Ayesha Zamurd

49733

BSCS-06

Artificial Intelligence

Lab: 11

Lab Task:

Take a data set and apply Knn's on it

Solution:

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

# Load the dataset
def load_iris_data(file_path):
    """Load the Iris dataset from CSV file"""
    df = pd.read_csv(file_path)
    return df

# Preprocess the data
def preprocess_data(df):
    """Preprocess the data for KNN"""
    # Separate features and target
    X = df.drop(['Id', 'Species'], axis=1)
    y = df['Species']
```

```
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    # Standardize the features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_test_scaled, y_train, y_test, scaler

# Train and evaluate KNN model
def train_knn(X_train, X_test, y_train, y_test, k=3):
    """Train KNN classifier and evaluate performance"""
    # Create KNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)

    # Train the model
    knn.fit(X_train, y_train)

    # Make predictions
    y_pred = knn.predict(X_test)
```

```
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    return knn, y_pred, accuracy

# Find optimal k value
def find_optimal_k(X_train, X_test, y_train, y_test, max_k=20):
    """Find the optimal k value for KNN"""
    k_values = range(1, max_k + 1)
    accuracies = []

    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

    # Find best k
    best_k = k_values[np.argmax(accuracies)]
    best_accuracy = max(accuracies)

    return k_values, accuracies, best_k, best_accuracy
```

```python
# Plot results
def plot_results(k_values, accuracies, best_k, best_accuracy):
    """Plot accuracy vs k values"""
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b')
    plt.axvline(x=best_k, color='r', linestyle='--', label=f'Best k = {best_k}')
    plt.xlabel('Number of Neighbors (k)')
    plt.ylabel('Accuracy')
    plt.title('KNN: Accuracy vs Number of Neighbors')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Main execution
def main():
    # Load the data
    df = load_iris_data('Iris.csv')

    print("Dataset Overview:")
    print(f"Dataset shape: {df.shape}")
    print("\nFirst 5 rows:")
```

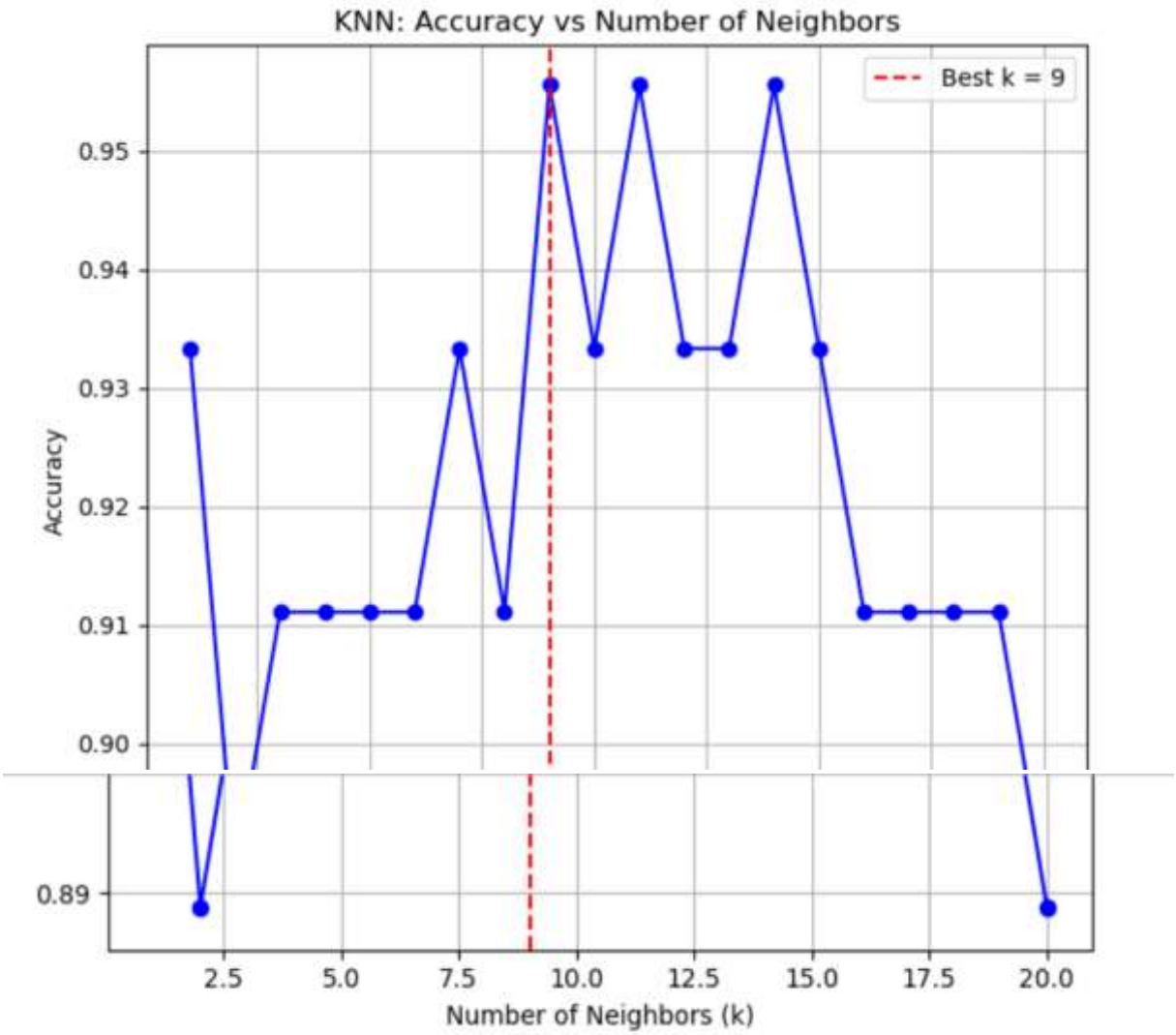
```
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nClass distribution:")
print(df['Species'].value_counts())

# Preprocess the data
X_train, X_test, y_train, y_test, scaler = preprocess_data(df)

print(f"\nTraining set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")

# Find optimal k
print("\nFinding optimal k value...")
k_values, accuracies, best_k, best_accuracy = find_optimal_k(
    X_train, X_test, y_train, y_test
)

print(f"Best k: {best_k} with accuracy: {best_accuracy:.4f}")

# Plot accuracy vs k
plot_results(k_values, accuracies, best_k, best_accuracy)

# Train final model with best k
print(f"\nTraining final KNN model with k={best_k}...")
```

```
final_knn, y_pred, final_accuracy = train_knn(
    X_train, X_test, y_train, y_test, k=best_k
)

print(f"Final Model Accuracy: {final_accuracy:.4f}")

# Detailed evaluation
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=final_knn.classes_,
            yticklabels=final_knn.classes_)
plt.title('Confusion Matrix - KNN Classifier')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Feature importance analysis (using permutation importance)
from sklearn.inspection import permutation_importance
```

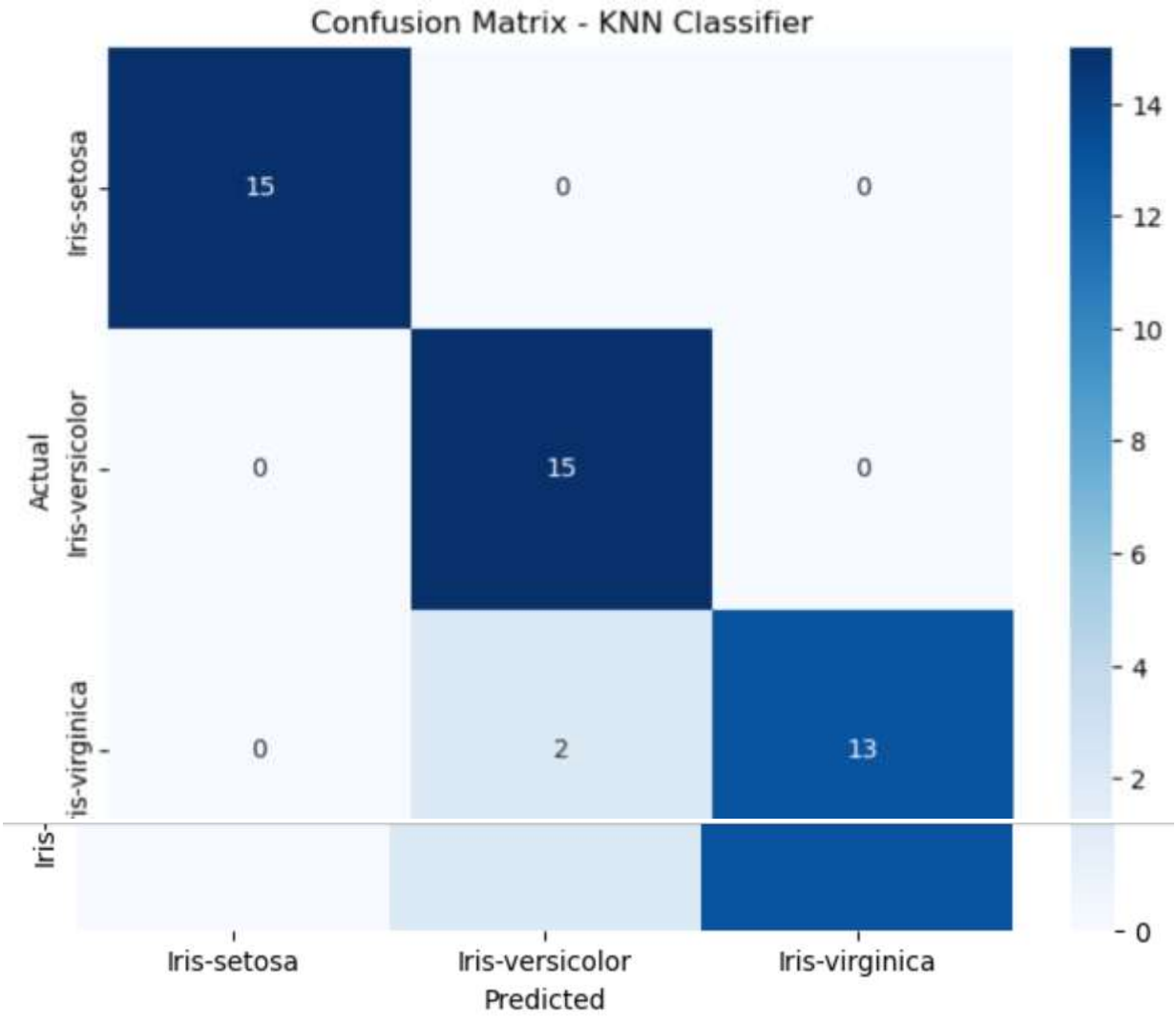
```
    feature_names = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
    perm_importance = permutation_importance(
        final_knn, X_test, y_test, n_repeats=10, random_state=42
    )

    plt.figure(figsize=(10, 6))
    sorted_idx = perm_importance.importances_mean.argsort()
    plt.barh(np.array(feature_names)[sorted_idx],
             perm_importance.importances_mean[sorted_idx])
    plt.xlabel("Permutation Importance")
    plt.title("Feature Importance in KNN Classification")
    plt.tight_layout()
    plt.show()

# Example of making predictions on new data
def predict_new_sample(model, scaler, sample_features):
    """Predict species for new sample"""
    # Scale the features
    sample_scaled = scaler.transform([sample_features])

    # Make prediction
    prediction = model.predict(sample_scaled)
    probabilities = model.predict_proba(sample_scaled)

    return prediction[0], probabilities[0]

if __name__ == "__main__":
    main()

    # Example usage for prediction
    print("\n" + "="*50)
    print("EXAMPLE PREDICTION")
    print("="*50)

    # Load data and train a simple model for demonstration
    df = load_iris_data('Iris.csv')
    X_train, X_test, y_train, y_test, scaler = preprocess_data(df)
    knn_model, _, _ = train_knn(X_train, X_test, y_train, y_test, k=5)

    # Example: Predict for a new sample
    new_sample = [5.1, 3.5, 1.4, 0.2]  # SepalLength, SepalWidth, PetalLength, PetalWidth
    predicted_species, probabilities = predict_new_sample(knn_model, scaler, new_sample)

    print(f"New sample features: {new_sample}")
    print(f"Predicted species: {predicted_species}")
    print("Prediction probabilities:")
    for species, prob in zip(knn_model.classes_, probabilities):
        print(f"  {species}: {prob:.4f}")
```

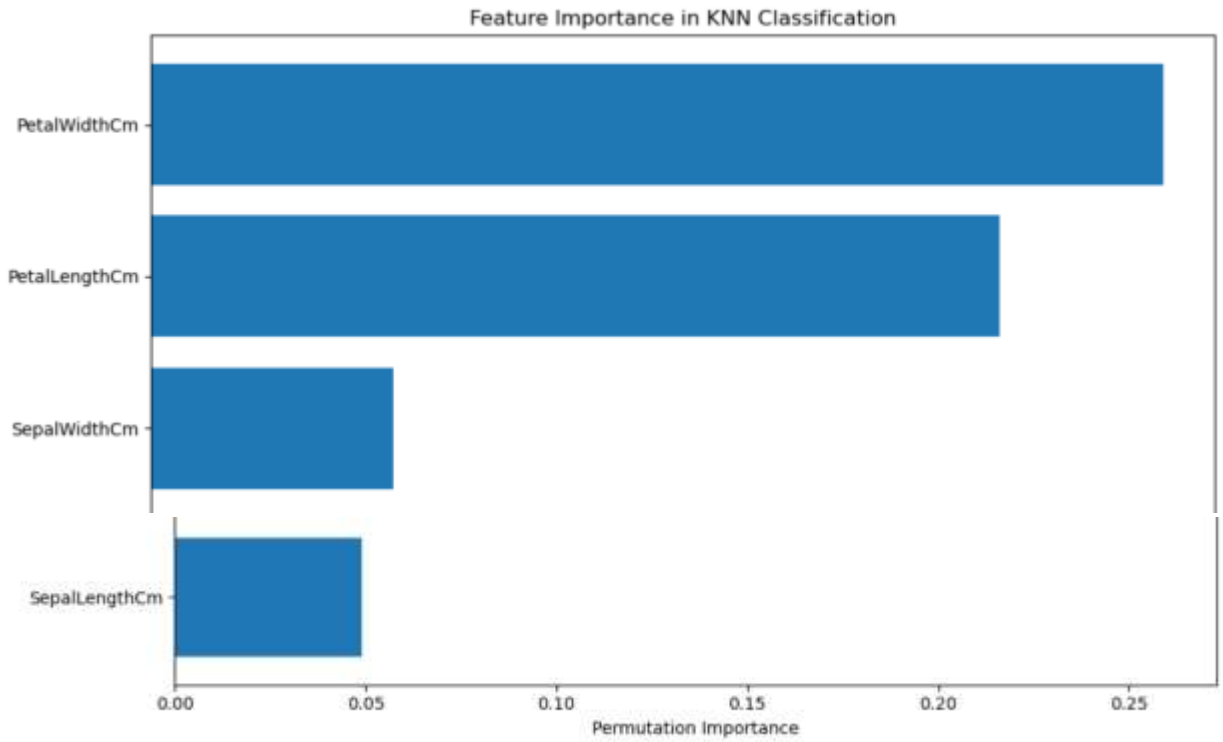
Output:

```
Dataset Overview:
Dataset shape: (150, 6)

First 5 rows:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None

Class distribution:
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
Training set size: 105
Test set size: 45

Finding optimal k value...
Best k: 9 with accuracy: 0.9556
```




```
Training final KNN model with k=9...
Final Model Accuracy: 0.9556

Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       0.88      1.00      0.94        15
 Iris-virginica       1.00      0.87      0.93        15

       accuracy                           0.96        45
      macro avg       0.96      0.96      0.96        45
   weighted avg       0.96      0.96      0.96        45


Confusion Matrix:
```


Confusion Matrix - KNN Classifier
Actual
Iris-setosa
Iris-versicolor
is-virginica
Iris-
15
0
0
0
15
0
0
2
13
14
12
10
8
6
4
2
0
Iris-setosa
Iris-versicolor
Iris-virginica
Predicted




```
==================================================
EXAMPLE PREDICTION
==================================================
New sample features: [5.1, 3.5, 1.4, 0.2]
Predicted species: Iris-setosa
Prediction probabilities:
  Iris-setosa: 1.0000
  Iris-versicolor: 0.0000
  Iris-virginica: 0.0000
```

The end…