Nithin S
221IT085

IT350 Lab Assignment 1

Q1. Implement the KNN classifier using the IRIS dataset.
Use a test set of 20% of the original dataset. Use Euclidean distance as the distance metric.
Implement the classifier for K=3 and 5. Please do not use built-in functions.
Perform the classification with and without cross-validation and analyse the results. For k-fold cross-validation, assume k=10.
Evaluate the classifier using the following metrics: Precision, Recall, F1-score, Accuracy and Confusion Matrix.
Once you have implemented the KNN classifier without using the built-in function, compare its results with using the built-in function.

Code

Scratch Implementation

Load the IRIS Dataset

[3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

# Load the IRIS dataset
iris = load_iris()
X = iris.data
y = iris.target
```

Train-Test Split

13]:
```
import random

def train_test_split(X, y, test_size=0.2, random_state=42):
    n_samples = len(X)
    random.seed(random_state)
    indices = list(range(n_samples))
    random.shuffle(indices)
    test_set_size = int(n_samples * test_size)
    test_indices = indices[:test_set_size]
    train_indices = indices[test_set_size:]
    X_train = [X[i] for i in train_indices]
    X_test = [X[i] for i in test_indices]
    y_train = [y[i] for i in train_indices]
    y_test = [y[i] for i in test_indices]
    return X_train, X_test, y_train, y_test

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

Implement KNN Classifier from Scratch

[14]:
```python
import math

def euclidean_distance(x1, x2):
    return math.sqrt(sum((a - b) ** 2 for a, b in zip(x1, x2)))

def predict(X_train, y_train, X_test, k):
    predictions = []
    for x in X_test:
        distances = [(euclidean_distance(x, x_train), y_train[i]) for i, x_train in enumerate(X_train)]
        distances.sort(key=lambda x: x[0])
        k_nearest_labels = [label for _, label in distances[:k]]
        most_common = max(set(k_nearest_labels), key=k_nearest_labels.count)
        predictions.append(most_common)
    return predictions
```

Evaluation Metrics

]:
```python
def precision_score(y_true, y_pred):
    true_positive = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp)
    predicted_positive = len(y_pred)
    return true_positive / predicted_positive

def recall_score(y_true, y_pred):
    actual_positive = len(y_true)
    true_positive = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp)
    return true_positive / actual_positive

def f1_score(y_true, y_pred):
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)

def accuracy_score(y_true, y_pred):
    correct_predictions = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp)
    return correct_predictions / len(y_true)

def confusion_matrix(y_true, y_pred):
    unique_labels = sorted(set(y_true))
    matrix = [[0 for _ in unique_labels] for _ in unique_labels]
    label_to_index = {label: index for index, label in enumerate(unique_labels)}
    for yt, yp in zip(y_true, y_pred):
        matrix[label_to_index[yt]][label_to_index[yp]] += 1
    return matrix

def evaluate(y_true, y_pred):
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    conf_matrix = confusion_matrix(y_true, y_pred)
    return precision, recall, f1, accuracy, conf_matrix

def print_evaluation_metrics(precision, recall, f1, accuracy, conf_matrix):
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-Score: {f1}")
    print(f"Accuracy: {accuracy}")
    print("Confusion Matrix:")
    for row in conf_matrix:
        print(row)
```

Implement Cross-Validation

```python
def k_fold_split(X, y, k_folds, random_state=42):
    n_samples = len(X)
    fold_size = n_samples // k_folds
    random.seed(random_state)
    indices = list(range(n_samples))
    random.shuffle(indices)
    folds = [indices[i*fold_size:(i+1)*fold_size] for i in range(k_folds)]
    return folds

def cross_validate(X, y, k_neighbors, k_folds=10):
    folds = k_fold_split(X, y, k_folds)
    precision_scores = []
    recall_scores = []
    f1_scores = []
    accuracy_scores = []
    confusion_matrices = []

    for i in range(k_folds):
        train_indices = [idx for j in range(k_folds) if j != i for idx in folds[j]]
        test_indices = folds[i]
        X_train = [X[idx] for idx in train_indices]
        X_test = [X[idx] for idx in test_indices]
        y_train = [y[idx] for idx in train_indices]
        y_test = [y[idx] for idx in test_indices]
        y_pred = predict(X_train, y_train, X_test, k_neighbors)
        precision, recall, f1, accuracy, conf_matrix = evaluate(y_test, y_pred)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f1_scores.append(f1)
        accuracy_scores.append(accuracy)
        confusion_matrices.append(conf_matrix)

    avg_precision = sum(precision_scores) / k_folds
    avg_recall = sum(recall_scores) / k_folds
    avg_f1 = sum(f1_scores) / k_folds
    avg_accuracy = sum(accuracy_scores) / k_folds
    total_conf_matrix = [[sum(confusion_matrices[j][i][k] for j in range(k_folds)) for k in range(len(confusion_matrices[0][0]))] for i in

    return avg_precision, avg_recall, avg_f1, avg_accuracy, total_conf_matrix

def print_cross_validation_metrics(avg_metrics):
    precision, recall, f1, accuracy, conf_matrix = avg_metrics
    print_evaluation_metrics(precision, recall, f1, accuracy, conf_matrix)
```

Run and Compare Results

```python
# K = 3 without cross-validation
print("K = 3, without cross-validation:")
y_pred = predict(X_train, y_train, X_test, k=3)
metrics = evaluate(y_test, y_pred)
print("Without built-in functions:")
print_evaluation_metrics(*metrics)

# K = 3 with 10-fold cross-validation
print("\nK = 3, with 10-fold cross-validation:")
avg_metrics = cross_validate(X, y, k_neighbors=3, k_folds=10)
print("Without built-in functions:")
print_cross_validation_metrics(avg_metrics)

# K = 5 without cross-validation
print("\nK = 5, without cross-validation:")
y_pred = predict(X_train, y_train, X_test, k=5)
metrics = evaluate(y_test, y_pred)
print("Without built-in functions:")
print_evaluation_metrics(*metrics)

# K = 5 with 10-fold cross-validation
print("\nK = 5, with 10-fold cross-validation:")
avg_metrics = cross_validate(X, y, k_neighbors=5, k_folds=10)
print("Without built-in functions:")
print_cross_validation_metrics(avg_metrics)
```

```
K = 3, without cross-validation:
Without built-in functions:
Precision: 0.9333333333333333
Recall: 0.9333333333333333
F1-Score: 0.9333333333333333
Accuracy: 0.9333333333333333
Confusion Matrix:
[8, 0, 0]
[0, 8, 2]
[0, 0, 12]

K = 3, with 10-fold cross-validation:
Without built-in functions:
Precision: 0.9666666666666668
Recall: 0.9666666666666668
F1-Score: 0.9666666666666668
Accuracy: 0.9666666666666668
Confusion Matrix:
[50, 0, 0]
[0, 47, 3]
[0, 2, 48]

K = 5, without cross-validation:
Without built-in functions:
Precision: 0.9666666666666667
Recall: 0.9666666666666667
F1-Score: 0.9666666666666667
Accuracy: 0.9666666666666667
Confusion Matrix:
[8, 0, 0]
[0, 9, 1]
[0, 0, 12]

K = 5, with 10-fold cross-validation:
Without built-in functions:
Precision: 0.9733333333333334
Recall: 0.9733333333333334
F1-Score: 0.9733333333333334
Accuracy: 0.9733333333333334
Confusion Matrix:
[50, 0, 0]
[0, 47, 3]
[0, 1, 49]
```

Library Implementation

```
]: from sklearn.datasets import load_iris
import pandas as pd

# Load the IRIS dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert to DataFrame for easier handling
X_df = pd.DataFrame(X, columns=iris.feature_names)
y_df = pd.DataFrame(y, columns=['species'])
```

```
]: from sklearn.model_selection import train_test_split

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

```
]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, confusion_matrix

def evaluate(y_true, y_pred):
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    accuracy = accuracy_score(y_true, y_pred)
    conf_matrix = confusion_matrix(y_true, y_pred)
    return precision, recall, f1, accuracy, conf_matrix

def print_evaluation_metrics(precision, recall, f1, accuracy, conf_matrix):
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-Score: {f1}")
    print(f"Accuracy: {accuracy}")
    print("Confusion Matrix:")
    print(conf_matrix)
```

```
2]: # K=3 without cross-validation
knn_3 = KNeighborsClassifier(n_neighbors=3)
knn_3.fit(X_train, y_train)
y_pred_3 = knn_3.predict(X_test)
print("K = 3, without cross-validation:")
metrics_3 = evaluate(y_test, y_pred_3)
print_evaluation_metrics(*metrics_3)

# K=5 without cross-validation
knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5.fit(X_train, y_train)
y_pred_5 = knn_5.predict(X_test)
print("\nK = 5, without cross-validation:")
metrics_5 = evaluate(y_test, y_pred_5)
print_evaluation_metrics(*metrics_5)
```

```
K = 3, without cross-validation:
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

K = 5, without cross-validation:
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

[23]:

```python
from sklearn.model_selection import cross_val_score, cross_val_predict

def cross_validate_knn(X, y, k_neighbors, k_folds=10):
    knn = KNeighborsClassifier(n_neighbors=k_neighbors)
    y_pred = cross_val_predict(knn, X, y, cv=k_folds)
    precision, recall, f1, accuracy, conf_matrix = evaluate(y, y_pred)
    return precision, recall, f1, accuracy, conf_matrix

# K=3 with 10-fold cross-validation
print("\nK = 3, with 10-fold cross-validation:")
avg_metrics_3_cv = cross_validate_knn(X, y, k_neighbors=3, k_folds=10)
print_evaluation_metrics(*avg_metrics_3_cv)

# K=5 with 10-fold cross-validation
print("\nK = 5, with 10-fold cross-validation:")
avg_metrics_5_cv = cross_validate_knn(X, y, k_neighbors=5, k_folds=10)
print_evaluation_metrics(*avg_metrics_5_cv)
```

```
K = 3, with 10-fold cross-validation:
Precision: 0.9667867146858743
Recall: 0.9666666666666667
F1-Score: 0.9666633329999667
Accuracy: 0.9666666666666667
Confusion Matrix:
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]

K = 5, with 10-fold cross-validation:
Precision: 0.9677505687140372
Recall: 0.9666666666666667
F1-Score: 0.9666366396423448
Accuracy: 0.9666666666666667
Confusion Matrix:
[[50  0  0]
 [ 0 46  4]
 [ 0  1 49]]
```