MP4

November 7, 2025

1 AI 221: Machine Exercise 4 - Handwritten Digits Analysis

This exercise focuses on dimensionality reduction and classification techniques applied to the 8x8 handwritten digits dataset from sklearn.

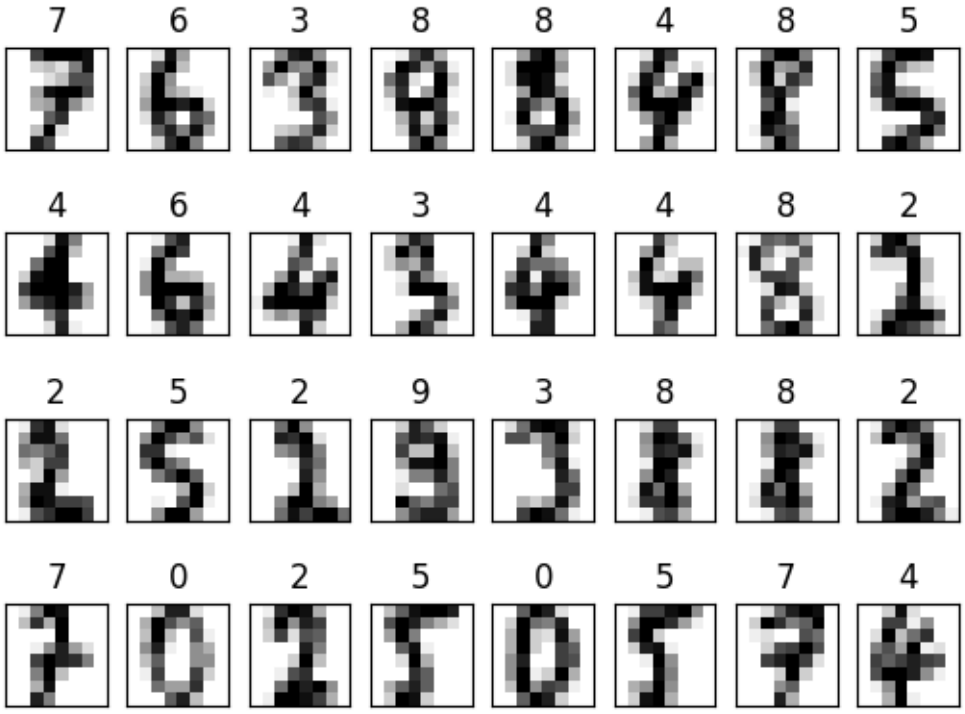
2 Dataset Preparation

This section contains the loading the handwritten digits and showing random samples.

[4]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
def plot_digits(X, y):

    random.seed(0)

    rows, cols = 4, 8
    fig, ax = plt.subplots(rows, cols, sharex='col', sharey='row',
                           subplot_kw=dict(xticks=[], yticks=[]))

    for row in range(rows):
        for col in range(cols):
            n = np.random.randint(1796)+1    # show random samples
            im = ax[row, col].imshow(X[n].reshape((8,8)), cmap=plt.cm.binary)
            ax[row, col].set_title(y[n])
            im.set_clim(0, 16)
            #print(X[n])
```

[5]:
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
import numpy as np
import random
X, y = load_digits(return_X_y=True)
print(X.shape)

plot_digits(X, y)
plt.show()
```

```
(1797, 64)
```



3 Exercises

This section contains the core implementations of Machine Exercise 4 and is subdivided in to 3 subsections;

- Part A: Dimensionality Reduction Visualization
- Part B: Variance Analysis
- Part C: Classification Comparison

3.1 Part A: Dimensionality Reduction Visualization

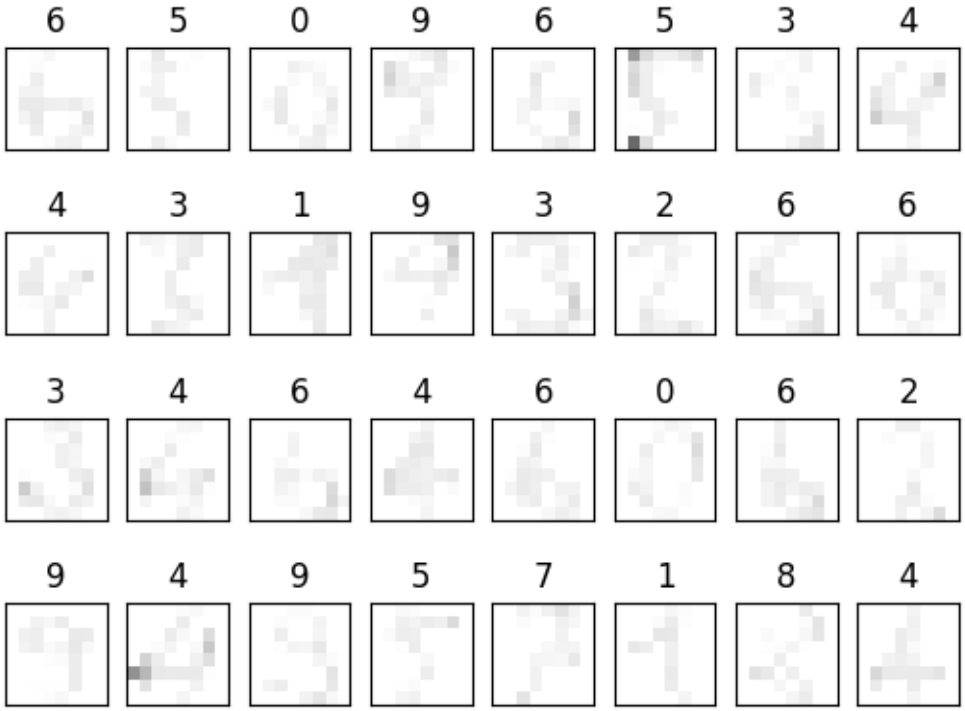
Normalize the X data using Standard Scaler. Then, project all the X data into 2 dimensions using 6 dimensionality reduction techniques:
i. Local Linear Embedding (n_neighbors = 200, random_state = 0)
ii. t-SNE (perplexity = 50, random_state = 0)
iii. Isomap (n_neighbors = 200)
iv. Laplacian Eigenmap (n_neighbors = 200)
v. Kernel PCA (kernel = 'rbf', gamma = 0.01)
vi. PCA

3.1.1 Tasks:

1. Apply StandardScaler normalization

2. Implement 6 dimensionality reduction techniques:
    - Local Linear Embedding (n_neighbors=200, random_state=0)
    - t-SNE (perplexity=50, random_state=0)
    - Isomap (n_neighbors=200)
    - Laplacian Eigenmap (n_neighbors=200)
    - Kernel PCA (kernel='rbf', gamma=0.01)
    - PCA
3. Visualization requirements:
    - Project data to 2D for each method
    - Color points by digit labels
    - Compare clustering quality

[6]:
```
# prompt: Normalize the data using StandardScaler
from sklearn.preprocessing import StandardScaler

# Normalize the data using StandardScaler
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

plot_digits(X_normalized, y)
```

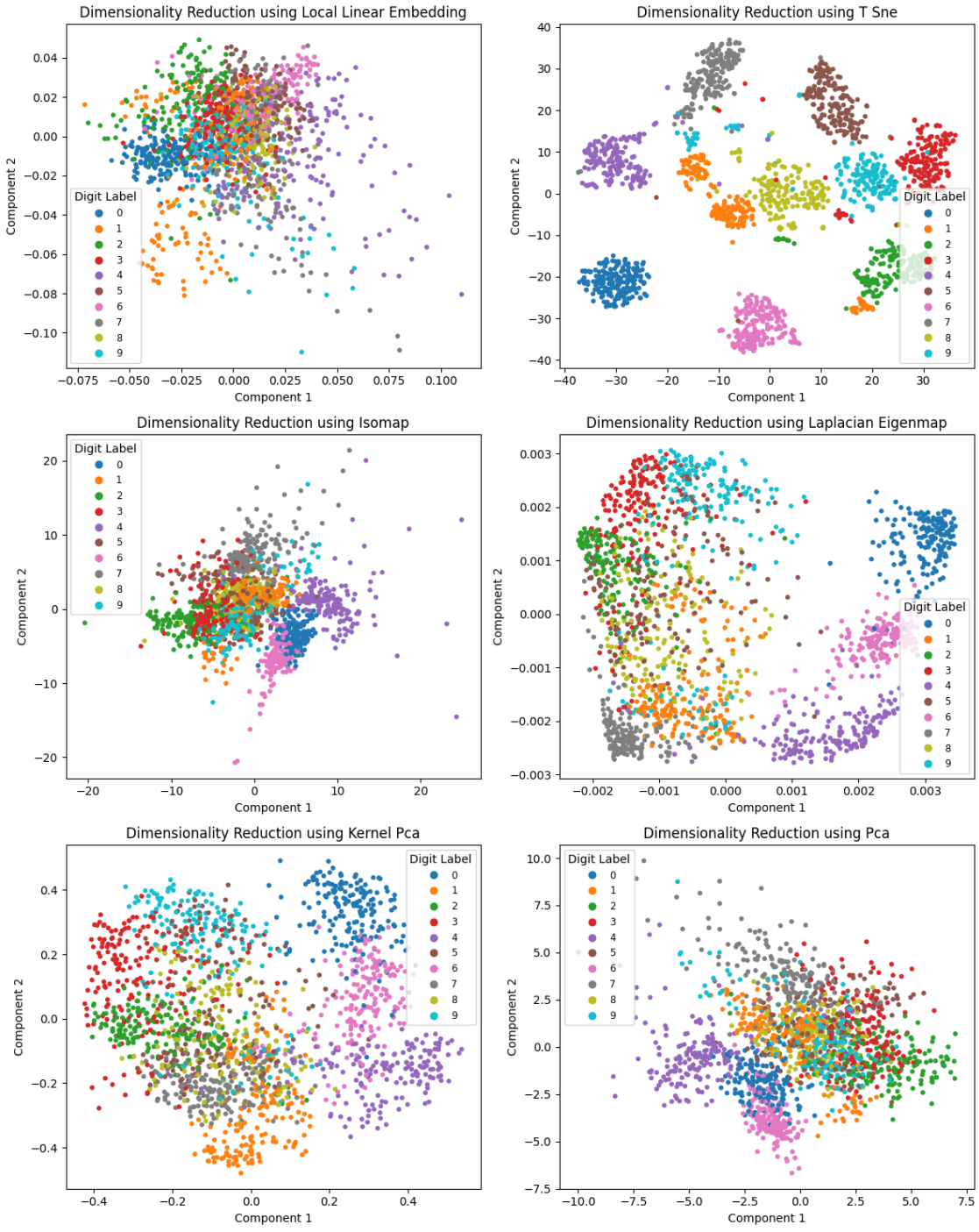


[7]:
```
# Prompt: Create a function that accepts a parameters reduction_technique, and␣
 ↪X data, and returns the transformed data using the specified dimensionality␣
 ↪reduction technique.
# The function should support:
#  local_linear_embedding:  Local Linear Embedding (n_neighbors = 200,␣
 ↪random_state = 0)
#  t_sne: t-SNE (perplexity = 50, random_state = 0)
#  isomap: Isomap (n_neighbors = 200)
#  laplacian_eigenmap: Laplacian Eigenmap (n_neighbors=200)
#  kernel_pca: Kernel PCA (kernel='rbf', gamma=0.01)
#  pca: PCA

from sklearn.manifold import LocallyLinearEmbedding, TSNE, Isomap,␣
 ↪SpectralEmbedding
from sklearn.decomposition import KernelPCA, PCA

def apply_dimensionality_reduction(reduction_technique, X):
    if reduction_technique == 'local_linear_embedding':
        model = LocallyLinearEmbedding(n_neighbors=200, n_components=2,␣
 ↪random_state=0)
    elif reduction_technique == 't_sne':
        model = TSNE(n_components=2, perplexity=50, random_state=0)
    elif reduction_technique == 'isomap':
        model = Isomap(n_neighbors=200, n_components=2)
    elif reduction_technique == 'laplacian_eigenmap':
        model = SpectralEmbedding(n_neighbors=200, n_components=2)
    elif reduction_technique == 'kernel_pca':
        model = KernelPCA(n_components=2, kernel='rbf', gamma=0.01)
    elif reduction_technique == 'pca':
        model = PCA(n_components=2)
    else:
        raise ValueError("Unsupported reduction technique")

    return model.fit_transform(X)
```

[8]:
```
reduction_techniques = [
    'local_linear_embedding',
    't_sne',
    'isomap',
    'laplacian_eigenmap',
    'kernel_pca',
    'pca'
]

results = {}
for technique in reduction_techniques:
    results[technique] = apply_dimensionality_reduction(technique, X_normalized)
```

[9]:
```
def plot_2d_scatter(data_list, y):
    """
    Displays 2D scatter plots for the given list of transformed data arrays in␣
 ↪a single figure.

    Parameters:
    data_list (list of tuples): Each tuple contains (X_transformed, title),␣
 ↪where:
        X_transformed (numpy.ndarray): 2D array with shape (n_samples, 2)␣
 ↪representing the transformed data.
        title (str): Title of the plot.
    y (numpy.ndarray): 1D array with shape (n_samples,) representing the labels.
    """
    # Ensure even number of plots
    if len(data_list) % 2 != 0:
        data_list = data_list[:-1]

    n_data = len(data_list)
    n_cols = 2
    n_rows = (n_data + 1) // n_cols

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows * 5))
    axes = axes.flatten()

    for i, (X_transformed, title) in enumerate(data_list):
        scatter = axes[i].scatter(X_transformed[:, 0], X_transformed[:, 1],␣
 ↪c=y, cmap=plt.colormaps.get_cmap('tab10'), s=10)
        axes[i].set_title(title)
        axes[i].set_xlabel('Component 1')
        axes[i].set_ylabel('Component 2')

        # Add discrete legend
        handles, labels = scatter.legend_elements(prop="colors", num=10,␣
 ↪fmt="{x:.0f}")
        axes[i].legend(handles, labels, title="Digit Label", loc="best",␣
 ↪fontsize="small")

    # Remove unused subplots
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()


data_list = [(results[technique], f'Dimensionality Reduction using {technique.
 ↪replace("_", " ").title()}') for technique in reduction_techniques]
```

```
plot_2d_scatter(data_list, y)
```



3.1.2 Results

Which of the methods produced clear clusters of data points?

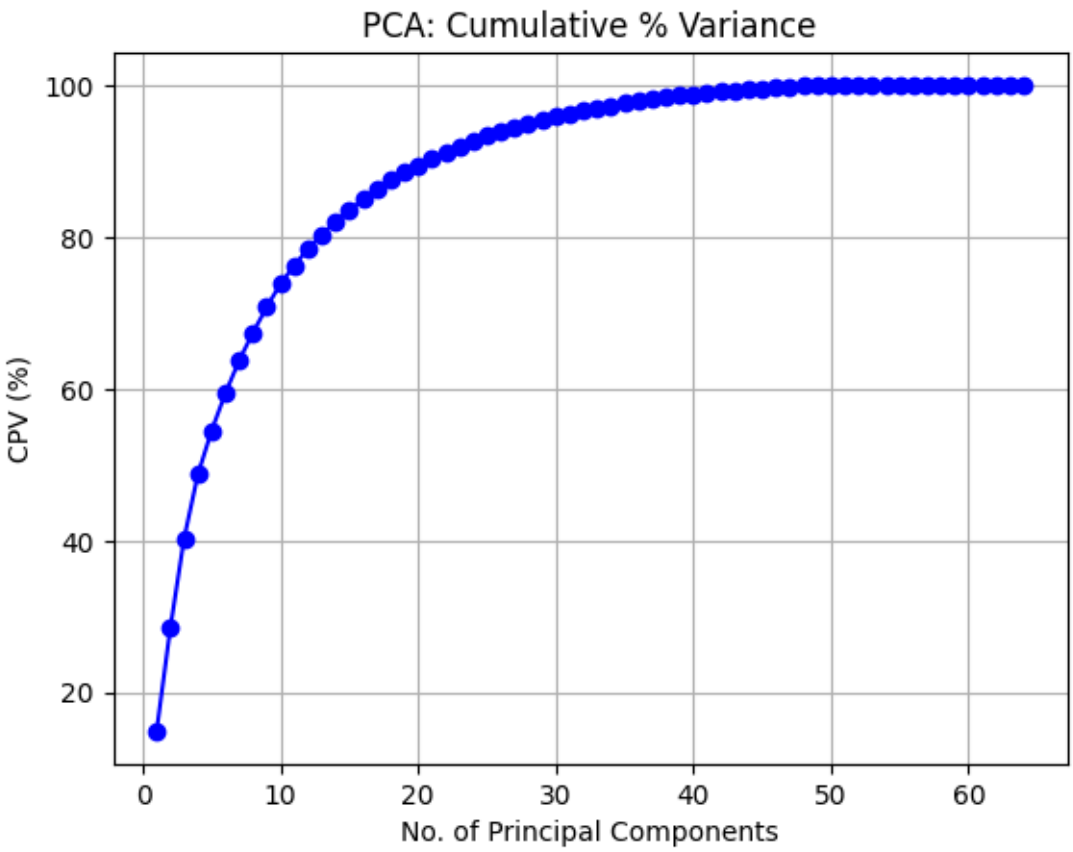
As shown in the plots above, it is evident that t-sne have produced clear clusters of data points.

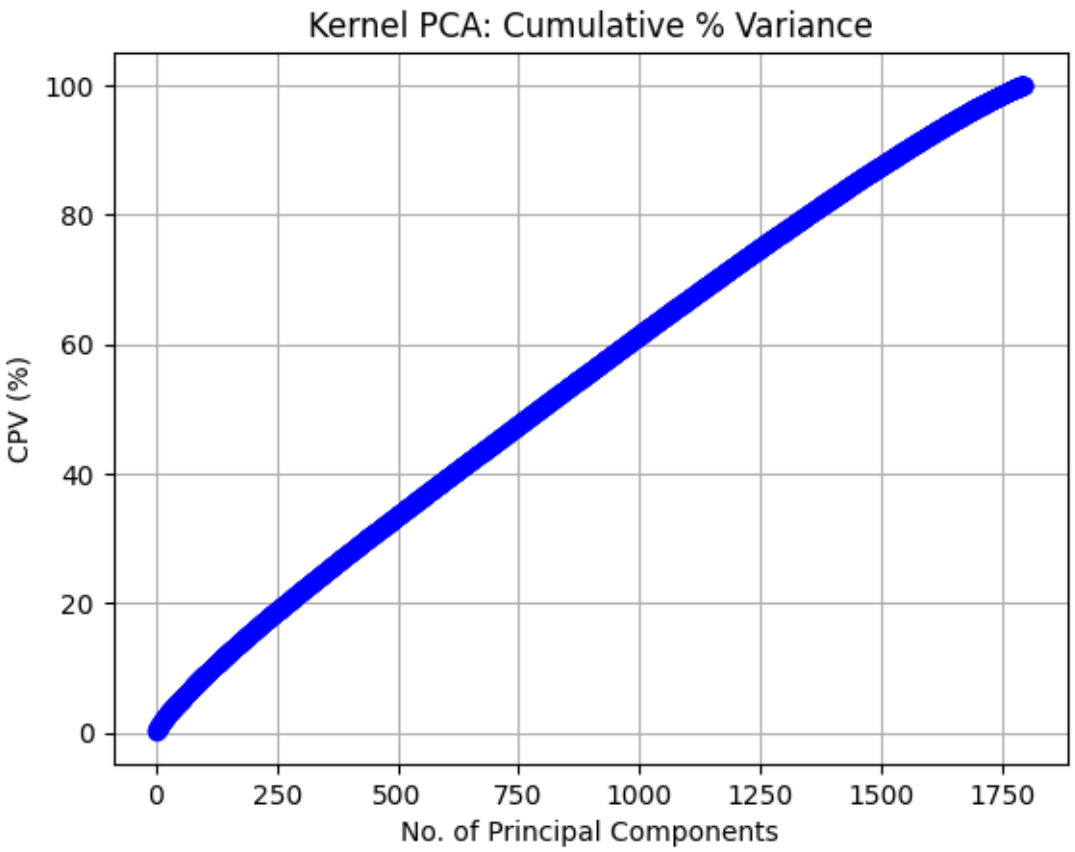
3.2 Part B: Variance Analysis

3.2.1 Task Breakdown:

1. Generate CPV (Cumulative Percent Variance) plots for:
   - Kernel PCA
   - Standard PCA
2. Analysis requirements:
   - Determine number of components for 95% variance retention
   - Compare results between KPCA and PCA

[ ]:
```
# Prompt: Cre

# Perform PCA to view the CPV plot
pca = PCA().fit(X)
var = pca.explained_variance_ratio_
cpv = np.cumsum(var)*100
plt.plot(np.arange(cpv.size)+1,cpv,'bo-')
plt.title('PCA: Cumulative % Variance')
plt.xlabel('No. of Principal Components')
plt.ylabel('CPV (%)')
plt.grid()
plt.show()


kpca = KernelPCA(kernel='rbf', gamma=0.01).fit(X)
X_transformed = kpca.fit_transform(X)

# Extract eigenvalues
eigenvalues = kpca.eigenvalues_

# Calculate cumulative percentage of variance (CPV)
cumulative_variance = np.cumsum(eigenvalues) / np.sum(eigenvalues) * 100
plt.plot(np.arange(cumulative_variance.size)+1,cumulative_variance,'bo-')
plt.title('Kernel PCA: Cumulative % Variance')
plt.xlabel('No. of Principal Components')
plt.ylabel('CPV (%)')
plt.grid()
plt.show()
```


PCA: Cumulative % Variance
CPV (%)
100
80
60
40
20
0
10
20
30
40
50
60
No. of Principal Components



3.3 Part C: Classification Comparison [60 pts]

3.3.1 Data Preparation:

- Split data: 70% training, 30% testing
- Ensure stratified sampling by class label

3.3.2 Classification Methods:

1. Method 1: Kernel PCA Pipeline [20 pts]
   - StandardScaler
   - Kernel PCA (kernel='sigmoid', n_components=40)
   - SVC (default parameters)
   - Report accuracy and F1-score
2. Method 2: LDA Pipeline [20 pts]
   - StandardScaler
   - LDA (n_components=9)
   - SVC (default parameters)
   - Report accuracy and F1-score
   - Explain LDA component limitation
3. Method 3: Baseline SVC [20 pts]

- StandardScaler
- SVC (default parameters)
- No dimensionality reduction
- Report accuracy and F1-score

3.3.3 Analysis Requirements:

- Compare performance metrics
- Explain best performing method
- Discuss trade-offs

[ ]:
```
# Prompt: Split dataset into training and testing sets (70% train, 30% test)␣
 ↪using train_test_split with random_state=42
from sklearn.model_selection import train_test_split

# Split dataset into training and testing sets (70% train, 30% test) using␣
 ↪train_test_split with random_state=42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪stratify=y, random_state=42)
```

3.3.4 Kernel PCA Pipeline

- StandardScaler
- Kernel PCA (kernel='sigmoid', n_components=40)
- SVC (default parameters)
- Report accuracy and F1-score

[ ]:
```
# Prompt: Create a pipeline that includes StandardScaler, KernelPCA␣
 ↪(kernel='sigmoid', n_components=40), SVC (default parameters), and report␣
 ↪accuracy and F1-score on the test set.

from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score

# Create a pipeline with StandardScaler, KernelPCA, and SVC
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('kernel_pca', KernelPCA(kernel='sigmoid', n_components=40)),
    ('svc', SVC())
])

# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = pipeline.predict(X_test)
```

```
# Calculate accuracy and F1-score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.9889
F1-score: 0.9889
```

3.3.5 LDA Pipeline [20 pts]

- StandardScaler
- LDA (n_components=9)
- SVC (default parameters)
- Report accuracy and F1-score
- Explain LDA component limitation

[16]:
```
# Prompt: Create a pipeline that includes StandardScaler, LDA (n_components=9),␣
↪SVC (default parameters), and report accuracy and F1-score on the test set.
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create a pipeline with StandardScaler, LDA, and SVC
lda_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lda', LinearDiscriminantAnalysis(n_components=9)),
    ('svc', SVC())
])

# Fit the pipeline on the training data
lda_pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred_lda = lda_pipeline.predict(X_test)

# Calculate accuracy and F1-score
accuracy_lda = accuracy_score(y_test, y_pred_lda)
f1_lda = f1_score(y_test, y_pred_lda, average='weighted')

print(f"LDA Pipeline Accuracy: {accuracy_lda:.4f}")
print(f"LDA Pipeline F1-score: {f1_lda:.4f}")
```

```
LDA Pipeline Accuracy: 0.9630
LDA Pipeline F1-score: 0.9631
```

3.3.6 Baseline SVC [20 pts]

- StandardScaler
- SVC (default parameters)

- No dimensionality reduction
- Report accuracy and F1-score

[17]:
```python
# Prompt: Create a pipeline that includes StandardScaler, SVC (default parameters), and report accuracy and F1-score on the test set.
# Create a pipeline with StandardScaler and SVC
svc_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC())
])

# Fit the pipeline on the training data
svc_pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred_svc = svc_pipeline.predict(X_test)

# Calculate accuracy and F1-score
accuracy_svc = accuracy_score(y_test, y_pred_svc)
f1_svc = f1_score(y_test, y_pred_svc, average='weighted')

print(f"SVC Pipeline Accuracy: {accuracy_svc:.4f}")
print(f"SVC Pipeline F1-score: {f1_svc:.4f}")
```

```
SVC Pipeline Accuracy: 0.9833
SVC Pipeline F1-score: 0.9833
```

3.4 Submission Guidelines

1. Format Requirements:
   - Submit as PDF file
   - Export from Jupyter Notebook
   - Highlight final answers clearly
2. Code Organization:
   - Clear section headers
   - Well-documented implementations
   - Proper visualization labels
3. Analysis Documentation:
   - Detailed explanations for each part
   - Comparative analysis between methods
   - Clear conclusions and insights
4. Submit through UVLE platform