Challenge: A Beginner's Journey Into Image Data Analysis", Journal of Science Engineering Technology and Management Science, Vol. 02, Issue 06, June 2025,pp:148-163, DOI: http://doi.org/10.63590/jsetms.2025.v02.i06.pp148-163
Submitted: 20-04-2025 *Accepted: 27-05-2025* *Published: 06-06-2025*

ABSTRACT
Handwritten digit classification is a fundamental problem in the field of machine learning and computer vision. It involves recognizing handwritten digits (0-9) from images. The MNIST dataset, consisting of 28x28 pixel grayscale images of handwritten digits, is a widely used benchmark for this task. In this we embark on the MNIST Handwritten Digit Classification Challenge, aiming to delve into image data analysis and explore different machine learning algorithms for accurate classification. Traditionally, handwritten digit classification was done manually by human experts. With the advent of computer vision and machine learning, automated systems have been developed to tackle this task. However, traditional systems often relied heavily on handcrafted features and simplistic algorithms, which limited their accuracy and scalability. Moreover, these systems struggled with variations in handwriting styles, noise in images, and computational inefficiency. The problem we address is to build an accurate and efficient system for classifying handwritten digits using machine learning techniques. Accurate handwritten digit classification has numerous real-world applications, including postal automation, bank check processing, and digit recognition in forms. By developing robust classification models, we can enhance the efficiency and reliability of such applications, leading to improved productivity and cost savings. The proposed system employs modern machine learning techniques, including Generalized Learning Vector Quantization (GLVQ) and Support Vector Machine (SVM), to classify handwritten digits effectively. These algorithms offer advantages such as automatic feature extraction, robustness to variations, and scalability to large datasets. Additionally, we leverage data visualization and performance evaluation techniques to gain insights into the classification process and ensure model reliability. Overall, our system aims to provide an accessible and insightful journey into image data analysis for beginners in the field.

Keywords: Handwritten Digit Classification, Machine Learning, MNIST Dataset, Image Recognition, Computer Vision, GLVQ, Support Vector Machine, Feature Extraction, Classification Accuracy, Digit Recognition.

1.INTRODUCTION

1.1 History

The history of handwritten digit classification traces back to the early days of computer vision and pattern recognition. In the 1960s, researchers began exploring ways to automate the recognition of handwritten characters to alleviate the burden of manual data entry. One of the pioneering systems was developed by David H. Shepard in 1968, which employed template matching techniques to recognize handwritten digits. However, these early systems were limited in their ability to handle variations in handwriting styles and often required extensive preprocessing of images.

In the 1980s and 1990s, with advancements in machine learning algorithms and computing power, researchers started to explore more sophisticated approaches to handwritten digit classification. The introduction of neural networks, particularly the multi-layer perceptron (MLP), revolutionized the field by enabling automatic feature extraction from raw pixel data. LeNet-5, developed by Yann LeCun and his colleagues in 1998, was one of the first convolutional neural networks (CNNs) specifically designed for handwritten digit recognition. It achieved remarkable accuracy on the MNIST dataset and laid the groundwork for future developments in deep learning.

The early 2000s witnessed a surge of interest in ensemble methods and support vector machines (SVMs) for handwritten digit classification. Researchers explored ensemble techniques such as bagging and boosting to combine multiple classifiers and improve classification performance. SVMs, known for their ability to find optimal decision boundaries in high-dimensional spaces, emerged as a popular choice for binary classification tasks within the MNIST dataset. Additionally, researchers investigated feature engineering techniques to enhance the discriminative power of classification models.

2.LITERATURE SURVEY

In recent decades, the image classification problem has been widely addressed in the literature, and it is still an active research field in image processing today. In this field, convolutional neural networks have made a substantial breakthrough in visual recognition, especially handwritten digit recognition [1,2,3,4,5]. These networks have a great ability for learning and extracting image features easily. CNN architectures for image classification have two different types of layers: convolutional layers for extracting image features and fully connected layers for performing the classification task based on the features extracted by the preceding convolutional layers [6,7]. The handwritten digit recognition problem is a topic of heated debate in recent years. Despite that there are enormous convolutional neural network algorithms proposed for handwritten digit recognition, issues such as recognition accuracy and computation time still require further improvement [8]. Ali et al. [9] utilized the DL4J framework and a convolutional neural network classifier to achieve a commendable accuracy of 99.21% on the MNIST handwritten digits dataset. Schaetti et al. [10] proposed the use of reservoir computing paradigm, achieving an accuracy of 93% and surpassing other models in terms of error rate and complexity. Hermans et al. [11] focused on optimizing input encoding to improve classification performance on both MNIST and TIMIT datasets, demonstrating significant improvements over random masks.Mohapatra et al. [12] introduced a novel method for classifying MNIST handwritten digit images, leveraging the discrete cosine space-frequency transform for feature extraction and artificial neural network classifiers for classification. Kussul and Baidyk [13] proposed the Limited Receptive Area (LIRA) classifier, achieving a high classification accuracy of 99.41% on MNIST handwritten digit images. Ahlawata and Choudharyb [14] integrated convolutional neural networks with support vector machines to build a hybrid classification model, achieving an accuracy of 99.28%.Chazal et al. [15] compared different weight optimization methods using identical network topologies, finding that extreme learning machine algorithm resulted in faster optimization compared to backpropagation. Ma and Zhang [16] employed deep analysis with multi-feature extraction to build a handwritten digit classification method, achieving promising results by extracting projection

features from pre-processed images. Lopes et al. [17] utilized the Optimum-Path Forest classifier and achieved an average accuracy of 99.53% on the MNIST dataset.

Aly and Allotairi [18] proposed an unsupervised deep convolutional self-organizing maps network, achieving remarkable results on the MNIST dataset. Man et al. [19] applied a single-hidden layer feedforward network to classify handwritten digit images, demonstrating improvements in sensitivity by adjusting regularization parameters. Cardoso and Wichert [21] utilized a biologically inspired model for visual recognition as a feature space, achieving competitive results on both MNIST and USPS handwritten digit datasets.Niu and Suen [22] combined convolutional neural networks with support vector machines, achieving a recognition rate of 94.40% on the MNIST handwritten digit database. However, the complexity of this hybrid model may limit its practicality for certain applications such as self-driving cars.

3. PROPOSED SYSTEM

The Project presents a comprehensive overview of a project focused on handwritten digit classification using machine learning techniques. Here's an overview of the project:

- **Data Import and Visualization:**
  The project is importing the dataset containing handwritten digit images. The dataset is visualized using matplotlib to gain insights into the structure and distribution of the data.
- **Data Analysis and Preprocessing:**
  Data analysis techniques such as info, describe, and isnull are applied to understand the dataset's characteristics.
  The dataset is preprocessed, which involves splitting it into independent features (pixel values) and dependent labels (digit categories). Additionally, the dataset is split into training and testing sets using the train_test_split function.
- **Model Building:**
  Two machine learning models are trained and evaluated for handwritten digit classification: Generalized Learning Vector Quantization (GLVQ) and Support Vector Machine (SVM).
  The GLVQ model and SVM model are implemented using the sklearn_lvq.GlvqModel and sklearn.svm.SVC classes, respectively.
- **Model Evaluation:**
  Performance metrics such as accuracy, precision, recall, and F1-score are calculated for each model using the precision_score, recall_score, f1_score, and accuracy_score functions.
  Confusion matrices and classification reports are generated to assess the models' performance in classifying handwritten digits.
- **Results and Visualization:**
  The results of the model evaluation are presented in tabular form, summarizing the performance metrics for each algorithm.
  Additionally, the models' predictions on the test dataset are visualized using matplotlib, displaying the predicted labels alongside the corresponding digit images.

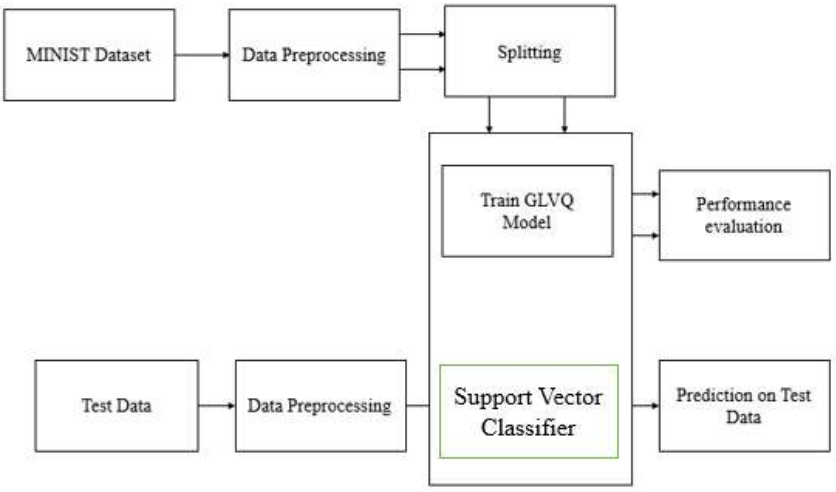

Fig. 1: Block Diagram of Proposed System.

3.1 Data Preprocessing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is no t always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set

Importing Libraries: To perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

import numpy as nm

Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. Here, we have used pd as a short name for this library. Consider the below image:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

```

Handling Missing data: The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset. There are mainly two ways to handle missing data, which are:

- By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.
- By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc.

Encoding Categorical data: Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, Country, and Purchased. Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

Feature Scaling: Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no variable dominates the other variable. A machine learning model is based on Euclidean distance, and if we do not scale the variable, then it will cause some issue in our machine learning model. Euclidean distance is given as:

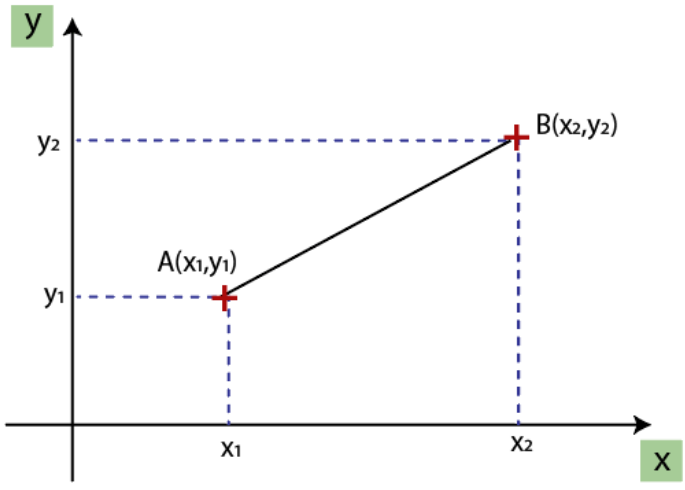

Fig. 2: Feature scaling

If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So, to remove this issue, we need to perform feature scaling for machine learning.

3.2 Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

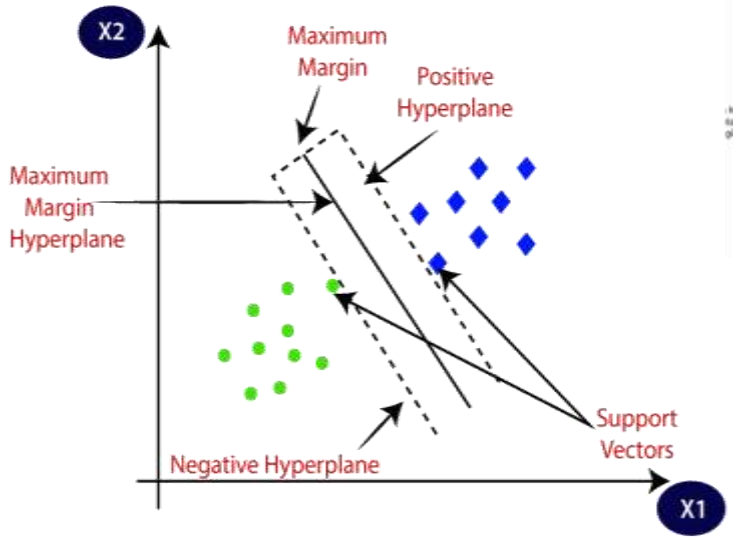

Fig. 3: Analysis of SVM

Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

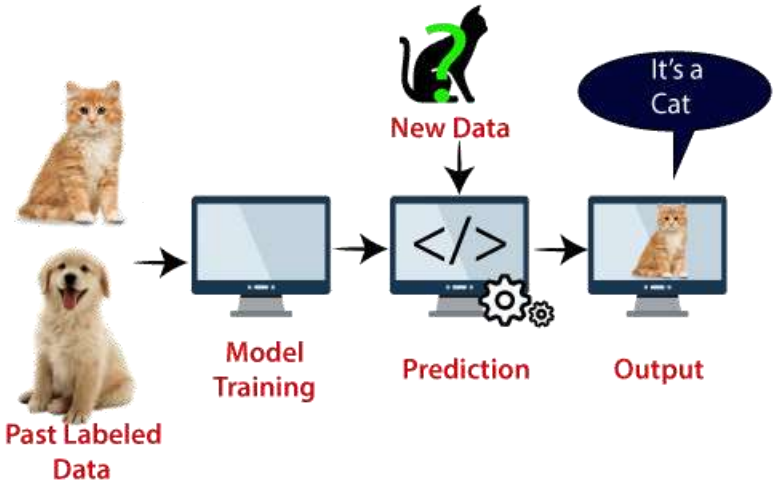

Fig. 4: Basic classification using SVM

Types of SVM: SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

3.3 SVM working

Linear SVM: The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair (x1, x2) of coordinates in either green or blue. Consider the below image:

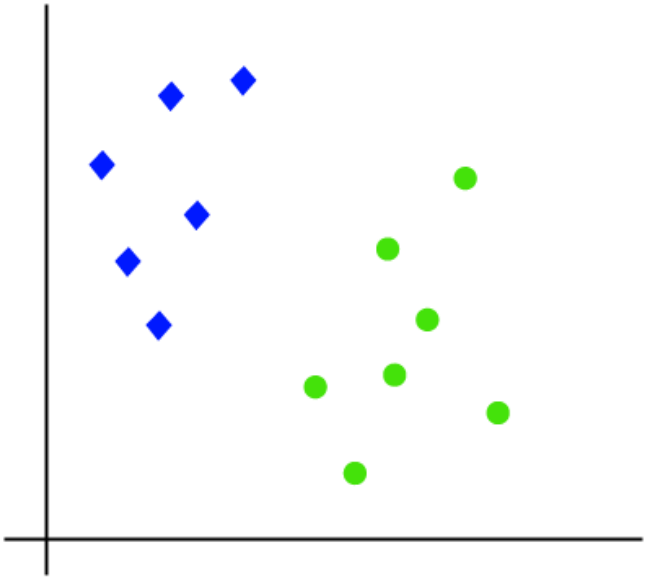

Fig. 5: Linear SVM

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

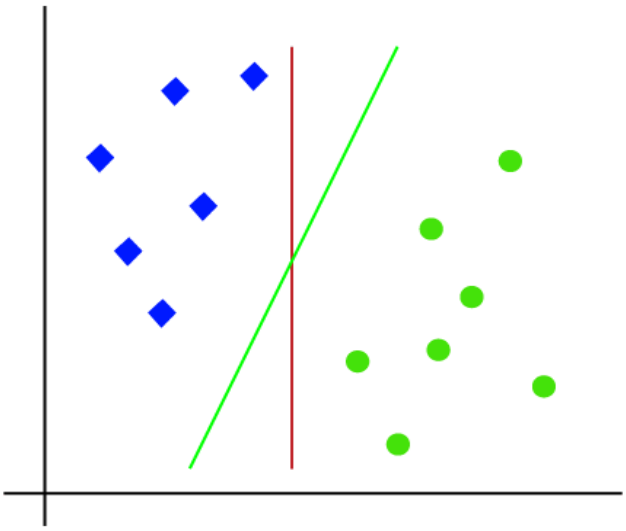

Fig. 6: Test-Vector in SVM

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

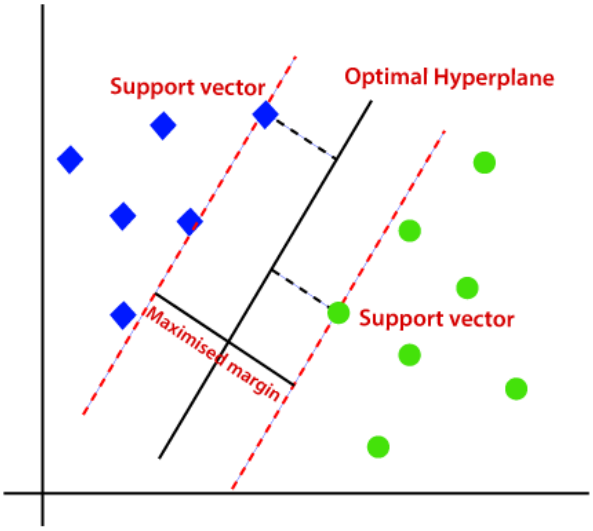

Fig. 7: Classification in SVM

Non-Linear SVM: If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

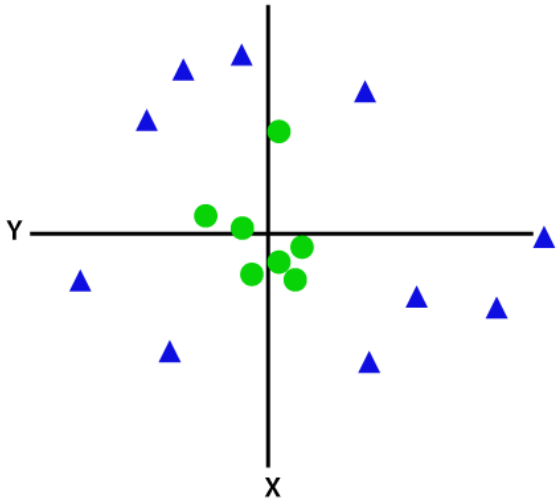

Fig. 8: Non-Linear SVM

So, to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third-dimension z. It can be calculated as:

$z=x^2 +y^2$

By adding the third dimension, the sample space will become as below image:

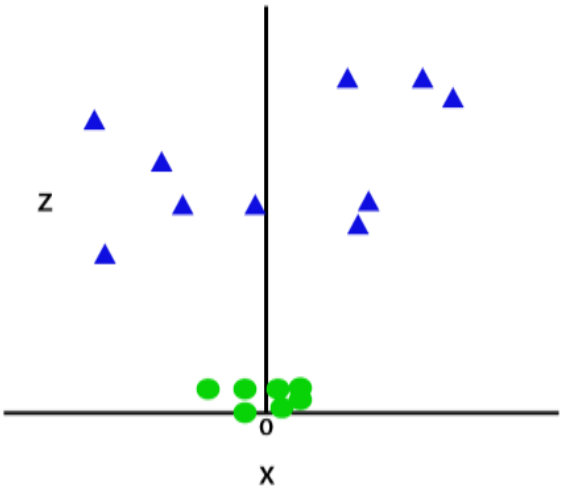

Fig. 9: Non-Linear SVM data seperation

So now, SVM will divide the datasets into classes in the following way. Consider the below image:

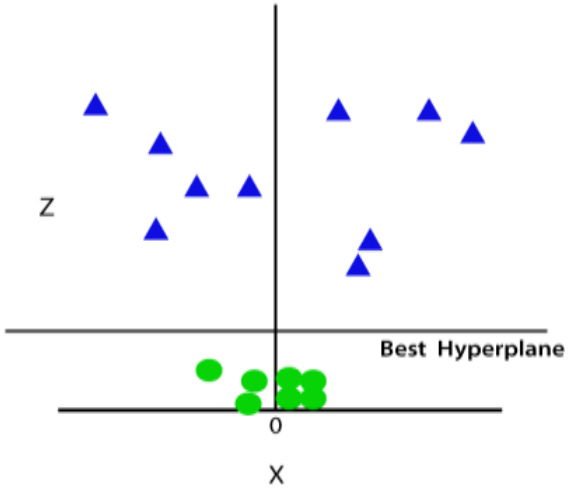

Fig. 10: Non-Linear SVM best hyperplane

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:

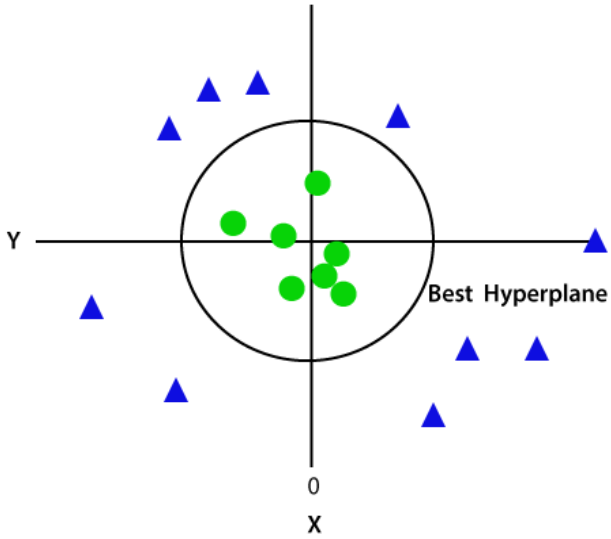

Fig. 11: Non-Linear SVM with ROC

Hence, we get a circumference of radius 1 in case of non-linear data.

4.RESULTS AND DESCRIPTION

4.1 Dataset Description

The labels of the MNIST dataset correspond to the handwritten digits represented in the images. Specifically, each image in the dataset is associated with a label, which is an integer ranging from 0 to 9. These labels indicate the digit that the handwritten image represents.

- **Dataset Composition:** The MNIST dataset contains a total of 70,000 grayscale images of handwritten digits, with each image being a 28x28 pixel square. These images are divided into two main subsets: a training set and a test set. The training set consists of 60,000 images, while the test set contains 10,000 images. Each image is associated with a corresponding label, indicating the digit it represents (0 through 9).
- **Data Collection Process:** The images in the MNIST dataset were collected from a variety of sources, including high school students and employees of the United States Census Bureau. Participants were asked to write digits on a blank canvas, resulting in a diverse collection of handwriting styles and variations. The images were then digitized and preprocessed to create the final dataset.
- **Data Representation:** As mentioned earlier, each image in the MNIST dataset is represented as a 28x28 pixel matrix. Each pixel value represents the intensity of the grayscale color, ranging from 0 (black) to 255 (white). Therefore, the dataset can be viewed as a collection of 28x28 matrices, with each matrix corresponding to a single handwritten digit.
- **Data Labeling:** In addition to the image data, the MNIST dataset also includes corresponding labels for each image. These labels are integers ranging from 0 to 9, indicating the digit represented by the image. For example, an image labeled with "5" represents the handwritten digit "5".

.Here's is the labels in the MNIST dataset:

- **Label 0**: Represents the digit zero. This label is assigned to images containing handwritten representations of the number 0.
- **Label 1**: Represents the digit one. This label is assigned to images containing handwritten representations of the number 1.
- **Label 2**: Represents the digit two. This label is assigned to images containing handwritten representations of the number 2.
- **Label 3**: Represents the digit three. This label is assigned to images containing handwritten representations of the number 3.
- **Label 4**: Represents the digit four. This label is assigned to images containing handwritten representations of the number 4.
- **Label 5**: Represents the digit five. This label is assigned to images containing handwritten representations of the number 5.
- **Label 6**: Represents the digit six. This label is assigned to images containing handwritten representations of the number 6.
- **Label 7**: Represents the digit seven. This label is assigned to images containing handwritten representations of the number 7.
- **Label 8**: Represents the digit eight. This label is assigned to images containing handwritten representations of the number 8.
- **Label 9**: Represents the digit nine. This label is assigned to images containing handwritten representations of the number 9.

Each label corresponds to a specific digit, and the MNIST dataset contains examples of handwritten representations of each digit. These labels are used for training machine learning models to recognize and classify handwritten digits accurately.

10.3 Results Description

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

5 rows × 784 columns

Fig. 12: Presents the Sample dataset of the minist dataset.

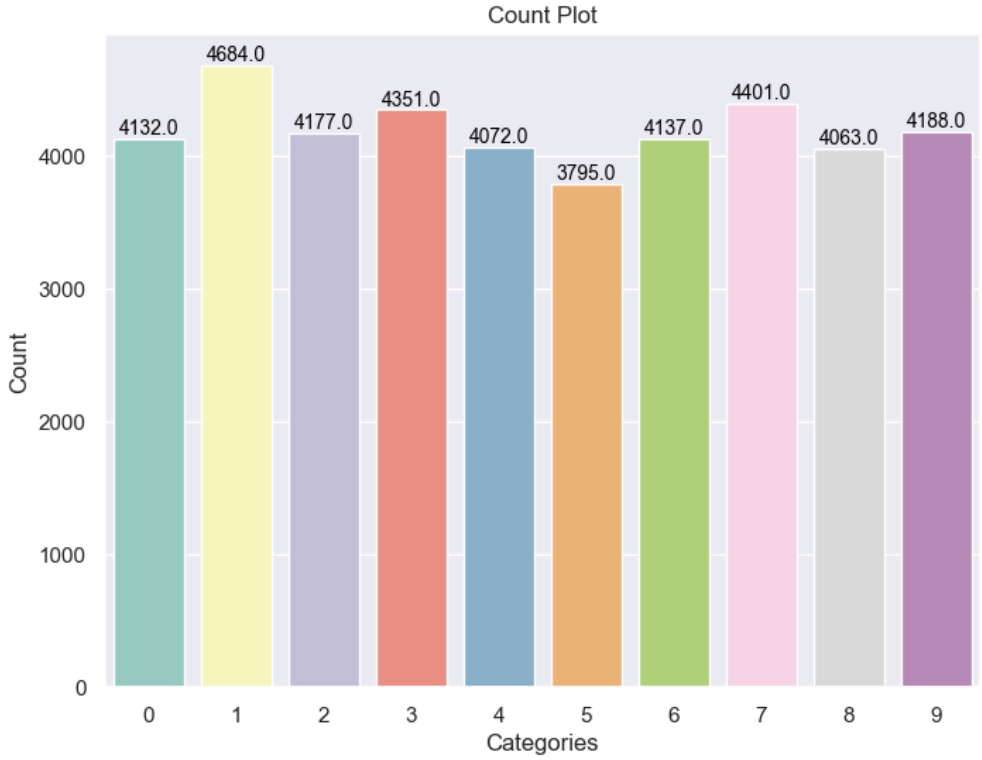

Fig. 13: Presents the count plot of minst dataset.

```
GLvq Accuracy     : 81.14285714285714
GLvq Precision    : 81.49284332813522
GLvq Recall       : 80.87953946539814
GLvq FSCORE       : 80.9612995902674

 GLvq classification report
              precision    recall  f1-score   support

           0       0.87      0.92      0.90       770
           1       0.97      0.76      0.85      1169
           2       0.76      0.87      0.81       744
           3       0.77      0.77      0.77       931
           4       0.81      0.80      0.80       854
           5       0.69      0.72      0.70       671
           6       0.85      0.87      0.86       771
           7       0.84      0.90      0.87       829
           8       0.74      0.82      0.78       755
           9       0.79      0.73      0.76       906

    accuracy                           0.81      8400
   macro avg       0.81      0.81      0.81      8400
weighted avg       0.82      0.81      0.81      8400
```

Fig. 14: Shows a classification report of a GLvq model.

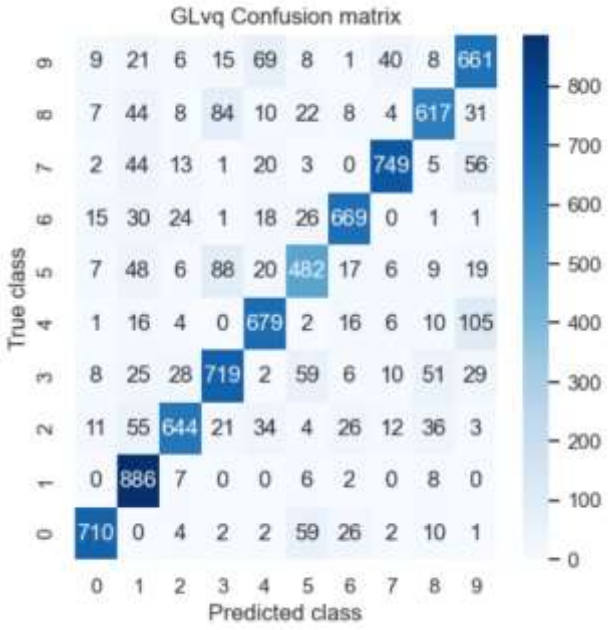

Fig. 15: Confusion matrix of GLvq model.

```
SVC Accuracy    : 97.3452380952381
SVC Precision   : 97.3416373747596
SVC Recall      : 97.35339772335251
SVC FSCORE      : 97.34602615363983

 SVC classification report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       819
           1       0.99      0.98      0.99       916
           2       0.97      0.98      0.97       839
           3       0.96      0.97      0.96       926
           4       0.97      0.96      0.97       847
           5       0.97      0.97      0.97       698
           6       0.99      0.98      0.98       795
           7       0.97      0.98      0.97       889
           8       0.97      0.97      0.97       837
           9       0.96      0.96      0.96       834

    accuracy                           0.97      8400
   macro avg       0.97      0.97      0.97      8400
weighted avg       0.97      0.97      0.97      8400
```

Fig. 16: Shows a classification report of a Support Vector Classifier model.

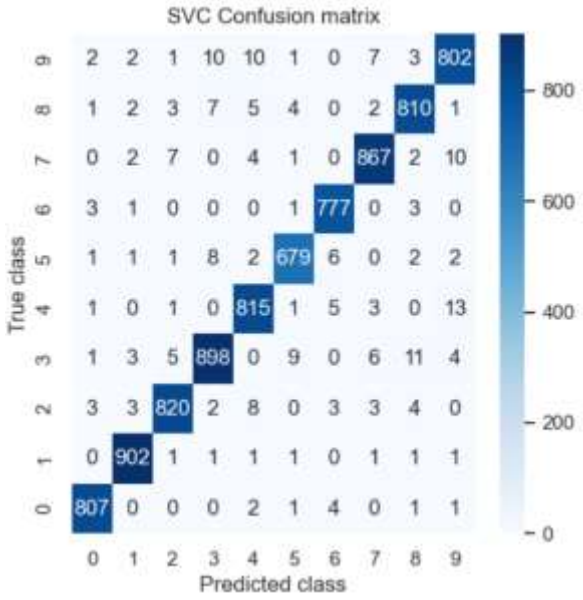

Fig. 17: Confusion matrix of Support Vector Classifier model.

The figure 4 confusion matrix of the Glvq model visually represents the performance of the model in classifying different categories of mouth diseases. It provides a clear overview of the true positive, true negative, false positive, and false negative predictions made by the model for each class. The figure 5 classification report of the Support Vector Classifier model presents a detailed summary of the model's performance in terms of precision, recall, F1-score, and support for each class. It offers insights into the model's ability to correctly classify instances of each disease category. The figure 6 confusion matrix of the Support Vector Classifier model illustrates the model's performance but specifically for this classifier. It provides a visual representation of how well the model predicts the actual classes of mouth diseases, aiding in understanding its strengths and weaknesses.

	Algorithm Name	Precison	Recall	FScore	Accuracy
0	GLvq Classifier	81.492843	80.879539	80.961300	81.142857
1	SVC Classifier	97.341637	97.353398	97.346026	97.345238

Fig. 17: Presents the Comparison table of performance metrics

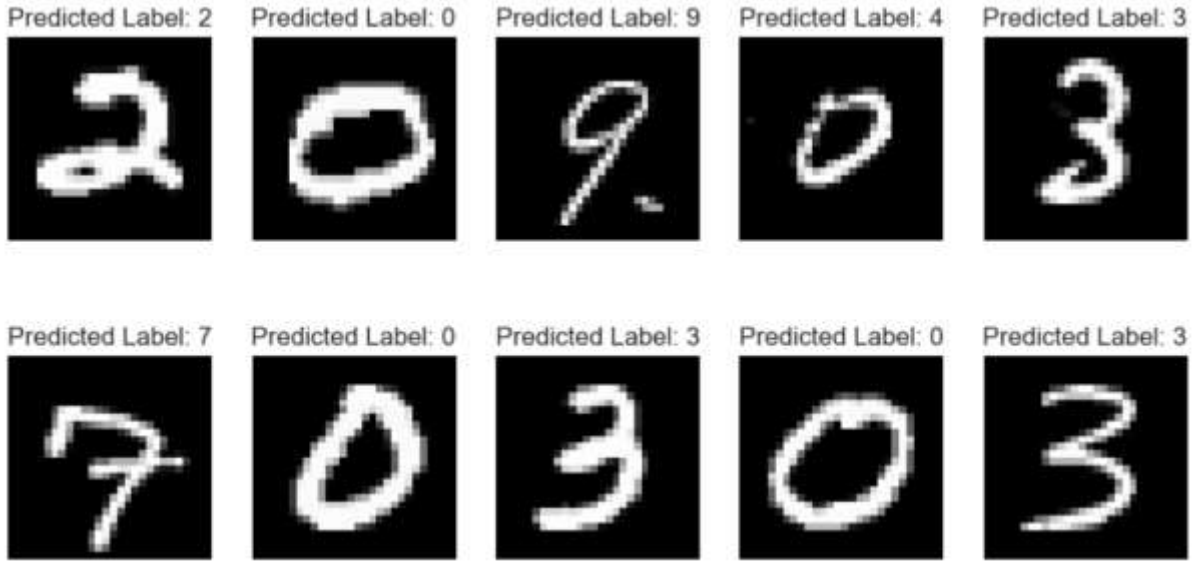

Fig. 18: Proposed SVC model Prediction of disease on test image.

The figure 7 comparison table of performance metrics presents a comprehensive overview of the performance of different classifiers, such as GLvq and SVC Classifier. It allows for a direct comparison of metrics such as accuracy, precision, recall, and F1-score, enabling stakeholders to make informed decisions about model selection. The figure 8 proposed SVC Classifier model's prediction of disease on a test image demonstrates the practical application of the model. By inputting an image of a mouth condition, the model predicts the corresponding disease category, showcasing its potential utility in real-world diagnostic scenarios.

5.CONCLUSION

The project has successfully demonstrated the implementation of machine learning models for handwritten digit classification using the MNIST dataset. Through various steps such as data loading, preprocessing, model building, evaluation, and testing, the project has provided a comprehensive understanding of the entire machine learning pipeline.

Specifically, the project utilized two classification algorithms, Generalized Learning Vector Quantization (GLVQ) and Support Vector Machine (SVM), to classify handwritten digits into their respective categories. Performance metrics such as accuracy, precision, recall, and F1-score were computed to evaluate the models' effectiveness in classifying digits accurately.

Furthermore, the project included visualizations of the dataset, including sample images of handwritten digits, count plots of digit distributions, confusion matrices, and classification reports. These visualizations offer insights into the dataset's characteristics and the models' performance.

REFERENCES

[1] Hossain M.A., Ali M.M. Recognition of handwritten digit using convolutional neural network (CNN) Glob. J. Comput. Sci. Technol. Neural Artif. Intell. 2019;19:27–33.

[2] He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); Las Vegas, NV, USA. 27–30 June 2016; pp. 1–12.

[3] Krizhevsky A., Sutskever I., Hinton G. Imagenet classification with deep convolutional neural networks. Commun. ACM. 2017;60:84–90.

[4] Arif R.B., Siddique M.A.B., Khan M.M.R., Oishe M.R. Study and observation of the variations of accuracies for handwritten digits Recognition with various hidden layers and epochs using convolutional neural network; Proceedings of the 4th IEEE International

Conference on Electrical Engineering and Information & Communication Technology (iCEEiCT); Dhaka, Bangladesh. 13–15 September 2018; pp. 1–6.
[5] Yang X., Pu J. MDig: Multi-Digit Recognition Using Convolutional Nerual Network on Mobile. pp. 1–10.
[6] Bora M.B., Daimary D., Amitab K., Kandar D. Handwritten character recognition from images using CNN-ECOC. Procedia Comput. Sci. 2020;167:2403–2409.
[7] Maitra D.S., Bhattacharya U., Parui S.K. CNN based common approach to handwritten character recognition; Proceedings of the 13th International Conference on Document Analysis and Recognition (ICDAR); Tunis, Tunisia. 23–26 August 2015.
[8] Qiao J., Wang G., Li W., Chenc M. An adaptive deep Q-learning strategy for handwritten digit recognition. Neural Netw. 2018;107:61–71.
[9] Ali S., Shaukat Z., Azeem M., Sakhawat Z., Mahmood T., Rehman K. An efficient and improved scheme for handwritten digit recognition based on convolutional neural network. SN Appl. Sci. 2019;1:1125.
[10] Schaetti N., Salomon M., Couturier R. Echo state networks-based reservoir computing for MNIST handwritten digits recognition; Proceedings of the IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES); Paris, France. 24–26 August 2016; pp. 484–491.
[11] Hermans M., Soriano M.C., Dambre J., Bienstman P., Fischer I. Photonic delay systems as machine learning implementations. J. Mach. Learn. Res. 2015;16:2081–2097.
[12] Mohapatra R.K., Majhi B., Jena S.K. Classification performance analysis of MNIST Dataset utilizing a Multi-resolution Technique; Proceedings of the International Conference on Computing, Communication and Security (ICCCS); Pamplemousses, Mauritius. 4–5 December 2015; pp. 1–5.
[13] Kussul E., Baidyk T. Improved method of handwritten digit recognition tested on MNIST database. Image Vis. Comput. 2004;22:971–981.
[14] Ahlawata S., Choudhary A. Hybrid CNN-SVM classifier for handwritten digit recognition. Procedia Comput. Sci. 2020;167:2554–2560.
[15] Chazal P., Tapson J., Schaik A. A comparison of extreme learning machines and back-propagation trained feed-forward networks processing the mnist database; Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); Brisbane, Australia. 19–24 April 2015; pp. 2165–2168.
[16] Ma C., Zhang H. Effective handwritten digit recognition based on multi-feature extraction and deep analysis; Proceedings of the 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD); Zhangjiajie, China. 15–17 August 2015; pp. 297–301.
[17] Lopes G.S., Vieira D.C.S., Rodrigues A.W.O., Filho P.P.R. Recognition of handwritten digits using the signature features and Optimum-Path Forest Classifier. IEEE Lat. Am. Trans. 2016;14:2455–2460.
[18] Aly S., Almotairu S. Deep convolutional self-organizing map network for robust handwritten digit recognition. IEEE Access. 2020;8:107035–107045.
[19] Man Z., Lee K., Wang D., Cao Z., Khoo S. An optimal weight learning machine for handwritten digit image recognition. Signal Process. 2013;93:1624–1638.
[20] Kulkarni S.R., Rajendran B. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. Neural Netw. 2018;103:118–127.

[21] Cardoso A., Wichert A. Handwritten digit recognition using biologically inspired features. Neurocomputing. 2013;99:575–580.
[22] Niu X.X., Suen C.Y. A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognit. 2012;45:1318–1325.–1325.