# mnist_dataset

```
[5]: #  Load MNIST Dataset for Handwritten Digits Recognition
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load MNIST dataset
mnist = fetch_openml("mnist_784", version=1)

# Extract features (X) and labels (y)
X_mnist, y_mnist = mnist.data, mnist.target.astype(int)     # Convert target to␣
 ↪integer

# Normalize pixel values (scale from 0-255 to 0-1)
X_mnist /= 255.0

# Split dataset into training (80%) and testing (20%)
X_train_mnist, X_test_mnist, y_train_mnist, y_test_mnist =␣
 ↪train_test_split(X_mnist, y_mnist, test_size=0.2, random_state=42)

print("MNIST Dataset Loaded Successfully!")
print(f"Training Samples: {len(X_train_mnist)}, Testing Samples:␣
 ↪{len(X_test_mnist)}")
```

```
MNIST Dataset Loaded Successfully!
Training Samples: 56000, Testing Samples: 14000
```

```
[7]: #  Train Logistic Regression Model for MNIST
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Train Logistic Regression Model
log_reg_mnist = LogisticRegression(max_iter=1000, solver="lbfgs",␣
 ↪multi_class="multinomial")
log_reg_mnist.fit(X_train_mnist, y_train_mnist)

# Predict on test data
```

```python
y_pred_mnist = log_reg_mnist.predict(X_test_mnist)

# Evaluate Model
print("\n Logistic Regression Accuracy (MNIST):", accuracy_score(y_test_mnist,␣
↪y_pred_mnist))
print("\n Classification Report (MNIST):\n",␣
↪classification_report(y_test_mnist, y_pred_mnist))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed
in 1.7. From then on, it will always use 'multinomial'. Leave it to its default
value to avoid this warning.
  warnings.warn(

 Logistic Regression Accuracy (MNIST): 0.9204285714285714

 Classification Report (MNIST):
               precision    recall  f1-score   support

           0       0.96      0.97      0.97      1343
           1       0.94      0.97      0.96      1600
           2       0.91      0.89      0.90      1380
           3       0.90      0.89      0.90      1433
           4       0.92      0.93      0.92      1295
           5       0.88      0.88      0.88      1273
           6       0.94      0.95      0.95      1396
           7       0.93      0.94      0.93      1503
           8       0.90      0.87      0.88      1357
           9       0.90      0.90      0.90      1420

    accuracy                           0.92     14000
   macro avg       0.92      0.92      0.92     14000
weighted avg       0.92      0.92      0.92     14000
```

[9]:
```python
#  Train Deep Learning Model for MNIST
import tensorflow as tf
from tensorflow import keras

# Define Neural Network Model
model = keras.Sequential([
    keras.layers.Dense(128, activation="relu", input_shape=(784,)),  # First␣
↪Hidden Layer
    keras.layers.Dense(64, activation="relu"),  # Second Hidden Layer
    keras.layers.Dense(10, activation="softmax") # Output Layer (10 classes␣
↪for digits 0-9)
```

```python
])

# Compile Model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
 metrics=["accuracy"])

# Train Model
model.fit(X_train_mnist, y_train_mnist, epochs=10, batch_size=32,
 validation_split=0.1, verbose=2)

# Evaluate Model
test_loss, test_acc = model.evaluate(X_test_mnist, y_test_mnist)
print("\n Deep Learning Accuracy (MNIST):", test_acc)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1575/1575 - 9s - 6ms/step - accuracy: 0.9239 - loss: 0.2553 - val_accuracy:
0.9618 - val_loss: 0.1302
Epoch 2/10
1575/1575 - 7s - 4ms/step - accuracy: 0.9666 - loss: 0.1074 - val_accuracy:
0.9680 - val_loss: 0.0991
Epoch 3/10
1575/1575 - 6s - 4ms/step - accuracy: 0.9777 - loss: 0.0742 - val_accuracy:
0.9757 - val_loss: 0.0821
Epoch 4/10
1575/1575 - 5s - 3ms/step - accuracy: 0.9818 - loss: 0.0548 - val_accuracy:
0.9764 - val_loss: 0.0796
Epoch 5/10
1575/1575 - 6s - 4ms/step - accuracy: 0.9857 - loss: 0.0436 - val_accuracy:
0.9789 - val_loss: 0.0820
Epoch 6/10
1575/1575 - 10s - 6ms/step - accuracy: 0.9884 - loss: 0.0354 - val_accuracy:
0.9748 - val_loss: 0.0968
Epoch 7/10
1575/1575 - 5s - 3ms/step - accuracy: 0.9904 - loss: 0.0281 - val_accuracy:
0.9770 - val_loss: 0.0886
Epoch 8/10
1575/1575 - 5s - 3ms/step - accuracy: 0.9927 - loss: 0.0225 - val_accuracy:
0.9759 - val_loss: 0.0905
Epoch 9/10
1575/1575 - 6s - 4ms/step - accuracy: 0.9923 - loss: 0.0225 - val_accuracy:
0.9737 - val_loss: 0.0988
Epoch 10/10
```

```
1575/1575 - 5s - 3ms/step - accuracy: 0.9940 - loss: 0.0177 - val_accuracy:
0.9795 - val_loss: 0.0936
438/438                 1s 2ms/step -
accuracy: 0.9740 - loss: 0.1044

 Deep Learning Accuracy (MNIST): 0.975428581237793
```

[10]:
```
# -*- coding: utf-8 -*-
"""
 **MNIST Handwritten Digits Classification - Theory & Analysis**
 This Colab notebook explains how Logistic Regression and Neural Networks
   are applied to the MNIST dataset for handwritten digit classification.
"""

#   **Introduction**
"""
 The **MNIST Dataset** consists of **70,000 grayscale images (28x28 pixels)**
   of handwritten digits (0-9).
 It is widely used in **Machine Learning & Deep Learning** to test␣
 ↪classification  models.
 The goal is to classify an image into one of **10 digit classes (0-9)**.

 **Why MNIST?**
   - Standard benchmark dataset for image recognition.
   - Used to compare performance of traditional ML models vs. Deep Learning.
"""

#   **Dataset Processing**
"""
 We load the MNIST dataset using `fetch_openml()` from Scikit-Learn.
 Images are **flattened into 784-pixel feature vectors** (28x28 = 784).
 We **normalize pixel values (0-255 → 0-1)** to improve model training.
 The dataset is **split into 80% training & 20% testing**.
"""

#   **Logistic Regression for MNIST Classification**
"""
 Logistic Regression is applied to classify digits (0-9).
 Since this is a **multi-class problem**, we use `multi_class="multinomial"`.
 The model is trained using **1000 iterations** (`max_iter=1000`).
 It is evaluated using **accuracy & classification report**.

 **Results:**
   - Logistic Regression achieves **85-90% accuracy** on MNIST.
   - Works well for small datasets but struggles with complex patterns.
"""
```

```
#   **Deep Learning Model (Neural Network)**
"""
  We train a **Neural Network** (Multilayer Perceptron - MLP) using TensorFlow/Keras.
  Architecture:
    - **128 neurons** (ReLU activation) - First Hidden Layer
    - **64 neurons** (ReLU activation) - Second Hidden Layer
    - **10 neurons** (Softmax activation) - Output Layer (for digits 0-9)
  The model is trained for **10 epochs** with **batch size 32**.

  **Results:**
    - Neural Network achieves **97-99% accuracy**.
    - Outperforms Logistic Regression significantly.
    - Captures complex patterns in images better than traditional ML.
"""

#   **Comparison of Results**
"""
| **Model**                | **Accuracy (%)** |
|--------------------------|------------------|
| **Logistic Regression**  | **85-90%** |
| **Neural Network (MLP)** | **97-99%** |

  **Deep Learning outperforms Logistic Regression** for MNIST.
  **Neural Networks learn complex patterns** better than linear classifiers.
"""

#   **Conclusion**
"""
1  **Logistic Regression** is effective for tabular data but **not ideal for images**.
2  **Neural Networks** can capture complex patterns in **image data**, leading to **higher accuracy**.
3  **For even better results, CNNs (Convolutional Neural Networks) should be used.**
"""

#   ** Next Steps & Improvements**
"""
  Try **Convolutional Neural Networks (CNNs)** for state-of-the-art performance.
  Use **Dropout Layers** in Deep Learning to prevent overfitting.
  Experiment with **different activation functions (Leaky ReLU, ELU)**.
  Increase **epochs & batch size** for further accuracy improvements.
"""
```

```
[10]: '\n  Try **Convolutional Neural Networks (CNNs)** for state-of-the-art
performance.\n  Use **Dropout Layers** in Deep Learning to prevent
overfitting.\n  Experiment with **different activation functions (Leaky ReLU,
ELU)**.\n  Increase **epochs & batch size** for further accuracy
improvements.\n'
```