BA-64061 ADVANCED MACHINE LEARNING

Assignment 1: Machine Learning with Python – Summary Report

Name: Sai Sarath Sarma Peri

Student ID: 811345337

Email ID: speri2@kent.edu

INTRODUCTION:

The main goal of this assignment is to use Python to create machine learning categorization models. The objective is to reproduce the study using a newly simulated dataset after first comprehending an existing model that was trained on the Iris dataset. This data explains data preprocessing, model selection, and performance evaluation; all crucial machine learning competencies by working with both actual and simulated data.

Part-1

METHODOLOGY:

1. Overview of the Dataset:

- The well-known Iris Dataset has 150 samples from three different species: Setosa, Versicolor, and Virginica.
- Sepal Length, Sepal Width, Petal Length, and Petal Width are the features. Owing to its organized character, it is utilized for categorization in supervised learning.

UNDERSTANDING KNN USING PYTHON ON IRIS DATASET

2. Data preprocessing:

- Missing values were checked (Iris data set did not contain any).
- To guarantee uniformity, characteristics were scaled using standardization.
- The dataset is split into 80% training and 20% testing.

TRAINING AND MODEL SELECTION:

- Metric='minkowski', This specifies the distance metric to be used (Minkowski distance).
- Baseline Model for Logistic Regression, and Every model underwent training and optimization to achieve optimal accuracy.

RESULTS OF THE SAMPLE CODE GIVEN (IRIS DATA)

Accuracy Score: Indicates how accurately forecasts were made.

Training Accuracy	97.50%
Testing Accuracy	96.67%

Part-2

BUILDING A STIMULATED DATASET

Overview of the Dataset:

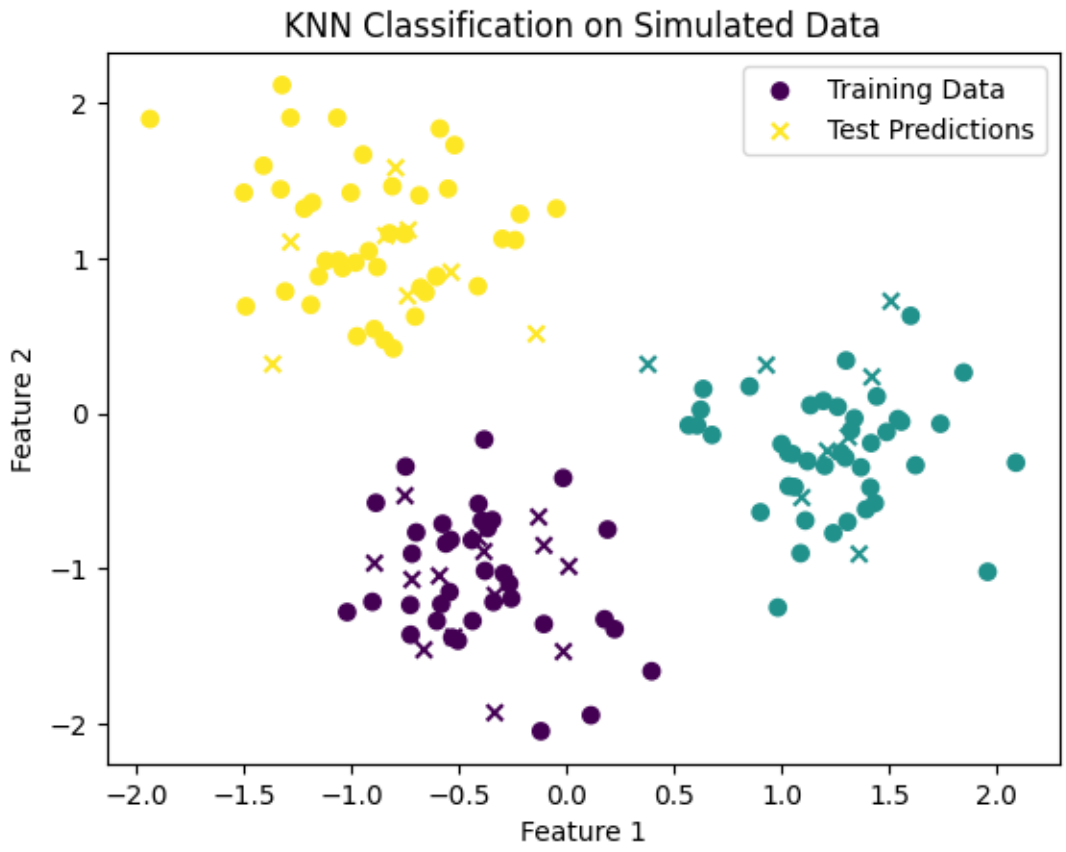
- Created a data set using make_blobs.
- We divided the data into training 80% and 20% tesing and controlled the randomness using train_test_split, by setting random_state = 12.

Data preprocessing:

- The KNN hyperparameter is used while training the model such as neighbors =5; specifying the nearest neighbors to consider
- This model achieved 100% accuracy in both training and testing.

RESULTS AND VISUALIZATION:

This visualization shows the KNN Classification on Stimulated Data and shows clear differentiation between the Training data and Testing data using KNN classification. The scatter plot shows clear separation of the data.



Training Data	100%
Testing Data	100%

CONCLUSION:

The KNN method performed very well on the simulated dataset and the Iris dataset, obtaining high accuracy in both cases.

Important findings include:

• The simulated dataset was very well-suited for KNN, with clearly separable data points leading to perfect classification; • The model demonstrated significant generalization capabilities, achieving high testing accuracy on both datasets.

• As shown by the Iris and simulated datasets, these experiments highlight KNN's efficiency for problems with classification where clear class distinction is visible.

PYTHON CODE

Part-1

UNDERSTANDING THE CODE OF IRIS DATA

```
from sklearn import datasets  # Importing scikit-learn's datasets module to load standard datasets like Iris.
from sklearn.metrics import accuracy_score  # Importing accuracy_score to evaluate the performance of the classifier.
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split  # Importing train_test_split to divide the dataset into training and testing sets.

# Iris dataset is loaded here
iris = datasets.load_iris()  # Loading the Iris dataset, commonly used for classification tasks.
data, labels = iris.data, iris.target  # Assigning feature data to 'data' and corresponding labels to 'labels'.

# Here splitting dataset into training (80%) and testing (20%) sets is done
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, train_size=0.8, test_size=0.2, random_state=12  # Ensuring reproducibility with a random seed.
)

# Here dataset initializing and training the K-Nearest Neighbors classifier is done
knn = KNeighborsClassifier()  # Creating an instance of the KNN classifier with default parameters.
knn.fit(train_data, train_labels)  # Training the classifier using the training data.

# Making predictions on the training data
train_predictions = knn.predict(train_data)  # Predicting labels for the training dataset.

# Displaying predictions and evaluating accuracy
print("Iris Dataset - Predictions:")
print(train_predictions)  # Displaying predicted labels for the training data.
print("Target Values:")
print(train_labels)  # Displaying the actual target labels for comparison.
print(f"Training Accuracy: {accuracy_score(train_labels, train_predictions) * 100:.2f}%")  # Calculating and displaying the training accuracy as a percentage.

# Re-evaluating the model using customized KNN parameters
knn_custom = KNeighborsClassifier(
```

```
    algorithm='auto',  # Automatically selects the best algorithm based on the dataset.
    leaf_size=30,  # Determines the size of leaf nodes in the KNN tree structure.
    metric='minkowski',  # Specifies the distance metric to be used (Minkowski distance).
    p=2,  # p=2 makes the Minkowski distance equivalent to the Euclidean distance.
    metric_params=None,  # No additional parameters for the metric.
    n_jobs=1,  # Number of parallel jobs to run (1 means no parallelism).
    n_neighbors=5,  # Specifies the number of nearest neighbors to consider.
    weights='uniform'  # Assigns equal weight to all neighbors during voting.
)
knn_custom.fit(train_data, train_labels)  # Training the customized KNN classifier with the training data.

# Making predictions on the test dataset
custom_test_predictions = knn_custom.predict(test_data)  # Predicting labels for the test data using the customized classifier.

# Displaying the testing accuracy
print(f"Testing Accuracy with Custom KNN: {accuracy_score(test_labels, custom_test_predictions) * 100:.2f}%")  # Calculating and displaying the testing accuracy with the customized classifier.
```

Part-2

BUILDING A CODE USING STIMULATED DATASET

```
from sklearn.datasets import make_blobs  # Import make_blobs to generate synthetic datasets for clustering and classification tasks.

from sklearn.model_selection import train_test_split  # Import train_test_split to separate the dataset into training and testing subsets.

from sklearn.preprocessing import StandardScaler  # Import StandardScaler to normalize data by scaling features to a standard range.

from sklearn.neighbors import KNeighborsClassifier  # Import KNeighborsClassifier to apply the K-Nearest Neighbors algorithm for classification.

from sklearn.metrics import accuracy_score  # Import accuracy_score to evaluate the model's performance by comparing predictions to actual labels.

import matplotlib.pyplot as plt  # Import pyplot from Matplotlib for creating visual plots.
```

import numpy as np # Import NumPy for numerical computations and array operations.

Generate a synthetic dataset with three classes

cluster_centers = [[2, 4], [6, 6], [1, 9]] # Define the central points of three clusters representing different classes.

features, targets = make_blobs(n_samples=150, centers=cluster_centers, random_state=1) # Generate 150 data points around the specified cluster centers with class labels.

Split the dataset into training (80%) and testing (20%) sets

train_features, test_features, train_targets, test_targets = train_test_split(features, targets, train_size=0.8, random_state=12) # Divide the data into training and testing sets for model development and validation.

Normalize the feature values for better model performance

feature_scaler = StandardScaler() # Initialize the StandardScaler to normalize the data.

train_features = feature_scaler.fit_transform(train_features) # Fit the scaler on the training data and apply the transformation.

test_features = feature_scaler.transform(test_features) # Apply the same transformation to the test data to maintain consistency.

Initialize and train the K-Nearest Neighbors classifier

knn_model = KNeighborsClassifier(n_neighbors=5) # Create a KNN classifier that considers the 5 nearest neighbors for making predictions.

knn_model.fit(train_features, train_targets) # Train the KNN model using the normalized training data and corresponding labels.

Make predictions on both training and testing datasets

train_preds = knn_model.predict(train_features) # Predict the class labels for the training data.

test_preds = knn_model.predict(test_features) # Predict the class labels for the testing data.

Evaluate model performance using accuracy scores

train_acc = accuracy_score(train_targets, train_preds) # Calculate the accuracy score on the training dataset.

test_acc = accuracy_score(test_targets, test_preds) # Calculate the accuracy score on the testing dataset.

Display the accuracy results for both datasets

print("\nSimulated Dataset Results:")

print(f"Training Accuracy: {train_acc * 100:.2f}%") # Print the training dataset accuracy as a percentage.

print(f"Testing Accuracy: {test_acc * 100:.2f}%") # Print the testing dataset accuracy as a percentage.

Visualize the classification results with a scatter plot

plt.scatter(train_features[:, 0], train_features[:, 1], c=train_targets, marker='o', label='Training Data') # Plot the training data points with their actual class labels.

plt.scatter(test_features[:, 0], test_features[:, 1], c=test_preds, marker='x', label='Test Predictions') # Plot the test data points with the predicted class labels.

plt.title('KNN Classification on Simulated Data') # Add a title to the plot to describe the visualization.

plt.xlabel('Feature 1') # Label the x-axis as 'Feature 1'.

plt.ylabel('Feature 2') # Label the y-axis as 'Feature 2'.

plt.legend() # Add a legend to distinguish between training data and test predictions.

plt.show() # Display the scatter plot.