

dm lab



Perform K-Means with T-SNE on Digit dataset.

write about the topic and explain it in an easy way and impleemnt this in python



K-Means with t-SNE on the Digits Dataset: Explained

1. What is K-Means?

K-Means is a clustering algorithm that groups similar data points into clusters. It works by:

1. Initializing cluster centroids randomly.
2. Assigning each data point to the nearest centroid.
3. Updating centroids based on the mean of data points in each cluster.
4. Repeating until centroids stabilize or a maximum number of iterations is reached.

2. What is t-SNE?

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique that maps high-dimensional data to 2D or 3D for visualization. It preserves local similarities, making it excellent for visualizing clusters in complex datasets.

3. Why Combine K-Means and t-SNE?

t-SNE helps visualize high-dimensional data in a reduced space. When K-Means is applied on this reduced space:

- t-SNE simplifies patterns in the data.
- K-Means identifies clusters efficiently in the simplified space.

4. Digits Dataset

The Digits dataset contains images of handwritten digits (0-9) represented as 8x8 pixel grids, with 64 features (pixel intensities). The task is to cluster these digits into 10 groups.

5. Steps for Implementation

1. Load the Digits dataset.
2. Perform t-SNE to reduce dimensionality.
3. Apply K-Means clustering.
4. Visualize the clusters in the t-SNE-reduced space.

Here's the Python implementation:

python

```
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

# Load Digits dataset
digits = load_digits()
X = digits.data
y = digits.target # True labels (for visualization, not used in clustering)

# Step 1: Perform t-SNE for dimensionality reduction
tsne = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=300)
X_tsne = tsne.fit_transform(X)

# Step 2: Apply K-Means clustering
kmeans = KMeans(n_clusters=10, random_state=42)
clusters = kmeans.fit_predict(X_tsne)

# Step 3: Visualize the clusters
plt.figure(figsize=(10, 7))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=clusters, cmap='tab10', s=50, alpha=0.7)
plt.colorbar(scatter, ticks=range(10), label="Cluster ID")
plt.title("K-Means Clustering on t-SNE-Reduced Digits Data", fontsize=14)
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.show()
```

Explanation of the Code

1. Dataset Loading:

- `load_digits()` provides digit images as feature vectors.

2. t-SNE Dimensionality Reduction:

- Reduces 64 features to 2D for visualization while maintaining local structures.

3. K-Means Clustering:

- Groups the reduced 2D points into 10 clusters, corresponding to the digits.

4. Visualization:

- `plt.scatter` shows the clusters in a scatter plot, with colors distinguishing clusters.

Output

You'll see a scatter plot with 10 clusters, ideally separating digits. While the clusters won't perfectly align with true labels, patterns of similarity will emerge.

Applications

- Analyzing handwritten digit data.
- Visualizing patterns in high-dimensional datasets.
- Exploring unsupervised learning in real-world problems.



in r language



Here's how to perform **K-Means clustering with t-SNE** on the Digits dataset in R:

Step-by-Step Explanation

1. **Load Data:** Use the ``digits`` dataset from Python, or an equivalent dataset in R.
2. **Install Libraries:** Use libraries like ``Rtsne`` for t-SNE and ``stats`` for K-Means.
3. **t-SNE for Dimensionality Reduction:** Reduce high-dimensional data to 2D.
4. **Apply K-Means:** Cluster the t-SNE-reduced data.
5. **Visualize Clusters:** Use ``ggplot2`` to plot clusters.

R Implementation

```
R

# Install necessary packages if not already installed
if (!requireNamespace("Rtsne", quietly = TRUE)) install.packages("Rtsne")
if (!requireNamespace("ggplot2", quietly = TRUE)) install.packages("ggplot2")

library(Rtsne)
library(ggplot2)

# Step 1: Load the Digits dataset
# For demonstration, we'll use the 'digits' dataset from R or use random high-dimensional data
# Replace this with your dataset as required
set.seed(42)
n_samples <- 1000
n_features <- 64
digits_data <- matrix(runif(n_samples * n_features, min = 0, max = 1), ncol = n_features)
true_labels <- sample(0:9, n_samples, replace = TRUE) # True labels (optional for visualization)

# Step 2: Perform t-SNE for dimensionality reduction
tsne_result <- Rtsne(digits_data, dims = 2, perplexity = 30, max_iter = 300)

# Extract t-SNE coordinates
tsne_data <- as.data.frame(tsne_result$Y)
colnames(tsne_data) <- c("Component_1", "Component_2")

# Step 3: Apply K-Means clustering
k <- 10 # Number of clusters
kmeans_result <- kmeans(tsne_data, centers = k)

# Add cluster assignments to the t-SNE data
tsne_data$Cluster <- as.factor(kmeans_result$cluster)

# Optionally, add true labels for visualization (if available)
tsne_data$True_Label <- as.factor(true_labels)

# Step 4: Visualize the clusters using ggplot2
ggplot(tsne_data, aes(x = Component_1, y = Component_2, color = Cluster)) +
  geom_point(alpha = 0.7, size = 3) +
  labs(title = "K-Means Clustering on t-SNE-Reduced Digits Data",
       x = "t-SNE Component 1",
       y = "t-SNE Component 2") +
```

```
theme_minimal() +
scale_color_brewer(palette = "Set3")
```

Explanation of the Code

1. Dataset Simulation:

- We simulate a dataset with 64 features (like the digits dataset) for simplicity.
- Replace this with the actual Digits dataset in R or imported from Python.

2. t-SNE with `Rtsne`:

- Reduces high-dimensional data to 2D using parameters like `perplexity` and `max_iter`.

3. K-Means with `kmeans`:

- Groups the 2D data into 10 clusters.

4. Visualization with `ggplot2`:

- Scatter plot shows clusters with colors distinguishing them.

Expected Output

- A scatter plot where each color represents a cluster. Ideally, the clusters will align with digit classes.

Notes

- Adjust `perplexity` and `k` values to improve clustering and visualization.
- Replace the simulated data with actual datasets for meaningful results.



Perform K-Means with PCA on Digit dataset

write about the topic and explain it in an easy way and impleemnt this in python



K-Means with PCA on the Digits Dataset: Explained

1. What is PCA?

PCA (Principal Component Analysis) is a dimensionality reduction technique that transforms high-dimensional data into fewer dimensions while preserving most of the data's variance. It identifies the directions (principal components) where the data varies the most.

2. What is K-Means?

K-Means is an unsupervised clustering algorithm. It groups similar data points into clusters by:

- Assigning each point to the nearest centroid.

- Updating centroids based on the mean position of points in each cluster.

3. Why Combine K-Means and PCA?

- PCA reduces the dataset's dimensions, making clustering faster and easier by removing noise and redundant information.
- K-Means works better on the reduced data, as it focuses on the most significant features identified by PCA.

4. Digits Dataset

The Digits dataset contains images of handwritten digits (0-9) represented as 8x8 pixel grids (64 features). The task is to cluster these images into 10 groups.

5. Steps for Implementation

1. Load the Digits dataset.
2. Apply PCA to reduce dimensions.
3. Use K-Means on the PCA-transformed data.
4. Visualize the clustering results.

Python Implementation

python

```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Digits dataset
digits = load_digits()
X = digits.data
y = digits.target # True labels (optional, for evaluation and visualization)

# Step 1: Apply PCA for dimensionality reduction
pca = PCA(n_components=2) # Reduce to 2 dimensions for visualization
X_pca = pca.fit_transform(X)

# Step 2: Perform K-Means clustering
kmeans = KMeans(n_clusters=10, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Step 3: Visualize the clusters
plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette="tab10", s=50)
plt.title("K-Means Clustering on PCA-Reduced Digits Data")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title="Cluster")
plt.show()

# Step 4: Optional - Evaluate clustering with true labels
# (This is just for analysis, not used in unsupervised learning)
from sklearn.metrics import adjusted_rand_score
```

```
score = adjusted_rand_score(y, clusters)
print(f"Adjusted Rand Index: {score:.2f}")
```

Explanation of the Code

1. Dataset Loading:

- The `digits` dataset contains 64 features (pixel intensities) for each image.

2. PCA Dimensionality Reduction:

- Reduces the dataset to 2 components (`n_components=2`) for visualization.
- PCA helps retain the most significant patterns while discarding noise.

3. K-Means Clustering:

- Groups the reduced data into 10 clusters (corresponding to digits 0-9).

4. Visualization:

- A scatter plot shows clusters based on the two principal components. Colors represent different clusters.

5. Evaluation (Optional):

- Adjusted Rand Index (ARI) compares the clustering to the true labels for validation (though true labels are not typically available in real unsupervised tasks).

Expected Output

1. Scatter Plot:

- Clusters are represented in 2D space, with colors indicating different groups.

2. Adjusted Rand Index (Optional):

- A measure of similarity between K-Means clusters and true labels, where 1.0 indicates perfect agreement.

Applications

- Identifying patterns in high-dimensional datasets.
- Simplifying data for faster and more interpretable clustering.
- Exploring real-world datasets where labels are unknown.

By combining PCA with K-Means, you leverage the power of dimensionality reduction for efficient clustering!



- 3)
 - a) Construct decision tree classifier on weather.numeric dataset
 - b) Perform model evaluation metrics from the above.

write about the topic and explain it. and impleemnt this in both python and r programming languages



Decision Tree Classifier: Explained

1. What is a Decision Tree Classifier?

A decision tree is a machine learning algorithm used for classification and regression tasks. It splits the data into subsets based on feature values, creating a tree structure where:

- **Nodes** represent features.
- **Branches** represent decisions/rules.
- **Leaves** represent final outcomes (classes).

2. Weather Numeric Dataset

The `weather.numeric` dataset contains numerical weather-related features (e.g., temperature, humidity) and a target variable, often indicating whether to play a game (e.g., "PlayTennis").

3. Model Evaluation Metrics

To evaluate the decision tree classifier, common metrics include:

1. **Accuracy:** Proportion of correct predictions.
2. **Precision:** Proportion of true positive predictions among all positive predictions.
3. **Recall (Sensitivity):** Proportion of true positive predictions among all actual positives.
4. **F1-Score:** Harmonic mean of precision and recall.
5. **Confusion Matrix:** A table showing true vs. predicted values.

Python Implementation

Code for Decision Tree and Model Evaluation in Python:

```
python

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load a sample dataset similar to weather.numeric
data = {
    'Temperature': [85, 80, 83, 70, 68, 65, 75, 72, 69, 81],
    'Humidity': [85, 90, 86, 96, 80, 70, 80, 95, 70, 75],
    'Windy': [0, 1, 0, 0, 0, 1, 1, 0, 1, 1],
    'PlayTennis': [0, 0, 1, 1, 1, 1, 1, 0, 1, 0],
}
df = pd.DataFrame(data)

# Split dataset into features and target
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the Decision Tree Classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Model predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy:", accuracy_score(y_test, y_pred))

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, class_names=["No", "Yes"], filled=True)
plt.show()
```

R Implementation

Code for Decision Tree and Model Evaluation in R:

```
R

# Load necessary libraries
if (!require("rpart")) install.packages("rpart")
if (!require("rpart.plot")) install.packages("rpart.plot")
if (!require("caret")) install.packages("caret")

library(rpart)
library(rpart.plot)
library(caret)

# Load or create a dataset similar to weather.numeric
data <- data.frame(
  Temperature = c(85, 80, 83, 70, 68, 65, 75, 72, 69, 81),
  Humidity = c(85, 90, 86, 96, 80, 70, 80, 95, 70, 75),
  Windy = c(0, 1, 0, 0, 0, 1, 1, 0, 1, 1),
  PlayTennis = factor(c(0, 0, 1, 1, 1, 1, 1, 0, 1, 0))
)

# Train-test split
set.seed(42)
train_index <- createDataPartition(data$PlayTennis, p = 0.7, list = FALSE)
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# Create and train the Decision Tree model
model <- rpart(PlayTennis ~ ., data = train_data, method = "class")

# Visualize the Decision Tree
rpart.plot(model, type = 4, extra = 101, main = "Decision Tree for Weather Data")

# Make predictions
pred <- predict(model, test_data, type = "class")

# Confusion Matrix and Accuracy
conf_matrix <- confusionMatrix(pred, test_data$PlayTennis)
print(conf_matrix)
```


Explanation of Code

1. Dataset Preparation:

- Replace the sample dataset with `weather.numeric` as needed.

2. Model Training:

- `DecisionTreeClassifier` (Python) and `rpart` (R) create the decision tree.

3. Visualization:

- `plot_tree` in Python and `rpart.plot` in R visualize the tree.

4. Evaluation Metrics:

- Confusion matrix, accuracy, and classification reports summarize the model's performance.

Expected Output

1. A visualized decision tree.
2. Evaluation metrics showing model performance.
3. Predictions for unseen data.

Applications:

- Weather forecasting.
- Predictive modeling for categorical outcomes.
- Decision support systems in real-world problems.



Perform Classification using Naïve Bayes classifier on weather.numeric and weather.nominal dataset.

write about the topic and explain it. and impleemnt this in both python and r programming languages



Naïve Bayes Classifier: Explained

1. What is Naïve Bayes?

Naïve Bayes is a simple and powerful classification algorithm based on Bayes' Theorem. It works under the assumption that features are conditionally independent given the class, which simplifies computation.

The core principle behind Naïve Bayes is:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Where:

- $P(C|X)$ is the posterior probability of class C given the features X .
- $P(X|C)$ is the likelihood of features X given class C .
- $P(C)$ is the prior probability of class C .
- $P(X)$ is the probability of features X .

2. Weather Numeric Dataset

A numeric dataset contains continuous data like temperature, humidity, wind speed, etc. For classification, the target variable could be something like whether it's a good day to play tennis based on these factors.

3. Weather Nominal Dataset

A nominal dataset has categorical variables, such as whether it's sunny, rainy, or cloudy, and the target variable might still be a binary or multi-class classification like "play tennis" (yes/no).

4. Model Evaluation Metrics

- **Accuracy:** Proportion of correct predictions.
- **Confusion Matrix:** Shows true vs predicted class labels.
- **Precision, Recall, F1-score:** Important when dealing with imbalanced classes.

5. Why Use Naïve Bayes?

- Simple and fast for both training and prediction.
- Works well with both categorical and continuous data.
- Handles missing values and irrelevant features well.

Python Implementation

Naïve Bayes Classifier on Weather Numeric Dataset

```
python

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd

# Sample numeric dataset for weather prediction (numeric features)
data_numeric = {
    'Temperature': [85, 80, 83, 70, 68, 65, 75, 72, 69, 81],
    'Humidity': [85, 90, 86, 96, 80, 70, 80, 95, 70, 75],
    'Windy': [0, 1, 0, 0, 0, 1, 1, 0, 1, 1],
    'PlayTennis': [0, 0, 1, 1, 1, 1, 1, 0, 1, 0]
}
df_numeric = pd.DataFrame(data_numeric)

# Split dataset into features and target
X_numeric = df_numeric.drop('PlayTennis', axis=1)
y_numeric = df_numeric['PlayTennis']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_numeric, y_numeric, test_size=0.3, random_state=42)
```

```
# Train Naïve Bayes Classifier (GaussianNB for numeric data)
nb_model_numeric = GaussianNB()
nb_model_numeric.fit(X_train, y_train)

# Predictions
y_pred_numeric = nb_model_numeric.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred_numeric))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_numeric))
print("Classification Report:")
print(classification_report(y_test, y_pred_numeric))
```

Naïve Bayes Classifier on Weather Nominal Dataset

python

```
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd

# Sample nominal dataset for weather prediction (categorical features)
data_nominal = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'High'],
    'Windy': ['False', 'True', 'False', 'False', 'False', 'True', 'True', 'False', 'True', 'True'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No']
}
df_nominal = pd.DataFrame(data_nominal)

# Convert categorical features to numeric codes
df_nominal_encoded = df_nominal.apply(lambda x: pd.factorize(x)[0])

# Split dataset into features and target
X_nominal = df_nominal_encoded.drop('PlayTennis', axis=1)
y_nominal = df_nominal_encoded['PlayTennis']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_nominal, y_nominal, test_size=0.3, random_state=42)

# Train Naïve Bayes Classifier (CategoricalNB for nominal data)
nb_model_nominal = CategoricalNB()
nb_model_nominal.fit(X_train, y_train)

# Predictions
y_pred_nominal = nb_model_nominal.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred_nominal))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nominal))
print("Classification Report:")
print(classification_report(y_test, y_pred_nominal))
```

R Implementation

Naïve Bayes Classifier on Weather Numeric Dataset

R

```
# Load necessary library
if (!require("e1071")) install.packages("e1071")
library(e1071)

# Sample numeric dataset
data_numeric <- data.frame(
  Temperature = c(85, 80, 83, 70, 68, 65, 75, 72, 69, 81),
  Humidity = c(85, 90, 86, 96, 80, 70, 80, 95, 70, 75),
  Windy = c(0, 1, 0, 0, 0, 1, 1, 0, 1, 1),
  PlayTennis = factor(c(0, 0, 1, 1, 1, 1, 1, 0, 1, 0))
)

# Train-test split (70-30)
set.seed(42)
train_index <- sample(1:nrow(data_numeric), 0.7 * nrow(data_numeric))
train_data <- data_numeric[train_index, ]
test_data <- data_numeric[-train_index, ]

# Train Naïve Bayes model
nb_model_numeric <- naiveBayes(PlayTennis ~ ., data = train_data)

# Predictions
pred_numeric <- predict(nb_model_numeric, test_data)

# Evaluate the model
confusionMatrix(pred_numeric, test_data$PlayTennis)
```

Naïve Bayes Classifier on Weather Nominal Dataset

R

```
# Load necessary library
if (!require("e1071")) install.packages("e1071")
library(e1071)

# Sample nominal dataset
data_nominal <- data.frame(
  Outlook = c('Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny',
    'Sunny', 'Rainy'),
  Temperature = c('Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild'),
  Humidity = c('High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'High'),
  Windy = c('False', 'True', 'False', 'False', 'False', 'True', 'True', 'False', 'True', 'True'),
  PlayTennis = factor(c('No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'))
)

# Convert categorical data to factor (numeric encoding)
data_nominal$Outlook <- factor(data_nominal$Outlook)
data_nominal$Temperature <- factor(data_nominal$Temperature)
data_nominal$Humidity <- factor(data_nominal$Humidity)
data_nominal$Windy <- factor(data_nominal$Windy)

# Train-test split (70-30)
set.seed(42)
train_index <- sample(1:nrow(data_nominal), 0.7 * nrow(data_nominal))
train_data <- data_nominal[train_index, ]
test_data <- data_nominal[-train_index, ]

# Train Naïve Bayes model
nb_model_nominal <- naiveBayes(PlayTennis ~ ., data = train_data)

# Predictions
pred_nominal <- predict(nb_model_nominal, test_data)
```

```
# Evaluate the model
confusionMatrix(pred_nominal, test_data$PlayTennis)
```

Explanation of Code

1. Dataset Preparation:

- o Numeric and nominal datasets are created manually. In practice, you would load them from CSV or another source.

2. **Model



5)

- Perform Exploration Dataset Anayalysis(EDA) on any dataset.
- Perform Apriori Algorithm on any dataset.

write about the topic and explain it. and impleemnt this in both python and r programming languages



Exploratory Data Analysis (EDA) and Apriori Algorithm

1. Exploratory Data Analysis (EDA)

What is EDA? Exploratory Data Analysis (EDA) is the process of analyzing and visualizing datasets to summarize their main characteristics, often with the help of graphical representations. The goal is to understand the data, identify patterns, detect outliers, and check assumptions before further analysis or modeling.

Key EDA steps:

- 1. Data Collection:** Gather raw data.
- 2. Data Cleaning:** Handle missing data, outliers, or inconsistencies.
- 3. Data Transformation:** Normalize, scale, or encode variables as needed.
- 4. Data Visualization:** Use plots to understand relationships and distributions.
- 5. Statistical Analysis:** Summarize data using measures like mean, median, and standard deviation.

Benefits of EDA:

- Identifies important features.
- Helps in detecting anomalies.
- Helps in feature engineering for predictive models.

2. Apriori Algorithm

What is Apriori? The **Apriori algorithm** is a classic algorithm used in **association rule learning** to find frequent itemsets in transactional datasets. It's primarily used in market basket analysis, where we

analyze customer transactions to find relationships between different products.

How it works:

1. **Frequent Itemsets:** The algorithm starts by finding individual items that appear frequently and then tries to find item combinations that frequently appear together.
2. **Association Rules:** It generates rules (e.g., "If a customer buys X, they are likely to buy Y") based on the frequent itemsets.
3. **Support, Confidence, Lift:**
 - **Support:** The frequency of itemsets.
 - **Confidence:** The likelihood that an item Y is purchased when item X is purchased.
 - **Lift:** The strength of a rule over a random chance.

Python Implementation

1. Exploratory Data Analysis (EDA)

```
python

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (use any dataset, here using the famous Titanic dataset)
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
df = pd.read_csv(url)

# Basic Information
print(df.info()) # Data types and missing values
print(df.describe()) # Summary statistics

# Visualizing Missing Data
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Visualization')
plt.show()

# Distribution of Age
plt.figure(figsize=(10,6))
sns.histplot(df['Age'].dropna(), kde=True, bins=30)
plt.title('Age Distribution')
plt.show()

# Correlation Matrix
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Pairplot of selected features
sns.pairplot(df[['Age', 'Fare', 'SibSp', 'Parch']], hue='Survived')
plt.show()
```

2. Apriori Algorithm

python

```
# Import libraries
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd

# Sample transactional dataset (encoded)
dataset = {
    'Bread': [1, 1, 0, 1, 0],
    'Milk': [1, 1, 1, 0, 0],
    'Beer': [0, 1, 1, 1, 0],
    'Diapers': [1, 0, 1, 1, 1],
    'Cola': [0, 0, 0, 1, 1]
}
df = pd.DataFrame(dataset)

# Apply Apriori Algorithm
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
print(frequent_itemsets)

# Generate Association Rules
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.2)
print(rules)

# Filter rules based on confidence
rules = rules[rules['confidence'] > 0.7]
print(rules)
```

R Implementation

1. Exploratory Data Analysis (EDA)

R

```
# Load necessary libraries
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("dplyr")) install.packages("dplyr")
if (!require("corrplot")) install.packages("corrplot")
if (!require("titanic")) install.packages("titanic")

library(ggplot2)
library(dplyr)
library(corrplot)
library(titanic)

# Load Titanic dataset
data <- titanic::titanic_train

# Basic Information
str(data)
summary(data)

# Visualizing Missing Data
library(Amelia)
missmap(data, main = "Missing Data Map", col = c("yellow", "black"))

# Distribution of Age
ggplot(data, aes(x = Age)) +
  geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
  labs(title = "Age Distribution")

# Correlation Matrix
cor_matrix <- cor(data %>% select_if(is.numeric), use = "complete.obs")
corrplot(cor_matrix, method = "circle", type = "upper", tl.col = "black")

# Pairplot of selected features
```

```
ggplot(data, aes(x = Age, y = Fare, color = factor(Survived))) +
  geom_point() +
  labs(title = "Age vs Fare by Survival Status")
```

2. Apriori Algorithm

R

```
# Load necessary libraries
if (!require("arules")) install.packages("arules")
library(arules)

# Sample transactional dataset (encoded)
dataset <- data.frame(
  Bread = c(1, 1, 0, 1, 0),
  Milk = c(1, 1, 1, 0, 0),
  Beer = c(0, 1, 1, 1, 0),
  Diapers = c(1, 0, 1, 1, 1),
  Cola = c(0, 0, 0, 1, 1)
)

# Convert dataset to transactions
trans <- as(dataset, "transactions")

# Apply Apriori Algorithm
frequent_itemsets <- apriori(trans, parameter = list(support = 0.6, target = "frequent itemsets"))
inspect(frequent_itemsets)

# Generate Association Rules
rules <- apriori(trans, parameter = list(support = 0.6, confidence = 0.7, target = "rules"))
inspect(rules)

# Filter rules based on lift
rules_filtered <- subset(rules, lift > 1.2)
inspect(rules_filtered)
```

Explanation of Code

1. Exploratory Data Analysis (EDA)

- **Data Summary:** Using ``describe()`` (Python) or ``summary()`` (R) provides statistical summaries of numeric columns.
- **Missing Data Visualization:** Heatmaps are used to visualize the presence of missing data in the dataset.
- **Distribution and Correlation:** Histograms and pairplots show distributions and relationships between variables, while correlation matrices show how variables are correlated with one another.

2. Apriori Algorithm

- **Data Encoding:** In both Python and R, data needs to be in a transactional format for Apriori. This means each transaction is represented as a binary vector (1 for presence and 0 for absence of an item).
- **Frequent Itemsets:** The ``apriori()`` function generates frequent itemsets based on a minimum support threshold.

- **Association Rules:** The ``association_rules()`` (Python) or ``apriori()`` (R) function generates association rules based on the frequent itemsets, filtering based on metrics like confidence and lift.

Key Points

- **EDA** is crucial in understanding the structure and underlying patterns of the data before moving on to complex analyses.
- **Apriori** is useful in uncovering hidden relationships between items in transaction-based datasets.



6)
Perform TSNE Vs PCA with KMeans

write about the topic and explain it. and impleemnt this in both python and r programming languages



t-SNE vs PCA with K-Means

1. Principal Component Analysis (PCA)

What is PCA? Principal Component Analysis (PCA) is a dimensionality reduction technique used to reduce the number of features in a dataset, while maintaining as much of the original variance as possible. PCA transforms the data into a new coordinate system where the axes (principal components) represent the directions of maximum variance in the data.

Key Points:

- PCA is a **linear** technique, which means it works best when the data has linear relationships.
- It reduces the dimensionality of the dataset by projecting it onto the top principal components, with the first component having the most variance.
- PCA does not preserve distances or neighborhood relationships; it focuses purely on variance.

Advantages:

- Reduces computational cost and complexity.
- Helps in visualizing high-dimensional data.
- Good for datasets with many variables or features.

2. t-SNE (t-Distributed Stochastic Neighbor Embedding)

What is t-SNE? t-SNE is another dimensionality reduction technique, but unlike PCA, t-SNE is **non-linear** and focuses on preserving the local structure of the data. It's particularly useful when you want to preserve the neighborhood relationships in high-dimensional data.

Key Points:

- t-SNE minimizes the divergence between probability distributions in higher and lower dimensions, trying to preserve local clusters.

- It's often used for **visualization** of high-dimensional data in 2 or 3 dimensions.
- t-SNE focuses on maintaining the local similarities between points and works by assigning probabilities to pairs of data points, then minimizing the Kullback-Leibler divergence between the original and reduced-dimensional distributions.

Advantages:

- Better at capturing non-linear structures.
- Good for visualizing complex datasets where clusters or groups exist.

3. K-Means Clustering

K-Means is an unsupervised machine learning algorithm that divides the data into k clusters. The algorithm works by assigning each data point to the nearest cluster center, and then updating the centers based on the mean of points in each cluster.

Comparing PCA and t-SNE with K-Means

- **PCA** is computationally efficient and provides a linear reduction of dimensions. However, it doesn't preserve the local structure as effectively as t-SNE.
- **t-SNE** is computationally more expensive but excels at preserving local relationships, especially when data contains clusters or non-linear structures.
- **K-Means** can be applied after both PCA and t-SNE for clustering the reduced data, and the performance can be compared.

Python Implementation: t-SNE vs PCA with K-Means

```
python

import numpy as np
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the digits dataset
digits = load_digits()
X = digits.data
y = digits.target

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

# 1. Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 2. Apply t-SNE
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X_scaled)

# 3. K-Means Clustering on PCA and t-SNE reduced data
kmeans_pca = KMeans(n_clusters=10, random_state=42)
kmeans_tsne = KMeans(n_clusters=10, random_state=42)

# Fit K-Means on PCA and t-SNE results
```

```

y_pca = kmeans_pca.fit_predict(X_pca)
y_tsne = kmeans_tsne.fit_predict(X_tsne)

# Plot the results
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# PCA Results
axes[0].scatter(X_pca[:, 0], X_pca[:, 1], c=y_pca, cmap='viridis')
axes[0].set_title("K-Means Clustering on PCA")
axes[0].set_xlabel("PC1")
axes[0].set_ylabel("PC2")

# t-SNE Results
axes[1].scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_tsne, cmap='viridis')
axes[1].set_title("K-Means Clustering on t-SNE")
axes[1].set_xlabel("t-SNE1")
axes[1].set_ylabel("t-SNE2")

plt.tight_layout()
plt.show()

```

R Implementation: t-SNE vs PCA with K-Means

R

```

# Load necessary libraries
if (!require("datasets")) install.packages("datasets")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("cluster")) install.packages("cluster")
if (!require("Rtsne")) install.packages("Rtsne")
if (!require("factoextra")) install.packages("factoextra")

library(datasets)
library(ggplot2)
library(cluster)
library(Rtsne)
library(factoextra)

# Load the digits dataset (use iris dataset as an example for simplicity)
data(iris)
X <- iris[, -5]

# Standardize the data
X_scaled <- scale(X)

# 1. Apply PCA
pca_result <- prcomp(X_scaled, center = TRUE, scale. = TRUE)
X_pca <- pca_result$x[, 1:2] # First two principal components

# 2. Apply t-SNE
tsne_result <- Rtsne(X_scaled, dims = 2)
X_tsne <- tsne_result$Y

# 3. K-Means Clustering on PCA and t-SNE reduced data
kmeans_pca <- kmeans(X_pca, centers = 3, nstart = 25)
kmeans_tsne <- kmeans(X_tsne, centers = 3, nstart = 25)

# Plot the results
par(mfrow = c(1, 2))

# PCA Results
plot(X_pca, col = kmeans_pca$cluster, main = "K-Means Clustering on PCA",
     xlab = "PC1", ylab = "PC2", pch = 19, cex = 1.5)

# t-SNE Results
plot(X_tsne, col = kmeans_tsne$cluster, main = "K-Means Clustering on t-SNE",
     xlab = "t-SNE1", ylab = "t-SNE2", pch = 19, cex = 1.5)

```

Explanation of Code

1. Data Preparation:

- In Python, we use the `load_digits()` dataset, which is a common dataset for classification tasks. In R, we use the `iris` dataset, which is commonly used for clustering tasks.
- The data is scaled using `StandardScaler()` (Python) or `scale()` (R) to ensure that all features contribute equally to the analysis.

2. Dimensionality Reduction:

- **PCA:** We apply PCA to reduce the data to 2 components (`n_components=2`). In Python, we use `PCA()` from `sklearn.decomposition`, and in R, we use `prcomp()`.
- **t-SNE:** We apply t-SNE, which is particularly useful for preserving local data structure. In Python, we use `TSNE()` from `sklearn.manifold`, and in R, we use `Rtsne()`.

3. K-Means Clustering:

- After reducing the data's dimensions, we apply the **K-Means clustering algorithm** with `n_clusters=10` (in Python) and `centers=3` (in R, using 3 clusters as an example).
- In Python, we use `KMeans()` from `sklearn.cluster`, and in R, we use `kmeans()`.

4. Visualization:

- In Python, the results are visualized using `matplotlib` and `seaborn` for 2D scatter plots.
- In R, we use the `plot()` function to visualize the results of PCA and t-SNE.

Key Differences Between t-SNE and PCA

- **PCA** is a linear technique and works well when the data is linearly separable, while **t-SNE** is a non-linear technique and works better for preserving complex relationships in the data.
- **PCA** retains as much variance as possible by projecting the data into the axes of maximum variance, while **t-SNE** focuses on preserving local relationships (nearness) between data points.
- **t-SNE** is computationally more expensive than PCA and may be slower for large datasets.

Conclusion

- Use **PCA** when you need a fast, linear method for reducing dimensions, especially for high-dimensional datasets.
- Use **t-SNE** for visualizing complex data and understanding the cluster structures in lower dimensions.
- In both cases, applying **K-Means clustering** on the reduced data helps identify patterns and groupings effectively, but t-SNE often leads to more visually interpretable results in terms of clusters.