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

Title:

Implementing dimensionality reduction algorithms on a specific dataset and comparing its outcomes

---

Aim:

Implement the dimensionality reduction techniques and compare their outcomes. (PCA, LDA, t-SNE, MDS, SVD, etc)

Objective:

Students will learn:

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

Explanation / Stepwise Procedure / Algorithm

Dimensionality Reduction Techniques

Principal Component Analysis (PCA)

PCA reduces high-dimensional data while keeping most of its information. It identifies key directions (principal components) where data varies the most and projects it onto them.

Steps:

1. Standardize the data.
2. Compute the covariance matrix.
3. Find eigenvectors and eigenvalues.
4. Select top k eigenvectors.
5. Project data onto these eigenvectors.

Uses:

- Reducing dimensions
- Finding patterns in data
- Improving machine learning performance

Linear Discriminant Analysis (LDA)

LDA is a supervised technique that finds the best way to separate different classes. It is useful when features are many, but samples are few.

Steps:

1. Standardize the data.
2. Compute within-class and between-class scatter matrices.
3. Find eigenvectors and eigenvalues.
4. Select top k eigenvectors.
5. Project data onto these eigenvectors.

Uses:

- Reducing dimensions
- Improving classification performance
- Identifying key features for classification

---

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE maps high-dimensional data to a lower-dimensional space while preserving local relationships. It is mainly used for visualization.

Steps:

1. Compute data similarity using a Gaussian kernel.
2. Convert it into a probability distribution.
3. Define a cost function for differences between high- and low-dimensional data.
4. Minimize the cost function.

Uses:

- Visualizing high-dimensional data
- Detecting clusters and patterns
- Preserving local structure in data

Singular Value Decomposition (SVD)

SVD breaks a matrix into three smaller matrices, capturing key patterns. It is widely used in image compression, recommendations, and noise reduction.

Steps:

1. Decompose matrix X into U, Σ, and VT:
   - U: Left singular vectors
   - Σ: Singular values (importance)
   - VT: Right singular vectors
2. Keep top k singular values and vectors.
3. Use these components to create a lower-dimensional representation.

Uses:

- Reducing dimensions
- Removing noise
- Feature extraction
- Applications in text mining & image processing

Multidimensional Scaling (MDS)

MDS represents high-dimensional data in lower dimensions while maintaining pairwise distances. It helps visualize similarities in data.

Steps:

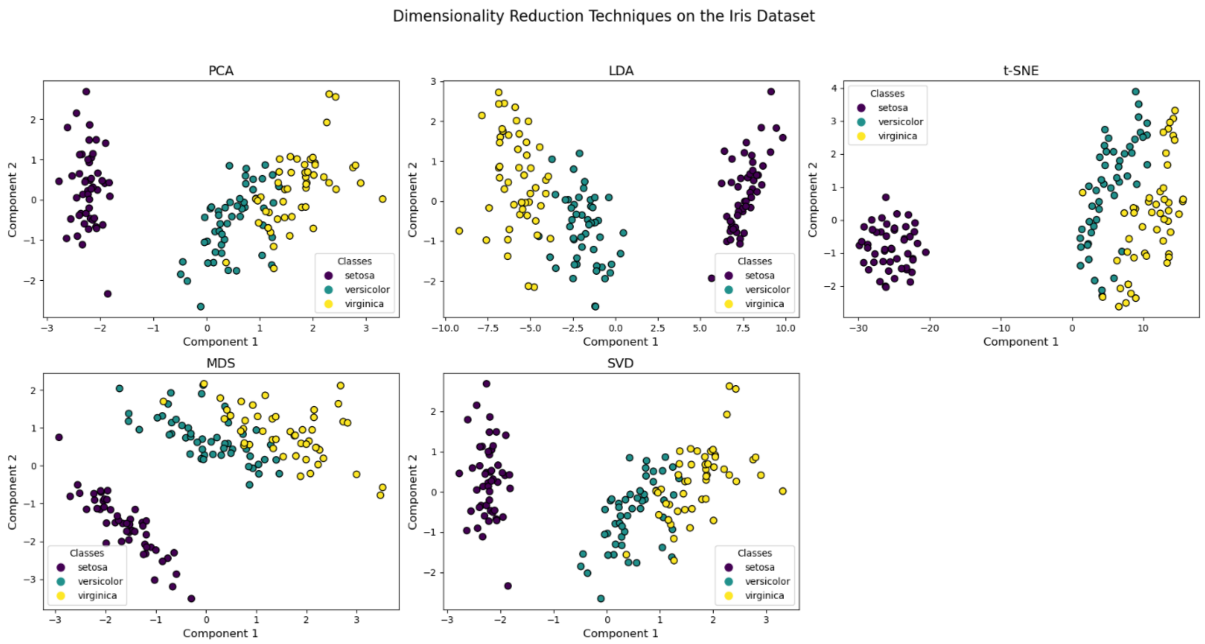
1. Compute the dissimilarity matrix.
2. Convert it for eigenvalue decomposition or define a cost function.
3. Perform decomposition or use an optimization algorithm.
4. Select top k dimensions.
5. Assign new coordinates to data points.

Uses:

- Visualizing data in 2D/3D
- Understanding relationships between points
- Preserving distances in dimensionality reduction
- Market research and psychology analysis

Figures/Diagrams

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



Challenges Encountered

1. Different techniques work in different ways, making it hard to choose the best one.
2. Some methods, like t-SNE and MDS, take longer to process large datasets.
3. Understanding the reduced data can be tricky, as some details may be lost.

Conclusion

- Dimensionality reduction makes data easier to analyse and improves performance.
- Each method has its strengths, so the choice depends on the data and purpose.
- Comparing results helps in selecting the most suitable technique.

all_comparison

February 24, 2025

[4]:
```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.manifold import TSNE, MDS
```

[5]:
```python
# Load the iris dataset
iris = load_iris()
X = iris.data        # Feature data
y = iris.target      # Class labels
target_names = iris.target_names

# Standardize the data for better performance of the algorithms
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

[6]:
```python
# Principal Component Analysis (PCA)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)
```

[7]:
```python
# Linear Discriminant Analysis (LDA)
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X_std, y)
```

[8]:
```python
# t-Distributed Stochastic Neighbor Embedding (t-SNE)
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_std)
```

```
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,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

[9]:

```
# Multidimensional Scaling (MDS)
mds = MDS(n_components=2, random_state=42)
X_mds = mds.fit_transform(X_std)
```

[10]:

```
# Singular Value Decomposition (SVD)
svd = TruncatedSVD(n_components=2, random_state=42)
X_svd = svd.fit_transform(X_std)
```

[13]:

```
# Function to plot the reduced dimensions
def plot_embedding(ax, X_emb, title):
    scatter = ax.scatter(X_emb[:, 0], X_emb[:, 1], c=y, cmap='viridis',
                         edgecolor='k', s=50)
    ax.set_title(title, fontsize=14)
    ax.set_xlabel('Component 1', fontsize=12)
    ax.set_ylabel('Component 2', fontsize=12)

    # Manually create the legend using unique class labels
    unique_classes = np.unique(y)
    legend_labels = [target_names[i] for i in unique_classes]
    handles = [plt.Line2D([0], [0], marker='o', color='w',␣
↪markerfacecolor=scatter.cmap(scatter.norm(i)), markersize=10)
               for i in unique_classes]

    ax.legend(handles, legend_labels, title="Classes", loc="best")
```

[14]:

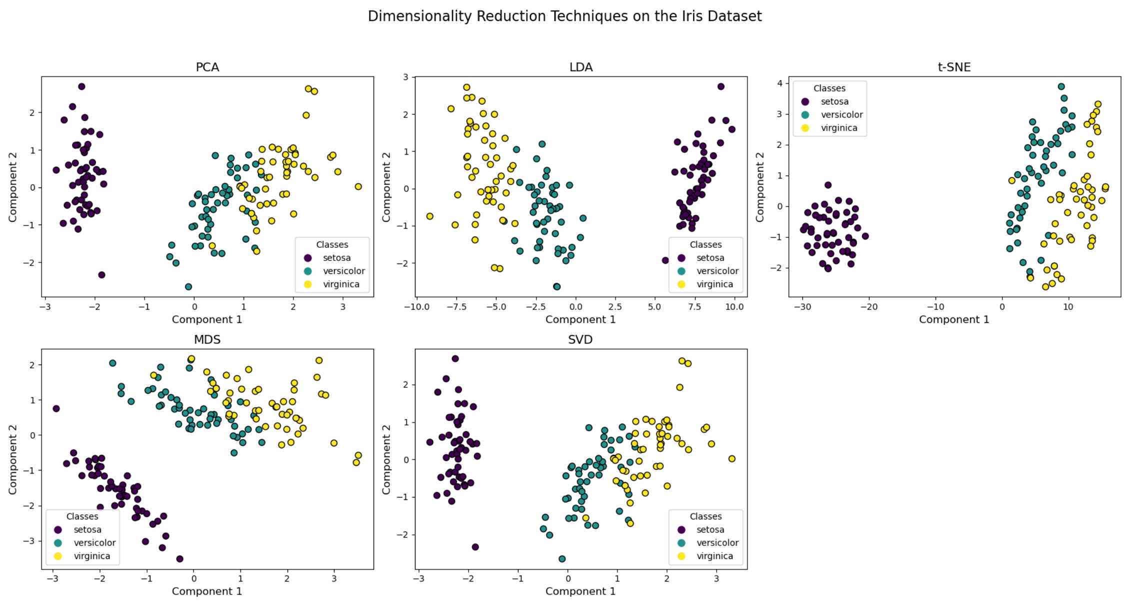
```
# Create subplots
fig, axs = plt.subplots(2, 3, figsize=(18, 10))
axs = axs.flatten()  # Flatten to easily iterate over subplots

# Plot each technique's output
plot_embedding(axs[0], X_pca, 'PCA')
plot_embedding(axs[1], X_lda, 'LDA')
plot_embedding(axs[2], X_tsne, 't-SNE')
```

```
plot_embedding(axs[3], X_mds, 'MDS')
plot_embedding(axs[4], X_svd, 'SVD')

# Remove the empty subplot
fig.delaxes(axs[5])

# Add a super title and adjust layout
plt.suptitle('Dimensionality Reduction Techniques on the Iris Dataset',␣
 ↪fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```




```
[ ]:
```