Digit Recognizer

Description

MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Goal

Your goal is to **correctly identify digits from a dataset of tens of thousands of handwritten images**.

Steps to perform

Load the Data - test.csv and train.csv (Using Pandas)
Pre-Process the data set by Splitting into features and target values
Optionally split further into train/validation sets using train_test_split() to evaluate during training
Normalize/Standardize the Features
Chose and Train the Model
Make Predictions
Evaluate the Model (Measure the error or loss)
Adjust parameters (weight and bias) to reduce the error
Repeat Steps 5 to 8, until the error is minimized (converged)

In [2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [4]:
```
 # Load the data
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")

print("Sample Training data............")
print(train_data.head())

print("Sample Testing data............")
print(test_data.head())
```

```
Sample Training data............
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
Sample Testing data............
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
4       0       0       0       0       0       0       0       0       0

   pixel9  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

In [5]:
```python
print(f"train_data dimension - {train_data.shape}") # 1st Column i.e. "label" is th
print(f"test_data dimension - {test_data.shape}")
```

```
train_data dimension - (42000, 785)
test_data dimension - (28000, 784)
```

In [6]:
```python
# Split the Data into Features and Values
X = train_data.drop("label", axis=1) # axis=1 means "drop column" (not row)
y = train_data["label"]
```

```
print(f"Features Data: \n {X}")
print("\n\n")
print(f"Target Data: \n {y}")
```

```
Features Data:
       pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0           0       0       0       0       0       0       0       0       0
1           0       0       0       0       0       0       0       0       0
2           0       0       0       0       0       0       0       0       0
3           0       0       0       0       0       0       0       0       0
4           0       0       0       0       0       0       0       0       0
...       ...     ...     ...     ...     ...     ...     ...     ...     ...
41995       0       0       0       0       0       0       0       0       0
41996       0       0       0       0       0       0       0       0       0
41997       0       0       0       0       0       0       0       0       0
41998       0       0       0       0       0       0       0       0       0
41999       0       0       0       0       0       0       0       0       0

       pixel9  ...  pixel774  pixel775  pixel776  pixel777  pixel778  \
0           0  ...         0         0         0         0         0
1           0  ...         0         0         0         0         0
2           0  ...         0         0         0         0         0
3           0  ...         0         0         0         0         0
4           0  ...         0         0         0         0         0
...       ...  ...       ...       ...       ...       ...       ...
41995       0  ...         0         0         0         0         0
41996       0  ...         0         0         0         0         0
41997       0  ...         0         0         0         0         0
41998       0  ...         0         0         0         0         0
41999       0  ...         0         0         0         0         0

       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
...         ...       ...       ...       ...       ...
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 784 columns]



Target Data:
 0        1
1        0
2        1
3        4
4        0
        ..
41995    0
41996    1
41997    7
41998    6
```

```
41999    9
Name: label, Length: 42000, dtype: int64
```

In [7]:
```
# Normalize the features (scale 0-255 to 0-1)
X = X / 255.0
test_data = test_data / 255.0
```

In [8]:
```
from sklearn.model_selection import train_test_split

X_train, X_validate, y_train, y_validate = train_test_split(X, y, test_size=0.2, ra
print(f"Training set size  : {X_train.shape[0]} samples")
print(f"Validation set size: {X_validate.shape[0]} samples")
```

```
Training set size  : 33600 samples
Validation set size: 8400 samples
```

In [9]:
```
# Apply PCA to improve the accuracy further
# from sklearn.decomposition import PCA

# pca = PCA(n_components=0.95, random_state=42)  # Keep 95% variance
# X_train_pca = pca.fit_transform(X_train)
# X_validate_pca = pca.transform(X_validate)
```

In [10]:
```
# Chose and Train the Model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# model = LogisticRegression(solver='lbfgs', max_iter=1000, multi_class='multinomia
model = RandomForestClassifier(n_estimators=100, random_state=42)
# model = LogisticRegression(max_iter=1000)
# model.fit(X_train, y_train)
model.fit(X_train, y_train)
print("Model training completed!")
```

```
Model training completed!
```

In [11]:
```
# Make Predictions
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# y_pred = model.predict(X_validate)
y_pred = model.predict(X_validate)
accuracy = accuracy_score(y_validate, y_pred)
print(f"\nModel Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_report(y_validate, y_pred))
```

```
Model Accuracy: 0.96

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       816
           1       0.98      0.99      0.99       909
           2       0.96      0.96      0.96       846
           3       0.96      0.95      0.96       937
           4       0.96      0.97      0.96       839
           5       0.96      0.96      0.96       702
           6       0.96      0.98      0.97       785
           7       0.97      0.95      0.96       893
           8       0.95      0.95      0.95       835
           9       0.93      0.94      0.94       838

    accuracy                           0.96      8400
   macro avg       0.96      0.96      0.96      8400
weighted avg       0.96      0.96      0.96      8400
```

In [ ]: