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)
   b. Applications of Classification Algorithms
   c. Advantages and Disadvantages of Classification Algorithms

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://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://github.com/2796gaurav/Naive-bayes-explained/blob/master/Naive%20bayes/Naive%20Bayes%20in%20scikit%20learn.ipynb**
- **https://medium.com/analytics-vidhya/naive-bayes-classifier-a-beginners-guide-to-master-the-fastest-and-simplest-classification-d6a368e6b737**

This notebook implements two classification algorithms:

1. **Naïve Bayes Classifier**
2. **K-Nearest Neighbors (KNN) Classifier**

We will:

- Load and preprocess the dataset
- Train both classifiers
- Make predictions
- Compare their accuracy
- Predict for a given input: `("Sunny", "Cool", "High", "Strong")`

Step 1: Import Required Libraries

We import essential libraries for:

- Data handling (`pandas`)
- Data visualization (`tabulate`)
- Machine learning models (`sklearn`)

In [ ]:

```
import pandas as pd
from tabulate import tabulate
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Step 2: Load and Display the Dataset

We read the dataset from a CSV file and display the rows using `tabulate` for better readability.

In [ ]:

```
df = pd.read_csv('/content/weather_data.csv')
print(tabulate(df, headers='keys', tablefmt='pretty', stralign="left"))
```

```
+----+-----+----------+-------------+----------+--------+------+
|    | Day | Outlook  | Temperature | Humidity | Wind   | Play |
+----+-----+----------+-------------+----------+--------+------+
| 0  | D1  | Sunny    | Hot         | High     | Weak   | No   |
| 1  | D2  | Sunny    | Hot         | High     | Strong | No   |
| 2  | D3  | Overcast | Hot         | High     | Weak   | Yes  |
| 3  | D4  | Rain     | Mild        | High     | Weak   | Yes  |
| 4  | D5  | Rain     | Cool        | Normal   | Weak   | Yes  |
| 5  | D6  | Rain     | Cool        | Normal   | Strong | No   |
| 6  | D7  | Overcast | Cool        | Normal   | Strong | Yes  |
| 7  | D8  | Sunny    | Mild        | High     | Weak   | No   |
| 8  | D9  | Sunny    | Cool        | Normal   | Weak   | Yes  |
| 9  | D10 | Rain     | Mild        | Normal   | Weak   | Yes  |
| 10 | D11 | Sunny    | Mild        | Normal   | Strong | Yes  |
| 11 | D12 | Overcast | Mild        | High     | Strong | Yes  |
| 12 | D13 | Overcast | Hot         | Normal   | Weak   | Yes  |
| 13 | D14 | Rain     | Mild        | High     | Strong | No   |
+----+-----+----------+-------------+----------+--------+------+
```

Step 3: Check Dataset Columns

We print the column names to understand the dataset's structure.

In [ ]:

```
print("Columns are:",df.columns)
```

```
Columns are: Index(['Day', 'Outlook', 'Temperature', 'Humidity', 'Win
d', 'Play'], dtype='object')
```

Step 4: Convert Categorical Data to Numeric

Since machine learning models work with numerical data, we use `LabelEncoder` to transform categorical values into numerical values.

In [ ]:

```
label_encoder_outlook = LabelEncoder()
label_encoder_temperature = LabelEncoder()
label_encoder_humidity = LabelEncoder()
label_encoder_wind = LabelEncoder()
label_encoder_play = LabelEncoder()

df['Outlook'] = label_encoder_outlook.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder_temperature.fit_transform(df['Temperature'])
df['Humidity'] = label_encoder_humidity.fit_transform(df['Humidity'])
df['Wind'] = label_encoder_wind.fit_transform(df['Wind'])
df['Play'] = label_encoder_play.fit_transform(df['Play'])

print("Dataset after Label Encoding is: ")
print(tabulate(df, headers='keys', tablefmt='pretty', stralign="left"))
```

```
Dataset after Label Encoding is:
+----+-----+---------+-------------+----------+------+------+
|    | Day | Outlook | Temperature | Humidity | Wind | Play |
+----+-----+---------+-------------+----------+------+------+
| 0  | D1  | 2       | 1           | 0        | 1    | 0    |
| 1  | D2  | 2       | 1           | 0        | 0    | 0    |
| 2  | D3  | 0       | 1           | 0        | 1    | 1    |
| 3  | D4  | 1       | 2           | 0        | 1    | 1    |
| 4  | D5  | 1       | 0           | 1        | 1    | 1    |
| 5  | D6  | 1       | 0           | 1        | 0    | 0    |
| 6  | D7  | 0       | 0           | 1        | 0    | 1    |
| 7  | D8  | 2       | 2           | 0        | 1    | 0    |
| 8  | D9  | 2       | 0           | 1        | 1    | 1    |
| 9  | D10 | 1       | 2           | 1        | 1    | 1    |
| 10 | D11 | 2       | 2           | 1        | 0    | 1    |
| 11 | D12 | 0       | 2           | 0        | 0    | 1    |
| 12 | D13 | 0       | 1           | 1        | 1    | 1    |
| 13 | D14 | 1       | 2           | 0        | 0    | 0    |
+----+-----+---------+-------------+----------+------+------+
```

Step 5: Split Dataset into Training and Testing Sets

We split the dataset into:

- **Features (X):** All columns except the target (`Play`)
- **Target (y):** The `Play` column (what we are predicting)
- **70% Training data** and **30% Testing data** for evaluation.

In [ ]:

```
X = df[['Outlook', 'Temperature', 'Humidity', 'Wind']]
y = df['Play']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

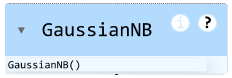
Step 6: Train a Naïve Bayes Classifier

We use `GaussianNB()` to train the Naïve Bayes model on the training dataset.

In [ ]:

```
model = GaussianNB()
model.fit(X_train, y_train)
```

Out[ ]:



Step 7: Make Predictions using Naïve Bayes

We use the trained model to predict values for the test set and evaluate its accuracy.

In [ ]:

```
y_pred_nb = model.predict(X_test)
accuracy_nb = accuracy_score(y_test, y_pred_nb)

print(f"Naïve Bayes Model Accuracy: {accuracy_nb:.2f}")
```

```
Naïve Bayes Model Accuracy: 0.60
```

Step 8: Define a Function to Predict Custom Input

We create a function to encode a given input (`"Sunny", "Cool", "High", "Strong"`) into numerical values and use the trained model to predict the outcome.

In [ ]:

```
def predict(outlook, temperature, humidity, wind):

    try:
        outlook_encoded = label_encoder_outlook.transform([outlook])[0]
        temperature_encoded = label_encoder_temperature.transform([temperature])[0]
        humidity_encoded = label_encoder_humidity.transform([humidity])[0]
        wind_encoded = label_encoder_wind.transform([wind])[0]
    except ValueError as e:
        print(f"Error in encoding input values: {e}")
        return None


    new_data = pd.DataFrame({
        'Outlook': [outlook_encoded],
        'Temperature': [temperature_encoded],
        'Humidity': [humidity_encoded],
        'Wind': [wind_encoded]
    })


    prediction = model.predict(new_data)
    predicted_class = label_encoder_play.inverse_transform(prediction)

    return predicted_class[0]

result = predict("Sunny", 'Cool', 'High', 'Strong')
print(f'Prediction for Sunny, Cool, High, Strong: {result}')
```

```
Prediction for Sunny, Cool, High, Strong: No
```

Step 9: Train a KNN Classifier

We use `KNeighborsClassifier(n_neighbors=3)` to train a KNN model with 3 neighbors.

In [ ]:

```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
```

Out[ ]:

Step 10: Make Predictions using KNN

We use the trained KNN model to predict values for the test set and evaluate its accuracy.

In [ ]:

```
y_pred_knn = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

print(f"KNN Model Accuracy: {accuracy_knn:.2f}")
```

```
KNN Model Accuracy: 0.60
```