

A white rectangular box is centered on the background, containing the text "APAI 2024" and "Lesson 4".

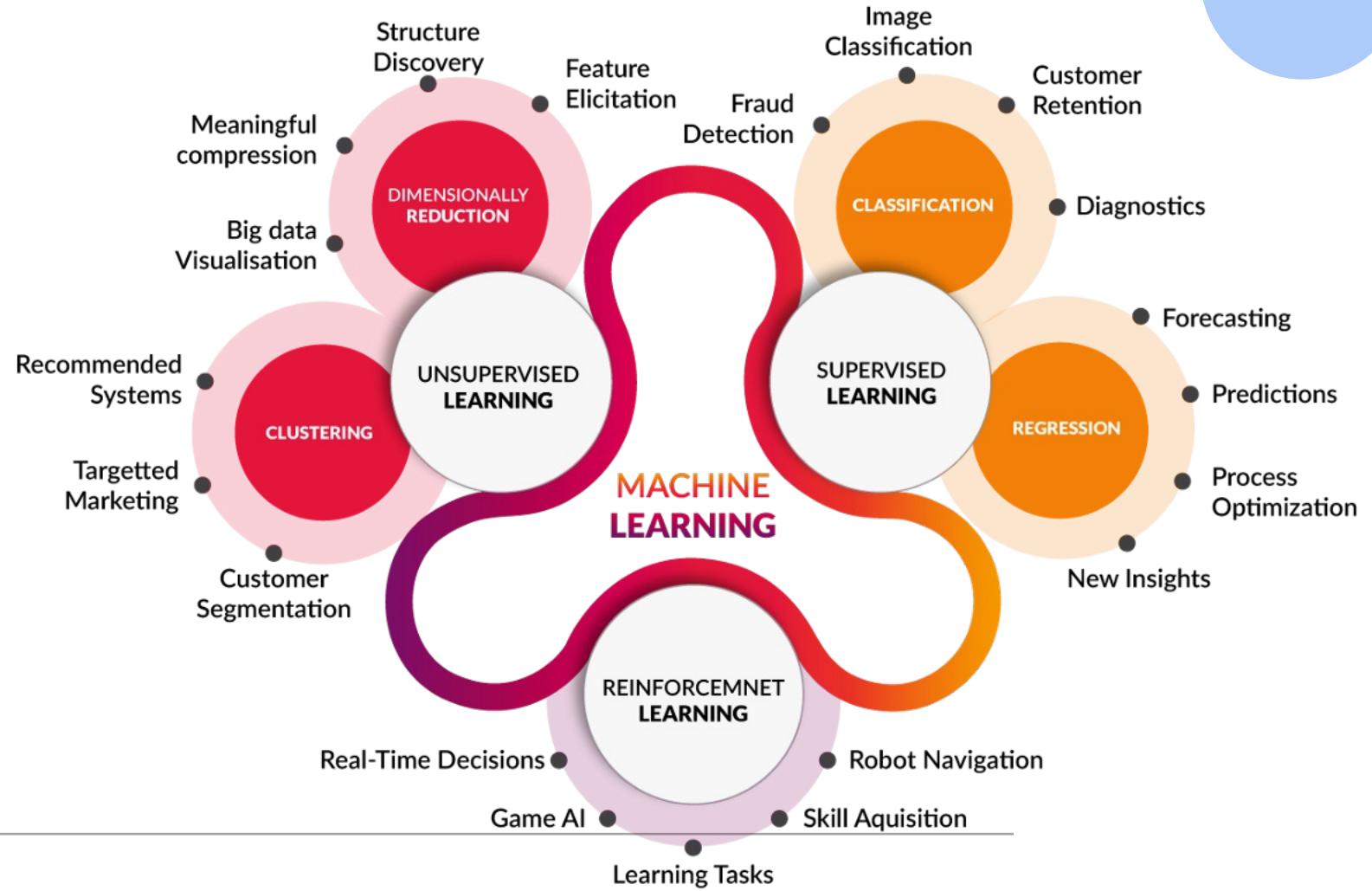
APAI 2024

Lesson 4

Machine Learning



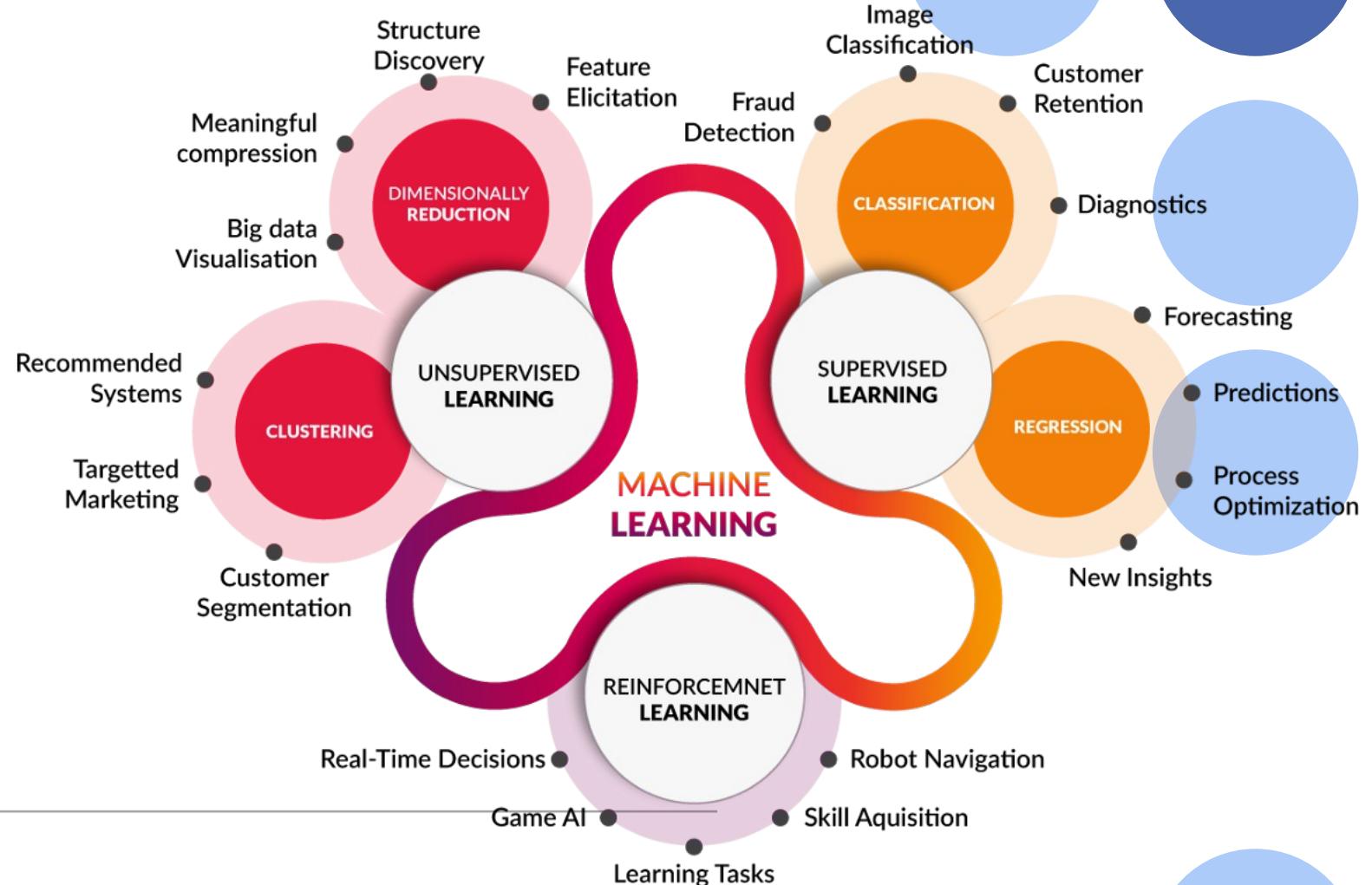
Machine Learning



Machine Learning

High level taxonomy

- Unsupervised
- Supervised
- Reinforcement



Machine Learning (Supervised)

- You work for a real estate agency
- you want to build a model that predicts the selling price of houses based on their **features**, such as
 - ✓ the number of bedrooms,
 - ✓ bathrooms,
 - ✓ square footage, and
 - ✓ location.



Machine Learning (Unsupervised)

- You are the marketing manager of a retail store.
- You know about your customer customers:
 - their annual income
 - the number of purchases made, and
 - the time spent in the store.
- You want to divide your customers into homogeneous groups so that you can better tailor your marketing strategies to each group.



Machine Learning (Reinforcement)

- You have a mobile robot equipped with sensors that can move in a 2D environment, represented as a maze.
- The robot's goal is to reach a specific position in the maze, called the 'goal', starting from an initial position.
- However, the maze contains obstacles, such as walls or barriers, that the robot must avoid while trying to reach the goal.



Scikit-learn

- is a free and open-source machine learning library for the [Python](#)
 - It provides:
 - Simple and efficient tools for predictive data analysis
 - Accessible to everybody, and reusable in various contexts
 - Built on NumPy, SciPy, and matplotlib
 - Open source, commercially usable - BSD license
-

Scikit-learn

- Official Webpage <https://scikit-learn.org/>
-

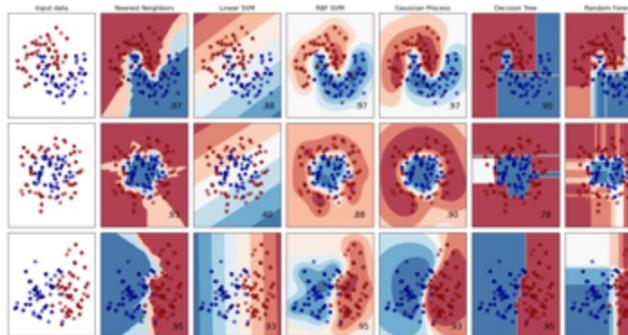
Scikit-learn

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

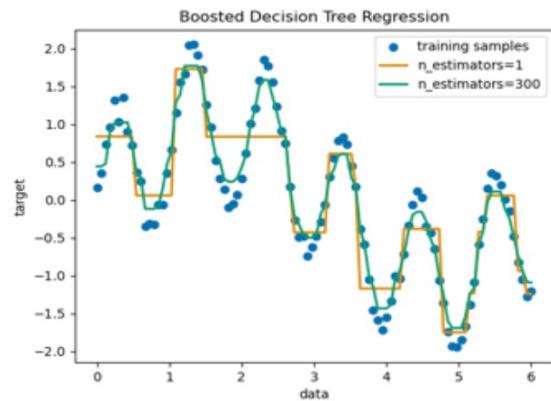


Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...



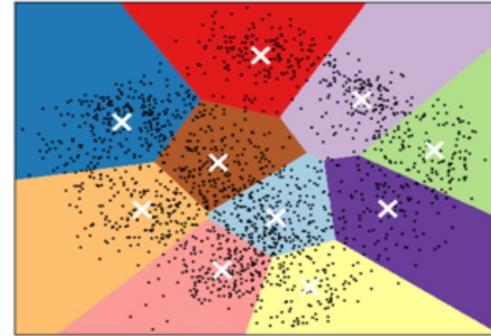
Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross



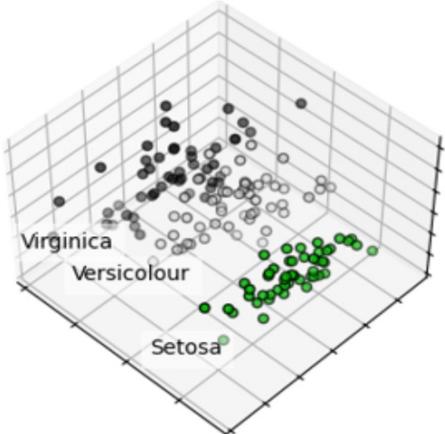
Scikit-learn

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

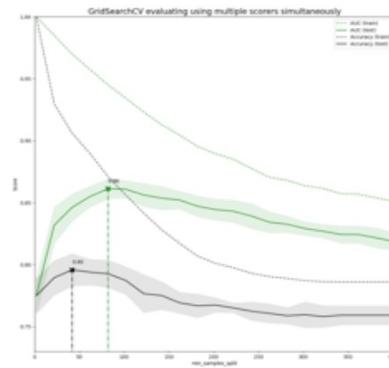


Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...

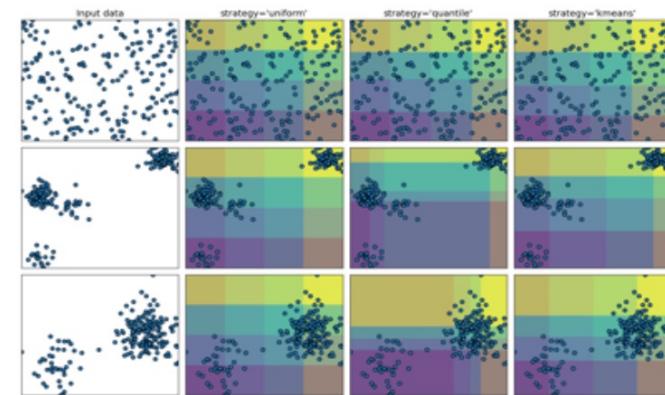


Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



Scikit-learn

Install:

```
$ pip install -U scikit-learn
```

Dependency	Minimum Version
numpy	1.19.5
scipy	1.6.0
joblib	1.2.0
threadpoolctl	2.0.0
cython	3.0.8
matplotlib	3.3.4
scikit-image	0.17.2
pandas	1.1.5
seaborn	0.9.0
memory_profiler	0.57.0
pytest	7.1.2
pytest-cov	2.9.0
ruff	0.0.272
black	23.3.0
mypy	1.3
pyamg	4.0.0
polars	0.19.12
pyarrow	12.0.0
sphinx	6.0.0
sphinx-copybutton	0.5.2
sphinx-gallery	0.15.0
numpydoc	1.2.0
Pillow	7.1.2
pooch	1.6.0
sphinx-prompt	1.3.0
sphinxext-opengraph	0.4.2
plotly	5.14.0
conda-lock	2.4.2

Scikit-learn, SciPy and Numpy



TASK 000

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object = selected_obj
# mirror_mod.mirror_object
# mirror_mod.mirror_x = True
# mirror_mod.mirror_y = False
# mirror_mod.mirror_z = False
# operation == "MIRROR_X"
# mirror_mod.use_x = True
# mirror_mod.use_y = False
# mirror_mod.use_z = False
# operation == "MIRROR_Y"
# mirror_mod.use_x = False
# mirror_mod.use_y = True
# mirror_mod.use_z = False
# operation == "MIRROR_Z"
# mirror_mod.use_x = False
# mirror_mod.use_y = False
# mirror_mod.use_z = True

selection at the end -add
# ob.select= 1
# mirr_ob.select=1
context.scene.objects.active = eval("Selected" + str(modifier))
# ob.select = 1
# bpy.context.selected_objects.append(modifier)
# data.objects[one.name].select = 1
# int("please select exactly one object")
# - OPERATOR CLASSES ---

types.Operator):
    X mirror to the selected
    object.mirror_mirror_x"
    "mirror X"
    context):
        ext.active_object is not
```

Statistical Analysis

The function task000() performs statistical analysis:

- Generate two normal distributions with different means and standard deviations, each containing 100 elements, taking care of the "seed" of the random number generators.
- Conducting a t-test to compare two groups and calculate the associated p-value.
- Determine whether the p-value is significant or not

```
def task000():
    # Generate two random samples with different means
    np.random.seed(0)  # To make the result reproducible
    sample1 = np.random.normal(loc=5, scale=1, size=100)  # Sample 1 with mean 5 and standard deviation 1
    sample2 = np.random.normal(loc=7, scale=1, size=100)  # Sample 2 with mean 7 and standard deviation 1

    # Perform t-test
    t_statistic, p_value = stats.ttest_ind(sample1, sample2)

    # Show the results
    print("Test t statistic:", t_statistic)
    print("Associated p-value:", p_value)

    # Interpret the result
    alpha = 0.05
    if p_value < alpha:
        print("The difference between the means of the two groups is statistically significant (reject H0)")
    else:
        print("There is not enough evidence to say that the means of the two groups are different (fail to reject H0)")
```

TASK 001

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object
# mirror operation
if operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

# selection at the end - add
# mirror_mod.select= 1
# mirror_mod.select=1
context.scene.objects.active = bpy.context.selected_objects[0]
("Selected" + str(modifier))
mirror_mod.select = 0
bpy.context.selected_objects[0].select = 1
data.objects[one.name].select = 1

int("please select exactly one object")
# --- OPERATOR CLASSES ---
# --- OPERATOR CLASSES ---

types.Operator):
    X mirror to the selected
    object.mirror_mirror_x"
    "mirror X"
    "context):
        context.active_object is not
```

Visualizing and Analyzing the Digits Dataset

The function `task001()` loads the Digits dataset, visualizes some images, and provides basic statistics:

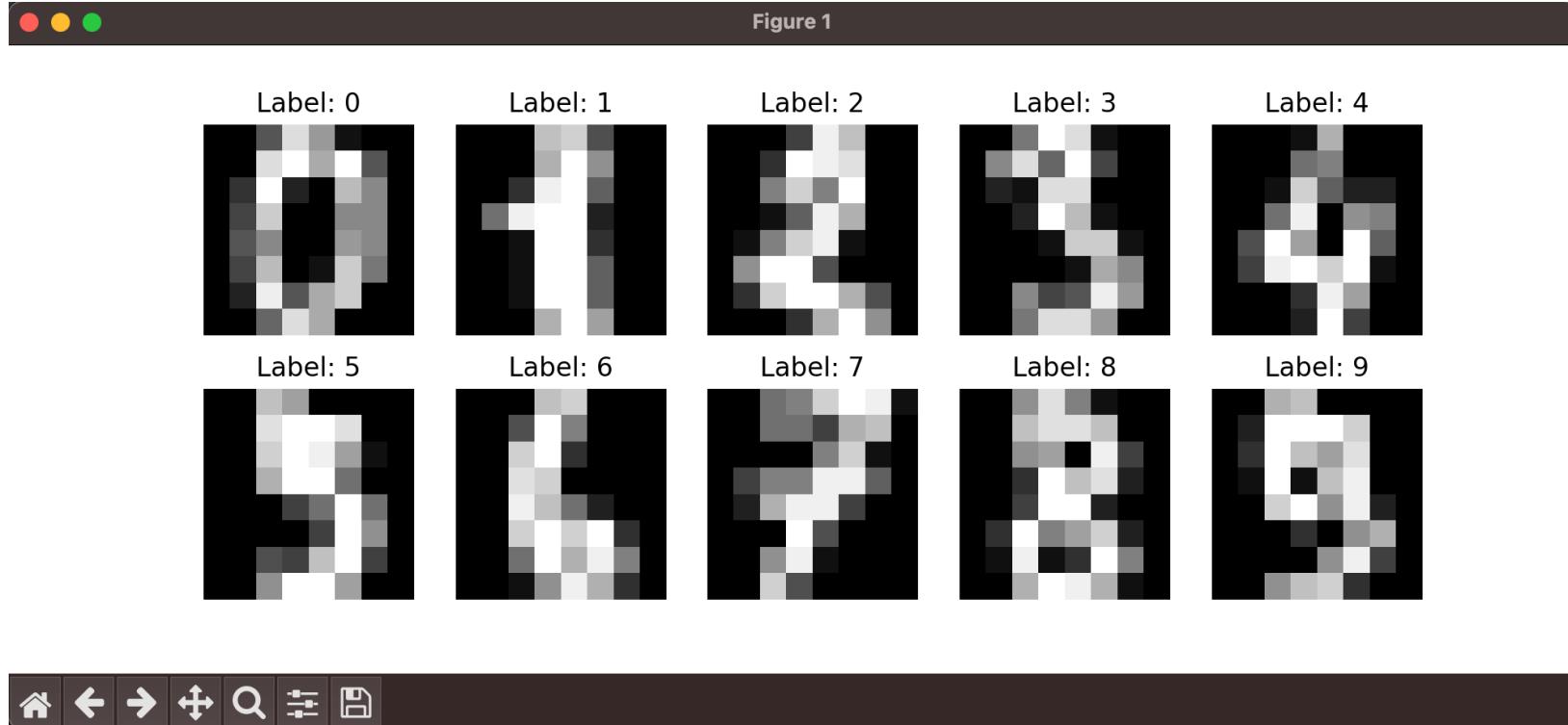
- Load the Digits dataset.
- Visualize some images from the dataset, displaying 10 handwritten digits.
- Print some basic statistics:
 - Number of samples in the dataset.
 - Number of features per sample.
 - Number of classes in the dataset.
 - Class distribution in the dataset.

```
def task001():
    """
    This function loads the Digits dataset, visualizes some images, and provides basic statistics.
    """

    # Load the Digits dataset
    digits = load_digits()

    # Visualize some images
    fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 4))
    for ax, image, label in zip(axes.flat, digits.images, digits.target):
        ax.imshow(image, cmap='gray')
        ax.set_title(f"Label: {label}")
        ax.axis('off')
    plt.show()

    # Print some basic statistics
    print("Number of samples:", len(digits.data))
    print("Number of features per sample:", len(digits.data[0]))
    print("Number of classes:", len(digits.target_names))
    print("Class distribution:")
    for i in range(len(digits.target_names)):
        print(f"Class {i}: {sum(digits.target == i)} samples")
```



TASK 002

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object
operation = "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True

selection at the end -add
mirror_mod.select= 1
mirror_mod.select=1
context.scene.objects.active
("Selected" + str(modifier))
mirror_mod.select = 0
bpy.context.selected_objects
data.objects[one.name].select
int("please select exactly one
- OPERATOR CLASSES ---

types.Operator):
X mirror to the selected
object.mirror_mirror_x"
mirror X"

context):
next.active_object is not
```

Feature Engineering and Model Building

The function task002() focuses on feature engineering and model building:

- Loading a dataset of iris flowers
https://en.wikipedia.org/wiki/Iris_flower_data_set
- Create a scatter plot of the iris dataset with the first two features as the x and y coordinates, respectively. Points are colored based on the species of iris, and a legend is added to indicate the classes.
- Preprocessing the data.
- Splitting the data into training and testing sets.
- Training a decision tree classifier to predict flower species.

```
def task002():
    """
    This function performs a simple classification task using scikit-learn on the Digits dataset.
    """

    # Load the Digits dataset
    digits = load_digits()

    # Split the dataset into features (X) and target (y)
    X = digits.data
    y = digits.target

    # Split the dataset 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)

    # Initialize the Support Vector Classifier
    clf = SVC()

    # Train the classifier using the training set
    clf.fit(X_train, y_train)

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

    # Calculate the accuracy of the predictions
    accuracy = accuracy_score(y_test, predictions)

    print("Accuracy:", accuracy)
```

TASK 003

```
mirror_mod = modifier_ob.  
# mirror object to mirror  
# mirror_mod.mirror_object  
  
if operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
  
    if operation == "MIRROR_Y":  
        mirror_mod.use_x = False  
        mirror_mod.use_y = True  
        mirror_mod.use_z = False  
  
    if operation == "MIRROR_Z":  
        mirror_mod.use_x = False  
        mirror_mod.use_y = False  
        mirror_mod.use_z = True  
  
# selection at the end - add  
# mirror_mod.select= 1  
# mirror_mod.select=1  
# context.scene.objects.active  
# ("Selected" + str(modifier))  
# mirror_mod.select = 0  
# bpy.context.selected_objects  
# data.objects[one.name].select  
  
int("please select exactly one object")  
  
- OPERATOR CLASSES ---  
  
@types.Operator:  
    X mirror to the selected  
    object.mirror_mirror_x"  
    or X"  
  
@context:  
    context.active_object is not
```

Clustering

The function task003() performs clustering using scikit-learn on a synthetic blobs dataset:

- Generates synthetic blobs dataset.
- Applies K-Means clustering.
- Visualizes the clustering result.

```
def task003():
    """
    This function performs clustering using scikit-learn on a synthetic blobs dataset.
    """

    # Generate synthetic blobs dataset
    X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

    # Initialize K-Means clustering model
    kmeans = KMeans(n_clusters=4, random_state=42)

    # Fit the model to the data
    kmeans.fit(X)

    # Get cluster labels
    cluster_labels = kmeans.labels_

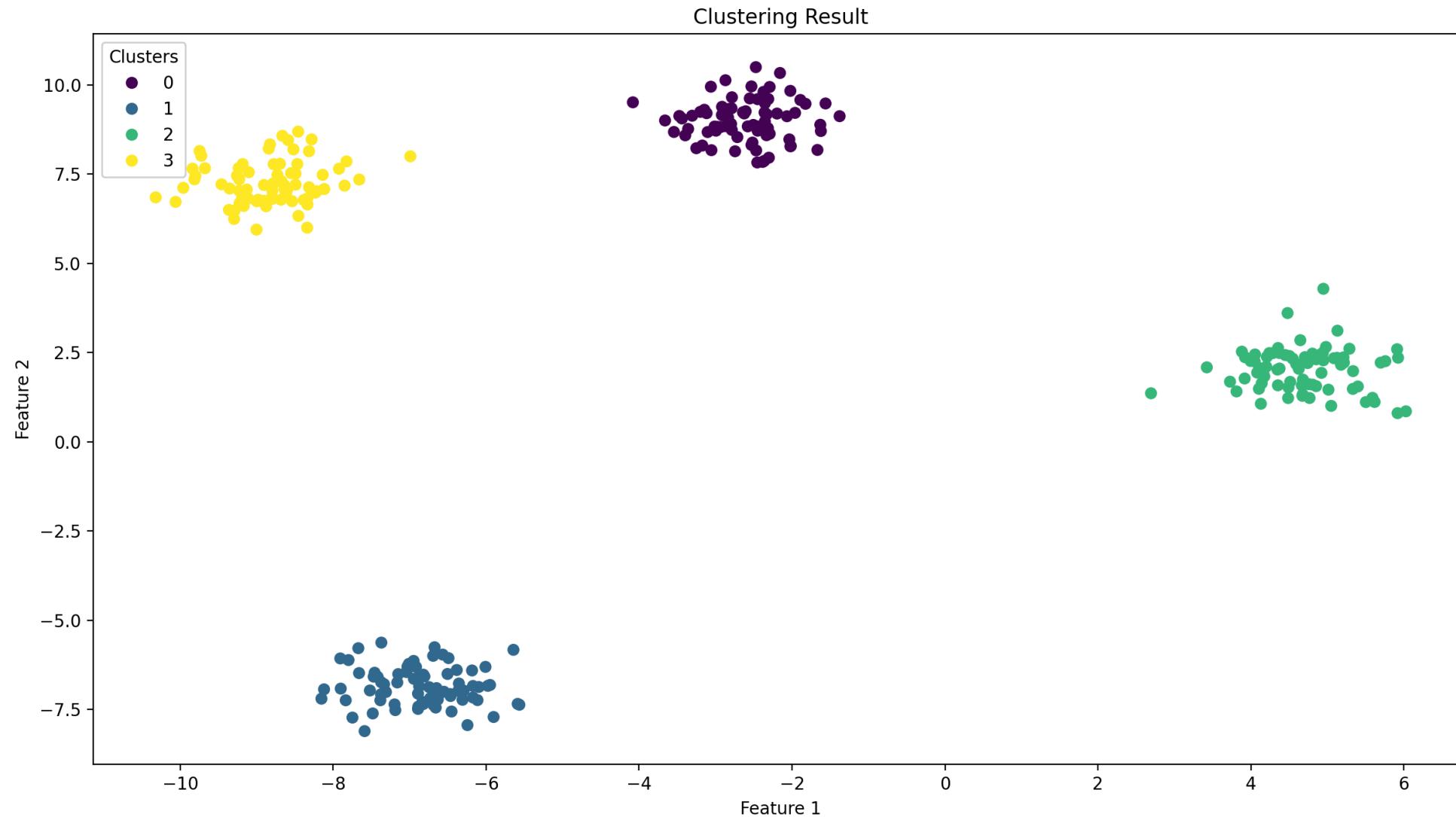
    # Visualize the clustering result
    fig, ax = plt.subplots()

    scatter = ax.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_title('Clustering Result')

    # Add legend
    legend = ax.legend(*scatter.legend_elements(), title="Clusters", loc="upper left")
    ax.add_artist(legend)

    plt.show()
```

Figure 1



TASK 004

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object = True
if operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

# selection at the end - add
if bpy.context.scene.objects.active:
    if bpy.context.scene.objects.active.name != "Selected" + str(modifier_index):
        mirror_mod.select = 0
    else:
        bpy.context.selected_objects.append(bpy.context.scene.objects[one.name])
        bpy.context.scene.objects[one.name].select = True
else:
    int("please select exactly one object")
    return

# OPERATOR CLASSES ---
# Operator class for mirroring objects
class MirrorOperator(bpy.types.Operator):
    bl_idname = "object.mirror"
    bl_label = "Mirror Object"
    bl_description = "Mirrors selected objects across the X, Y, or Z axis"

    def execute(self, context):
        # Get selected objects
        selected_objects = context.selected_objects
        if len(selected_objects) < 1:
            self.report({'ERROR'}, "Select at least one object")
            return {'CANCELLED'}

        # Create a mirror modifier for each selected object
        for obj in selected_objects:
            if obj.type == 'MESH':
                modifier = obj.modifiers.new("Mirror", 'MIRROR')
                modifier.mirror_object = obj
                modifier.use_x = True
                modifier.use_y = False
                modifier.use_z = False
            else:
                self.report({'WARNING'}, "Only mesh objects can be mirrored")
                continue
```

Dimensionality Reduction

The function `task004()` demonstrates the use of Principal Component Analysis (PCA) using scikit-learn:

- Loads the Iris dataset.
- Standardizes the features.
- Applies PCA to reduce the dimensionality to 2 dimensions.
- Visualizes the transformed data in the new feature space.

```
def task004():
    """
    This function demonstrates the use of Principal Component Analysis (PCA) using scikit-learn.
    """

    # Load the Iris dataset
    iris = load_iris()

    # Extract features
    X = iris.data

    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Initialize PCA model
    pca = PCA(n_components=2)

    # Fit the model to the data
    pca.fit(X_scaled)

    # Transform the data to the new feature space
    X_pca = pca.transform(X_scaled)
```

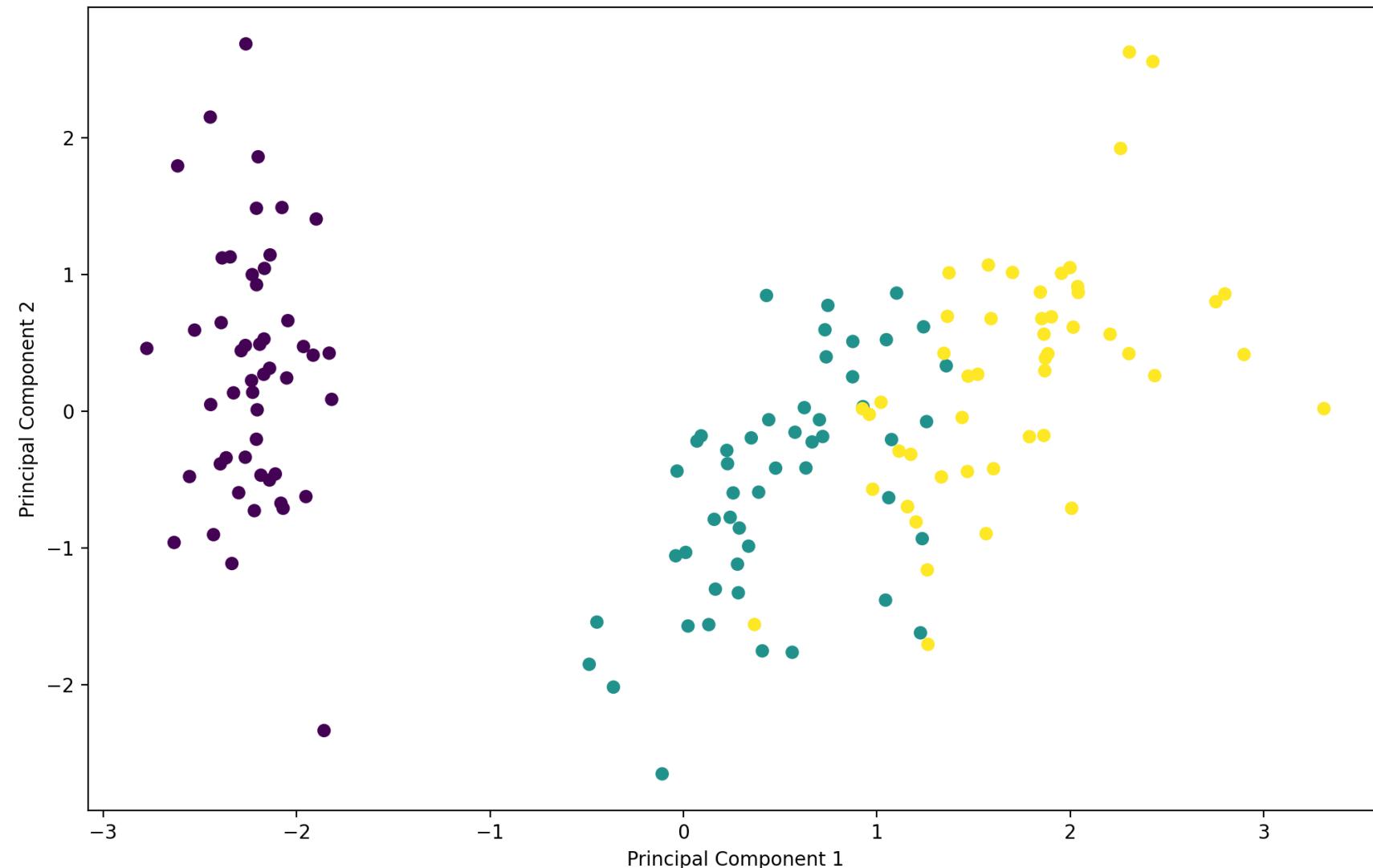
```
# Transform the data to the new feature space
X_pca = pca.transform(X_scaled)

# Visualize the transformed data
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=iris.target, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.show()

# Explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

Figure 1

PCA of Iris Dataset



TASK 005

```
mirror_mod = modifier_obj.mirror  
# mirror object to mirror  
# mirror_mod.mirror_object  
  
if operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
  
    if mirror_mod.use_x:  
        mirror_mod.operation = "MIRROR_Y"  
        mirror_mod.use_x = False  
        mirror_mod.use_y = True  
        mirror_mod.use_z = False  
  
        if mirror_mod.use_y:  
            mirror_mod.operation = "MIRROR_Z"  
            mirror_mod.use_x = False  
            mirror_mod.use_y = False  
            mirror_mod.use_z = True  
  
            if mirror_mod.use_z:  
                selection_at_the_end = True  
  
                if len(modifier_obj.select) > 1:  
                    bpy.context.scene.objects.active = modifier_obj  
                    bpy.context.selected_objects.append(modifier_obj)  
                    bpy.context.selected_objects.remove(modifier_obj)  
                    bpy.context.scene.objects[one.name].select = True  
  
                    if selection_at_the_end:  
                        bpy.context.scene.objects.active = modifier_obj  
                        bpy.context.selected_objects.append(modifier_obj)  
                        bpy.context.selected_objects.remove(modifier_obj)  
                        bpy.context.scene.objects[one.name].select = True  
  
                else:  
                    bpy.context.scene.objects.active = modifier_obj  
                    bpy.context.selected_objects.append(modifier_obj)  
                    bpy.context.selected_objects.remove(modifier_obj)  
                    bpy.context.scene.objects[one.name].select = True  
  
            else:  
                if len(modifier_obj.select) > 1:  
                    bpy.context.scene.objects.active = modifier_obj  
                    bpy.context.selected_objects.append(modifier_obj)  
                    bpy.context.selected_objects.remove(modifier_obj)  
                    bpy.context.scene.objects[one.name].select = True  
  
                else:  
                    bpy.context.scene.objects.active = modifier_obj  
                    bpy.context.selected_objects.append(modifier_obj)  
                    bpy.context.selected_objects.remove(modifier_obj)  
                    bpy.context.scene.objects[one.name].select = True  
  
        else:  
            if len(modifier_obj.select) > 1:  
                bpy.context.scene.objects.active = modifier_obj  
                bpy.context.selected_objects.append(modifier_obj)  
                bpy.context.selected_objects.remove(modifier_obj)  
                bpy.context.scene.objects[one.name].select = True  
  
            else:  
                bpy.context.scene.objects.active = modifier_obj  
                bpy.context.selected_objects.append(modifier_obj)  
                bpy.context.selected_objects.remove(modifier_obj)  
                bpy.context.scene.objects[one.name].select = True  
  
    else:  
        if len(modifier_obj.select) > 1:  
            bpy.context.scene.objects.active = modifier_obj  
            bpy.context.selected_objects.append(modifier_obj)  
            bpy.context.selected_objects.remove(modifier_obj)  
            bpy.context.scene.objects[one.name].select = True  
  
        else:  
            bpy.context.scene.objects.active = modifier_obj  
            bpy.context.selected_objects.append(modifier_obj)  
            bpy.context.selected_objects.remove(modifier_obj)  
            bpy.context.scene.objects[one.name].select = True  
  
else:  
    if len(modifier_obj.select) > 1:  
        bpy.context.scene.objects.active = modifier_obj  
        bpy.context.selected_objects.append(modifier_obj)  
        bpy.context.selected_objects.remove(modifier_obj)  
        bpy.context.scene.objects[one.name].select = True  
  
    else:  
        bpy.context.scene.objects.active = modifier_obj  
        bpy.context.selected_objects.append(modifier_obj)  
        bpy.context.selected_objects.remove(modifier_obj)  
        bpy.context.scene.objects[one.name].select = True  
  
# --- OPERATOR CLASSES ---  
  
class MIRROR_OT_Mirror(bpy.types.Operator):  
    bl_idname = "mirror.mirror"  
    bl_label = "X mirror to the selected object"  
    bl_options = {'REGISTER', 'UNDO'}  
  
    def execute(self, context):  
        if context.active_object is not None:  
            if context.active_object.type == "MESH":  
                self.execute_mesh(context)  
            else:  
                self.execute_other(context)  
        else:  
            self.execute_other(context)  
  
        return {'FINISHED'}
```

Classification and Model Evaluation

The function `task005()` loads the Iris dataset, preprocesses the data, trains a logistic regression model, and evaluates its performance on training and testing sets:

- Loads the Iris dataset.
- Creates a scatter plot.
- Preprocesses the data.
- Trains a **logistic regression** model.
- Evaluates the model's performance.



github.com/clipsound/APAI2024_student

It's your turn now.

TASK 006

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object = modifier
if modifier.operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif modifier.operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif modifier.operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

# selection at the end - add
if bpy.context.scene.objects.active == modifier:
    modifier.select = 1
else:
    modifier.select = 0
bpy.context.selected_objects.append(modifier)
data.objects[one.name].select = 1

int("please select exactly one object")
# - OPERATOR CLASSES ---

@types.Operator:
def execute(self, context):
    # X mirror to the selected
    obj = context.object
    obj.mirror_mirror_x = True
    if context:
        # context.active_object is not
        # valid here, because we have
        # already selected the mirror
        # object above
        # so we can't use it here
        # to get the original object
        # to mirror to
```

Classification and ROC Curve

The function `task006()` loads the Credit Card Fraud Detection dataset, preprocesses the data, trains a logistic regression model, and plots the ROC curve:

- Loads the Credit Card Fraud Detection dataset.
- Preprocesses the data.
- Trains a logistic regression model.
- Plots the ROC curve to evaluate the model's performance.

```
def task006():
    """
    This function loads the Credit Card Fraud Detection dataset, preprocesses the data, trains a logistic regression model and plots the ROC curve.
    """

    # Carica il dataset Credit Card Fraud Detection
    data = fetch_openml(name='creditcard', version=1)

    # Divide i dati in feature e target
    X = data.data
    y = data.target

    # Dividi i dati in set di training e di testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Standardizza le feature
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Plot della curva ROC per la regressione logistica
    y_train = y_train.astype(int)
    y_test = y_test.astype(int)
    plot_roc_curve_logistic_regression(X_train, y_train, X_test, y_test)
```

```
def plot_roc_curve_logistic_regression(X_train, y_train, X_test, y_test):
    """
    This function plots the ROC curve for the logistic regression model on the training and testing data.
    """

    # Initialize and train logistic regression model
    log_reg = LogisticRegression()
    log_reg.fit(X_train, y_train)

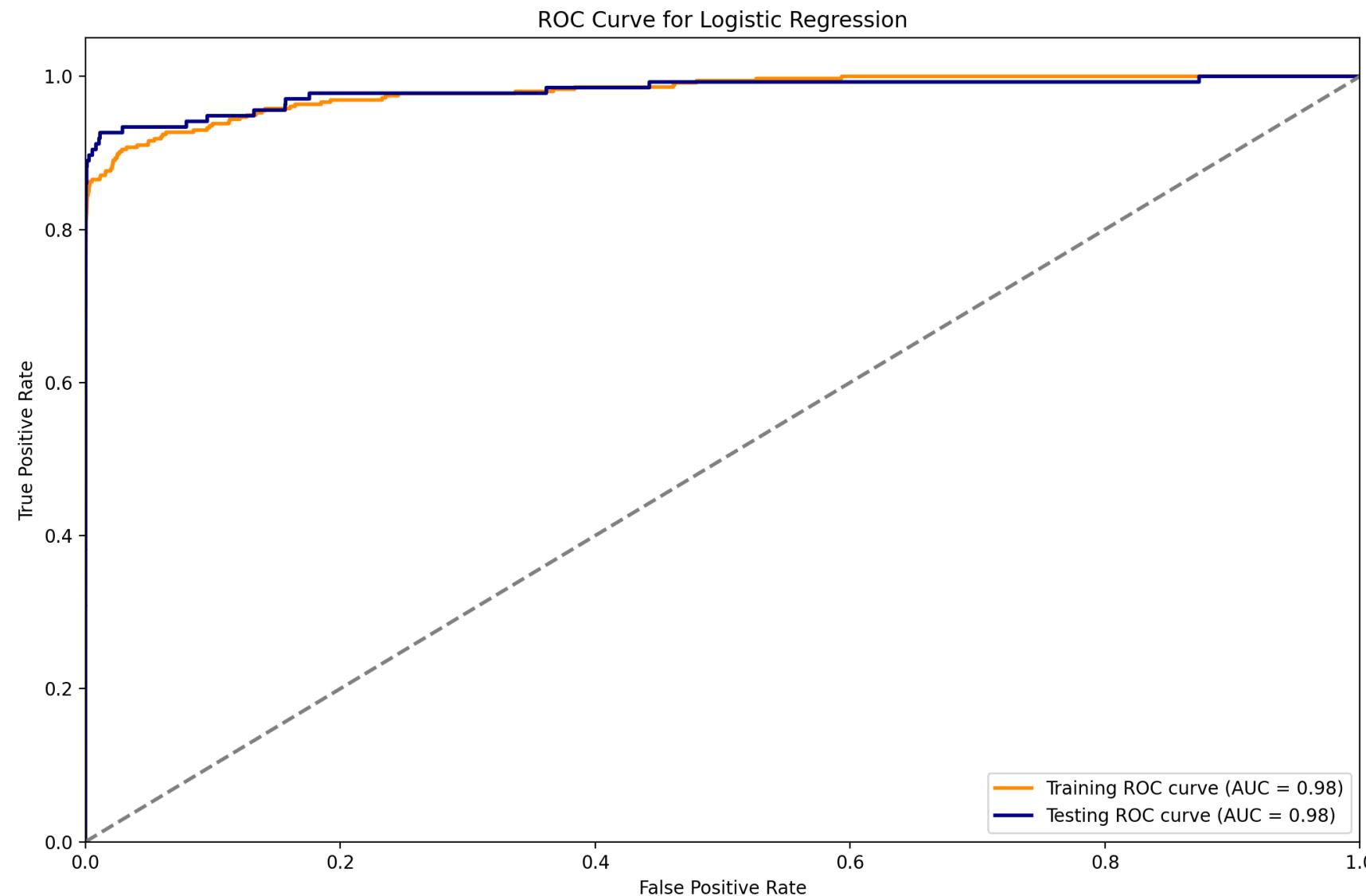
    # Predict probabilities for positive class
    y_train_prob = log_reg.predict_proba(X_train)[:, 1]
    y_test_prob = log_reg.predict_proba(X_test)[:, 1]

    # Compute ROC curve and ROC area for training set
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_prob)
    roc_auc_train = auc(fpr_train, tpr_train)

    # Compute ROC curve and ROC area for testing set
    fpr_test, tpr_test, _ = roc_curve(y_test, y_test_prob)
    roc_auc_test = auc(fpr_test, tpr_test)

    # Plot ROC curve
    plt.figure()
    lw = 2
    plt.plot(fpr_train, tpr_train, color='darkorange', lw=lw, label=f'Training ROC curve (AUC = {roc_auc_train:.2f})')
    plt.plot(fpr_test, tpr_test, color='navy', lw=lw, label=f'Testing ROC curve (AUC = {roc_auc_test:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Logistic Regression')
    plt.legend(loc="lower right")
    plt.show()
```

Figure 2



Thanks

