```
import numpy as np
import pandas as pd
import time
from sklearn.metrics import confusion_matrix
import urllib.request
import io


# Load Breast Cancer Wisconsin Diagnostic Dataset from UCI repository
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-w
data = urllib.request.urlopen(url).read().decode('utf-8')
df_wdbc = pd.read_csv(io.StringIO(data), header=None)

# Define column names
columns = ["ID", "Diagnosis"]
for i in range(1, 31):
    columns.append(f"Feature_{i}")
df_wdbc.columns = columns

# Drop ID column and convert Diagnosis to binary
df_wdbc.drop("ID", axis=1, inplace=True)
df_wdbc["Diagnosis"] = df_wdbc["Diagnosis"].map({"M": 1, "B": 0})

# Separate features and target
X_wdbc = df_wdbc.drop("Diagnosis", axis=1).values  # Shape (569, 30)
y_wdbc = df_wdbc["Diagnosis"].values  # Shape (569, )

print("WDBC Dataset shape:", X_wdbc.shape, y_wdbc.shape)
print("Class distribution (M=1, B=0):")
print(pd.Series(y_wdbc).value_counts())
```

```
WDBC Dataset shape: (569, 30) (569,)
Class distribution (M=1, B=0):
0    357
1    212
Name: count, dtype: int64
```

```
def euclidean_distance(x1, x2):
    """
    Calculate Euclidean distance between two points (NumPy arrays).

    Parameters:
    -----------
    x1 : numpy.ndarray
        First point
    x2 : numpy.ndarray
        Second point

    Returns:
    --------
    float: Euclidean distance between x1 and x2
    """
    return np.sqrt(np.sum((x1 - x2) ** 2))
```

```
def knn_classify(X_train, y_train, x_test, k=3):
    """
    Perform K-Nearest Neighbors classification for a single test sample.

    Parameters:
    -----------
    X_train : numpy.ndarray, shape (N, d)
        Training feature matrix
    y_train : numpy.ndarray, shape (N,)
        Training labels (0 or 1)
    x_test : numpy.ndarray, shape (d,)
        Test sample to classify
    k : int, optional (default=3)
        Number of nearest neighbors to consider

    Returns:
    --------
    int: Predicted label (0 or 1)
    """
    # Calculate distances to all training points
    distances = []
    for i in range(len(X_train)):
        dist = euclidean_distance(X_train[i], x_test)
        distances.append((dist, y_train[i]))

    # Sort distances
    distances.sort(key=lambda x: x[0])

    # Select k nearest neighbors
    neighbors = distances[:k]

    # Predict based on majority vote
    labels = [n[1] for n in neighbors]
    prediction = max(set(labels), key=labels.count)

    return prediction


def cross_validation_knn_classification(X, y, k=3, n_folds=6):
    """
    Perform k-fold cross-validation using K-Nearest Neighbors classifier.

    Parameters:
    -----------
    X : numpy.ndarray, shape (N, d)
        Feature matrix
    y : numpy.ndarray, shape (N,)
        Target labels (0 or 1)
```

```
    k : int, optional (default=3)
        Number of neighbors for KNN
    n_folds : int, optional (default=6)
        Number of cross-validation folds

    Returns:
    --------
    tuple:
        - Average accuracy (float)
        - Confusion matrices for each fold (list)
        - Average runtime (float)
    """
    # Prepare for cross-validation
    N = len(X)
    fold_size = N // n_folds

    # Shuffle indices
    indices = np.arange(N)
    np.random.shuffle(indices)

    # Metrics storage
    accuracies = []
    all_conf_matrices = []
    fold_times = []

    # Cross-validation loop
    for fold_idx in range(n_folds):
        start_time = time.time()

        # Define test and training indices
        test_start = fold_idx * fold_size
        test_end = test_start + fold_size
        test_indices = indices[test_start:test_end]
        train_indices = np.concatenate((indices[:test_start], indices[test_end:

        # Split data
        X_train, y_train = X[train_indices], y[train_indices]
        X_test, y_test = X[test_indices], y[test_indices]

        # Predict for each test sample
        y_pred = []
        for x_test in X_test:
            pred = knn_classify(X_train, y_train, x_test, k=k)
            y_pred.append(pred)
        y_pred = np.array(y_pred)

        # Calculate accuracy
        acc = np.mean(y_pred == y_test)
```

```
        accuracies.append(acc)

        # Compute confusion matrix
        cm = confusion_matrix(y_test, y_pred, labels=[0,1])
        all_conf_matrices.append(cm)

        # Record runtime
        end_time = time.time()
        fold_times.append(end_time - start_time)

    # Compute average metrics
    avg_accuracy = np.mean(accuracies)
    avg_time = np.mean(fold_times)

    return avg_accuracy, all_conf_matrices, avg_time
```

```
# Experiment configuration
k_value = 3
n_folds = 6

# Perform cross-validation
avg_acc, conf_mats, runtime = cross_validation_knn_classification(
    X_wdbc, y_wdbc, k=k_value, n_folds=n_folds
)

# Print results
print("==== KNN Classification (WDBC Dataset) ====")
print(f"K = {k_value}, Number of Folds = {n_folds}")
print(f"Average Accuracy: {avg_acc:.4f}")
print(f"Average Runtime (seconds): {runtime:.4f}")

# Print confusion matrices for each fold
for i, cm in enumerate(conf_mats, 1):
    print(f"\nConfusion Matrix for Fold {i} (row=actual, col=predicted) [B(0),
```

```
==== KNN Classification (WDBC Dataset) ====
K = 3, Number of Folds = 6
Average Accuracy: 0.9379
Average Runtime (seconds): 1.5567

Confusion Matrix for Fold 1 (row=actual, col=predicted) [B(0), M(1)]:
 [[55  2]
 [ 6 31]]

Confusion Matrix for Fold 2 (row=actual, col=predicted) [B(0), M(1)]:
 [[51  3]
 [ 3 37]]

Confusion Matrix for Fold 3 (row=actual, col=predicted) [B(0), M(1)]:
 [[49  2]
 [ 6 37]]

Confusion Matrix for Fold 4 (row=actual, col=predicted) [B(0), M(1)]:
 [[62  2]
 [ 2 28]]

Confusion Matrix for Fold 5 (row=actual, col=predicted) [B(0), M(1)]:
 [[57  3]
 [ 2 32]]

Confusion Matrix for Fold 6 (row=actual, col=predicted) [B(0), M(1)]:
 [[66  0]
 [ 4 24]]
```

1. Data Preparation

- Directly downloaded from UCI Machine Learning Repository
- 569 total samples, 30 feature dimensions, binary classsification
- Removed ID column(non-predictive feature)
- Converted categorical diagnosis to binary:
    - Malignant(M) -> 1
    - Benign(B) -> 0
- Seperated features (X) and target variable (y)

---

2. Distance Metric Selection: Euclidean Distance

- Measures straight-line distance between data points
- Captures multi-ddimensional feature interactions
- Simple yet effective for medical diagnostic data

---

3. KNN Classification Mechanism

- Calculate distances to all training points
- Sort distances in ascending order
- Select k-nearest neighbors (k=3 in this case)

---

4. Cross-Validation Strategy

- 6-fold cross-validation
- Randommly shuffle and split data
- Ensures robust performance estimation
- Prevents overfitting
- provides comprehensive model evaluation

Summary

Our machine learning approach for breast cancer diagnosis achieved remarkable results through the K-Nearest Neighbors (**KNN**) algorithm. By meticulously preprocessing the data and evaluating 30 distinct features using Euclidean distance, we systematically identified similarities between data points. Through a rigorous 6-fold cross-validation process, we

comprehensively tested the model's performance, achieving an impressive 93.26% accuracy. The low misclassification rates and consistent outcomes demonstrated the significant potential of this simple yet powerful algorithm in providing diagnostic support, showcasing how intelligent computational methods can effectively augment medical decision-making.