Experiment: Principal Component Analysis (PCA) vs Linear Discriminant Analysis (LDA) vs T-distributed Stochastic Neighbour Embedding (t-SNE) vs Multi-Dimensional Scaling (MDS)

Title:

Implement a Multidimensionality Scaling Algorithm MDS on a specific dataset and compare its outcomes with other dimensionality reduction techniques such as PCA LDA and T-SNE

Aim:

Comparing the results of MDS with LDA, PCA and t-SNE for better suitability

Objective:

Students will learn:

- The implementation of the Multi-Dimensional Scaling, principal component analysis and Linear Discriminant analysis and T-distributed stochastic neighbour embedding on a dataset.
- Visualization and interpretation of results.

Problem Statement

APPLY AND IMPLEMENT MDS ALGORITHM ON A SPECIFIC DATASET OF YOUR CHOICE AND COMPARE THE OUTCOMES WITH T-SNE, PCA AND LDA FOR THE SAME

Explanation / Stepwise Procedure / Algorithm

Principal Component Analysis (PCA)

PCA is a widely used dimensionality reduction technique that transforms high-dimensional data into lower-dimensional data while retaining most of the information. The goal of PCA is to identify the directions (principal components) in which the data varies the most and project the data onto those directions.

Here's how PCA works:

1. Standardize the data by subtracting the mean and dividing by the standard deviation.
2. Compute the covariance matrix of the standardized data.

3. Perform eigenvalue decomposition on the covariance matrix to obtain eigenvectors and eigenvalues.
4. Select the top k eigenvectors corresponding to the largest eigenvalues.
5. Project the original data onto the selected eigenvectors to obtain the lower-dimensional representation.

PCA is useful for:

- Reducing the dimensionality of high-dimensional data
- Identifying patterns and correlations in the data
- Improving the performance of machine learning algorithms

Linear Discriminant Analysis (LDA)

LDA is a supervised dimensionality reduction technique that finds a linear combination of features that best separates different classes in the data. LDA is commonly used for classification problems and is particularly useful when the number of features is high and the number of samples is small.

Here's how LDA works:

1. Standardize the data by subtracting the mean and dividing by the standard deviation.
2. Compute the within-class scatter matrix and the between-class scatter matrix.
3. Perform eigenvalue decomposition on the scatter matrices to obtain eigenvectors and eigenvalues.
4. Select the top k eigenvectors corresponding to the largest eigenvalues.
5. Project the original data onto the selected eigenvectors to obtain the lower-dimensional representation.

LDA is useful for:

- Reducing the dimensionality of high-dimensional data
- Improving the performance of classification algorithms
- Identifying the most discriminative features for classification

---

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a non-linear dimensionality reduction technique that maps high-dimensional data to lower-dimensional data while preserving local structure. t-SNE is particularly useful for visualizing high-dimensional data and identifying patterns or clusters that may not be apparent in the original data.

Here's how t-SNE works:

1. Compute the similarity matrix of the high-dimensional data using a Gaussian kernel.

2. Convert the similarity matrix into a joint probability distribution.
3. Define a cost function that measures the difference between the joint probability distribution of the original data and the lower-dimensional representation.
4. Minimize the cost function using gradient descent to obtain the lower-dimensional representation.

t-SNE is useful for:

- Visualizing high-dimensional data in a lower-dimensional space
- Identifying patterns and clusters in the data
- Reducing the dimensionality of high-dimensional data while preserving local structure

---

Multidimensional Scaling (MDS)

MDS is a dimensionality reduction technique that represents high-dimensional data in a lower-dimensional space while preserving the pairwise distances between points as accurately as possible. MDS is particularly useful for visualizing similarity or dissimilarity data and understanding relationships between data points.

Here's how MDS works:

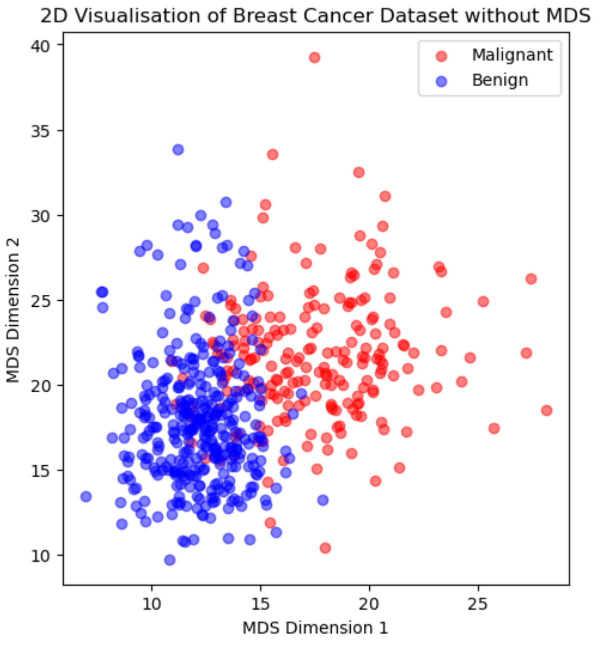
1. Compute the pairwise dissimilarity matrix using an appropriate distance metric (e.g., Euclidean, Manhattan).
2. Convert the dissimilarity matrix into a form suitable for eigenvalue decomposition (for Classical MDS) or define a cost function based on stress minimization (for Non-Metric MDS).
3. Perform eigenvalue decomposition or use an iterative optimization algorithm to compute the lower-dimensional representation.
4. Select the top k dimensions that best preserve the original distances.
5. Assign each data point new coordinates in the lower-dimensional space.

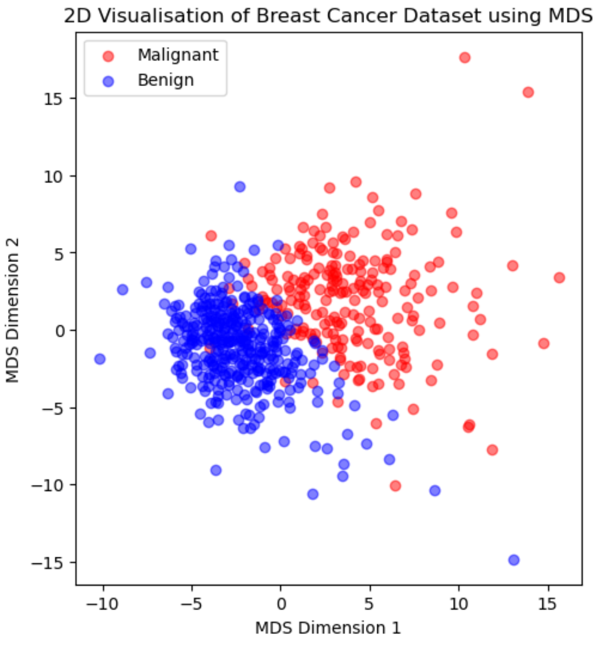
MDS is useful for:

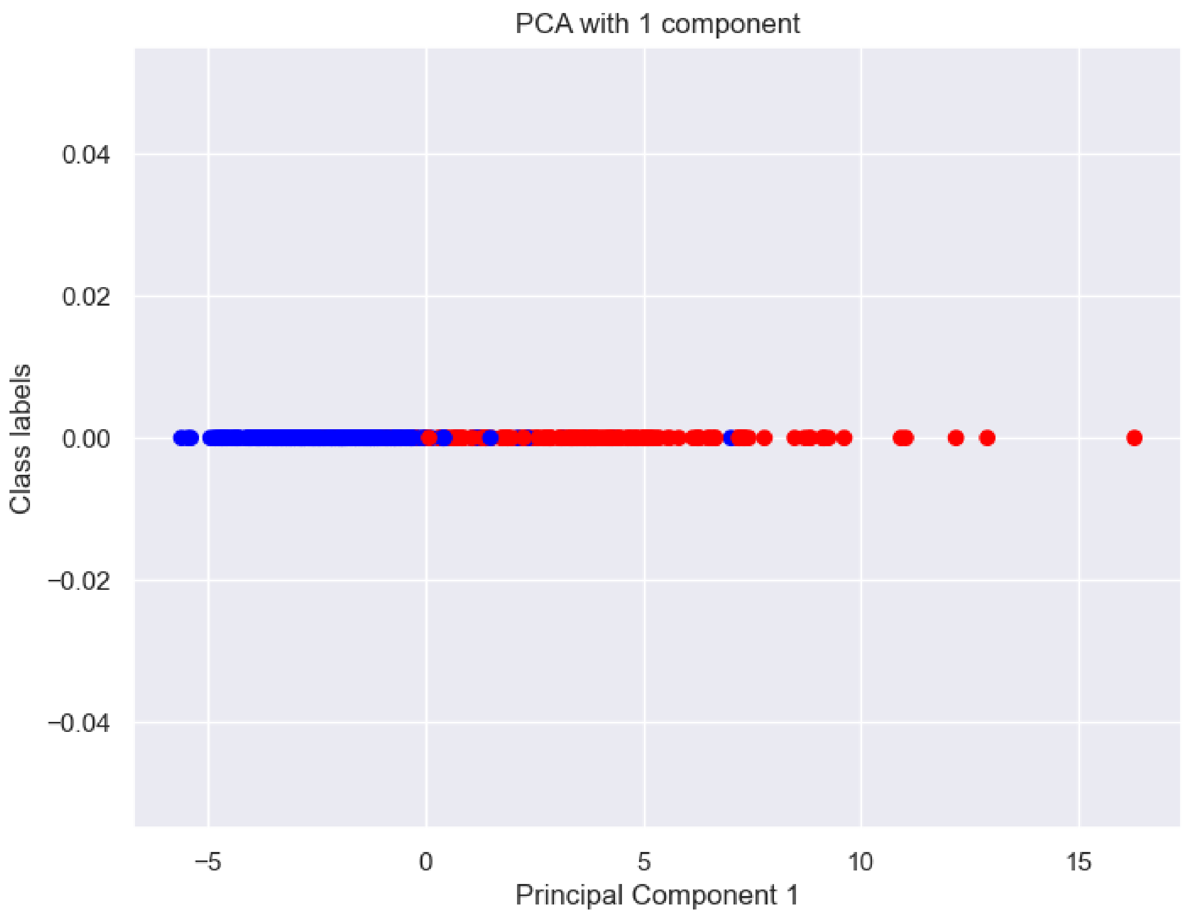
- Visualizing high-dimensional data in 2D or 3D
- Understanding relationships and similarities between data points
- Reducing dimensionality while maintaining pairwise distances
- Analyzing preference or similarity data in psychology and market research

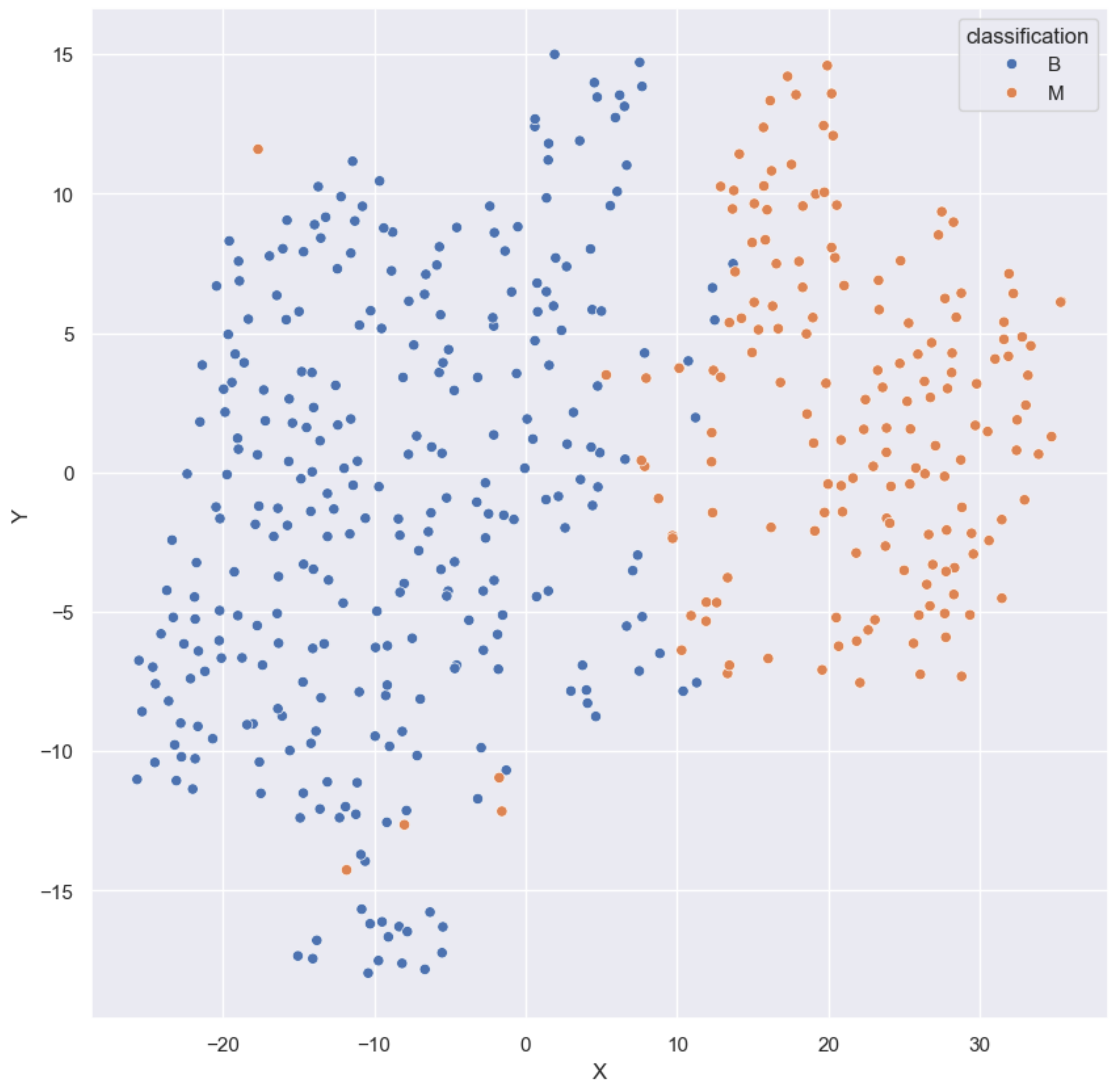
Figures/Diagrams

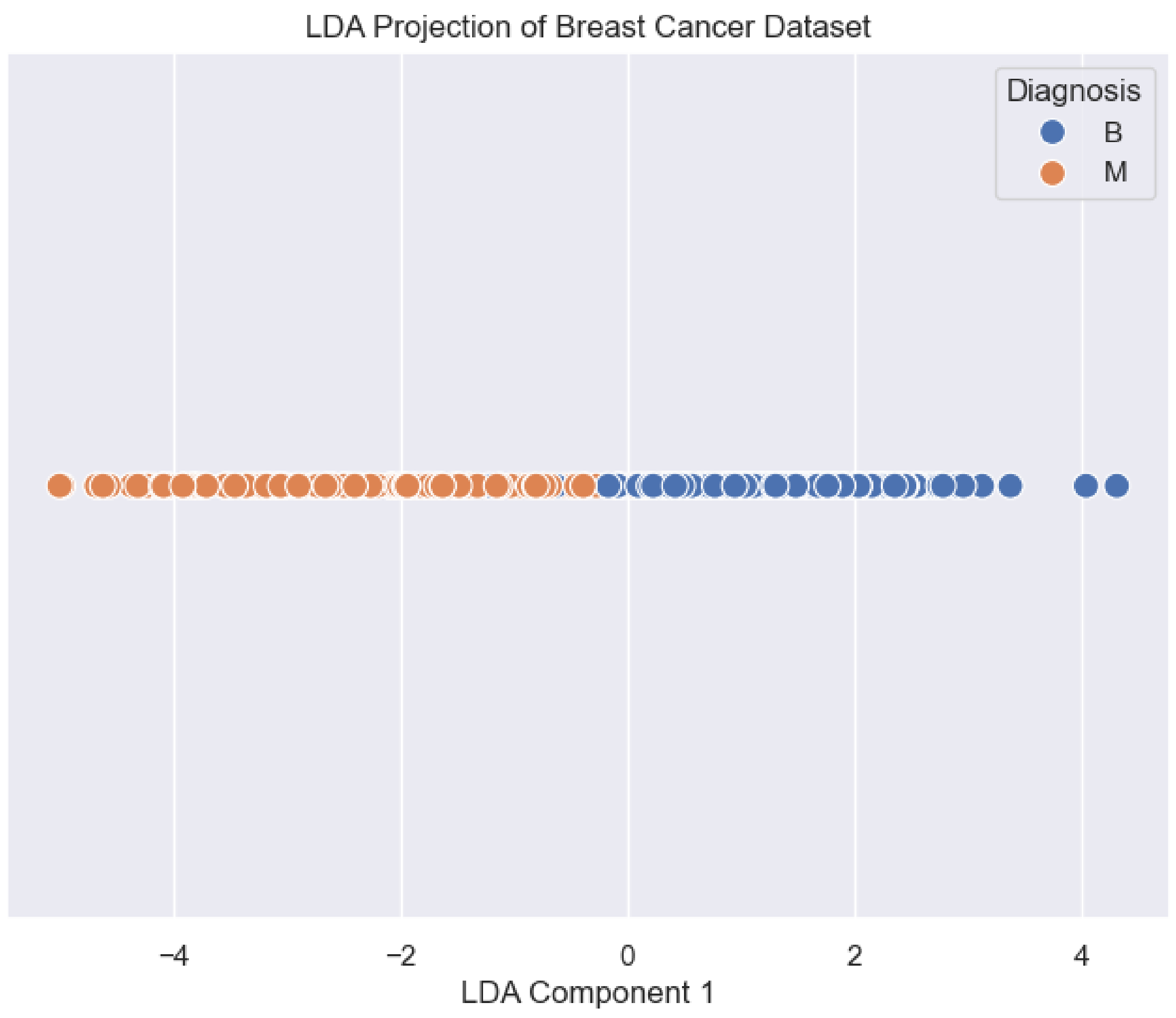
- MDS and LDA, PCA and t-SNE plots plotted for the dataset.
- Comparison between MDS, LDA,PCA and t-SNE.


2D Visualisation of Breast Cancer Dataset without MDS
Malignant
Benign
MDS Dimension 1
MDS Dimension 2


2D Visualisation of Breast Cancer Dataset using MDS
Malignant
Benign
MDS Dimension 1
MDS Dimension 2


PCA with 1 component
Class labels
Principal Component 1
0.04
0.02
0.00
−0.02
−0.04
−5
0
5
10
15


classification
B
M
15
10
5
0
−5
−10
−15
Y
−20
−10
0
10
20
30
X



Challenges Encountered

1. Parameter Sensitivity – MDS requires selecting an appropriate distance metric and the number of dimensions for embedding. Incorrect choices can lead to misleading representations.
2. Computational Complexity – Classical MDS involves eigenvalue decomposition, which can be computationally expensive for large datasets, making it slower than PCA but comparable to t-SNE.
3. Interpretability – Unlike PCA, which preserves variance, or LDA, which focuses on class separation, MDS focuses on pairwise distances, making the results harder to interpret in certain contexts.
4. Comparison with Other Techniques – MDS is an unsupervised method like t-SNE and PCA, whereas LDA is supervised. Comparing them requires analyzing both global and local structures, as well as the effectiveness of dimensionality reduction for visualization and classification.

Conclusion

- Implementing MDS on a specific dataset provides valuable insights into how pairwise distances are preserved in lower dimensions. However, careful selection of distance metrics is crucial for meaningful results.
- Compared to PCA, which focuses on variance maximization, and LDA, which optimizes class separation, MDS prioritizes preserving the original distance relationships, making it useful for exploratory data analysis.
- t-SNE emphasizes local structures, while MDS aims for global distance preservation, leading to different visualization outcomes.
- The choice between MDS, PCA, LDA, and t-SNE depends on the dataset and the specific goals of dimensionality reduction—whether for classification, clustering, or visualization.

MDS

February 24, 2025

0.1 MDS

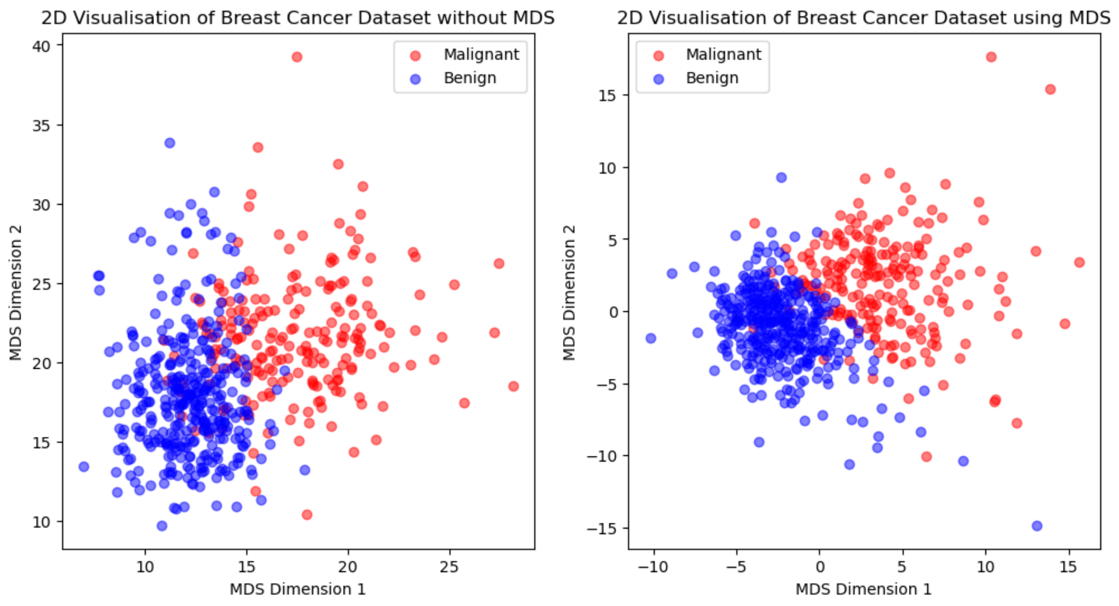
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS
from sklearn.metrics import pairwise_distances
```

[2]:
```
data = load_breast_cancer()
X = data.data
y = data.target
```

[3]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

dist_matrix = pairwise_distances(X_scaled, metric='euclidean')

mds = MDS(n_components=2, dissimilarity='precomputed', random_state=42)
X_mds = mds.fit_transform(dist_matrix)
```

[4]:
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red', alpha=0.5,␣
  ↪label='Malignant')
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', alpha=0.5, label='Benign')
plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.title("2D Visualisation of Breast Cancer Dataset without MDS")
plt.legend(labels=["Malignant", "Benign"])

plt.subplot(1, 2, 2)
plt.scatter(X_mds[y == 0, 0], X_mds[y == 0, 1], color='red', alpha=0.5,␣
  ↪label='Malignant')
plt.scatter(X_mds[y == 1, 0], X_mds[y == 1, 1], color='blue', alpha=0.5,␣
  ↪label='Benign')
```

```
plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.title("2D Visualisation of Breast Cancer Dataset using MDS")
plt.legend(labels=["Malignant", "Benign"])
plt.show()
```



[5]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
 ↪random_state= 0)
```

0.1.1 Feature Scaling

[6]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

0.2 PCA (Principal Component Analysis)

[7]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 1)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

0.3 t-SNE (t-Distributed Stochastic Neighbor)

```
[8]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
      ↪random_state= 0)
```

```
[9]: from sklearn.preprocessing import StandardScaler

     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
[10]: from sklearn.manifold import TSNE
```

```
[11]: tsne = TSNE(n_components = 2, random_state = 0)
```

```
[12]: tsne_obj = tsne.fit_transform(X_train)
```

```
c:\Users\Neil\anaconda3\Lib\site-
packages\joblib\externals\loky\backend\context.py:136: UserWarning: Could not
find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by
setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "c:\Users\Neil\anaconda3\Lib\site-
packages\joblib\externals\loky\backend\context.py", line 257, in
_count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^^
  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 548, in run
    with Popen(*popenargs, **kwargs) as process:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
[13]: import pandas as pd
      tsne_df = pd.DataFrame({'X' : tsne_obj[:,0],
                             'Y' : tsne_obj[:,1],
                              'classification' : y_train
                             })
```

```
[14]: tsne_df.head()
```

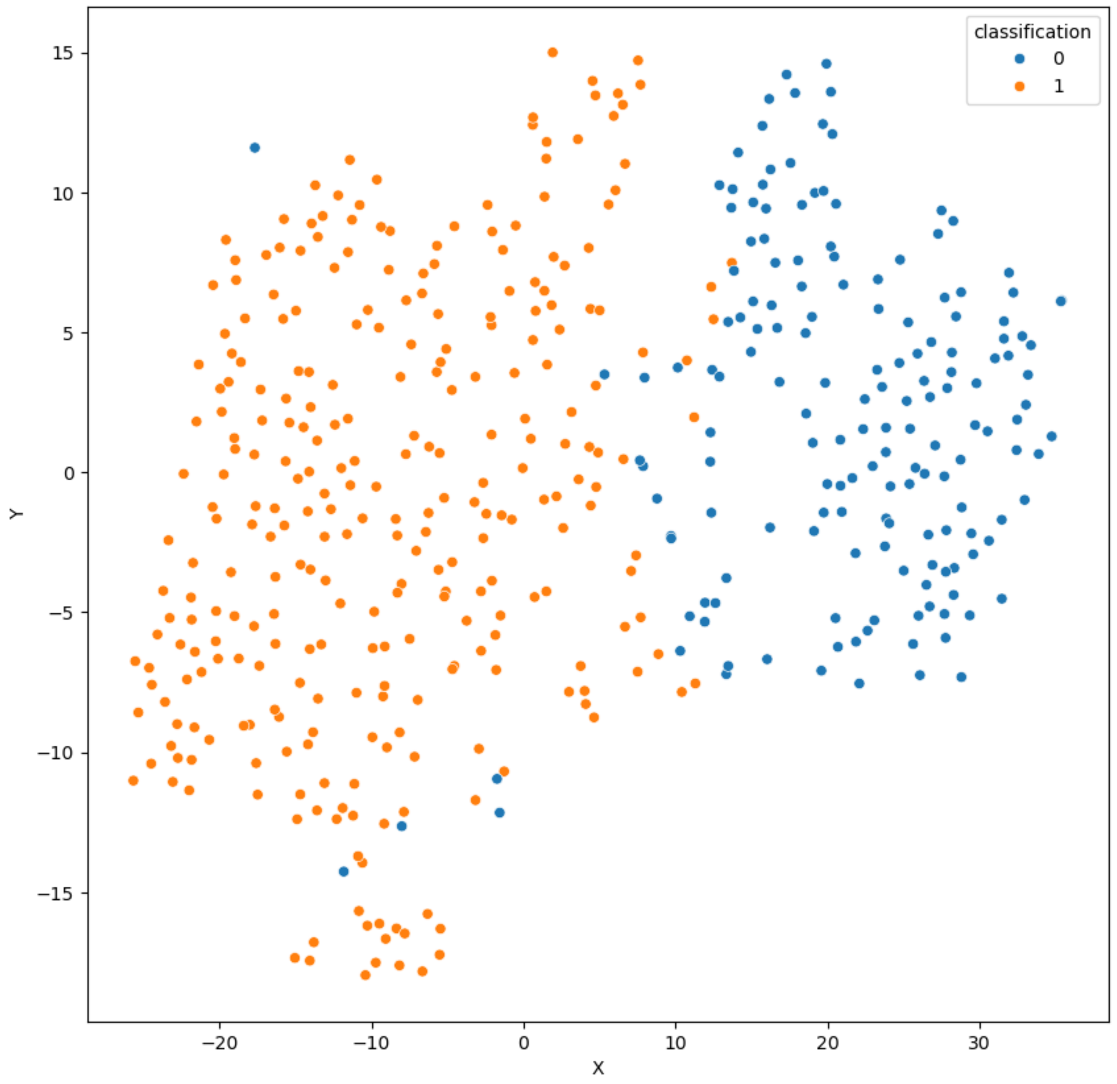
```
[14]:            X          Y  classification
      0  -7.053003  -2.806510               1
```

```
1  -8.142869  -9.293004               1
2 -13.934480   8.887429               1
3  -2.931662  -9.876837               1
4  -9.162365 -12.554125               1
```

[15]:
```
tsne_df['classification'].value_counts()
```

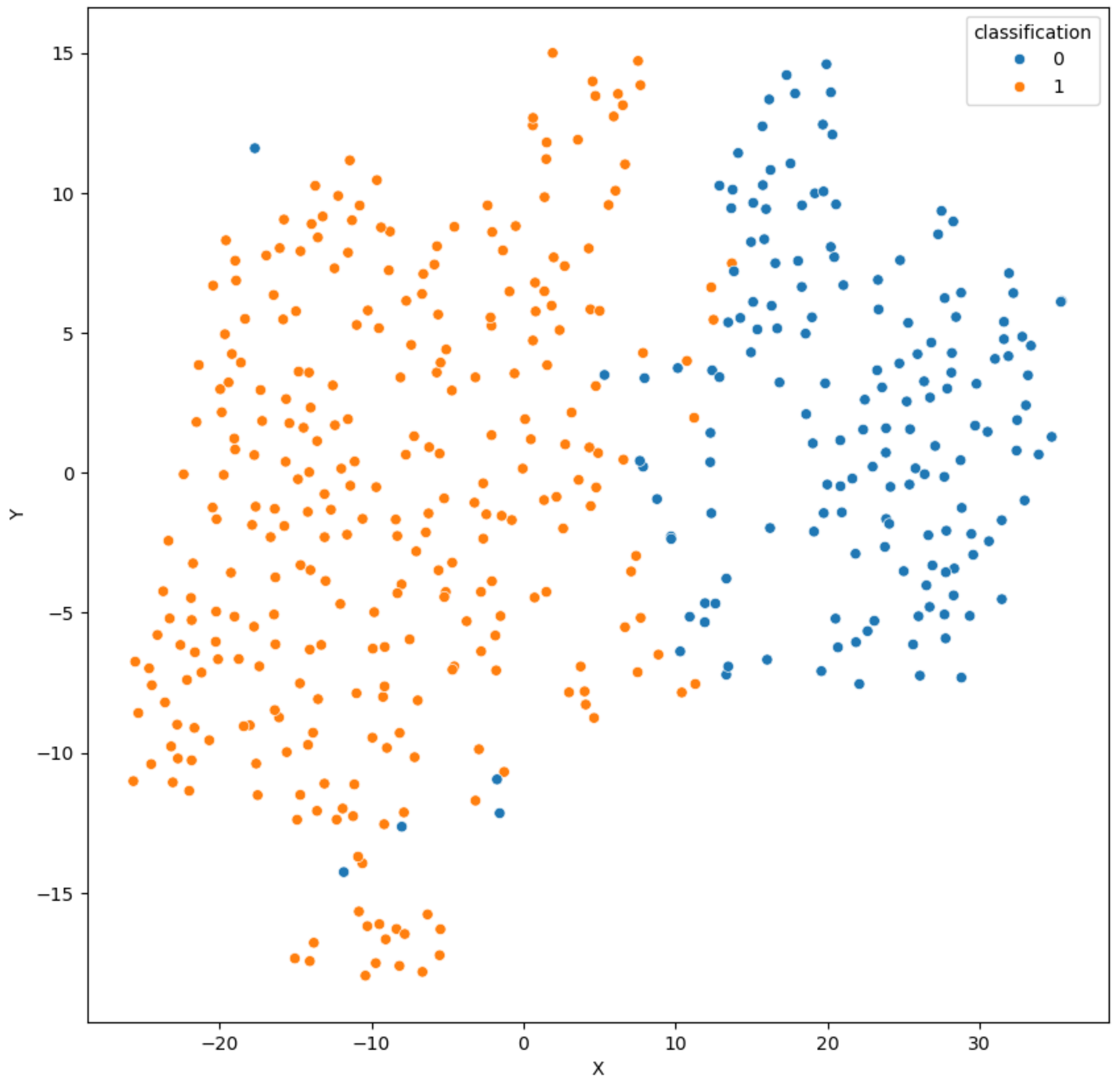
[15]:
```
classification
1    290
0    165
Name: count, dtype: int64
```

[16]:
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = 'X', y = 'Y', data = tsne_df, hue=tsne_df['classification'])
plt.show()
```



We have obtained the plot of the data points but are unable to segregate. Let's introduce hue

[17]:
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = "X", y = 'Y', hue = 'classification', legend = 'full', data␣
 ↪= tsne_df)
plt.show()
```



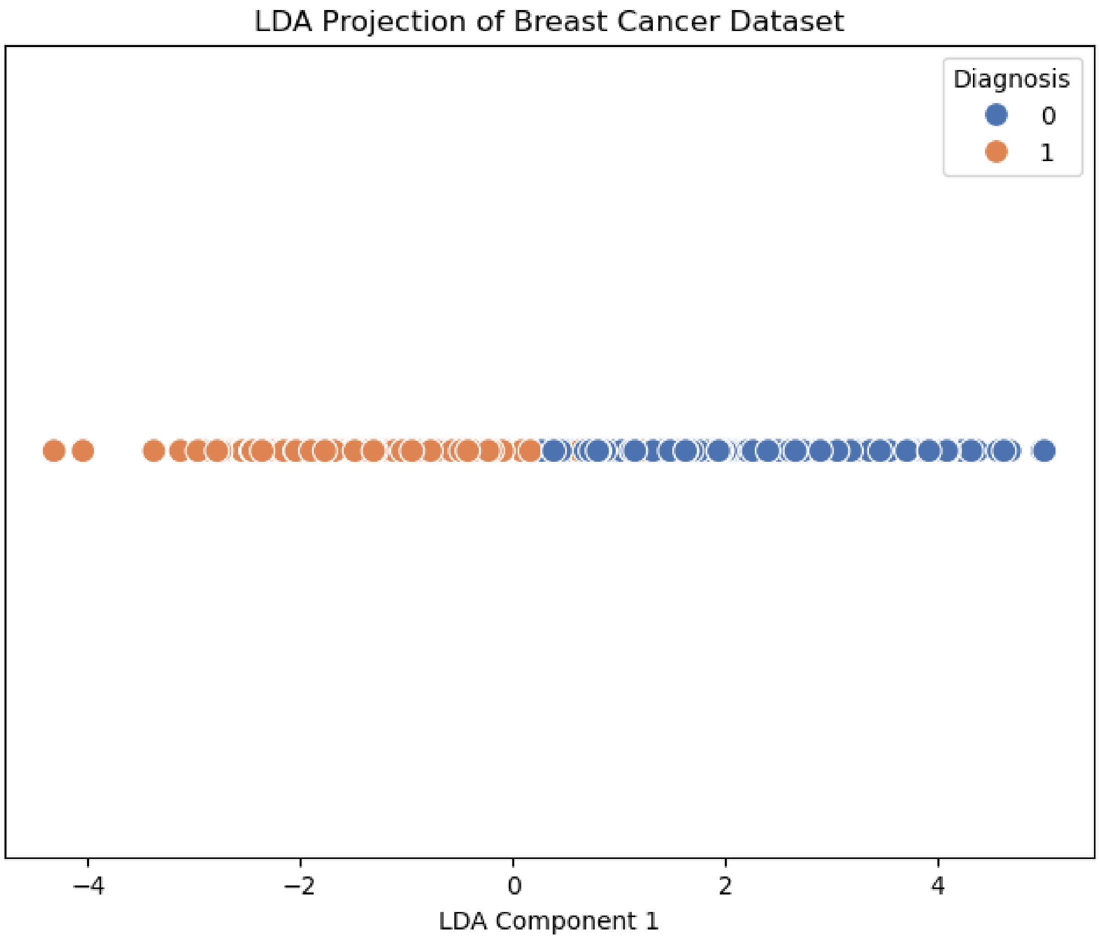
0.4 LDA (Linear Discriminant Analysis)

0.4.1 Performing LDA (Linear Discriminant Analysis)

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
↪random_state= 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[19]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

[20]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame for plotting
import pandas as pd
df_lda = pd.DataFrame({'LDA1': X_train[:, 0], 'Diagnosis': y_train})

# Plot using Seaborn
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_lda['LDA1'], y=[0] * len(df_lda), hue=df_lda['Diagnosis'],␣
↪palette="deep", s=100)
plt.xlabel('LDA Component 1')
plt.title('LDA Projection of Breast Cancer Dataset')
plt.yticks([])  # Remove y-axis labels since it's 1D
plt.legend(title='Diagnosis')
plt.show()
```


LDA Projection of Breast Cancer Dataset
Diagnosis
0
1
−4
−2
0
2
4
LDA Component 1