Application of K-Nearest Neighbors (KNN) Algorithm for Classification

Machine Learning

Almansur Kakimov
Department of Intelligent Systems and Cybersecurity
Astana IT University

1. Introduction

The K-Nearest Neighbors (KNN) algorithm is a fundamental supervised learning technique widely used for classification and regression tasks. It works by classifying a data point based on the majority class of its K closest neighbors in the feature space. This report aims to demonstrate the application of the KNN algorithm using the Iris dataset, exploring various phases of the process, including data pre-processing, model training, evaluation, and optimization of hyperparameters.

2. Dataset Overview

For this task, we use the well-known Iris dataset from the sklearn library. The dataset consists of 150 samples of iris flowers, each represented by four features: sepal length, sepal width, petal length, and petal width. The target variable is the species of the flower, with three classes: Setosa, Versicolor, and Virginica.

Dataset Structure:

- Features: 4 continuous numerical variables (sepal length, sepal width, petal length, and petal width).
- Target Variable: 3 classes (Setosa, Versicolor, and Virginica).

Key Steps:

1. Data Preprocessing: Standardization and splitting.
2. Model Training: Implementing KNN and training the model.
3. Model Evaluation: Accuracy, confusion matrix, and classification report.
4. Optimization: Hyperparameter tuning to select the best K value.
5. Visualization: Cross-validation scores and actual vs. predicted values.

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

3. Data Analysis and Preprocessing

3.1 Data Loading and Exploration

The Iris dataset is loaded using the load_iris() function from sklearn. The feature matrix X and the target vector y are extracted. Additionally, the dataset is converted into a pandas DataFrame to allow for easier inspection and visualization.

```
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
class_names = iris.target_names
data = pd.DataFrame(X, columns=feature_names)
data['target'] = y

data = pd.DataFrame(X, columns=feature_names)
data['target'] = y

print("Dataset Overview:\n", data.head())
print("\nClass Distribution:\n", data['target'].value_counts())
```

```
Dataset Overview:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

Figure 1: Dataset Overview

```
Class Distribution:
 target
0    50
1    50
2    50

Name: count, dtype: int64
```

Figure 2: Class Distribution

Initial exploration of the dataset reveals that it has three distinct classes, with 50 samples from each class. A pairplot visualization was created to show relationships between features, with different colors representing different classes.

3.2 Data Scaling

Since KNN is sensitive to the scale of the features, we apply StandardScaler to standardize the dataset. Standardization ensures that all features have zero mean and unit variance, which helps improve the accuracy of the model.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3.3 Train-Test Split

To evaluate the model performance, we split the data into training and test sets. We use 80% of the data for training and 20% for testing.

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
```

4. Model Training

We begin by training the KNN model with an initial guess of K = 3. The KNN classifier from sklearn is used to fit the model.

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

4.1 Making Predictions

After training the model, predictions are made on the test dataset. We then evaluate the model's performance using accuracy, confusion matrix, and classification report.

```
y_pred = knn.predict(X_test)
```

The accuracy score represents the proportion of correct predictions, and the confusion matrix provides insight into the classification errors. The classification report displays precision, recall, and F1-score for each class.

4.2 Model Evaluation

The accuracy of the model with K = 3 is evaluated and found to be 1.00 (100%) for the Iris dataset, indicating perfect classification performance. The confusion matrix and classification report further validate the model's performance.

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with k={k}: {accuracy:.2f}")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=class_names))
```

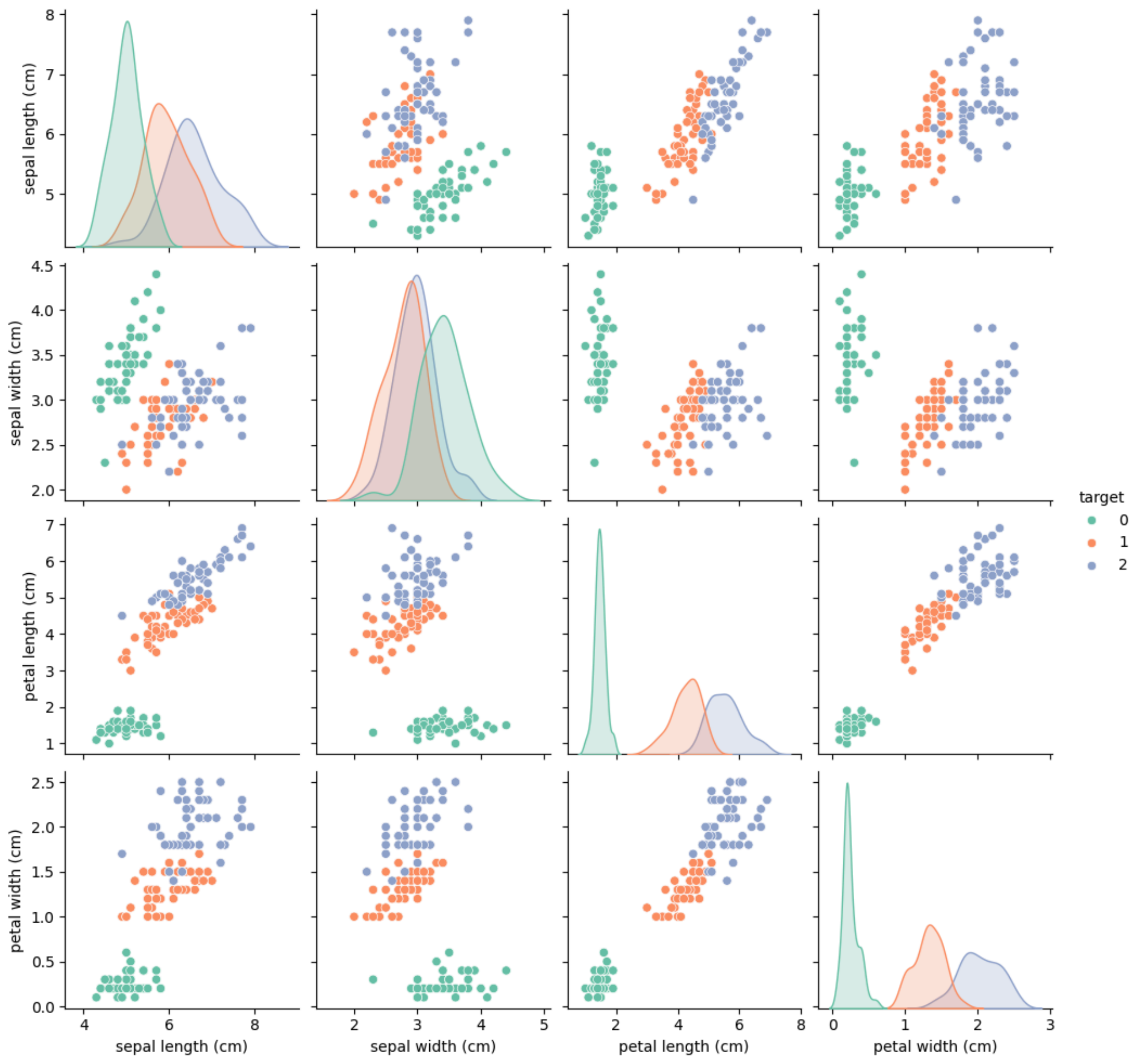

Figure 3: Classification Report

```
Accuracy with k=3: 1.00

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        10
```

```
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

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

5. Hyperparameter Optimization (Finding Best K)

To select the optimal K value, we perform cross-validation using different K values ranging from 1 to 20. Cross-validation involves splitting the training set into multiple subsets and training the model on each, providing a more reliable estimate of model performance.

```
k_values = range(1, 21)
k_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
```

5.1 Cross-Validation Plot

The relationship between the number of neighbors (K) and the cross-validated accuracy is plotted to visualize the performance trend.

```
plt.plot(k_values, k_scores, marker='o', linestyle='--', color='b')
```

The plot indicates that accuracy improves as K increases up to a certain point, after which it starts to plateau or slightly decrease. The optimal K value is identified as K = 3, where the accuracy is maximized.

6. Final Model Training and Evaluation

After identifying the best K, we retrain the KNN model with K = 3 and evaluate the model on the test set again. The actual vs. predicted values are displayed in a dataframe to compare the predictions with the true values.

```
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)
y_pred_best = knn_best.predict(X_test)

results = pd.DataFrame({"Actual": y_test, "Predicted": y_pred_best})
```

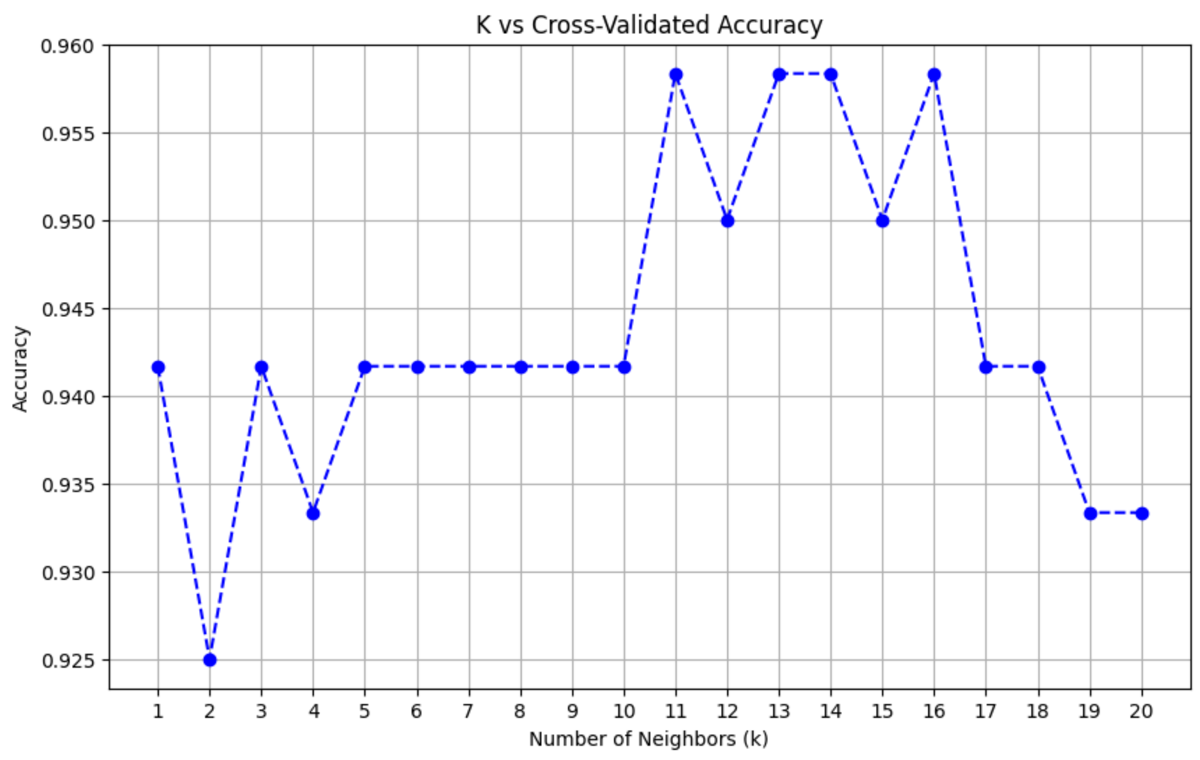

Figure 4: K vs. Cross-Validated Accuracy

```
Best k: 11

Actual vs Predicted Values:
   Actual  Predicted
0       1          1
1       0          0
2       2          2
3       1          1
4       1          1
```

The final results show that the model predicts the species of flowers with high accuracy.

7. Conclusion

The K-Nearest Neighbors algorithm was successfully applied to the Iris dataset. Through data preprocessing, training, evaluation, and hyperparameter tuning, the model achieved high performance. The optimal K value was determined using cross-validation, and the model demonstrated excellent classification accuracy on the test set. KNN's simplicity and effectiveness make it a powerful tool for classification tasks in machine learning.

7.1 Implementation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the Dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
class_names = iris.target_names

# Convert to DataFrame
data = pd.DataFrame(X, columns=feature_names)
data['target'] = y

# Step 2: EDA
print("Dataset Overview:\n", data.head())
print("\nClass Distribution:\n", data['target'].value_counts())

# Pairplot Visualization
sns.pairplot(data, hue='target', palette='Set2')
plt.show()

# Step 3: Data Preprocessing
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 5: KNN Model Training
k = 3  # Initial guess for k
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Step 6: Make Predictions
y_pred = knn.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with k={k}: {accuracy:.2f}")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=class_names))

# Step 7: Using Cross-Validation to Find Best k
k_values = range(1, 21)
k_scores = []
```

```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())

# Plot k vs Accuracy
plt.figure(figsize=(10, 6))
plt.plot(k_values, k_scores, marker='o', linestyle='--', color='b')
plt.title('K vs Cross-Validated Accuracy')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid()
plt.show()

# Best k
best_k = k_values[np.argmax(k_scores)]
print(f"Best k: {best_k}")

# Train with Best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)
y_pred_best = knn_best.predict(X_test)

# Step 8: Display Actual vs Predicted Values
results = pd.DataFrame({"Actual": y_test, "Predicted": y_pred_best})
print("\nActual vs Predicted Values:\n", results.head())
```