NAVTCC-SSUET Karachi

Course Code: AI-2002	**Course:** Artificial IntelligencE
Instructors	Sohail Ahmed Malik

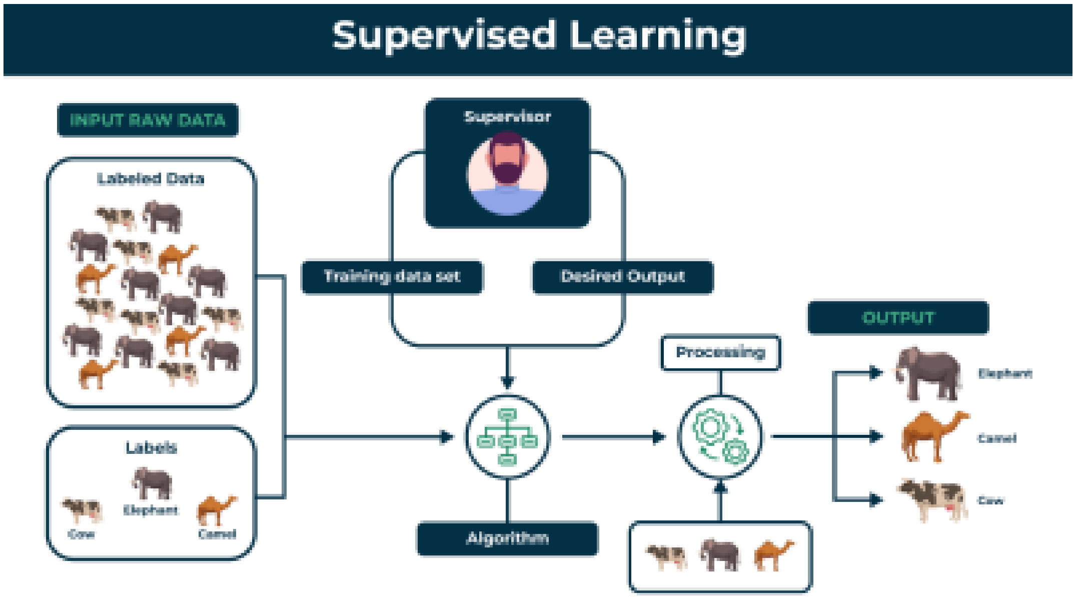
Objectives:

- Supervised Learning
- Unsupervised Learning
- Performance Measuring

What is Supervised learning?
Supervised learning is a type of machine learning algorithm that learns from labeled data. Labeled data is data that has been tagged with a correct answer or classification.

Supervised learning, as the name indicates, has the presence of a supervisor as a teacher. Supervised learning is when we teach or train the machine using data that is well-labelled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labeled data.

For example, a labeled dataset of images of Elephant, Camel and Cow would have each image tagged with either “Elephant”, “Camel “or “Cow.”



Key Points:

- Supervised learning involves training a machine from labeled data.
- Labeled data consists of examples with the correct answer or classification.
- The machine learns the relationship between inputs (fruit images) and outputs (fruit labels).
- The trained machine can then make predictions on new, unlabeled data.

Example:
Let’s say you have a fruit basket that you want to identify. The machine would first analyze the image to extract features such as its shape, color, and texture. Then, it would compare these features to the features of the fruits it has already learned about. If the new image’s features are most similar to those of an apple, the machine would predict that the fruit is an apple.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all the different fruits one by one like this:

• If the shape of the object is rounded and has a depression at the top, is red in color, then it will be labeled as –**Apple**. • If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.

Since the machine has already learned the things from previous data and this time has to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category. Thus the machine learns the things from training data (basket containing fruits) and then applies the knowledge to test data (new fruit).

Types of Supervised Learning

Supervised learning is classified into two categories of algorithms:

- **Regression: -** A regression problem is when the output variable is a real value, such as "dollars" or "weight". • **Classification: -** A classification problem is when the output variable is a category, such as "Red" or "blue", "disease" or "no disease".

Supervised learning deals with or learns with "labeled" data. This implies that some data is already tagged with the correct answer.

Regression

Regression is a type of supervised learning that is used to predict continuous values, such as house prices, stock prices, or customer churn. Regression algorithms learn a function that maps from the input features to the output value.

Some common regression algorithms include:

- Linear Regression
- Polynomial Regression
- Support Vector Machine Regression
- Decision Tree Regression
- Random Forest Regression

Code Implementations

Linear Regression

Definition- Linear regression is a statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (features) by fitting a linear equation to the observed data.

Use Cases

- Predicting house prices based on features like area, number of bedrooms, etc.
- Forecasting sales revenue based on advertising spend. • Estimating the demand for a product based on historical sales data.

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Sample data (house areas in sq. ft. and prices in USD) X =
np.array([[1500], [1800], [2200], [2500]]) # Input features (house
areas)
y = np.array([300000, 350000, 400000, 450000]) # Target
variable (house prices)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict price for a new house with area 2000 sq. ft.
new_area = np.array([[2000]])
predicted_price = model.predict(new_area)
print("Predicted Price:", predicted_price) # Output:
[383333.33333333]
```

Polynomial Regression

Definition- Polynomial regression extends linear regression by adding polynomial terms to the model, allowing it to capture non linear relationships between features and target.

Use Cases

- Modeling physical phenomena where the relationship between variables is non-linear.
- Predicting the growth rate of organisms based on time and environmental factors.
- Capturing the effect of interactions between variables in the model.

```
from sklearn.preprocessing import
PolynomialFeatures from sklearn.linear_model
import LinearRegression
import numpy as np

# Sample data (input features and target variable)
X = np.array([[1], [2], [3], [4]])
y = np.array([1, 4, 9, 16]) # Quadratic relationship (y = x^2)

# Transform features to polynomial features
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)

# Create and fit the polynomial regression model
model_poly = LinearRegression()
model_poly.fit(X_poly, y)

# Predict for new values
X_new = np.array([[5]])
X_new_poly = poly_features.transform(X_new)
predicted_value = model_poly.predict(X_new_poly)
print("Predicted Value (Polynomial Regression):",
predicted_value)  # Output: [25.]
```

Support Vector Machine (SVM) Regression

Definition- Support Vector Machine (SVM) regression finds a hyperplane that best fits the data by maximizing the margin between support vectors and the hyperplane.

Use Cases

- Predicting stock prices based on historical data.
- Forecasting future demand for products or services. • Estimating the price of real estate based on property features.

```
from sklearn.svm import SVR
import numpy as np

# Sample data (input features and target variable)
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 3, 4, 5]) # Linear relationship (y = x + 1)

# Create and fit the SVM regression model
svm_regressor = SVR(kernel='linear')
svm_regressor.fit(X, y)

# Predict for new values
X_new = np.array([[5]])
predicted_value = svm_regressor.predict(X_new)
print("Predicted Value (SVM Regression):", predicted_value) #
Output: [6.]
```

Decision Tree Regression

Definition- Decision tree regression divides the feature space into regions and predicts the average target value within each region.

Use Cases

- Predicting the price of a used car based on its age, mileage, and other features.
- Estimating the energy consumption of a building based on weather conditions.
- Forecasting the sales volume of a product based on marketing campaigns.

```
from sklearn.tree import DecisionTreeRegressor
import numpy as np

# Sample data (input features and target variable)
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 3, 4, 5]) # Linear relationship (y = x + 1)

# Create and fit the decision tree regression model
tree_regressor = DecisionTreeRegressor()
tree_regressor.fit(X, y)

# Predict for new values
X_new = np.array([[5]])
predicted_value = tree_regressor.predict(X_new)
print("Predicted Value (Decision Tree Regression):",
predicted_value) # Output: [6.]
```

Random Forest Regression

Definition- Random forest regression is an ensemble learning method that builds multiple decision trees and averages their predictions to improve accuracy and reduce overfitting.

Use Cases

- Predicting the yield of a crop based on weather conditions and soil properties.
- Estimating the price of a product based on customer demographics and market data.
- Forecasting the sales revenue of a retail store based on historical data.

```
from sklearn.ensemble import
RandomForestRegressor import numpy as np

# Sample data (input features and target variable)
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 3, 4, 5]) # Linear relationship (y = x + 1)

# Create and fit the random forest regression model
forest_regressor = RandomForestRegressor()
forest_regressor.fit(X, y)

# Predict for new values
X_new = np.array([[5]])
predicted_value = forest_regressor.predict(X_new)
print("Predicted Value (Random Forest Regression):",
predicted_value) # Output: [6.]
```

Classification

Classification is a type of supervised learning that is used to predict categorical values, such as whether a customer will churn or not, whether an email is spam or not, or whether a medical image shows a tumor or not. Classification algorithms learn a function that maps from the input features to a probability distribution over the output classes.

Some common classification Algorithms include:

- Logistic Regression
- Support Vector Machines
- Decision Trees
- Random Forests
- Naive Bayes

Code Implementations

Logistic Regression

Definition- Logistic regression is a statistical method used for binary or multiclass classification by modeling the probability of a categorical outcome using a logistic (sigmoid) function.

Use Cases

- Predicting whether an email is spam or not.
- Classifying customer churn (yes/no).
- Identifying the type of iris flower based on petal and sepal measurements.

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (Logistic Regression):", accuracy)
```

Support Vector Machines (SVM)

Definition- Support Vector Machines (SVM) are supervised learning models used for classification and regression tasks by finding the optimal hyperplane that best separates classes in a high-dimensional space.

Use Cases

- Text classification (e.g., sentiment analysis).
- Image classification.
- Anomaly detection in network traffic.

```
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit the SVM classifier
model = SVC(kernel='linear')
model.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (SVM):", accuracy)
```

Decision Trees

Definition- Decision trees are hierarchical models that partition the feature space into regions and assign class labels to each region based on majority voting.

Use Cases

- Customer segmentation based on purchasing behavior. •
Medical diagnosis based on patient symptoms.
- Credit scoring for loan approval.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit the decision tree classifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (Decision Tree):", accuracy)
```

Random Forests

Definition- Random forests are ensemble learning methods that build multiple decision trees and aggregate their predictions to improve accuracy and reduce overfitting.

Use Cases

- Predicting the species of plants based on botanical features.
- Detecting fraudulent transactions in financial transactions.
- Recommender systems based on user preferences.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit the random forest classifier
model = RandomForestClassifier()
model.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (Random Forest):", accuracy)
```

Naive Bayes

Definition- Naive Bayes classifiers apply Bayes' theorem with strong (naive) independence assumptions between features.

Use Cases

- Text classification tasks such as spam detection and sentiment analysis.
- Document categorization based on content.
- Medical diagnosis based on symptoms and test results.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (Naive Bayes):", accuracy)
```

Evaluating Supervised Learning Models

Evaluating supervised learning models is an important step in ensuring that the model is accurate and generalizable. There are a number of different matrics that can be used to evaluate supervised learning models, but some of the most common ones include:

For Regression

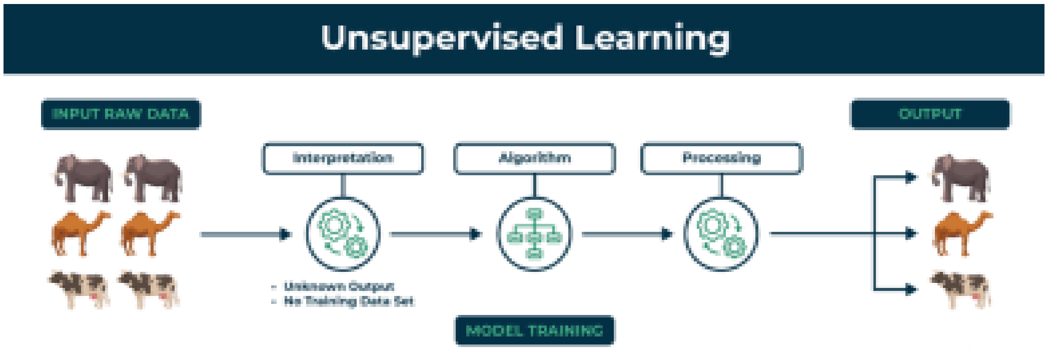
- **Mean Squared Error (MSE):** MSE measures the average squared difference between the predicted values and the actual values. Lower MSE values indicate better model performance.
- **Root Mean Squared Error (RMSE):** RMSE is the square root of MSE, representing the standard deviation of the prediction errors. Similar to MSE, lower RMSE values indicate better model performance.
- **Mean Absolute Error (MAE):** MAE measures the average absolute difference between the predicted values and the actual values. It is less sensitive to outliers compared to MSE or RMSE.
- **R-squared (Coefficient of Determination):** R-squared measures the proportion of the variance in the target variable that is explained by the model. Higher R-squared values indicate better model fit.

For Classification

- **Accuracy:** Accuracy is the percentage of predictions that the model makes correctly. It is calculated by dividing the number of correct predictions by the total number of predictions.
- **Precision:** Precision is the percentage of positive predictions that the model makes that are actually correct. It is calculated by dividing the number of true positives by the total number of positive predictions.
- **Recall:** Recall is the percentage of all positive examples that the model correctly identifies. It is calculated by dividing the number of true positives by the total number of positive examples.
- **F1 score:** The F1 score is a weighted average of precision and recall. It is calculated by taking the harmonic mean of precision and recall.
- **Confusion matrix:** A confusion matrix is a table that shows the number of predictions for each class, along with the actual class labels. It can be used to visualize the performance of the model and identify areas where the model is struggling.

What is Unsupervised learning?

Unsupervised learning is a type of machine learning that learns from unlabeled data. This means that the data does not have any pre-existing labels or categories. The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.



Key Points

- Unsupervised learning allows the model to discover patterns and relationships in unlabeled data.
- Clustering algorithms group similar data points together based on their inherent characteristics.
- Feature extraction captures essential information from the data, enabling the model to make meaningful distinctions. •
Label association assigns categories to the clusters based on the extracted patterns and characteristics.

Example

Imagine you have a machine learning model trained on a large dataset of unlabeled images, containing both dogs and cats. The model has never seen an image of a dog or cat before, and it has no pre-existing labels or categories for these animals. Your task is to use unsupervised learning to identify the dogs and cats in a new, unseen image.

For instance, suppose it is given an image having both dogs and cats which it has never seen.

Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having **dogs** in them and the second part may contain all pics having **cats** in them.

Types of Unsupervised Learning

Unsupervised learning is classified into two categories of algorithms:

1. **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
2. **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Clustering

Clustering is a type of unsupervised learning that is used to group similar data points together. Clustering Algorithms work by iteratively moving data points closer to their cluster centers and further away from data points in other clusters.

- Exclusive (partitioning)
- Agglomerative
- Overlapping
- Probabilistic

Clustering Types: -

- Hierarchical clustering
- K-means clustering
- Principal Component Analysis
- Gaussian Mixture Models (GMMs)
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Hierarchical Clustering

Definition- Hierarchical clustering is a method of cluster analysis that builds a hierarchy of clusters by either recursively merging or splitting clusters based on distance criteria.

Use Cases

- Grouping similar documents or articles in text mining. •
Creating taxonomies or dendrograms in biological sciences. •
Customer segmentation based on purchasing behavior.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60,random_state=0)

# Perform Hierarchical Clustering
agg_clustering = AgglomerativeClustering(n_clusters=4)
labels = agg_clustering.fit_predict(X)

# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("Hierarchical Clustering")
plt.show()
```

K-means Clustering

Definition: K-means clustering partitions data into K clusters by iteratively assigning data points to the nearest cluster centroid and updating centroids based on the mean of points assigned to each cluster.

Use Cases:

- Market segmentation based on customer demographics. • Image segmentation for object recognition.
- Analyzing clickstream data for user behavior patterns.

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Perform K-means Clustering
kmeans = KMeans(n_clusters=4)
labels = kmeans.fit_predict(X)

# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200,
c='red', marker='X', label='Centroids')
plt.title("K-means Clustering")
plt.legend()
plt.show()
```

Principal Component Analysis (PCA)

Definition: Principal Component Analysis (PCA) is a dimensionality reduction technique that projects data onto a lower-dimensional space while preserving the maximum variance.

Use Cases:

- Feature extraction and data visualization.
- Noise reduction in signal processing.
- Enhancing the performance of machine learning models by reducing multicollinearity.

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform PCA for visualization (2D projection)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Visualize PCA-transformed data
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title("PCA Visualization of Iris Dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

Gaussian Mixture Models (GMMs)

Definition: Gaussian Mixture Models (GMMs) assume that the data is generated from a mixture of several Gaussian distributions and estimate the parameters of these distributions to perform probabilistic clustering.

Use Cases:

- Image segmentation in computer vision.
- Density estimation for anomaly detection.
- Modeling complex data distributions in unsupervised learning.

```
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Perform Gaussian Mixture Model (GMM) clustering
gmm = GaussianMixture(n_components=4)
labels = gmm.fit_predict(X)

# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("Gaussian Mixture Model (GMM) Clustering")
plt.show()
```

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Definition: DBSCAN is a density-based clustering algorithm that groups together closely packed points based on a density threshold.

Use Cases:

- Identifying spatial hotspots in geographic data.
- Anomaly detection in cybersecurity.
- Clustering large datasets with irregular shapes and noise.

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Perform DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)

# Visualize the clusters (including noise points)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("DBSCAN Clustering")
plt.show()
```

Association rule learning

Association rule learning is a type of unsupervised learning that is used to identify patterns in a data. Association Rule learning algorithms work by finding relationships between different items in a dataset.

Some common association rule learning algorithms include:

- Apriori Algorithm
- FP-Growth Algorithm

Apriori Algorithm

Definition: The Apriori algorithm is an association rule mining algorithm that discovers frequent itemsets in a transaction database. It uses an iterative approach to generate candidate itemsets and prune the search space by leveraging the "apriori" property.

Use Cases:

- Market basket analysis to identify frequently co-occurring items in customer transactions.
- Recommender systems to suggest related products based on purchase history.
- Web usage mining to understand sequential patterns in user behavior.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# Sample transaction dataset (list of lists)
dataset = [['bread', 'milk', 'eggs'],
 ['bread', 'butter'],
 ['milk', 'butter'],
 ['bread', 'milk', 'butter'],
 ['bread', 'milk']]

# One-hot encode the dataset
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df, min_support=0.4,
use_colnames=True) print(frequent_itemsets)
```

FP-Growth Algorithm

Definition: The FP-Growth (Frequent Pattern Growth) algorithm is a frequent itemset mining algorithm that constructs a compact data structure called the FP-tree to efficiently mine frequent itemsets without generating candidate itemsets.

Use Cases:

- Market basket analysis for identifying association rules. • Bioinformatics for identifying frequent sequence patterns in DNA sequences.
- Telecommunication for analyzing call patterns and customer behaviors.

```
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# Sample transaction dataset (list of lists)
dataset = [['bread', 'milk', 'eggs'],
 ['bread', 'butter'],
 ['milk', 'butter'],
 ['bread', 'milk', 'butter'],
 ['bread', 'milk']]

# One-hot encode the dataset
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply FP-Growth algorithm to find frequent itemsets
frequent_itemsets = fpgrowth(df, min_support=0.4,
use_colnames=True) print(frequent_itemsets)
```

Evaluating Non-Supervised Learning Models

Evaluating non-supervised learning models is an important step in ensuring that the model is effective and useful. However, it can be more challenging than evaluating supervised learning models, as there is no ground truth data to compare the model’s predictions to. There are a number of different metrics that can be used to evaluate non-supervised learning models, but some of the most common ones include:

- **Silhouette score:** The silhouette score measures how well each data point is clustered with its own cluster members and separated from other clusters. It ranges from -1 to 1, with higher scores indicating better clustering.
- **Calinski-Harabasz score:** The Calinski-Harabasz score measures the ratio between the variance between clusters and the variance within clusters. It ranges from 0 to infinity, with higher scores indicating better clustering.
- **Adjusted Rand index:** The adjusted Rand index measures the similarity between two clusterings. It ranges from -1 to 1, with higher scores indicating more similar clusterings.
- **Davies-Bouldin index:** The Davies-Bouldin index measures the average similarity between clusters. It ranges from 0 to infinity, with lower scores indicating better clustering.
- **F1 score:** The F1 score is a weighted average of precision and recall, which are two metrics that are commonly used in supervised learning to evaluate classification models. However, the F1 score can also be used to evaluate non-supervised learning models, such as clustering models.

Tasks:

1. Classification Task with Breast Cancer Dataset:
    - Load the Breast Cancer dataset (load_breast_cancer()) from sklearn.datasets.
    - Split the dataset into training and testing sets.
    - Implement a classification model (e.g., Logistic Regression, Decision Tree) to predict cancer diagnosis (malignant or benign). • Evaluate the model's accuracy and other metrics.

2. Regression Task with Boston Housing Dataset:
    - Load the Boston Housing dataset (load_boston()) from sklearn.datasets.
    - Split the dataset into training and testing sets.
    - Implement a regression model (e.g., Linear Regression, Random Forest) to predict house prices.
    - Calculate and analyze the model's performance using metrics like mean squared error.

3. Clustering Task with Iris Dataset:
    - Use the Iris dataset (load_iris()) from sklearn.datasets.
    - Apply K-means clustering to group similar iris flowers based on their features.
    - Visualize the clusters and compare them with the actual species labels.

4. Dimensionality Reduction Task with Wine Dataset:
    - Load the Wine dataset (load_wine()) from sklearn.datasets.
    - Apply PCA (Principal Component Analysis) or t-SNE (t-distributed Stochastic Neighbor Embedding) for dimensionality reduction.
    - Visualize the reduced-dimensional data and observe the separation of wine classes.

***Advanced Tasks**

1. Hyperparameter Tuning with Grid Search:
    - Select any dataset and supervised learning algorithm (e.g., SVM, Random Forest).
    - Use GridSearchCV to perform hyperparameter tuning by searching for the best combination of parameters. • Evaluate the tuned model on a hold-out test set and compare it with the default model.

2. Ensemble Learning Task:
    - Choose a classification or regression dataset (e.g., load_iris() or load_diabetes()).
    - Implement an ensemble learning method such as Random Forest or Gradient Boosting.
    - Compare the ensemble model's performance with individual base models.

3. Flower Classification- Dataset: Use the Flowers Recognition dataset available from Kaggle, containing images of flowers categorized into different classes (e.g., daisy, tulip, rose, sunflower). Use any Supervised Algorithm to Perform Classification and Use Stramlit to Give the UI to the Classification Task.