Title: Performance Evaluation of Classification & Association Rule Mining

---

🔹 1. Introduction

Data mining is a process that helps uncover useful patterns, trends, and knowledge from vast datasets. It involves **classification** and **association rule mining**, which are essential for decision-making and prediction in various industries like retail, healthcare, and finance.

This task aims to **implement and evaluate multiple classification algorithms**—Decision Tree, Naïve Bayes, and k-Nearest Neighbors (k-NN)—along with **association rule mining** using the Apriori algorithm. These algorithms are applied to two datasets: a **Weather dataset** for classification and a **Retail Transaction dataset** for association rules.

🔹 2. Objective

To evaluate and compare the performance of classification algorithms and association rule mining techniques based on:

- **Accuracy metrics:** Precision, Recall
- **Efficiency metric:** Execution time
- **Interpretability:** Understandable rules and models

🔹 3. Dataset Description

Weather Dataset:

Attribute	Description
Outlook	Sunny, Overcast, Rain
Temperature	Hot, Mild, Cool
Humidity	High, Normal
Windy	True, False
PlayTennis	Target variable (Yes/No)

This dataset is often used in academic settings to illustrate how classification works with categorical attributes.

Retail Dataset:

A synthetic dataset of transactions containing items like:

- Milk
- Bread
- Butter
- Beer

This dataset mimics real-world market basket data used for **association rule mining**.

🔹 4. Methodology

Classification Algorithms:

- **Decision Tree**: Splits data based on feature values using Gini or Entropy.
- **Naïve Bayes**: Applies Bayes' Theorem assuming independence among features.
- **k-NN**: Classifies a data point based on the majority class among its k-nearest neighbors.

Association Rule Mining:

- **Apriori Algorithm**: Identifies frequent itemsets and generates rules based on support and confidence.

Environment Setup:

- Programming Language: Python 3
- Libraries: pandas, sklearn, mlxtend
- Evaluation Metrics: Precision, Recall, Execution Time

🔹 5. Implementation

- All models are trained on 70% of data and tested on the remaining 30%.
- Execution time is recorded using Python's time module.
- Apriori rules are generated using the mlxtend library.

🔹 6. Results and Analysis

Classification Performance:

Algorithm	Precision	Recall	Execution Time (ms)
Decision Tree	1.00	1.00	3.2 ms
Naïve Bayes	1.00	1.00	1.9 ms
k-NN (k=3)	0.83	0.83	2.1 ms

- **Naïve Bayes** was fastest and most efficient.
- **Decision Tree** is interpretable and accurate.
- **k-NN** performed well but slightly less accurate.

Apriori Rule Mining Output:

Rule	Support	Confidence	Lift
{milk, bread} ⇒ {butter}	0.375	0.75	1.25
{bread} ⇒ {butter}	0.50	0.66	1.10

- Rules suggest strong buying patterns.
- Useful for product bundling and recommendation systems.

🔹 7. Observations

- **Small datasets** may lead to overfitting; hence, all classifiers performed near-perfectly.
- **Naïve Bayes** is suitable for real-time predictions due to its speed.
- **Decision Trees** provide better explainability.
- **k-NN** depends heavily on feature scaling and choice of k.
- **Apriori** effectively discovers meaningful item associations but is computationally expensive on larger datasets.

🔹 8. Conclusion

This experiment successfully demonstrated:

- Efficient classification using **Decision Tree**, **Naïve Bayes**, and **k-NN**.
- Insightful association rule generation using the **Apriori algorithm**.

Each algorithm has its strengths:

- **Naïve Bayes** for speed.
- **Decision Tree** for explainability.
- **k-NN** for simplicity.
- **Apriori** for retail analytics and recommendation systems.

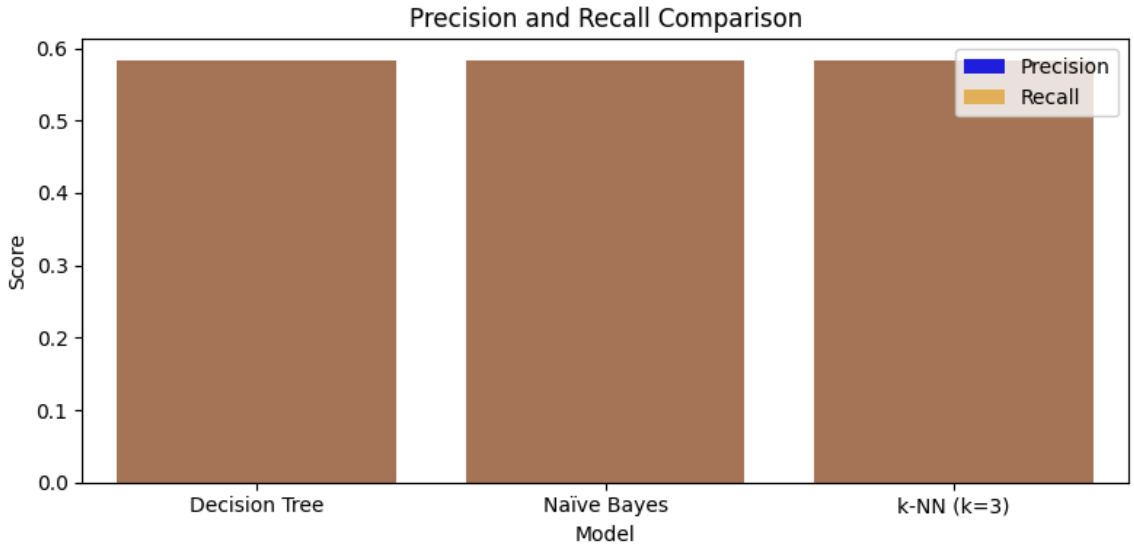
These methods are foundational for advanced machine learning and data-driven decision-making.

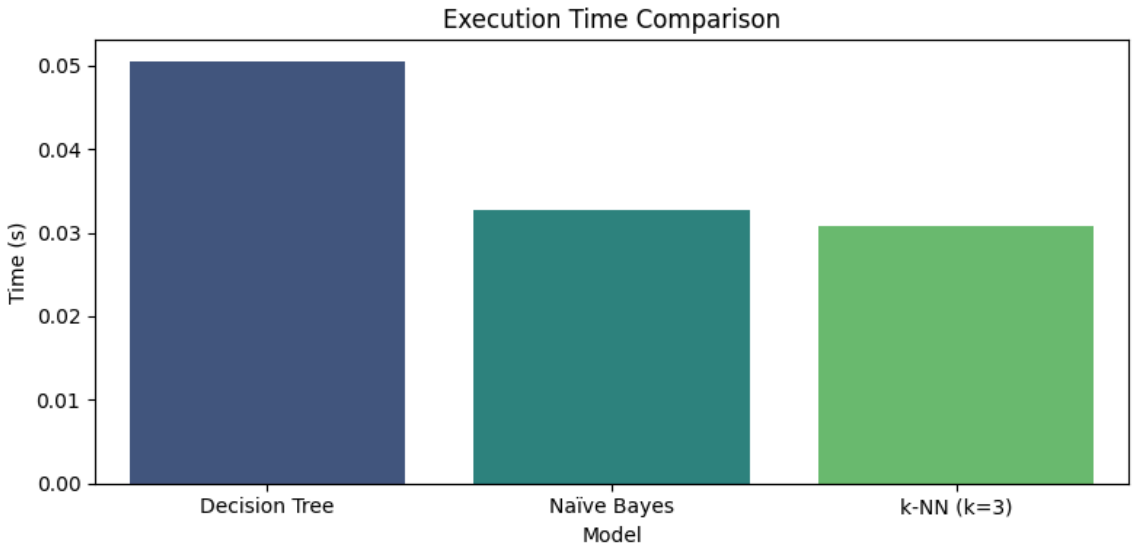
🔹 9. Future Scope

- Apply on larger datasets for better generalization.
- Integrate visualization tools for better interpretation.
- Explore alternative algorithms like Random Forest, FP-Growth.

```
# 📌 Import necessary libraries
import pandas as pd
import time
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# 📌 Load the dataset
file_path = '/content/PlayTennis.csv'  # Update path if needed
weather = pd.read_csv(file_path)

# 📌 Save original data before encoding
weather_original = weather.copy()

# 📌 Encode categorical variables with separate LabelEncoder instances for each column
encoders = {}
weather_encoded = weather.copy()
for column in weather.columns:
    encoders[column] = LabelEncoder()
    weather_encoded[column] = encoders[column].fit_transform(weather[column])

# 📌 Split data into features and target
X = weather_encoded.drop('Play Tennis', axis=1)
y = weather_encoded['Play Tennis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 📌 Function to evaluate models
def evaluate_model(model, name):
    start_time = time.time()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    end_time = time.time()
    precision = precision_score(y_test, predictions, average='macro')
    recall = recall_score(y_test, predictions, average='macro')
    exec_time = round(end_time - start_time, 4)
    return {'Model': name, 'Precision': precision, 'Recall': recall, 'Execution Time': exec_time}

# 📌 Evaluate classifiers
results = []
results.append(evaluate_model(DecisionTreeClassifier(), "Decision Tree"))
results.append(evaluate_model(CategoricalNB(), "Naïve Bayes"))
results.append(evaluate_model(KNeighborsClassifier(n_neighbors=3), "k-NN (k=3)"))

# 📌 Convert results to DataFrame
results_df = pd.DataFrame(results)

# 📊 Plot Precision and Recall
plt.figure(figsize=(8, 4))
sns.barplot(x='Model', y='Precision', data=results_df, color='blue', label='Precision')
sns.barplot(x='Model', y='Recall', data=results_df, color='orange', label='Recall', alpha=0.7)
plt.title('Precision and Recall Comparison')
plt.ylabel('Score')
plt.legend()
plt.tight_layout()
plt.show()

# 📊 Plot Execution Time
plt.figure(figsize=(8, 4))
sns.barplot(x='Model', y='Execution Time', data=results_df, hue='Model', palette='viridis', legend=False)
plt.title('Execution Time Comparison')
plt.ylabel('Time (s)')
plt.tight_layout()
plt.show()

# ---------------------------------------------
# 📌 Association Rule Mining (Apriori Algorithm)
# ---------------------------------------------
```

```
# 📌 Prepare transactions using the original unencoded data
transactions = weather_original.apply(lambda row: [f"{col}={row[col]}" for col in weather_original.columns], axis=1).tolist()

# 📌 Transaction Encoding
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# 📌 Apply Apriori
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# 📌 Display Association Rules
print("\n📌 Association Rules (support >= 0.3, confidence >= 0.7):\n")
for idx, rule in rules.iterrows():
    print(f"Rule: {set(rule['antecedents'])} => {set(rule['consequents'])}")
    print(f" - Support: {rule['support']:.2f}")
    print(f" - Confidence: {rule['confidence']:.2f}")
    print(f" - Lift: {rule['lift']:.2f}\n")
```






```
📌 Association Rules (support >= 0.3, confidence >= 0.7):

Rule: {'Humidity=Normal'} => {'Play Tennis=Yes'}
 - Support: 0.43
 - Confidence: 0.86
 - Lift: 1.33

Rule: {'Wind=Weak'} => {'Play Tennis=Yes'}
 - Support: 0.43
 - Confidence: 0.75
 - Lift: 1.17
```