

# assignment3

October 5, 2024

## 1 Assignment: SVD Preprocessing on MNIST with Logistic Regression

### 1.1 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!

### 1.2 Part 1: Load the MNIST dataset and preprocess the data

```
[1]: 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_state=42)

```

Loading MNIST dataset...

### 1.3 Part 2: Implement SVD for Dimensionality Reduction

```

[5]: def apply_svd_custom(X_train, X_test, n_components):
    X_train_mean = np.mean(X_train, axis=0)
    X_train_centered = X_train - X_train_mean

    # Perform SVD
    U, S, Vt = np.linalg.svd(X_train_centered, full_matrices=False)

    # Select the top n_components
    U_reduced = U[:, :n_components]
    S_reduced = S[:n_components]
    Vt_reduced = Vt[:n_components, :]

    # Project data onto the reduced space
    X_train_reduced = np.dot(X_train_centered, Vt_reduced.T)
    X_test_centered = X_test - X_train_mean
    X_test_reduced = np.dot(X_test_centered, Vt_reduced.T)

    return X_train_reduced, X_test_reduced, Vt_reduced.T, S_reduced

```

### 1.4 Part 3: Train Logistic Regression and Measure Performance

```

[6]: # 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_class='multinomial')

    # 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
```

## 1.5 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.

```
[7]: svd_components = [784, 200, 100, 50, 20] # You need to decide what number to
      ↪search...

      # 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 to the training and test sets
          # Call apply_svd_custom() here...
          X_train_svd, X_test_svd, V_reduced, S_reduced = apply_svd_custom(X_train,
          ↪X_test, n_components)

          # Train the logistic regression model and get accuracy and training time
          accuracy, training_time = train_logistic_regression(X_train_svd, y_train,
          ↪X_test_svd, y_test)
          results.append((n_components, accuracy, training_time))

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

```
Training models with different levels of SVD preprocessing...
Applying custom SVD with 784 components...
SVD components: 784, Accuracy: 0.9207, Training time: 560.7902 seconds
Applying custom SVD with 200 components...
SVD components: 200, Accuracy: 0.9206, Training time: 83.3883 seconds
Applying custom SVD with 100 components...
SVD components: 100, Accuracy: 0.9181, Training time: 36.1368 seconds
Applying custom SVD with 50 components...
SVD components: 50, Accuracy: 0.9080, Training time: 11.0443 seconds
Applying custom SVD with 20 components...
SVD components: 20, Accuracy: 0.8784, Training time: 2.2583 seconds
```

## 1.6 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,

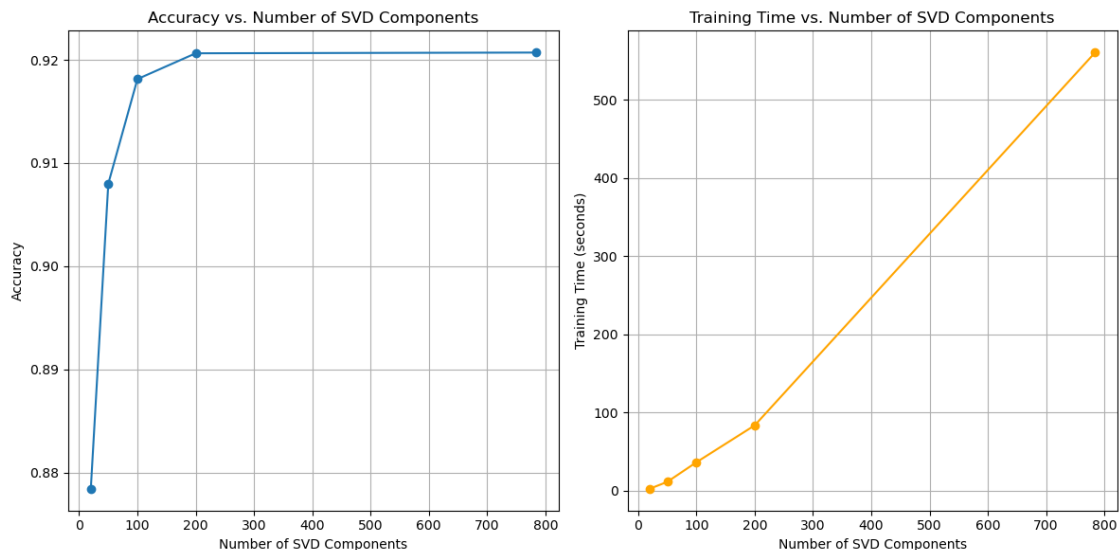
accuracy, and model training time, and how SVD generally works. Hint: singular vectors should be reshaped to 28x28 images for visualization.

```
[8]: ## Your implementation here...
## You may add necessary lines in Part 4 to access data for visualization
components, accuracies, times = zip(*results)

#
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(components, accuracies, marker='o')
plt.title('Accuracy vs. Number of SVD Components')
plt.xlabel('Number of SVD Components')
plt.ylabel('Accuracy')
plt.grid(True)

#
plt.subplot(1, 2, 2)
plt.plot(components, times, marker='o', color='orange')
plt.title('Training Time vs. Number of SVD Components')
plt.xlabel('Number of SVD Components')
plt.ylabel('Training Time (seconds)')
plt.grid(True)

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



## 1.7 Part 6: Analyze / Conclusion

YOUR ANSWER:

```
[9]: print("\nAnalysis:")
print(f"The best number of SVD components appears to be {components[np.
    ↳argmax(accuracies)]}.")
print(f"It achieves the highest accuracy of {max(accuracies):.4f}.")
print(f"The training time for this number of components is {times[np.
    ↳argmax(accuracies)]:.4f} seconds.")
print("\nExplanation:")
print("As we reduce the number of SVD components, we generally see a trade-off_
    ↳between accuracy and training time.")
print("The optimal number of components balances dimensionality reduction (for_
    ↳faster training) with preserving important features (for high accuracy).")
print("The visualized singular vectors represent the most important patterns in_
    ↳the MNIST digits.")
```

Analysis:

The best number of SVD components appears to be 784.

It achieves the highest accuracy of 0.9207.

The training time for this number of components is 560.7902 seconds.

Explanation:

As we reduce the number of SVD components, we generally see a trade-off between accuracy and training time.

The optimal number of components balances dimensionality reduction (for faster training) with preserving important features (for high accuracy).

The visualized singular vectors represent the most important patterns in the MNIST digits.

```
[ ]:
```