**St. Francis Institute of Technology, Mumbai-400 103**

**Department Of Information Technology**

**A.Y. 2024-2025**

**Class: TE-ITA/B, Semester: VI**

**Subject: Business Intelligence Lab**

**Experiment – 5: a) To implement a classifier- Naïve Bayes using any one Language (JAVA/Python)**

**b) To implement a KNN classifier using any one Language (JAVA/Python) (Topic Beyond Syllabus)**

**1.** **Aim:** a) To implement a classifier- Naïve Bayes using any one Language (JAVA/Python)

b) To implement a KNN classifier using any one Language (JAVA/Python) (Topic Beyond Syllabus)

**2.**     **Objectives:** After study of this experiment, the students will be able to

                           implement Naïve based algorithm and Random forest/SVM algorithm

**3.**     **Outcomes:** After study of this experiment, the students will be able to

**CO 3:** Design and Implement various classification data mining techniques and apply metrics to measure its performance

**4.** **Prerequisite:** Introduction to all the classifiers through algorithms & Problem solving approach.

**5.** **Requirements:** Personal Computer, Windows XP operating system/Windows 7, Internet

Connection, Microsoft Word, WEKA tool, Java/R/Python

**6.** **Theory:**

**a. Explain the Classification Algorithm (Naïve Bayes and KNN)**

ANS:

1. Naïve Bayes Classifier

Naïve Bayes is a probabilistic classification algorithm based on Bayes’ Theorem. It assumes that all features are independent (Naïve assumption) and contribute equally to the outcome.

##### Bayes' Theorem:

Where:

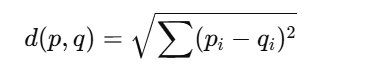
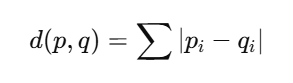
* P(A∣B)P(A|B)P(A∣B) is the posterior probability (probability of class AAA given evidence BBB)
* P(B∣A)P(B|A)P(B∣A) is the likelihood (probability of evidence BBB given class AAA)
* P(A)P(A)P(A) is the prior probability of class AAA
* P(B)P(B)P(B) is the probability of evidence

KNN is a non-parametric, instance-based learning algorithm that classifies a new data point based on the majority class of its K-nearest neighbors.

##### Algorithm Steps:

1. Choose the number of neighbors K.
2. Calculate the distance (Euclidean, Manhattan, or Minkowski) between the new data point and all existing data points.
3. Select the K closest neighbors.
4. Assign the class label based on majority voting.

##### Distance Metrics:

* **Euclidean Distance:** 
* **Manhattan Distance:** 

**b. Applications of Classification Algorithms**

ANS:

Spam Detection – Naïve Bayes is used to classify emails as spam or not.

Sentiment Analysis – Classifying reviews as positive, neutral, or negative.

Medical Diagnosis – Predicting diseases based on symptoms.

Fraud Detection – Identifying fraudulent transactions in banking.

Handwriting Recognition – KNN is commonly used for digit recognition

**c. Advantages and Disadvantages of Classification Algorithms**

ANS:

| **Naïve Bayes** | **ADVANTAGES** | **DISADVANTAGES** |
| --- | --- | --- |
| 1. Fast and efficient – Works well with large datasets.  2. Works with categorical & numerical data.  3. Requires less training data. | 1. Strong independence assumption – Rarely holds in real-world data.  2. Struggles with correlated features. |
| | **KNN** | | --- |  |  | | --- | | 1. Simple and easy to implement.  2. Works well with small datasets.  3. No need for the training phase. | 1. Computationally expensive for large datasets.  2. Sensitive to irrelevant features.  3. Requires careful choice of K. |

**7.**   **Laboratory Exercise:** Implementation of both (a&b) Classification Algorithm using JAVA/ R/ Python. Printout of implementation along with coding and Output.

**8.**   **Post-Experiments Exercise**

**a.** **Questions:**

**●** Compare and Contrast between Decision Tree & Naïve Bayes

* Compare and Contrast between Decision Tree and Random forest
* Solve a numerical on Naïve Bayes Algorithm

**b.**     **Conclusion:**

**●** Summary of Experiment

● Importance of Experiment

● Application of Experiment

**9.**     **Reference**: Data Mining: Concept & Techniques, 3rd Edition, Jiawei Han, Micheline Kamber, Jian Pei, Elsevier.

**Reference links:**

**·** [**https://scikit-learn.org/stable/modules/naive\_bayes.html**](https://scikit-learn.org/stable/modules/naive_bayes.html)

**·** [**https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn**](https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn)

**·** [**https://www.analyticsvidhya.com/blog/2021/11/implementation-of-gaussian-naive-bayes-in-python-sklearn/**](https://www.analyticsvidhya.com/blog/2021/11/implementation-of-gaussian-naive-bayes-in-python-sklearn/)

**·** [**https://github.com/2796gaurav/Naive-bayes-explained/blob/master/Naive%20bayes/Naive%20Bayes%20in%20scikit%20learn.ipynb**](https://github.com/2796gaurav/Naive-bayes-explained/blob/master/Naive%20bayes/Naive%20Bayes%20in%20scikit%20learn.ipynb)

a) To implement a classifier- Naïve Bayes using any one Language (Python)

import pandas as pd

data = {

"Outlook": ["Sunny", "Sunny", "Overcast", "Rainy", "Rainy", "Rainy", "Overcast", "Sunny", "Sunny", "Rainy",

"Sunny", "Overcast", "Overcast", "Rainy"],

"Temperature": ["Hot", "Hot", "Hot", "Mild", "Cool", "Cool", "Cool", "Mild", "Cool", "Mild", "Mild", "Mild", "Hot", "Mild"],

"Humidity": ["High", "High", "High", "High", "Normal", "Normal", "Normal", "High", "Normal", "Normal",

"Normal", "High", "Normal", "High"],

"Wind": ["Weak", "Strong", "Weak", "Weak", "Weak", "Strong", "Strong", "Weak", "Weak", "Weak", "Strong",

"Strong", "Weak", "Strong"],

"Play": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "Yes", "No"]

}

df = pd.DataFrame(data)

# Compute prior probabilities

total\_samples = len(df)

yes\_count = df[df["Play"] == "Yes"].shape[0]

no\_count = df[df["Play"] == "No"].shape[0]

P\_yes = yes\_count / total\_samples

P\_no = no\_count / total\_samples

# Given test sample

X\_new = {"Outlook": "Overcast", "Temperature": "Cool", "Humidity": "High", "Wind": "Strong"}

# Function to compute conditional probabilities

def conditional\_probability(feature, value, play\_value):

subset = df[df["Play"] == play\_value]

count = subset[subset[feature] == value].shape[0]

total = subset.shape[0]

return count / total if total != 0 else 1e-6 # Avoid zero probability

# Compute P(X | Play=Yes) and P(X | Play=No)

P\_X\_given\_yes = 1

P\_X\_given\_no = 1

print("Durva Kadam, 45")

# Step 1: Calculate prior probabilities

print(f"P(Play=Yes) = {yes\_count}/{total\_samples} = {round(P\_yes, 3)}")

print(f"P(Play=No) = {no\_count}/{total\_samples} = {round(P\_no, 3)}\n")

# Step 2: Calculate conditional probabilities for each feature

for feature, value in X\_new.items():

P\_feature\_given\_yes = conditional\_probability(feature, value, "Yes")

P\_feature\_given\_no = conditional\_probability(feature, value, "No")

P\_X\_given\_yes \*= P\_feature\_given\_yes

P\_X\_given\_no \*= P\_feature\_given\_no

print(f"P({feature}={value} | Play=Yes) = {round(P\_feature\_given\_yes, 3)}")

print(f"P({feature}={value} | Play=No) = {round(P\_feature\_given\_no, 3)}\n")

# Step 3: Calculate P(X)

P\_X = (P\_X\_given\_yes \* P\_yes) + (P\_X\_given\_no \* P\_no)

print(f"P(X | Play=Yes) = {round(P\_X\_given\_yes, 4)}")

print(f"P(X | Play=No) = {round(P\_X\_given\_no, 4)}\n")

print(f"P(X) = ({round(P\_X\_given\_yes, 4)} \* {round(P\_yes, 3)}) + ({round(P\_X\_given\_no, 4)} \* {round(P\_no, 3)})")

print(f" = {round(P\_X, 4)}\n")

# Step 4: Compute final probabilities

P\_X\_play\_yes = (P\_X\_given\_yes \* P\_yes) / P\_X

P\_X\_play\_no = (P\_X\_given\_no \* P\_no) / P\_X

print(f"P(Play=Yes | X) = ({round(P\_X\_given\_yes, 4)} \* {round(P\_yes, 3)}) / {round(P\_X, 4)}")

print(f" = {round(P\_X\_play\_yes, 4)}\n")

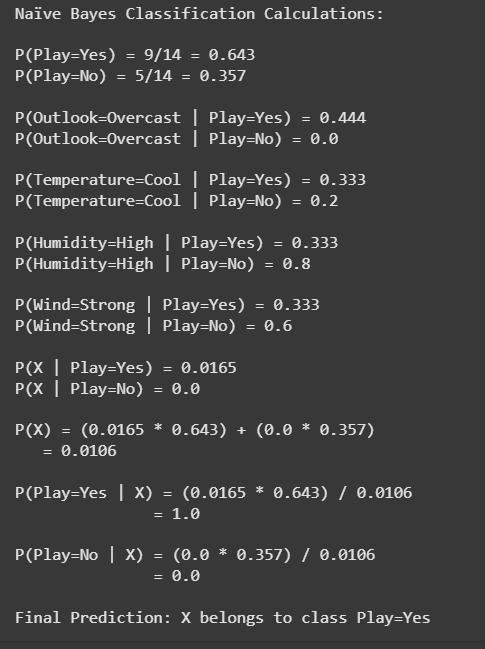
print(f"P(Play=No | X) = ({round(P\_X\_given\_no, 4)} \* {round(P\_no, 3)}) / {round(P\_X, 4)}")

print(f" = {round(P\_X\_play\_no, 4)}\n")

predicted\_class = "Yes" if P\_X\_play\_yes >= P\_X\_play\_no else "No"

print(f"Final Prediction: X belongs to class Play={predicted\_class}")

OUTPUT:



b) To implement a KNN classifier using any one Language (Python)

from collections import Counter

import pandas as pd

import math

# Dataset

data = {

"Outlook": ["Sunny", "Sunny", "Overcast", "Rainy", "Rainy", "Rainy", "Overcast", "Sunny", "Sunny", "Rainy",

"Sunny", "Overcast", "Overcast", "Rainy"],

"Temperature": ["Hot", "Hot", "Hot", "Mild", "Cool", "Cool", "Cool", "Mild", "Cool", "Mild", "Mild", "Mild",

"Hot", "Mild"],

"Humidity": ["High", "High", "High", "High", "Normal", "Normal", "Normal", "High", "Normal", "Normal",

"Normal", "High", "Normal", "High"],

"Wind": ["Weak", "Strong", "Weak", "Weak", "Weak", "Strong", "Strong", "Weak", "Weak", "Weak", "Strong",

"Strong", "Weak", "Strong"],

"Play": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "Yes", "No"]

}

df = pd.DataFrame(data)

# Convert categorical data to numerical values

df['Outlook'] = df['Outlook'].map({'Sunny': 0, 'Overcast': 1, 'Rainy': 2})

df['Temperature'] = df['Temperature'].map({'Hot': 0, 'Mild': 1, 'Cool': 2})

df['Humidity'] = df['Humidity'].map({'High': 0, 'Normal': 1})

df['Wind'] = df['Wind'].map({'Weak': 0, 'Strong': 1})

df['Play'] = df['Play'].map({'No': 0, 'Yes': 1})

# Function to calculate Euclidean distance

def euclidean\_distance(sample1, sample2):

distance = 0

for feature in sample1.keys():

if feature != 'Play': # Exclude target variable 'Play' from distance calculation

distance += (sample1[feature] - sample2[feature]) \*\* 2

return math.sqrt(distance)

# KNN classifier function

def knn\_classifier(df, query, k=3):

distances = []

for \_, row in df.iterrows():

distance = euclidean\_distance(query, row)

distances.append((distance, row['Play']))

# Sort distances in ascending order

distances.sort(key=lambda x: x[0])

print("\nKey Distance Calculations:")

for i, (distance, label) in enumerate(distances[:k]):

print(f"Neighbor {i + 1}: Distance = {distance:.3f}, Play = {'Yes' if label == 1 else 'No'}")

# Get k nearest labels

nearest\_neighbors = distances[:k]

labels = [neighbor[1] for neighbor in nearest\_neighbors]

# Majority vote

majority\_vote = Counter(labels).most\_common(1)[0][0]

return majority\_vote

# Query to classify

query = {

'Outlook': 1, # Overcast

'Temperature': 2, # Cool

'Humidity': 0, # High

'Wind': 1 # Strong

}

# Run KNN classifier

predicted\_class = knn\_classifier(df, query, k=3)

# Output final result

predicted\_class\_label = 'Yes' if predicted\_class == 1 else 'No'

print(f"\nFinal Prediction is: {predicted\_class\_label}")

OUTPUT:

