Assignment: SVD Preprocessing on MNIST with Logistic Regression

Instructions:

In this assignment, you will apply **Singular Value Decomposition (SVD)** as a preprocessing step to the **MNIST dataset** and train a **logistic regression classifier**. You will compare the model performance and training time when using different levels of SVD for dimensionality reduction.

In this assignment, you will need to:

1. Load the MNIST dataset and normalize it.
2. Perform SVD and reduce the dimensions of the data.
3. Train a logistic regression model on the original and SVD-reduced data.
4. Measure and compare the training time and accuracy of the model with varying SVD components.
5. Plot the results and analyze how SVD impacts the performance and efficiency of the model.

Your tasks include:

1. Implement SVD algorithm. You are not allowed to directly use SVD implemented by other packages, but you may use functions in NumPy. (Part 2)
2. Explore the accuracy and time performance from different numbers of SVD components. (Part 4)
3. Visualize the accuracy, time performance and top 5 singular vectors in the dataset, analyze and explain which number of SVD component looks best to you? (Part 4,5&6) Hint: singular vectors should be reshaped to 28x28 images for visualization.

Note that you may not import any other function or package. Let's get started!

Part 1: Load the MNIST dataset and preprocess the data

```
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from sklearn.datasets import fetch_openml
from sklearn.metrics import accuracy_score, classification_report

# Load MNIST dataset
print("Loading MNIST dataset...")
mnist = fetch_openml('mnist_784', version=1)
X = mnist.data
y = mnist.target

# Normalize the data
X = X / 255.0

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

```
Loading MNIST dataset...
```

Part 2: Implement SVD for Dimensionality Reduction

```
def apply_svd_custom(X_train, X_test, n_components):
    """
    Perform custom SVD decomposition and reduce dimensions to n_components.
    """
    # Perform SVD on the training set
    U_train, s_train, Vt_train = np.linalg.svd(X_train, full_matrices=False)

    # Reduce the dimensionality using the top n_components
    U_train_reduced = U_train[:, :n_components]
    S_train_reduced = np.diag(s_train[:n_components])
    Vt_train_reduced = Vt_train[:n_components, :]

    # Reconstruct the compressed version of the training set
    X_train_svd = np.dot(U_train_reduced, np.dot(S_train_reduced, Vt_train_reduced))

    # Do the same for the test set
    U_test, s_test, Vt_test = np.linalg.svd(X_test, full_matrices=False)
    U_test_reduced = U_test[:, :n_components]
    S_test_reduced = np.diag(s_test[:n_components])
    Vt_test_reduced = Vt_test[:n_components, :]
    X_test_svd = np.dot(U_test_reduced, np.dot(S_test_reduced, Vt_test_reduced))

    return X_train_svd, X_test_svd
```

Part 3: Train Logistic Regression and Measure Performance

```
# Function to train logistic regression and track training time
def train_logistic_regression(X_train, y_train, X_test, y_test):
    model = LogisticRegression(max_iter=1000, solver='saga', random_state=42, multi_

    # Measure training time
    start_time = time.time()
    model.fit(X_train, y_train)
    training_time = time.time() - start_time

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy, training_time
```

Part 4: Experiment with Different Levels of SVD

Now, apply SVD with varying numbers of components and observe how the dimensionality reduction impacts the model's performance. Record both the accuracy and training time for each number of components.

```
svd_components = [784, 300, 150, 100, 50, 20]  # Experiment with various numbers of

# Store the results
results = []

print("Training models with different levels of SVD preprocessing...")
for n_components in svd_components:
    print(f"Applying custom SVD with {n_components} components...")

    # Apply SVD
    X_train_svd, X_test_svd = apply_svd_custom(X_train, X_test, n_components)

    # Train logistic regression and get accuracy and training time
    accuracy, training_time = train_logistic_regression(X_train_svd, y_train, X_test

    # Store results
    results.append((n_components, accuracy, training_time))

    print(f"SVD components: {n_components}, Accuracy: {accuracy:.4f}, Training time:
```

```
Training models with different levels of SVD preprocessing...
Applying custom SVD with 784 components...
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 784, Accuracy: 0.9209, Training time: 690.2419 seconds
Applying custom SVD with 300 components...
```

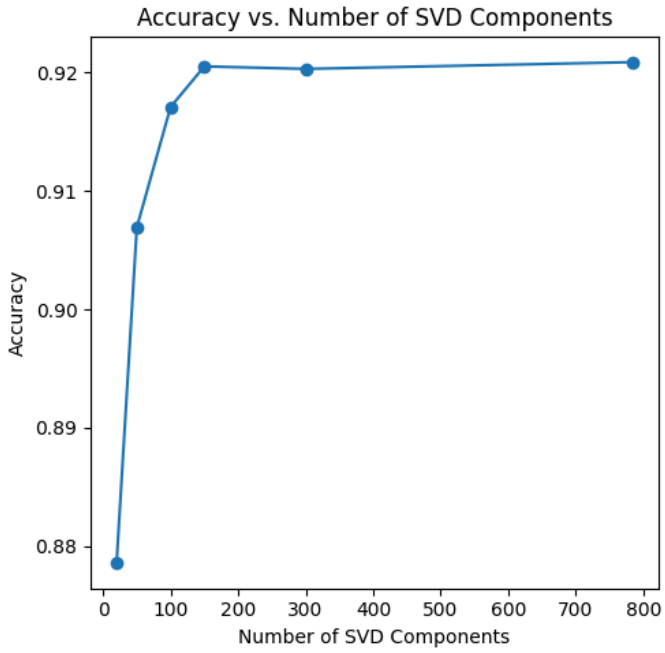
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 300, Accuracy: 0.9203, Training time: 530.6958 seconds
Applying custom SVD with 150 components...
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 150, Accuracy: 0.9205, Training time: 479.7990 seconds
Applying custom SVD with 100 components...
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 100, Accuracy: 0.9171, Training time: 404.7732 seconds
Applying custom SVD with 50 components...
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 50, Accuracy: 0.9069, Training time: 169.0424 seconds
Applying custom SVD with 20 components...
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
  warnings.warn(
SVD components: 20, Accuracy: 0.8786, Training time: 28.5699 seconds
```

Part 5: Visualize and Analyze the Results

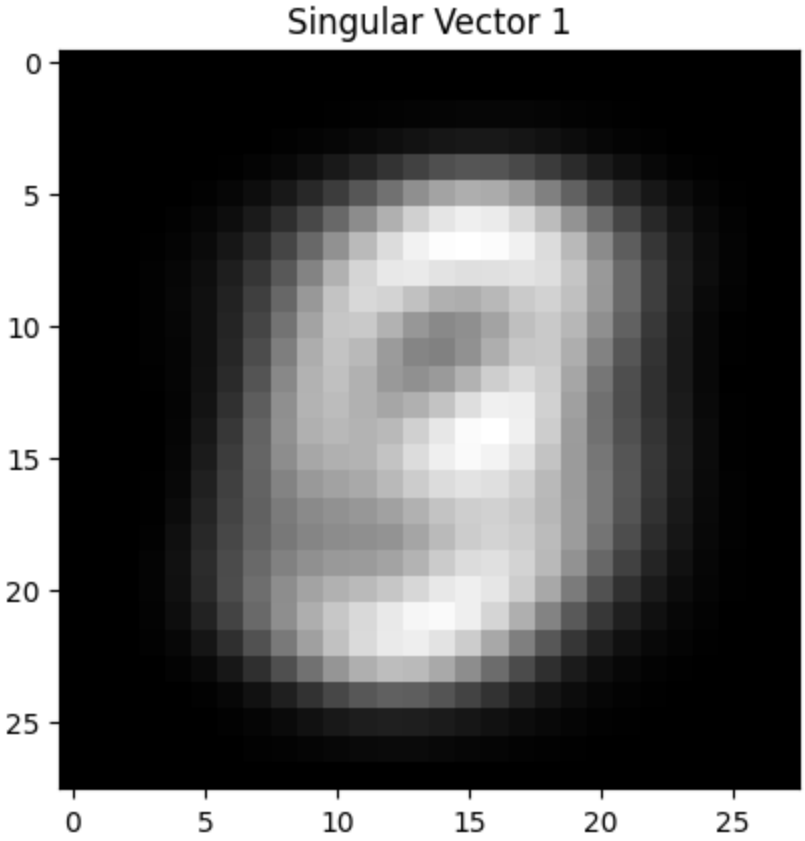
Finally, plot the accuracy, training time as a function of the number of SVD components, and top 5 singular vectors. This will help you understand the trade-off between dimensionality reduction, accuracy, and model training time, and how SVD generally works. Hint: singular vectors should be reshaped to 28x28 images for visualization.

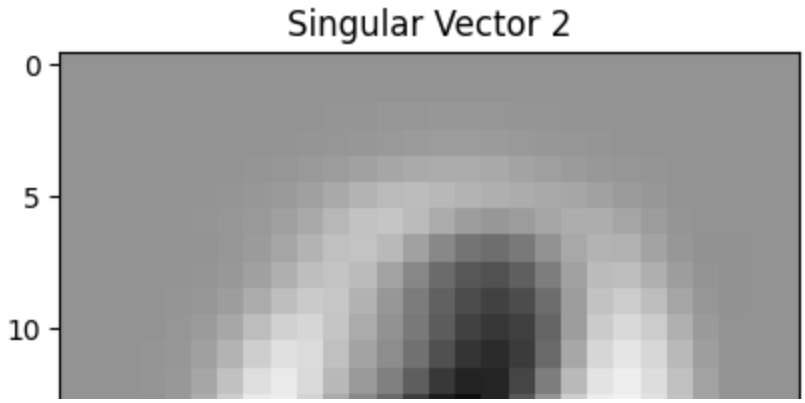
```
# Extract the results
components = [r[0] for r in results]
accuracies = [r[1] for r in results]
training_times = [r[2] for r in results]

# Plot accuracy vs SVD components
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(components, accuracies, marker='o')
plt.xlabel('Number of SVD Components')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Number of SVD Components')

# Plot training time vs SVD components
plt.subplot(1, 2, 2)
plt.plot(components, training_times, marker='o')
plt.xlabel('Number of SVD Components')
plt.ylabel('Training Time (seconds)')
plt.title('Training Time vs. Number of SVD Components')

plt.tight_layout()
plt.show()
```

```
# Visualize the top 5 singular vectors
U, s, Vt = np.linalg.svd(X_train, full_matrices=False)
for i in range(5):
    plt.figure()
    singular_vector = Vt[i].reshape(28, 28)
    plt.imshow(singular_vector, cmap='gray')
    plt.title(f'Singular Vector {i+1}')
    plt.show()
```


Accuracy vs. Number of SVD Components
Accuracy
Number of SVD Components
0.92
0.91
0.90
0.89
0.88
0
100
200
300
400
500
600
700
800


Training Time vs. Number of SVD Components
Training Time (seconds)
Number of SVD Components
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
800


Singular Vector 1
0
5
10
15
20
25


Singular Vector 2
0
5
10