January 2025

Classification and Evaluation of Machine Learning Algorithms on the MNIST Dataset

Felix Yeboah
University of Central Florida, fe528610@ucf.edu

computational requirements of the K Nearest Neighbour and Naive Bayes algorithms. Part of this project aims to detail the strengths and shortcomings of each algorithm in handling the challenges posed by the MNIST dataset.

II. Data

A. Data Description

The MNIST database (Modified National Institute of Standards and Technology database) is an extensive database of handwritten digits that is commonly used to train various machine learning algorithms. The database consists of handwritten digits collected among Census Bureau employees and high school students. The MNIST database comprises 60,000 training sample images and 10,000 testing sample images. Fifty percent of the training samples and the test samples were extracted from the National Institute of Standards and Technology (NIST) training dataset, while the other fifty percent of the training set and the test set were taken from NIST's testing dataset. The size of the images has been normalized and centered into 28x28 pixel size images [5]; a single instance is shown in Fig. 1. The distribution of each digit of the training sample is shown in Fig. 2

B. Exploratory Data Analysis

Having described the MNIST dataset, I will examine and visualize its main characteristics and patterns.Fig. 3 shows the distribution of pixel intensity. The graph reveals that about 65% of the pixels in the dataset are white, about 12% are dark, and the remaining 23% are distributed across various shades of gray. This finding is not all that surprising due to the gray-scale nature of the images. The dataset can be "binarized" into white

```
[[[0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.047 0.22  0.549 0.494 0.686 0.784 0.376 0.008 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.137 0.651 0.933 0.996 0.965 0.949 0.992 0.965 0.996 0.263 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.722 0.714 0.573 0.498 0.275 0.118 0.176 0.141 0.843 0.686 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.118 0.    0.    0.    0.    0.    0.    0.    0.812 0.965 0.055 0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.216 0.984 0.663 0.004 0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.043 0.843 0.91  0.078 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.078 0.745 0.98  0.239 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.094 0.463 0.808 0.996 0.973 0.557 0.424 0.071 0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.247 0.875 0.996 0.996 0.996 0.996 0.996 0.996 0.82  0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.204 0.682 0.506 0.373 0.063 0.063 0.063 0.416 0.976 0.49  0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.702 0.937 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.314 0.937 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.314 0.957 0.078 0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.392 0.937 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.918 0.937 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.016 0.549 0.02  0.    0.    0.    0.    0.    0.    0.012 0.588 0.996 0.506 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.251 0.996 0.71  0.149 0.    0.    0.    0.    0.133 0.737 0.996 0.82  0.078 0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.047 0.886 1.    0.875 0.345 0.267 0.502 0.616 0.949 0.996 0.812 0.09  0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.176 0.824 0.996 0.996 0.996 0.996 1.    0.996 0.733 0.192 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.161 0.506 0.937 0.898 0.702 0.357 0.063 0.012 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]
  [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    ]]]
```

Fig. 1. A 28x28 matrix of a single instance

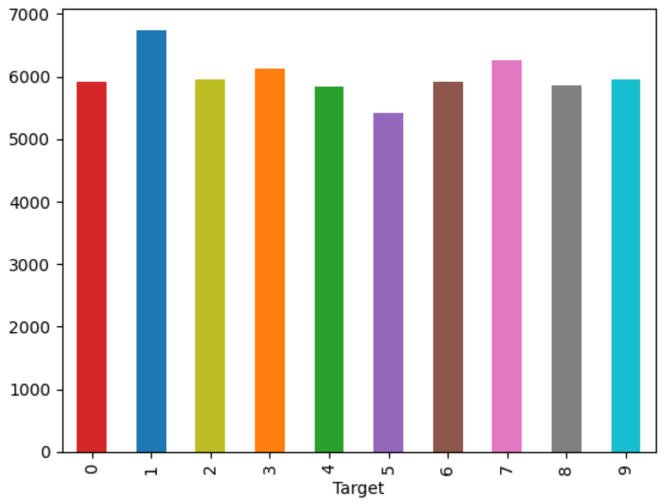
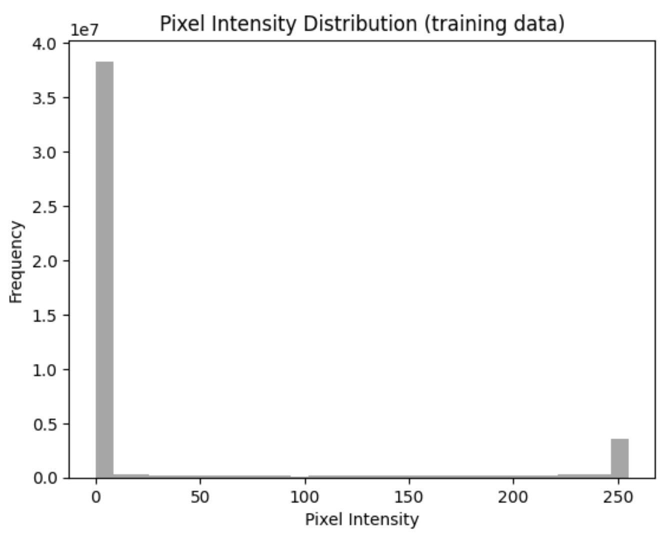


Fig. 2. Distribution of Target labels

and dark pixels due to their dominance. In addition, Fig. 3 can be seen as the distribution of target variables. Nearly all the target labels have more than 6000 instances, with target label ”1” having the most significant number of instances and target label ”5” having the minimum number of instances. With regard to visualizing the distribution of the whole training set, a method called “t-SNE” was utilized. T-Distributed Stochastic Neighbor Embedding (t-SNE) visualizes high-dimensional data by giving each data point a location in a two or three-dimensional plane [6]. “t-SNE” attempts to ensure that each point has an equal number of neighbors by creating an idea of which other points are its ”neighbors” for each point. After that, it attempts to embed them so that each point has an equal number of neighbors [7]. See Fig. 4. The next section presents two methods and evaluation metrics for classifying the MNIST dataset.



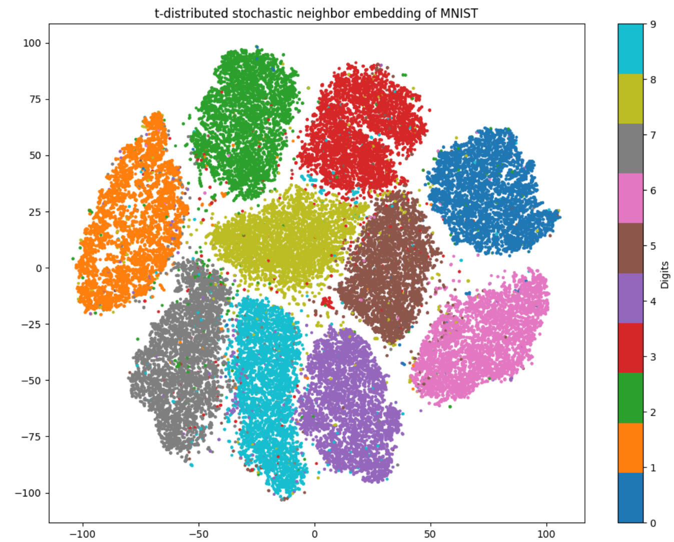


Fig. 4. Distribution of Training Set

III. Methodology

A. k-Nearest Neighbor

The K-nearest neighbors (KNN) algorithm is a nonparametric supervised machine learning approach to classification. KNN tries to categorize the accurate class for the test sample by calculating the distance between the test sample and all the training samples.The algorithm looks for the highest number of classes or the average of the neighboring data points with the most common characteristics with the newly added data point. [8].

Consider a test sample $\mathbf{x}$ that we wish to categorize into one of the K groups, we find the K observed data points that are closer to $\mathbf{x}$. The classification algorithm assigns $\mathbf{x}$ to the population with the highest number of observed data points out of the K-nearest neighbors. When there is no majority vote the new instances are classified to one of the majority populations at random or left unclassified [9].Algorithm 1 is pseudo-code for the KNN algorithm.

B. Naïve Bayes

Naive Bayes algorithm (NB) can broadly be defined as a family of machine learning classifiers that uses Bayes's theorem and conditional independence assumption to determine the probability that an instance belongs to a specific category. The algorithm is named naive because it assumes all the features are independent. This assumption is implausible in many real-world situations. Naive Bayes often produces competitive classification accuracy, although the assumption of independence is often violated in practice. Given a set of features $X_1, X_2, X_3, ...X_n$, the objective of classifying Y, is similar to maximizing
$\arg\max_{y \in Y} P(Y = y \mid X_1, X_2, X_3, ...X_n)$
Using Bayes' Theorem, we can rewrite this problem as

Algorithm 1 Algorithm for k-Nearest Neighbor

Input: Given a set of training samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{60000}$, where:

$$X_i = \text{ith feature instance},$$
$$y_i = \text{ith target label},$$
$$K = \text{Number of nearest neighbors},$$
$$\mathbf{x} = \text{New instance}.$$

Output: Class of test instance

for every $\mathbf{x}$ **do**
  Compute the distance: $d(\mathbf{x}, x) = \sqrt{\sum_{i=1}^{N} (\mathbf{x}_i - X_i)^2}$
end for
return $\hat{y} = \arg\max_{c \in C} \sum_{i=1}^{K} C(y_{(i)} = c)$

$$P(Y = y \mid X_1, X_2, \ldots X_n) = \frac{P(X_1, X_2, \ldots X_n \mid Y = y) P(Y = y)}{P(X_1, X_n, \ldots X_n)} \tag{1}$$

Expanding numerator in "(1)", we will have

$$\begin{aligned} &P(Y = y \mid X_1, X_2, \ldots X_n) \\ &= P(X_1, X_2, \ldots X_n \mid Y = y) \\ &= P(X_1, X_2, \ldots X_n \mid Y = y) P(X_2, \ldots X_n \mid Y = y) \times \\ &\quad P(X_1 \mid X_2, \ldots X_n, Y = y) P(X_2 \mid X_3 \ldots X_n, Y = y) \\ &\quad \ldots P(X_n \mid Y = y) P(Y = y) \end{aligned} \tag{2}$$

Modeling $P(Y = y) P(X_1, X_2 \ldots X_n \mid Y = y)$ is difficult because there are too many parameters, and in most cases, we will run out of space and training data. Instead we assume that the features $X_1, X_2, X_3, \ldots. X_n$ are independent given the labels Y.We can reformulate "(2)" as

$$P(Y = y \mid X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Y = y) \tag{3}$$

Therefore we have

$$P(Y = y \mid \mathbf{X}) = \frac{P(Y = y)}{P(\mathbf{X})} \prod_{i=1}^{n} P(X_i \mid Y = y) \tag{4}$$

"(4)" is equivalent to

$$\arg\max_{y \in Y} P(Y = y) \prod_{i=1}^{n} P(X_i \mid C = k) \tag{5}$$

Since $P(\mathbf{X})$ is constant and does not depend on y, we drop it. This transformation makes the problem computable [10]. The methods described above were implemented in Python using the Scikit-learn machine-learning library, and the results obtained from them are described in the next chapter.

IV. Results and Discussion

Having discussed the methods used in classifying the MNIST dataset, this section discusses, evaluates, and compares the results from these methods.
Table I shows the classification report after training the KNN and Naive Bayes algorithm on the 60000 training set and 10000 test set. Comparing KNN with k=3 nearest neighbors to Gaussian naive Bayes, KNN had an accuracy rate of 97% and an error rate of 3%, and naive Bayes had an accuracy of 56% and an error rate of 44%. The poor performance of the naive Bayes algorithm can be attributed to the nonnormal nature of the pixel distribution. Reproducing the analysis with a distribution that closely models the MNIST data can improve the performance of the Naive Bayes model.

TABLE I
Precision, Recall and F1-score for KNN and Naive Bayes

	KNN			Naive Bayes		
	Precision	Recall	F1-score	Precision	Recall	F1-score
0	0.96	0.99	0.98	0.79	0.89	0.84
1	0.95	1.00	0.98	0.85	0.95	0.90
2	0.98	0.96	0.97	0.90	0.26	0.40
3	0.96	0.97	0.97	0.71	0.35	0.47
4	0.98	0.96	0.97	0.88	0.17	0.29
5	0.97	0.97	0.97	0.55	0.05	0.09
6	0.98	0.99	0.98	0.65	0.93	0.77
7	0.96	0.96	0.96	0.88	0.27	0.42
8	0.99	0.94	0.96	0.28	0.67	0.40
9	0.96	0.95	0.95	0.37	0.95	0.53

[a] Error Rate KNN: 0.03.
[b] Accuracy Rate for KNN: 97%.
[c] Error Rate for Naive Bayes: 0.44.
[d] Accuracy Rate Naive Bayes: 56%.

The distribution of misclassified labels for KNN and naive Bayes is shown in Fig. 5 and Fig. 6, respectively, with KNN having fewer misclassified labels.

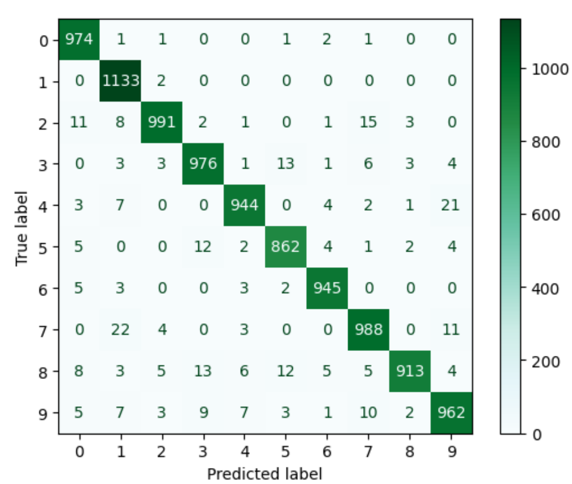


Fig. 5. Confusion Matrix for KNN

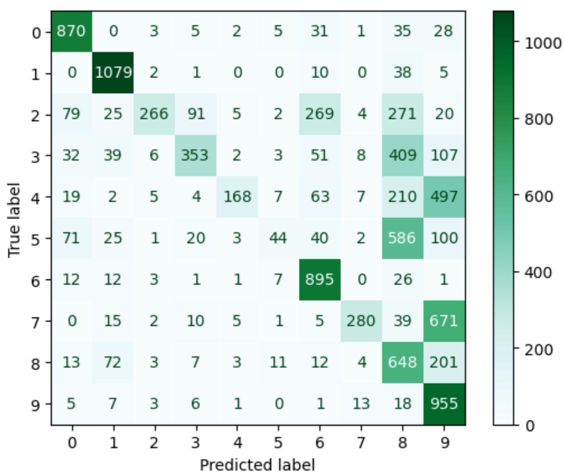


Fig. 6. Confusion Matrix for Naive Bayes

Receiver operating characteristic (ROC) curves of the KNN and Naive Bayes models are displayed in Fig. 7. The area under the ROC curve shows the probability that the model will perform better when given a new instance. Fig. 7 shows that the KNN model will likely perform better than the Naive Bayes model.

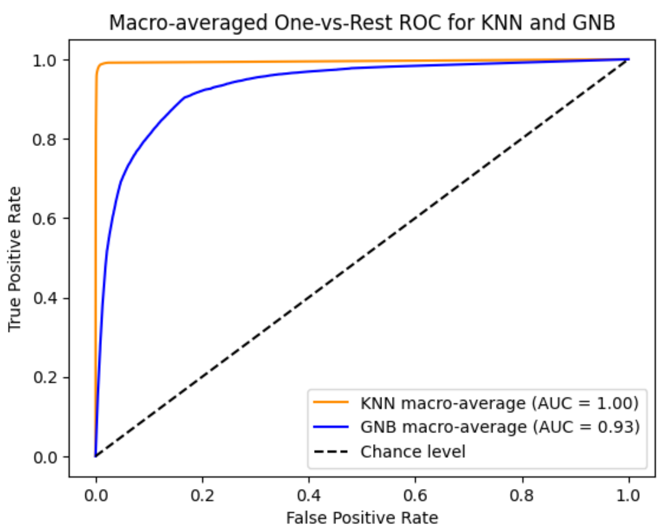


Fig. 7. ROC curve for KNN and Naive Bayes

V. Conclusion

This project was undertaken to explore, evaluate, and compare machine learning algorithms on the widely-used MNIST dataset. Due to the simplicity of the MNIST dataset, it has widely been used to benchmark most machine-learning algorithms. Based on the evaluation metrics used in this study, the K-Nearest Neighbors (KNN) algorithm came up as the best model compared with Naive Bayes. Even though the KNN algorithm is computationally expensive, it can still provide good performance and predictions. One source of weakness in this study that could have affected the Naive Bayes algorithm's performance is the pixel intensity distribution. For further studies, more information on pixel distribution would improve the performance of the Naive Bayes algorithm.

References

[1] M. Shinozuka and B. Mansouri, "Synthetic aperture radar and remote sensing technologies for structural health monitoring of civil infrastructure systems," Structural Health Monitoring of Civil Infrastructure Systems, pp. 113–151, 2009, doi: https://doi.org/10.1533/9781845696825.1.114.

[2] K. Huang, "Image Classification Using the Method of Convolutional Neural Networks," 2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), Dec. 2022, doi: https://doi.org/10.1109/tocs56154.2022.10016070.

[3] W. Sun, "Image classification based on CNN with three different networks," Applied and Computational Engineering, vol. 15, no. 1, pp. 181–186, Oct. 2023, doi: https://doi.org/10.54254/2755-2721/15/20230831.

[4] Yann LeCun et al., "Learning algorithms for classification: A comparison on handwritten digit recognition," 1995. Available: https://api.semanticscholar.org/CorpusID:13411815.

[5] L. V. D. Maaten and G. E. Hinton, "Visualizing Data using t-SNE," 2008. https://www.semanticscholar.org/paper/Visualizing-Data-using-t-SNE-Maaten-Hinton/1c46943103bd7b7a2c7be86859995a4144d1938b

[6] C. Olah, "Visualizing MNIST: An Exploration of Dimensionality Reduction - colah's blog," colah.github.io, Oct. 09, 2014. https://colah.github.io/posts/2014-10-Visualizing-MNIST/ (accessed Sep. 22, 2024).

[7] Rekha Gangula and Venkateswarlu B, "Exploring the Power and Practical Applications of K-Nearest Neighbours (KNN) in Machine Learning," vol. 2, no. 1, Feb. 2024, doi: https://doi.org/10.69996/jcai.2024002.

[8] R. C. Neath and M. S. Johnson, "Discrimination and classification," P. Peterson, E. Baker, and B. McGaw, Eds., Third Edition.Elsevier, 2010, pp. 135–141. doi: https://doi.org/10.1016/B978-0-08-044894-7.01312-9.

[9] Abedalmuhdi Almomany, W. R. Ayyad, and A. Jarrah, "Optimized implementation of an improved KNN classification algorithm using Intel FPGA platform: Covid-19 case study," Journal of King Saud University - Computer and Information Sciences, vol. 34, Art. no. 6, Part B, 2022, doi: https://doi.org/10.1016/j.jksuci.2022.04.006.

[10] I. Rish, "An empirical study of the naive Bayes classifier," 2001. Available: https://faculty.cc.gatech.edu/ isbell/reading/papers/Rish.pdf

VI. Appendix

```python
# %% [markdown]
# # Importing libraries and Dataset

# %%

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.manifold import TSNE
%matplotlib inline
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# %% [markdown]
# # Splitting, normalizing, rehspaing  the data.

# %%
# Normalizing Data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Reshaping Data
x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)


# %% [markdown]
# # Exploratoty Data Analysis

# %%
## How a single entry in  dataset looks like
instance1 = x_train[2]
print(instance1.reshape(1,28,28))


# %%
## Vizualizations
colorbar = ['tab:red',
            'tab:blue',
            'tab:olive',
            'tab:orange',
            'tab:green',
            'tab:purple',
            'tab:brown',
            'tab:pink',
            'tab:gray',
            'tab:cyan']

labels = pd.DataFrame(y_train)
labels.rename(columns= {0:'Target'}, inplace=True)
labels.groupby(['Target']).size().plot(kind='bar',color=colorbar)

# %%
# vizualizing all the 10 categories in the dataset
plt.figure(figsize=(12,10))
x, y = 10, 6
for i in range(30):
    plt.subplot(y, x, i+1)
    plt.imshow(x_train[i].reshape(28,28),cmap='magma')
plt.show()

# %%
### Distribution of pixel intensity
mean_all = np.mean(x_train)
median_all = np.median(x_train)
std_all = np.std(x_train)

print(f"Mean ( training data ): {mean_all:.2f}")
print(f"Median (training data): {median_all:.2f}")
print(f"Standard Deviation (training data): {std_all:.2f}")

# Plot histogram for all images
plt.hist(x_train.flatten(), bins=30, color='gray', alpha=0.7)
plt.title('Pixel Intensity Distribution (training data)')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()


# %%
##
Tsne = TSNE(n_components=2,perplexity=30,max_iter=1000,random_state=24)
model_tsne = Tsne.fit_transform(x_train)


# %%
plt.figure(figsize=(12,9))
scta_plt = plt.scatter(model_tsne[:,0], model_tsne[:,1],
                       c=y_train.astype(int), cmap='tab10', s= 4)
plt.colorbar(scta_plt, ticks=np.unique(y_train.astype(int)),
             label="Digits")
plt.title('t-distributed stochastic neighbor embedding of MNIST')
plt.show()

# %% [markdown]
# # Fiting the model

# %%
#### fiting the model
from sklearn.neighbors import KNeighborsClassifier
KNN_model = KNeighborsClassifier()
KNN_model.fit(x_train,y_train)


# %%
## Model prediction
predict = KNN_model.predict(x_test)


# %% [markdown]
# ## Model evaluation

# %%
```

```python

from sklearn import metrics
evalte = metrics.classification_report(y_test,predict)
print(evalte)

# %%
## vizulaize
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
conf_mat = confusion_matrix(y_test,predict)
conf_mat_disp = ConfusionMatrixDisplay(conf_mat)
conf_mat_disp.plot(cmap=plt.cm.BuGn)
plt.show()

# %%
from sklearn.metrics import precision_score,recall_score, accuracy_score, f1_score
accuracy_score(y_test,predict)
precision_score(y_test,predict, average='weighted')
recall_score(y_test,predict, average='weighted')
f1_score(y_test,predict,average='weighted')


# %%
### implement naive bayes Gaussian
from sklearn.naive_bayes import GaussianNB
model_gnb = GaussianNB()
model_gnb_fit= model_gnb.fit(x_train,y_train)


# %%
## predict
y_gnb_pred = model_gnb_fit.predict(x_test)

# %%
gnb_report = metrics.classification_report(y_test,y_gnb_pred)
print(gnb_report)

# %%
### confusion matrix
gnb_conf_mat = confusion_matrix(y_test,y_gnb_pred)
gnb_conf_mat_disp = ConfusionMatrixDisplay(gnb_conf_mat)
gnb_conf_mat_disp.plot(cmap=plt.cm.BuGn)
plt.show()

# %%
## evaluation metrics

accuracy_score(y_test,y_gnb_pred)
precision_score(y_test,y_gnb_pred, average='weighted')
recall_score(y_test,y_gnb_pred, average='weighted')
f1_score(y_test,y_gnb_pred,average='weighted')


# %% [markdown]
# ## Auc/ROC

# %%
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import RocCurveDisplay

label_binarizer = LabelBinarizer().fit(y_train)
y_onehot_test = label_binarizer.transform(y_test)
y_onehot_test.shape  # (n_samples, n_classes)



y_score = model_gnb.fit(x_train, y_train).predict_proba(x_test)

display = RocCurveDisplay.from_predictions(
    y_onehot_test.ravel(),
    y_score.ravel(),
    name="micro-average OvR",
    color="darkorange",
    plot_chance_level=True,
)
_ = display.ax_.set(
    xlabel="False Positive Rate",
    ylabel="True Positive Rate",
    title="Micro-averaged One-vs-Rest\nReceiver Operating Characteristic",
)


# %%
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import RocCurveDisplay
import matplotlib.pyplot as plt


# Binarize the labels
label_binarizer = LabelBinarizer().fit(y_train)
y_onehot_test = label_binarizer.transform(y_test)


y_score_knn = KNN_model.fit(x_train, y_train).predict_proba(x_test)

y_score_gnb = model_gnb.fit(x_train, y_train).predict_proba(x_test)

# Plot for KNN model
display_knn = RocCurveDisplay.from_predictions(
    y_onehot_test.ravel(),
    y_score_knn.ravel(),
    name="KNN micro-average OvR",
    color="darkorange",
    plot_chance_level=True,
)
_ = display_knn.ax_.set(
    xlabel="False Positive Rate",
    ylabel="True Positive Rate",
    title="Micro-averaged One-vs-Rest ROC for KNN and GNB"
)

# Plot for GNB model
```

```python
RocCurveDisplay.from_predictions(
    y_onehot_test.ravel(),
    y_score_gnb.ravel(),
    name="GNB micro-average OvR",
    color="blue",
    ax=display_knn.ax_,
    plot_chance_level=False,
)

# Show the plot
plt.show()
```