Lab Number: 12

Title
Programmatically demonstrate KNN classification

Objective
To implement the K-Nearest Neighbors (KNN) classification algorithm programmatically using Python, train the model on a suitable dataset, evaluate its accuracy, and interpret the classification results.

IDE/Tools Used
VS Code

Programming Language Used
Python

Theory
Classification Algorithms: Classification algorithms are a type of supervised learning algorithm in machine learning that categorize data into distinct classes or groups. They learn from labeled data to predict the correct category for new, unseen data, and are used for tasks like spam detection, image recognition, and medical diagnosis. Examples include decision trees, logistic regression, support vector machines (SVM), Naive Bayes, and k-nearest neighbors (KNN).

K-Nearest Neighbors (KNN): KNN is a simple, intuitive, and non-parametric classification algorithm. It assigns a class to an input sample based on the majority class among its K nearest neighbors in the feature space. It works by implementing the following steps:

1. Choose the value of K (e.g., K = 3 or 5).
2. Compute distance between the test sample and all training samples.
3. Select the K closest points.
4. Assign the class that appears most frequently among those neighbors.

Key Concepts in KNN
- Distance Metric: Usually Euclidean distance.
- K value selection:
    - Small value of K: sensitive to noise
    - Large value of K: smoother boundaries, but less flexible
- Scaling: Recommended because KNN is distance-based.

Advantages of KNN

- Simple to understand and implement
- Works well for small-to-medium datasets

Disadvantages

- Slower with large datasets
- Very sensitive to feature scaling

Scaler: A scaler is a preprocessing tool that transforms numerical features so they fall within a specific range or distribution. Scaling is often required for machine learning algorithms, especially those that rely on distances or gradients, to prevent variables with larger numeric ranges from dominating the model.

StandardScaler: It is a specific type of scaler that transforms each feature by subtracting the mean and dividing by the standard deviation. This results in values that follow a standard normal distribution with:

- mean = 0
- standard deviation = 1

Standard scaling ensures that all features contribute proportionally in distance-based algorithms such as K-Means, DBSCAN, K-NN, and PCA.

Encoding: Encoding is the process of converting categorical (non-numeric) data into numerical values so that machine learning algorithms can process them. Most algorithms, including KNN, cannot directly interpret text labels because they rely on mathematical operations like distance calculation.

Therefore, encoding translates human-readable categories into machine-readable numbers while preserving the meaning of the categories.

Label Encoding: Label Encoding is a simple form of encoding where each unique category in a feature is assigned a unique integer value. Label encoding is used in KNN because:

- All variables in the contact lens dataset are ordinal or nominal categories.
- KNN requires numeric values to compute distances.
- Label Encoding is efficient and straightforward for this type of dataset since the categories are discrete.

Implementation

1. Dataset

The dataset used is the contact_lenses.csv and it contains:

Feature	Description
age	young, pre-presbyopic, presbyopic
spectacle-prescrip	myope or hypermetrope
astigmatism	yes or no
tear-prod-rate	normal or reduced
contact-lenses (Target)	soft, hard, none

The goal is to classify patients into one of the three lens categories.

contact_lenses.csv > data

```
id,age,spectacle-prescrip,astigmatism,tear-prod-rate,contact-lenses
74,young,hypermetrope,no,normal,soft
19,presbyopic,myope,yes,reduced,none
119,young,myope,no,normal,soft
79,young,hypermetrope,no,reduced,none
77,young,hypermetrope,no,reduced,none
32,presbyopic,myope,no,normal,soft
65,presbyopic,myope,yes,reduced,none
142,pre-presbyopic,hypermetrope,no,normal,soft
69,pre-presbyopic,hypermetrope,no,normal,soft
83,presbyopic,myope,no,normal,soft
111,young,myope,yes,normal,hard
13,pre-presbyopic,hypermetrope,no,reduced,none
37,pre-presbyopic,hypermetrope,no,reduced,none
10,pre-presbyopic,myope,no,normal,soft
20,presbyopic,myope,yes,normal,hard
57,pre-presbyopic,myope,yes,reduced,none
105,presbyopic,myope,yes,normal,hard
70,young,hypermetrope,no,normal,soft
56,presbyopic,hypermetrope,no,normal,soft
133,presbyopic,myope,yes,normal,hard
30,young,myope,yes,normal,hard
128,young,myope,yes,reduced,none
27,young,hypermetrope,no,normal,soft
129,pre-presbyopic,myope,no,reduced,none
132,presbyopic,hypermetrope,no,normal,soft
146,presbyopic,hypermetrope,yes,reduced,none
109,presbyopic,hypermetrope,yes,reduced,none
144,pre-presbyopic,hypermetrope,yes,normal,none
46,presbyopic,myope,yes,normal,hard
31,pre-presbyopic,hypermetrope,yes,normal,none
23,presbyopic,hypermetrope,yes,reduced,none
16,pre-presbyopic,hypermetrope,yes,normal,none
66,pre-presbyopic,hypermetrope,no,reduced,none
12,pre-presbyopic,myope,yes,normal,hard
43,young,myope,yes,reduced,none
147,pre-presbyopic,hypermetrope,yes,reduced,none
```

Figure 1: Visualization of the dataset used

2. Code

The python code to implement the KNN algorithm is given below:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Loading the data data
df = pd.read_csv("contact_lenses.csv")

# Droping the ID column because it is not useful
df = df.drop(columns=['id'])

# Encoding the categorical columns into numeric
encoder = LabelEncoder()
for col in df.columns:
    df[col] = encoder.fit_transform(df[col])

# Splitting into features and class
X = df.drop('contact-lenses', axis=1)
y = df['contact-lenses']

# Train and test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# KNN Model where k is set to 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predictions of the test data
y_pred = knn.predict(X_test)

# Evaluation and report of the prediction
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Showing some of the sample predictions
pred_samples = pd.DataFrame({
    "Actual": y_test.values[:10],
    "Predicted": y_pred[:10]
})
print(pred_samples)
```

Output:

```
sayuj@Sayujya MINGW64 ~/OneDrive/Desktop/BIM054/BIM_7th/IT 274_Data Warehousing and Data Mining/Labs/Lab 12 KNN
$ python -u "c:\Users\sayuj\OneDrive\Desktop\BIM054\BIM_7th\IT 274_Data Warehousing and Data Mining\Labs\Lab 12 KNN\KNN.py"
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

   Actual  Predicted
0       2          2
1       2          2
2       1          1
3       2          2
4       1          1
5       1          1
6       2          2
7       1          1
8       2          2
9       2          2
```

Figure 2: Output in the console

Discussion

The KNN classification model was applied to the contact lenses dataset after converting all categorical variables into numerical form using Label Encoding. The model was trained using an 80–20 train-test split and evaluated using accuracy, classification report, and a predicted-vs-actual comparison.

The results show that the model achieved an accuracy of 1.0 (100%), meaning it correctly classified every instance in the test set. The classification report shows precision, recall, and F1-scores of 1.00 for all three classes (none, soft, and hard lenses), indicating perfect performance across all evaluation metrics. Such a result suggests that the dataset contains patterns that KNN can easily separate based on the encoded attributes age, spectacle prescription, astigmatism, and tear production rate.

The confusion comparison table also shows that all predictions matched the true labels, with no misclassified instances. This indicates that the class clusters in the feature space are well-defined, and KNN's distance-based approach effectively distinguishes between the three contact lens categories. However, it also suggests that the dataset may be simple or not highly varied, which often leads to perfect classification scores.

Conclusion

The programmatic implementation of KNN classification on the contact lenses dataset produced highly accurate results, with 100% accuracy and perfect precision, recall, and F1-scores. This demonstrates that KNN is a highly suitable algorithm for this dataset, successfully capturing the relationships between patient attributes and recommended lens types. The experiment confirms that when categorical attributes are properly encoded and the dataset contains clear class boundaries, KNN can perform exceptionally well in medical recommendation and classification tasks.