Dataset: (MNIST)

The MNIST dataset is one of the most well-known and widely used datasets in the field of machine learning and computer vision. It stands for "Modified National Institute of Standards and Technology" and consists of a large collection of handwritten digits. Here's a small report about the MNIST dataset:

1. Dataset Composition:

- The MNIST dataset contains 70,000 grayscale images of handwritten digits (0 through 9).
- Each image is a 28x28 pixel square, resulting in a total of 784 pixels per image.
- These images are accompanied by their corresponding labels, indicating the digit they represent.

2. Training and Testing Split:

- The dataset is typically divided into two parts: a training set and a testing set.
- The training set contains 60,000 images, while the testing set contains 10,000 images.
- This split ensures that models can be trained on one portion of the data and evaluated on another, independent portion.

3. Use in Machine Learning:

- MNIST has been a benchmark dataset for testing and comparing the performance of machine learning algorithms, particularly in the realm of image classification.
- Many researchers and practitioners use MNIST as a starting point for experimenting with various machine learning models, including neural networks, support vector machines, and decision trees.

4. Challenges and Limitations:

- While MNIST has been valuable for developing and benchmarking algorithms, its simplicity has led to some limitations.
- Real-world tasks often involve more complex images with variations in lighting, orientation, and scale, which MNIST does not fully capture.
- As a result, while algorithms may perform well on MNIST, their performance may not generalize effectively to more challenging datasets or real-world applications.

5. Legacy and Impact:

- MNIST has played a significant role in the advancement of machine learning, serving as a foundational dataset for numerous research papers, tutorials, and educational resources.

- It has also helped popularize techniques such as convolutional neural networks (CNNs), which have demonstrated exceptional performance on the MNIST dataset and later on more complex tasks.

Overall, the MNIST dataset remains an essential resource for researchers, educators, and practitioners in the field of machine learning, providing a standardized benchmark for evaluating and comparing different algorithms and techniques.

Goal:

Our goal was to use the MNIST dataset to train various models on classification to make a benchmark and determine which model achieves the highest score.

Data preprocessing:

We started by using the hog function from skimage, The HOG function from the skimage.feature module computes the Histogram of Oriented Gradients (HOG) feature descriptor for images. It extracts local gradient orientations and magnitudes, creating histograms within small cells. These histograms are then normalized within blocks to enhance robustness against illumination variations. The final feature vector is formed by concatenating these normalized block histograms. The HOG descriptor is widely used in computer vision tasks like object detection and recognition, serving as input features for machine learning algorithms such as support vector machines (SVMs) or neural networks.

1. Feature Extraction

2. Local Gradient Histograms

3. Normalization

4. Block Concatenation

5. Usage

Models:

1- K-neighbors classifier:

This model just takes the preprocessed data and consider them as neighbors, so if there's a new image needs to be classified the model just computes the distance between this image and K nearest neighbors to it and determines the class of the image due to the majority of the neighbors' class.

Performance:

- Sklearn accuracy: 97.5%
- Sklearn precision: 98%
- Sklearn recall: 97%
- Sklearn f1-Score: 97%

These numbers confirm that K-neighbors is one of the best algorithms to classify in this dataset context.

2- Decision Tree classifier:

This algorithm creates a tree that classifies data by going through the tree nodes starting from the initial node and splitting the data to different nodes by choosing the best feature to split on, and this is done by calculating the information gain for every feature attribute at the node we are standing on.

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$$IG(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S|}{|S_v|} \times Entropy(S_v)$$

Performance:

- Sklearn accuracy: 88.86%
- Sklearn precision: 89%
- Sklearn recall: 89%
- Sklearn f1-Score: 89%

The Decision tree classifier is not quite good for image classification because it works best on classifying based on written feature as for every data row or tuple its features have descriptive values not quantitative values as numbers.

3- Random Forest Classifier:

It's basically the same as the Decision tree classifier and it's based on it, though the difference is that the random forest is a trees ensemble of the Decision tree classifier and it classifies the example based on a criterion such as voting, weighted voting or even encoding the classes and averaging the answers.
and to make each tree in the forest different from each other, we train each tree by withdrawing samples from the dataset by sampling with replacement, so we guarantee that each tree is randomly different from the others.

Performance:

- Sklearn accuracy: 97.47%
- Sklearn precision: 97%
- Sklearn recall: 97%
- Sklearn f1-score: 97%

we notice that there's an observed positive change in performance when using the Random Forest rather than using the normal Decision tree, that's because creating different trees or trees ensembles with sampling with replacement allows each tree to focus on different aspects of the data, therefore given more reasonable answer when averaging since one model misclassification won't make that big change in the final answer.

4- Gaussian Naïve Bayes Classifier:

Gaussian Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming feature independence and following a Gaussian distribution for continuous features within each class. It calculates the likelihood of observing a new instance's feature values given each class using Gaussian probability density functions and combines them with prior probabilities to compute posterior probabilities. This method is computationally efficient, particularly suitable for large datasets, and performs well in scenarios where the independence assumption holds reasonably well. Despite its simplicity, it's widely used and effective, especially in text classification and similar tasks.

Performance:

- Sklearn accuracy: 81.08%
- Sklearn precision: 85%
- Sklearn recall: 81%
- Sklearn f1-score: 81%

Naive Bayes might not do well with the MNIST dataset because it assumes features are unrelated, but in MNIST, pixels are often related. Also, MNIST has lots of features (pixels)

which can confuse Naive Bayes. Plus, the pixel values don't always follow the pattern Naive Bayes expects. Lastly, recognizing handwritten digits is tricky, and Naive Bayes is a bit too simple for that job.

5- Logistic Regression Classifier:

Logistic regression is a classification algorithm that predicts binary outcomes. It uses a linear model to estimate the log odds of the probability of a given input belonging to a particular class. The sigmoid function converts these log odds into probabilities between 0 and 1. During training, logistic regression optimizes the parameters (weights) of the linear model to best fit the training data, typically using techniques like gradient descent. The decision boundary separates instances into different classes, and predictions are made based on the probability exceeding a threshold. Logistic regression is evaluated using metrics like accuracy and precision and is widely used for binary classification tasks due to its simplicity and effectiveness.

Performance:

- Sklearn accuracy: 98.15%
- Sklearn precision: 98%
- Sklearn recall: 98%
- Sklearn f1-Score: 98%

Logistic regression can work well for the MNIST dataset because it's good at handling binary classification tasks, which is what MNIST is about (recognizing digits). It's also efficient and easy to understand, making it a good choice for simpler problems like this one. Even though MNIST images are complex, logistic regression can still find patterns in them, especially the linear ones.

Benchmark:

	1-KNN	2-DT	3-RF	4-GNB	5-LR
Precision	98%	89%	97%	85%	98%
Recall	97%	89%	97%	81%	98%
F1-Scare	97%	89%	97%	81%	98%
Accuracy	97.5%	88.86%	97.47%	81.08%	98.15%

Seeing this benchmark approves that the best algorithm to be used on some visual data that's not too complicated is **Logistic regression**, especially if data patterns are stable like edges and brightness levels.