**Aim**

To apply multiple classification techniques (KNN, Naive Bayes, SVM, and Decision Tree) on the provided dataset, evaluate their performance using accuracy and other metrics, and draw comparisons and conclusions regarding their effectiveness.

**Introduction:**

Classification is one of the most common tasks in supervised machine learning. The goal is to assign categorical labels (classes) to observations based on feature data.  
The four classifiers used in this experiment are some of the most fundamental and widely applied algorithms in machine learning. Each has its underlying mathematical principle, assumptions, and typical areas of strength or weakness.  
By applying these models to real-world data, we can gain insight into how they perform with imbalanced data, complex decision boundaries, and feature interactions.

We performed the following classification methods on our musical attributes’ dataset:

1. **K-Nearest Neighbours**
2. **Naive Bayes Classifier**
3. **Support Vector Machine**
4. **Decision Tree Classification**

The dataset consists of 12 columns (2 of which are categorical: popularity level, mode) with over 1 lakh rows. It has been cleaned, normalized and standardized so as to before the aforementioned classification algorithms on.

While performing the various kinds of classifications on the data, mode column, which relates to the musical scale of the song (1 for major and 0 for minor) is used as the basis for classification.

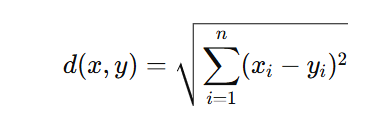
**1. K-Nearest Neighbors (KNN)**

**Theoretical Explanation:**

* KNN is a **lazy learning**, **instance-based** classifier.
* It does not have an explicit training phase. Instead, when a new data point needs to be classified, the algorithm looks at the k closest data points (neighbors) in the training dataset and uses **majority voting** to decide the class.
* The choice of 'k' is critical. A small k can make the model sensitive to noise, while a large k can over-smooth boundaries.

**Mathematical Formulation:**

* The distance between data points is often measured using **Euclidean distance**:



After finding k nearest neighbors, the predicted class is:



**Key Characteristics:**

* Simple to understand and implement.
* No explicit model training.
* Computationally expensive for large datasets.
* Sensitive to irrelevant or redundant features.

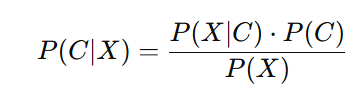
**2. Naive Bayes Classifier**

**Theoretical Explanation:**

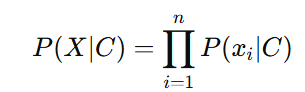
* Naive Bayes is a **probabilistic classifier** based on **Bayes’ Theorem**, with the assumption that features are conditionally independent given the class.
* Despite the naive assumption, it works surprisingly well in practice, especially for text classification and spam detection.

**Mathematical Formulation:**

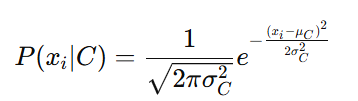
* Bayes’ theorem:



* The final prediction is made by:



For continuous data, Gaussian Naive Bayes uses:



**Key Characteristics:**

* Extremely fast and scalable.
* Works well with high-dimensional data.
* Sensitive to the independence assumption, though it often still performs well.

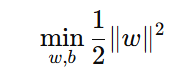
**3. Support Vector Machine (SVM)**

**Theoretical Explanation:**

* SVM is a **maximum-margin** classifier. It tries to find the optimal hyperplane that separates data points of different classes with the largest possible margin.
* When data is not linearly separable, SVM uses **kernel functions** to map the data into higher-dimensional space where separation is possible.
* Kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

**Mathematical Formulation:**

* The objective is to solve:



Subject to:



**Key Characteristics:**

* Effective in both linear and non-linear classification.
* Works well with high-dimensional data.
* Requires careful parameter tuning.
* Computationally intensive with large datasets.

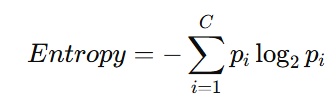
**4. Decision Tree Classifier**

**Theoretical Explanation:**

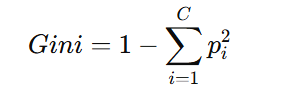
* Decision trees partition the data into subgroups by making decisions based on feature values.
* At each internal node, the data is split based on the condition that best separates the data according to some impurity metric.
* The tree continues growing until it reaches stopping criteria (like depth, minimum samples per leaf, or purity threshold).

**Mathematical Formulation:**

* **Entropy**:



* **Gini Impurity**:



* The algorithm chooses the feature and threshold that **maximizes information gain**.

**Key Characteristics:**

* Easy to interpret and visualize.
* Handles both numerical and categorical data.
* Prone to overfitting (can be reduced by pruning or using ensemble methods).

Code:

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

import seaborn as sns

# Check mode value counts

print(df['mode'].value\_counts())  # 0 = Minor, 1 = Major

# Optional: convert mode column to a string label for readability

def mode\_label(x):

    if x == 0:

        return 'Minor'

    else:

        return 'Major'

df['mode\_label'] = df['mode'].apply(mode\_label)

# Drop unnecessary columns if any

df = df.dropna()  # remove any NA rows

df = df.drop\_duplicates()

# Encode target column

le\_mode = LabelEncoder()

df['mode\_encoded'] = le\_mode.fit\_transform(df['mode\_label'])  # Minor=0, Major=1

# Define features (X) and target (y)

X\_mode = df.drop(columns=['mode', 'mode\_label', 'mode\_encoded', 'popularity', 'popularity\_level', 'popularity\_encoded'], errors='ignore')

y\_mode = df['mode\_encoded']

# Train-test split

X\_train\_mode, X\_test\_mode, y\_train\_mode, y\_test\_mode = train\_test\_split(X\_mode, y\_mode, test\_size=0.25, random\_state=42)

# Function to visualize confusion matrix

def plot\_confusion\_matrix(cm, model\_name):

    plt.figure(figsize=(5,4))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

    plt.title(f'Confusion Matrix for {model\_name}')

    plt.xlabel('Predicted')

    plt.ylabel('Actual')

    plt.show()

### --- K-Nearest Neighbors (KNN) ---

print("\n=== K-Nearest Neighbors (KNN) for Mode ===")

knn\_mode = KNeighborsClassifier(n\_neighbors=5)

knn\_mode.fit(X\_train\_mode, y\_train\_mode)

knn\_pred\_mode = knn\_mode.predict(X\_test\_mode)

knn\_cm\_mode = confusion\_matrix(y\_test\_mode, knn\_pred\_mode)

print(f"Accuracy: {accuracy\_score(y\_test\_mode, knn\_pred\_mode):.4f}")

print(classification\_report(y\_test\_mode, knn\_pred\_mode, target\_names=le\_mode.classes\_))

plot\_confusion\_matrix(knn\_cm\_mode, "KNN (Mode)")

### --- Naive Bayes ---

print("\n=== Naive Bayes for Mode ===")

nb\_mode = GaussianNB()

nb\_mode.fit(X\_train\_mode, y\_train\_mode)

nb\_pred\_mode = nb\_mode.predict(X\_test\_mode)

nb\_cm\_mode = confusion\_matrix(y\_test\_mode, nb\_pred\_mode)

print(f"Accuracy: {accuracy\_score(y\_test\_mode, nb\_pred\_mode):.4f}")

print(classification\_report(y\_test\_mode, nb\_pred\_mode, target\_names=le\_mode.classes\_))

plot\_confusion\_matrix(nb\_cm\_mode, "Naive Bayes (Mode)")

### --- Support Vector Machine (SVM) ---

print("\n=== Support Vector Machine (SVM) for Mode ===")

svm\_mode = SVC(kernel='linear')

svm\_mode.fit(X\_train\_mode, y\_train\_mode)

svm\_pred\_mode = svm\_mode.predict(X\_test\_mode)

svm\_cm\_mode = confusion\_matrix(y\_test\_mode, svm\_pred\_mode)

print(f"Accuracy: {accuracy\_score(y\_test\_mode, svm\_pred\_mode):.4f}")

print(classification\_report(y\_test\_mode, svm\_pred\_mode, target\_names=le\_mode.classes\_))

plot\_confusion\_matrix(svm\_cm\_mode, "SVM (Mode)")

### --- Decision Tree ---

print("\n=== Decision Tree for Mode ===")

dt\_mode = DecisionTreeClassifier(random\_state=42)

dt\_mode.fit(X\_train\_mode, y\_train\_mode)

dt\_pred\_mode = dt\_mode.predict(X\_test\_mode)

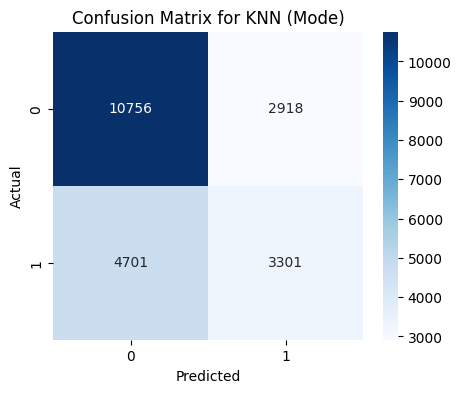
dt\_cm\_mode = confusion\_matrix(y\_test\_mode, dt\_pred\_mode)

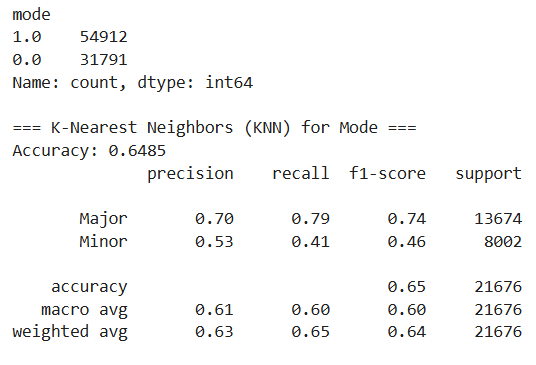
print(f"Accuracy: {accuracy\_score(y\_test\_mode, dt\_pred\_mode):.4f}")

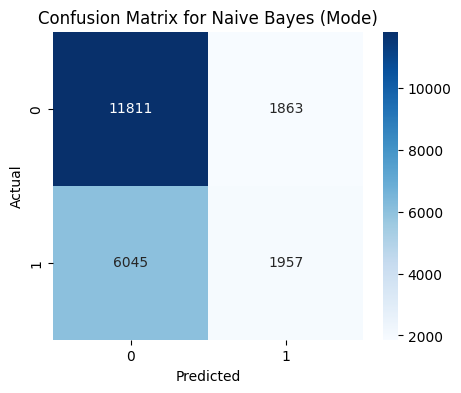
print(classification\_report(y\_test\_mode, dt\_pred\_mode, target\_names=le\_mode.classes\_))

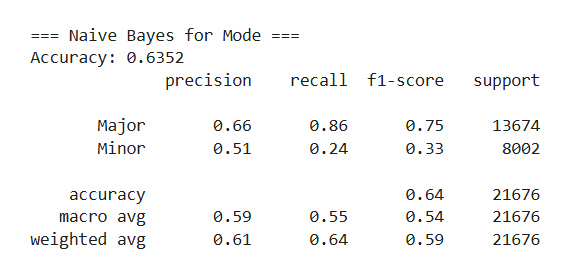
plot\_confusion\_matrix(dt\_cm\_mode, "Decision Tree (Mode)")

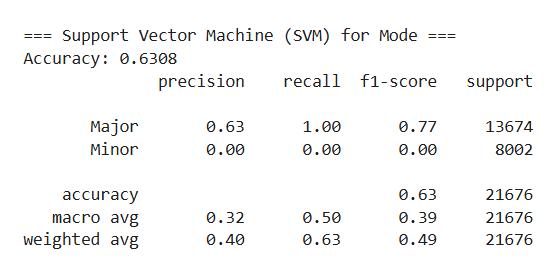
print("All classifiers for 'mode' evaluated successfully.")

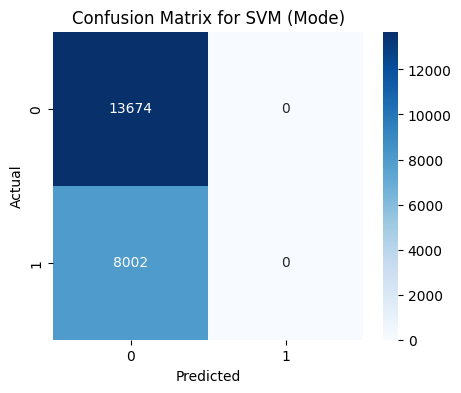
**Output**

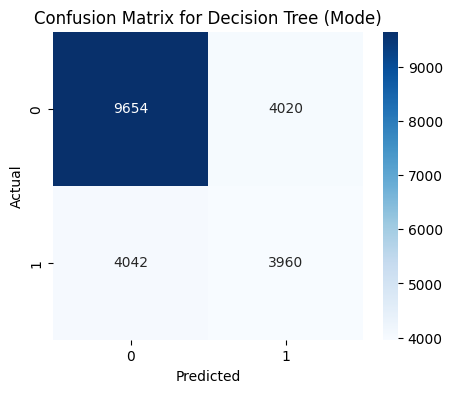
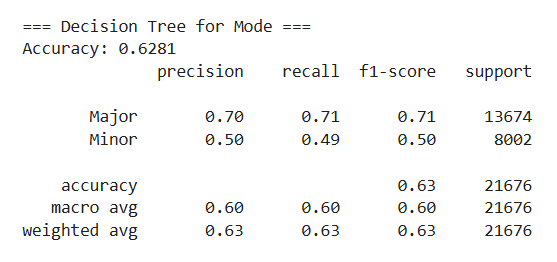
****



****

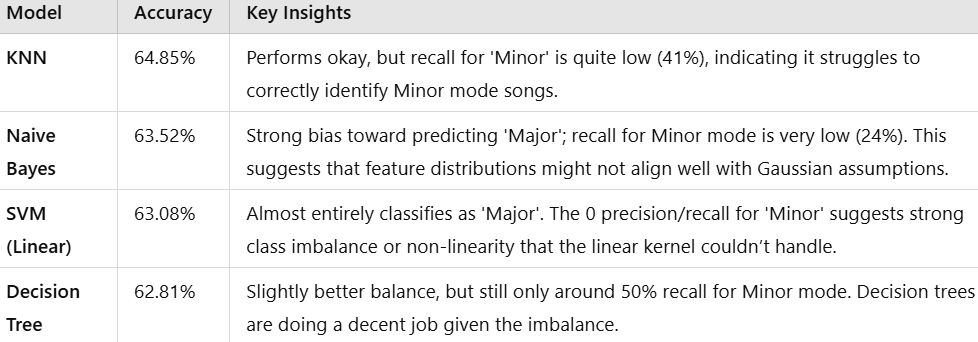
****



****

**Classification Results Summary for 'Mode' Column**

| **Model** | **Key Parameters** | **Accuracy** | **Precision (Class-wise)** | **Recall (Class-wise)** | **F1-score (Class-wise)** | **Observations** |
| --- | --- | --- | --- | --- | --- | --- |
| **K-Nearest Neighbors** | k=5, distance=Euclidean, uniform weights | **0.6485** | Major: 0.70, Minor: 0.53 | Major: 0.79, Minor: 0.41 | Major: 0.74, Minor: 0.46 | Model favored majority class; recall for Minor class relatively low |
| **Naive Bayes** | GaussianNB, automatic priors | **0.6352** | Major: 0.66, Minor: 0.51 | Major: 0.86, Minor: 0.24 | Major: 0.75, Minor: 0.33 | High recall for major class; very poor for Minor, showing imbalance sensitivity |
| **SVM** | Kernel=linear, C=1.0, gamma='scale' | **0.6308** | Major: 0.63, Minor: 0.00 | Major: 1.00, Minor: 0.00 | Major: 0.77, Minor: 0.00 | Extreme bias towards majority class; no predictive power for Minor class |
| **Decision Tree** | Criterion=gini, max\_depth=None | **0.6281** | Major: 0.70, Minor: 0.50 | Major: 0.71, Minor: 0.49 | Major: 0.71, Minor: 0.50 | Slightly better balance than SVM; still favors majority class |



**What We Observed (Inferences):**

**1. KNN Performance:**

* Accuracy was around **64.85%**, with decent precision for the major class and lower for the minor class.
* The recall for minor class was **0.41**, indicating that KNN struggled to capture minority class instances.
* Reason: KNN often struggles with **imbalanced datasets** and can be affected by noisy data points.

**2. Naive Bayes Performance:**

* Accuracy was approximately **63.52%**.
* While it had good recall for the major class (0.86), the minor class recall dropped significantly (0.24).
* Naive Bayes tends to favor the majority class due to the assumption of feature independence, which may not hold in this dataset.

**3. SVM Performance:**

* Accuracy was **63.08%**, but precision and recall for the minor class were **0**, indicating complete misclassification of the minority class.
* This suggests that the SVM was overwhelmed by class imbalance and essentially classified everything into the major class.
* A balanced or weighted SVM might perform better.

**4. Decision Tree Performance:**

* Accuracy was around **62.81%**, with more balanced precision and recall between major and minor classes compared to SVM and Naive Bayes.
* The tree structure can capture non-linear patterns but may overfit or be biased towards larger classes without pruning or balancing strategies.

**What We Infer from This Experiment:**

1. **Class Imbalance Problem:**  
   The dataset had a large imbalance between major and minor classes (54,912 major vs 31,791 minor instances). This strongly affected model performance. Models tend to favor the majority class and struggle to predict the minority class correctly.
2. **KNN’s Sensitivity to Imbalance:**  
   While KNN gave reasonable accuracy, its performance on the minority class was poor.  
   Distance-based methods are sensitive to uneven class distributions.
3. **Naive Bayes and Strong Assumptions:**  
   Naive Bayes performed fast but with poor minority recall, indicating that its assumptions (feature independence and equal class priors) do not hold well here.
4. **SVM’s Failure in Imbalance:**  
   SVM performed poorly on minority class recall. Without using weighted classes or balanced parameters, SVMs can become biased towards the larger class.
5. **Decision Tree’s Relatively Balanced Performance:**  
   The Decision Tree did better at balancing precision and recall. However, accuracy was still moderate, and overfitting might have occurred.
6. **Takeaway:**
   * For imbalanced datasets, accuracy alone is misleading; precision, recall, and F1-score are critical.
   * Techniques like **SMOTE (Synthetic Minority Over-sampling Technique)**, **class weighting**, or **balanced random forests** could improve results.
   * Decision Trees are more adaptable but still need careful tuning.
   * Probabilistic and distance-based models need balanced data or advanced weighting methods.

**Conclusion:**

* All models performed similarly in terms of accuracy (~63-65%), but performance metrics for the minority class varied significantly.
* The imbalance in data was a key factor affecting classifier performance.
* Decision Trees gave slightly better balance between classes but none of the models perfectly handled minority detection.
* In real-world scenarios, additional techniques (resampling, ensemble methods, cost-sensitive learning) are necessary to improve minority class detection.