**A Machine Learning project with Python**

**Assignment-1**

**Rajesh Nalliboyia**

Department of Information Systems and Business Analytics,

Kent State University

BA-64061-001 Advanced Machine Learning

Chaojiang (CJ) Wu, Ph.D.

09-17-2024

**INTRODUCTION**

This report presents the findings from a K-Nearest Neighbors (KNN) classification project. The project encompasses two primary tasks:

1. Applying KNN to the Iris dataset.
2. Reproducing the analysis using a simulated dataset generated with the make\_blobs function.

***Part-1***

**UNDERSTANDING KNN ON IRIS DATASET USING PYTHON**

**Data Loading and Splitting:**

* The Iris dataset was imported using datasets.load\_iris().
* The data was divided into training and testing sets in an 80-20 ratio using train\_test\_split, with randomness controlled by setting random\_state=12.

**Model Training and Evaluation:**

* A default KNN classifier was trained on the training data.
* The model achieved an accuracy of 97.5% on the training set and 96.67% on the test set.
* Hyperparameters such as n\_neighbors=5 and metric='minkowski' were specified for a re-trained model, but the results remained similar, confirming the effectiveness of the default settings.

**Results Summary for Iris Dataset:**

* **Training Accuracy:** 97.5%
* **Testing Accuracy:** 96.67%

***PART-2***

**IMPLEMENTATION USING A SIMULATED DATASET**

**Data Generation and Splitting:**

* A dataset was created using make\_blobs with three distinct centers.
* The data was divided into training (80%) and testing (20%) sets using train\_test\_split.

**Model Training and Evaluation:**

* A KNN classifier, set with n\_neighbors=5, was trained on the simulated dataset.
* The model achieved perfect accuracy, scoring 100% on both the training and testing sets.

**Visualization:**

* A scatter plot was produced to visually represent the data points and their labels. The plot confirmed clear cluster separation, indicating the dataset’s suitability for KNN classification.

A chart of different colored dots

Description automatically generated

**Results Summary for Simulated Dataset:**

* **Training Accuracy:** 100%
* **Testing Accuracy:** 100%

**PYTHON CODE**

***Part-1***

***UNDERSTANDING KNN ON IRIS DATASET USING PYTHON***

*# Importing necessary modules from the sklearn library for the project*

from sklearn import datasets

from sklearn.metrics import accuracy\_score

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

from sklearn.neighbors import KNeighborsClassifier

*# Loading and Preparing the Iris Dataset*

*# This dataset contains measurements of iris flowers in different categories*

iris = datasets.load\_iris()

data, labels = iris.data, iris.target

*# Split the dataset into training and testing sets using an 80-20 ratio*

*# 'train\_test\_split' automatically shuffles the data and splits it into train and test*

res = train\_test\_split(data, labels, train\_size=0.8,test\_size=0.2,random\_state=12)

train\_data, test\_data, train\_labels, test\_labels = res

knn = KNeighborsClassifier()

*# Fit the KNN model on the training data*

*# The model learns to classify based on the distances between the training data points*

knn.fit(train\_data, train\_labels)

*# Use the trained model to predict the labels of the training data*

learn\_data\_predicted = knn.predict(train\_data)

*# Output the predictions and the true labels of the training data*

print("Predictions from the classifier:")

print(learn\_data\_predicted)

print("Target values:")

print(train\_labels)

print('\n')

*# Calculate and print the accuracy score on the training set*

*# Accuracy is the ratio of correct predictions to total prediction*

*# Assuming learn\_data\_predicted and train\_labels are already defined*

accuracy = accuracy\_score(learn\_data\_predicted, train\_labels)

print('Accuracy of the Train lables: ' + str(accuracy))

*# Re-create the KNN classifier, this time with specific parameters*

knn2 = KNeighborsClassifier(algorithm='auto',  leaf\_size=30, metric='minkowski',p=2, n\_neighbors=5,  weights='uniform')

*# Fit the model to the training data again*

knn2.fit(train\_data, train\_labels)

test\_data\_predicted = knn2.predict(test\_data)

*# Calculate and print the accuracy score on the test data*

accuracy = (accuracy\_score(test\_data\_predicted, test\_labels))

print('Accuracy of the test lables :' + str(accuracy))

***PART-2***

***IMPLEMENTATION USING A SIMULATED DATASET***

*# Importing necessary libraries*

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

import numpy as np

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

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

*#Create a simulated dataset using make\_blobs*

*# Define the centers of the simulated blobs (i.e., clusters)*

centers = [[2, 4], [6, 6], [1, 9]]

data, labels = make\_blobs(n\_samples=150, centers=np.array(centers), random\_state=1)

*# Convert to DataFrame to include column names*

df = pd.DataFrame(data, columns=['Feature 1', 'Feature 2'])

df['Label'] = labels

*# Print the first 5 rows of the simulated dataset*

print("First 5 rows of the simulated dataset:")

print(df.head())

*# Split the simulated dataset into training (80%) and testing (20%)*

train\_data, test\_data, train\_labels, test\_labels = train\_test\_split(data, labels, train\_size=0.8, test\_size=0.2, random\_state=12)

*# Create a KNN classifier*

knn\_simulated = KNeighborsClassifier(n\_neighbors=5, algorithm='auto', metric='minkowski', p=2)

knn\_simulated.fit(train\_data, train\_labels)

*# Predict the train data labels*

train\_data\_predicted\_simulated = knn\_simulated.predict(train\_data)

*# Print predictions and target values for the training set*

print("\nTraining Set Predictions from the classifier:")

print(train\_data\_predicted\_simulated)

print("Training Set Target values:")

print(train\_labels)

*# Calculate and print accuracy for the simulated training data*

train\_accuracy\_simulated = accuracy\_score(train\_data\_predicted\_simulated, train\_labels)

print("Training Accuracy for simulated dataset: " + str(train\_accuracy\_simulated))

*# Predict the test data labels*

test\_data\_predicted\_simulated = knn\_simulated.predict(test\_data)

*# Calculate and print accuracy for the simulated test data*

test\_accuracy\_simulated = accuracy\_score(test\_data\_predicted\_simulated, test\_labels)

print("Test Accuracy for simulated dataset: "+ str(test\_accuracy\_simulated))

*# Plot the simulated dataset*

plt.figure(figsize=(8, 6))

plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')

plt.title('Simulated Dataset: Data Points and Their True Labels')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

**CONCLUSION**

The KNN algorithm demonstrated outstanding performance on both the Iris dataset and the simulated dataset, achieving high accuracy in both cases. Key observations include:

* The model exhibited strong generalization capabilities, achieving high testing accuracy on both datasets.
* The simulated dataset was particularly well-suited for KNN, with clearly separable data points leading to perfect classification.
* These experiments highlight KNN’s effectiveness for classification tasks where distinct class separation is evident, as seen in both the Iris and simulated datasets.