# Lab Sheet -11

# Dimensionality Reduction Methods

---------------------------------------------------------------------------------------------------

## INTRODUCTION:

In this labsheet, we will be discussing several dimensionality reduction techniques and try to perform tasks like classification. At the end, your job is to check if reducing the dimensions increased your accuracy. There will also be some questions which are generally things you should ask yourself before choosing a model.

## Question 1: Why reducing the dimensions?

1.**Dimensionality reduction** helps in **removing noise** and redundant features, thereby **simplifying the model** and making it less prone to overfitting.
2.Can be used as a **feature extraction** method, transforming the original features into a new set of features that are more informative and, in many cases, more compact.
3.**Processing high-dimensional data** is computationally expensive. Reducing the number of features can significantly speed up model training and inference time.

## Question 2: When do you think you should use these techniques?

**ABOUT THE DATASET:**

The **MNIST (Modified National Institute of Standards and Technology)** dataset is a large collection of handwritten digits, commonly used for training and evaluating image classification models. Here's a brief description of the dataset:

## Overview:

- **Size**: The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9).
  - **Training set**: 60,000 images.
  - **Test set**: 10,000 images.
- **Image Dimensions**: Each image is 28x28 pixels (784 total pixels).
- **Image Type**: Grayscale images, meaning each pixel has an intensity value between 0 and 255 (with 0 being black and 255 being white).
- **Labels**: Each image corresponds to a label representing the digit (0 through 9) depicted in the image.
- **Format**: The images are flattened into 1D vectors of size 784 (28x28), and the labels are integers representing the digit.

## DIFFERENT TECHNIQUES:

1.PRINCIPAL COMPONENT ANALYSIS
2.LINEAR DISCRIMINANT ANALYSIS
3.t-Distributed Stochastic Neighbor Embedding

## PRINCIPAL COMPONENT ANALYSIS:

**Principal Component Analysis (PCA)** is a statistical technique used for dimensionality reduction while preserving as much variance (information) in the data as possible.PCA identifies the **principal components** of the data, which are new variables (or features) created as linear combinations of the original features. These principal components capture the **directions of maximum variance** in the data. After transforming the data using PCA, the first few principal components usually account for most of the variability in the original dataset. In contrast, the later components account for less important variations.

CLASSIFICATION USING PCA:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 1. Load the MNIST dataset
# Fetch the MNIST dataset (28x28 images of digits 0-9)
mnist = fetch_openml('mnist_784', version=1)
X = mnist.data  # The pixel values of the images
y = mnist.target.astype(int)  # The corresponding labels (digits 0-9)

# 2. Standardize the data (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Apply PCA to reduce the dimensionality
pca = PCA(n_components=50)  # Reduce to 50 components
X_pca = pca.fit_transform(X_scaled)

# 4. Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.3, random_state=42)

# 5. Train a Logistic Regression classifier on the reduced data
clf = LogisticRegression(max_iter=10000)
clf.fit(X_train, y_train)

# 6. Make predictions on the test set
y_pred = clf.predict(X_test)

# 7. Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression classifier on PCA-reduced data:
{accuracy * 100:.2f}%')
```

## QUESTIONS:

1. Change the number of principal components and observe the change in accuracy.
2. Use the same dataset but now, do not perform any reduction and observe the accuracy?(you might also observe that it takes more time)
3. Look out for the hyperparameters from the documentation and try modifying them.

## Linear Discriminant Analysis:

**Linear Discriminant Analysis (LDA)** is a supervised dimensionality reduction technique primarily used for classification tasks. Unlike **Principal Component Analysis (PCA)**, which is unsupervised and focuses on preserving the overall variance in the data, **LDA** aims to find a projection that maximizes the separation between multiple classes.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)  # Reduce to 2 components
for visualization
X_lda = lda.fit_transform(X_scaled, y)

# 4. Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_lda, y,
test_size=0.3, random_state=42)

# 5. Train a Logistic Regression classifier on the LDA-reduced data
clf = LogisticRegression(max_iter=10000)
```

```
clf.fit(X_train, y_train)

# 6. Make predictions on the test set
y_pred = clf.predict(X_test)

# 7. Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression classifier on LDA-reduced data:
{accuracy * 100:.2f}%')
```

QUESTIONS:

1. Anything unusual about the accuracy for classification? Why do you think it is so?
2. Look out for the hyperparameters from the documentation and try modifying them.
3. What is the difference between pca and lda? Do you lda is good for non-linear data like images?

## t-Distributed Stochastic Neighbor Embedding:

**t-SNE** is a **non-linear dimensionality reduction** technique primarily used for the **visualization of high-dimensional data** in a lower-dimensional space (typically 2D or 3D). It is particularly useful for visualizing complex datasets such as image data, text data, or other high-dimensional structures where traditional linear techniques like **PCA** may not perform well.t-SNE is designed to preserve the **local structure** of the data by ensuring that points that are close in the high-dimensional space are also close in the lower-dimensional space.

PS. It might take a while to transform the data. If it keeps running even after 2-3 min . Try to understand how the algorithm works.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 1. Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X = mnist.data  # The pixel values of the images
y = mnist.target.astype(int)  # The corresponding labels (digits 0-9)
```

```
# 2. Standardize the data (important for most dimensionality reduction
techniques)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

# 4. Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tsne, y,
test_size=0.3, random_state=42)

# 5. Train a Logistic Regression classifier on the t-SNE reduced data
clf = LogisticRegression(max_iter=10000)
clf.fit(X_train, y_train)

# 6. Make predictions on the test set
y_pred = clf.predict(X_test)

# 7. Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression classifier on t-SNE-reduced data:
{accuracy * 100:.2f}%')
```

FINAL TASK:
Try to use the wine dataset from sklearn.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Wine dataset
wine = load_wine()
X = wine.data  # Features
y = wine.target  # Labels
```

1. Apply the reduction techniques and perform classification.[difference is that it is a linear dataset]
2. Also try to write some functions using matplotlib to visulize the various components.