**NNDL\_ICP5**

**AKSHITH REDDY SUNKA**

**700758358**

1. Implement Naïve Bayes method using scikit-learn library Use dataset available with name glass Use train\_test\_split to create training and testing part Evaluate the model on test part using score and classification\_report(y\_true, y\_pred)

# Step 1: Import necessary libraries and load the dataset

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report

# Load the Glass dataset (assuming you have it as a CSV file)

glass\_data = pd.read\_csv("glass.csv")

# Assuming the target variable is in a column called 'Type'

X = glass\_data.drop('Type', axis=1)

y = glass\_data['Type']

# Step 2: No preprocessing is needed for this example

# Step 3: Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train a Naïve Bayes classifier (GaussianNB) on the training data

nb\_classifier = GaussianNB()

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

# Step 5: Make predictions on the test data

y\_pred = nb\_classifier.predict(X\_test)

# Step 6: Evaluate the model

# Calculate the accuracy score

accuracy = nb\_classifier.score(X\_test, y\_test)

# Generate a classification report

classification\_rep = classification\_report(y\_test, y\_pred)

# Print the results

print("Accuracy:", accuracy)

print("\nClassification Report:\n", classification\_rep)

2. Implement linear SVM method using scikit library Use the same dataset above Use train\_test\_split to create training and testing part Evaluate the model on test part using score and classification\_report(y\_true, y\_pred) Which algorithm you got better accuracy? Can you justify why?

import pandas as pd

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

# Load the Glass dataset (assuming you have it as a CSV file)

glass\_data = pd.read\_csv("glass.csv")

# Assuming the target variable is in a column called 'Type'

X = glass\_data.drop('Type', axis=1)

y = glass\_data['Type']

# Step 2: No preprocessing is needed for this example

# Step 3: Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train a Linear SVM classifier on the training data

svm\_classifier = SVC(kernel='linear', C=1.0) # Linear SVM

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

# Step 5: Make predictions on the test data

y\_pred\_svm = svm\_classifier.predict(X\_test)

# Step 6: Evaluate the Linear SVM model

# Calculate the accuracy score for the Linear SVM model

accuracy\_svm = svm\_classifier.score(X\_test, y\_test)

# Generate a classification report for the Linear SVM model

classification\_rep\_svm = classification\_report(y\_test, y\_pred\_svm)

# Print the results for the Linear SVM model

print("Accuracy (Linear SVM):", accuracy\_svm)

print("\nClassification Report (Linear SVM):\n", classification\_rep\_svm)

To justify which model is better, consider factors like the dataset's distribution, feature characteristics, and any underlying assumptions. In some cases, SVM may perform better when the data has complex decision boundaries, while Naïve Bayes may work well for datasets with strong independence assumptions between features. It's essential to evaluate both models and potentially experiment with different algorithms to make an informed choice based on the dataset's performance metrics.