MACHINE LEARNING PRACTICAL JOURNAL

Submitted By:

Tushar Mukesh Pokhariyal

Roll No:

31031523043

MSC Computer Science Part-II

Department of Computer Science

S.K. Somaiya College,

Somaiya Vidyavihar University

Table of Contents
Machine Learning Lab Manual

1-Linear Regression (Diabetes Dataset)

November 11, 2024

Practical 1: Implement Linear Regression (Diabetes Dataset)

[2]:
```
# Import Dependencies

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model,metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns
```

[3]:
```
# Load the diabetes dataset

diabetes=datasets.load_diabetes()
```

[4]:
```
# X - feature vectors
# y - Target values

X=diabetes.data
y=diabetes.target
```

[5]:
```
# splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)
```

[6]:
```
# Create linear regression objest

lin_reg=linear_model.LinearRegression()
```

[7]:
```
# Train the model using trai and test data

lin_reg.fit(X_train,y_train)
```

[7]:
```
LinearRegression()
```

[8]:
```
# Predict values for X_test data
```

```
predicted = lin_reg.predict(X_test)
```

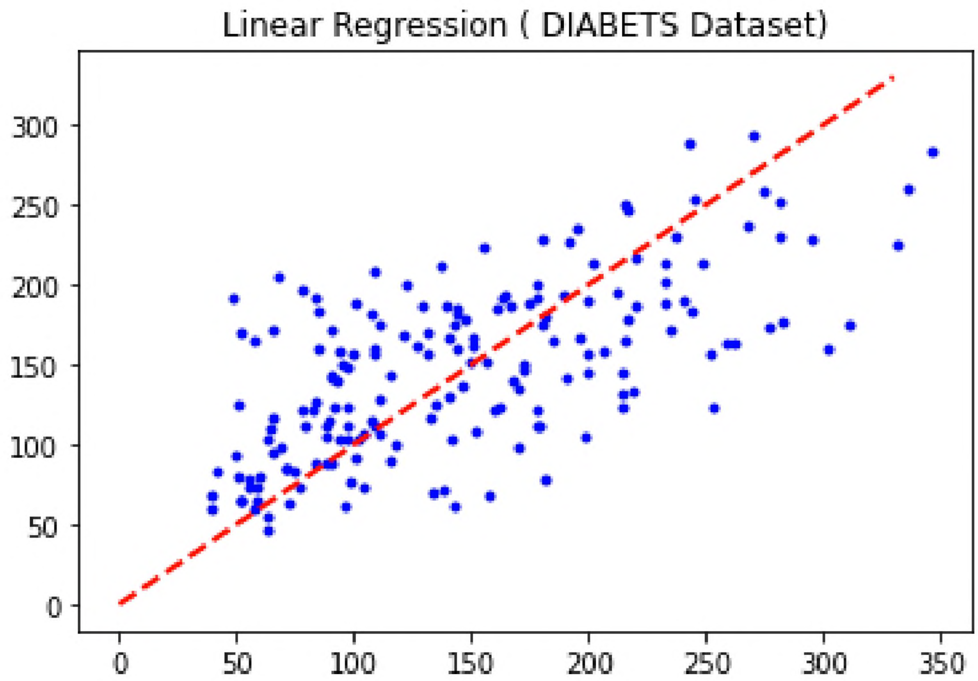
[9]:
```
# Regression coefficients

print('\n Coefficients are:\n',lin_reg.coef_)

# Intecept

print('\nIntercept : ',lin_reg.intercept_)

# variance score: 1 means perfect prediction

print('Variance score: ',lin_reg.score(X_test, y_test))
```

```
 Coefficients are:
 [ -59.73663337 -215.62170919  599.92621335  291.96724002 -829.65206295
  544.63994617  164.85191153  224.2392528   768.94426062   70.84982207]

Intercept :  152.89009028286725
Variance score:  0.4160439011127657
```

[10]:
```
# Mean Squared Erroe

print("Mean squared error: %.2f\n"
      % mean_squared_error(y_test, predicted))

# Original data of X_test

expected = y_test
```

```
Mean squared error: 2962.93
```

[11]:
```
# Plot a graph for expected and predicted values

plt.title('Linear Regression ( DIABETS Dataset)')
plt.scatter(expected,predicted,c='b',marker='.',s=36)
plt.plot(np.linspace(0, 330, 100),np.linspace(0, 330, 100), '--r', linewidth=2)

plt.show()
```


Linear Regression (DIABETS Dataset)
0
50
100
150
200
250
300
350

2-Logistic Regression

November 11, 2024

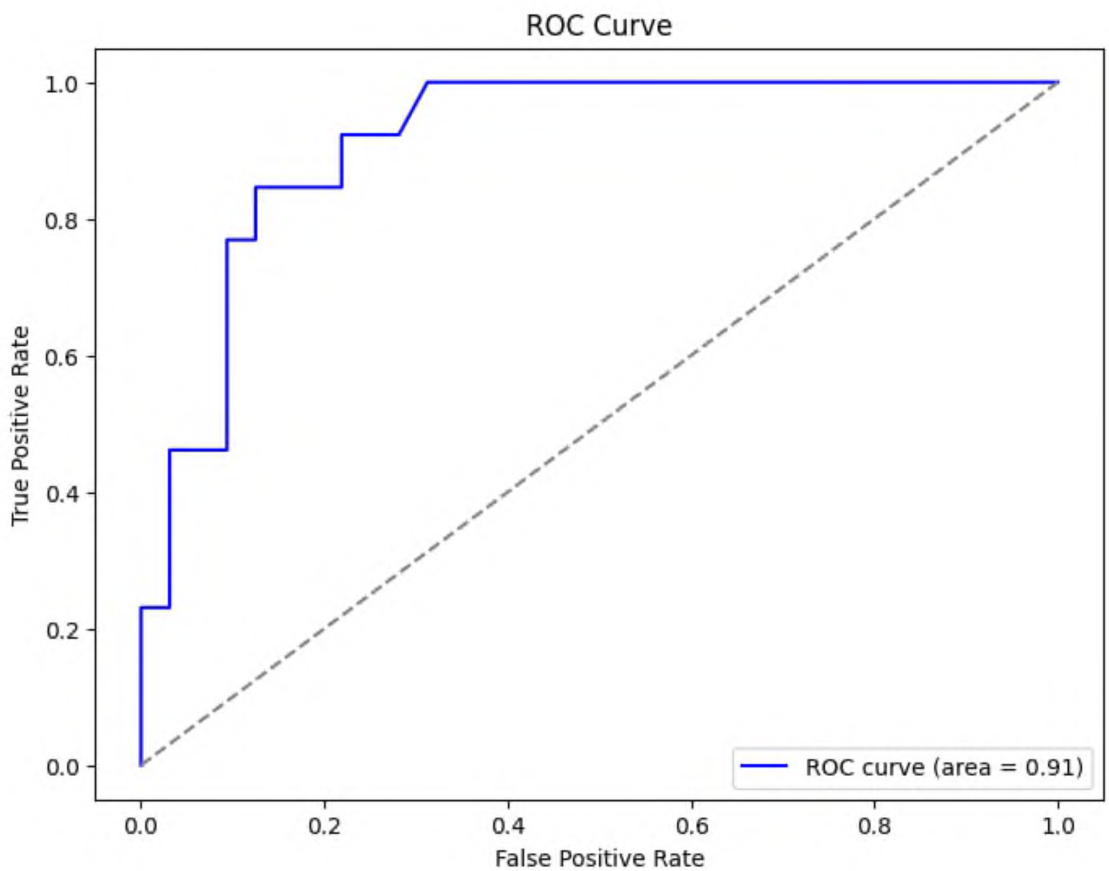
[2]:
```
# Using LogisticRegression()
```

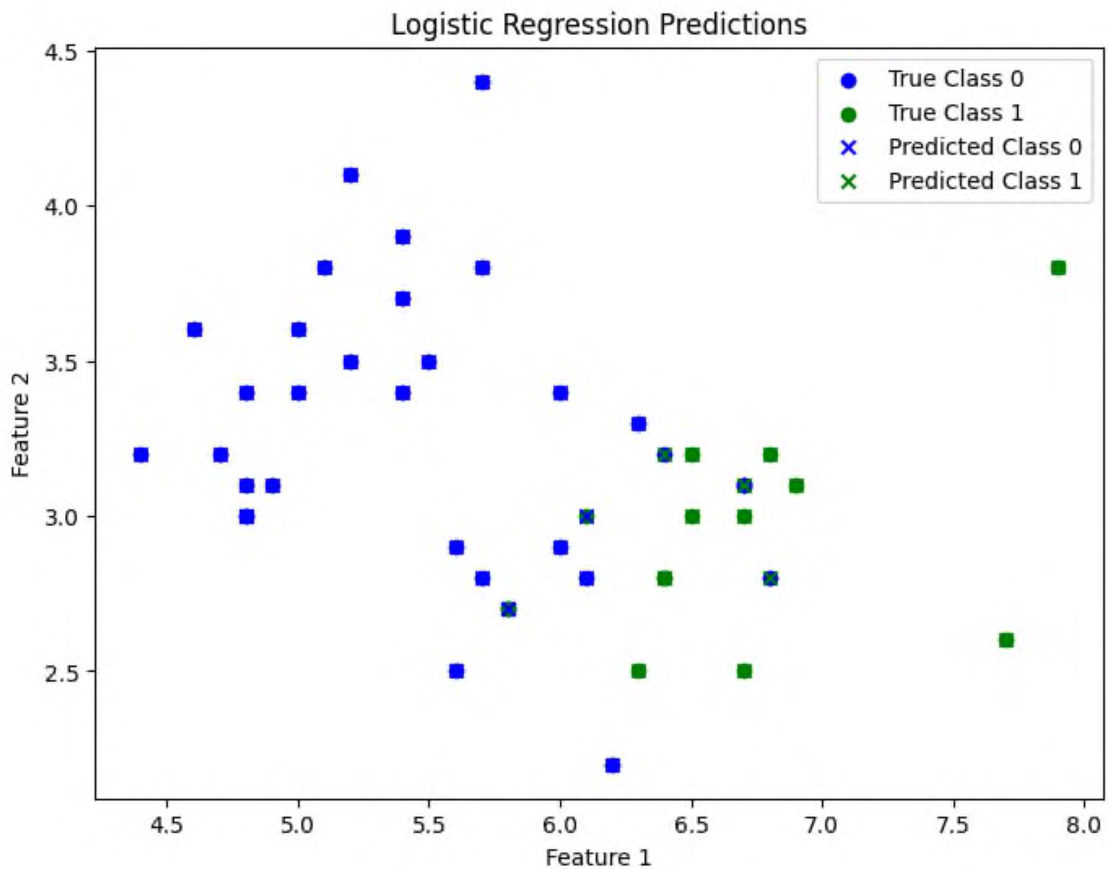
[3]:
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪roc_auc_score, roc_curve, confusion_matrix

# Load Iris dataset
data = load_iris()
X = data.data
y = data.target

# For binary classification, we only select the "Iris-Virginica" class (class 2)
# Convert it to a binary problem (1 if Virginica, 0 otherwise)
y = (y == 2).astype(int)

# Use only two features for easier 2D visualization
X = X[:, :2]  # Select the first two features for simplicity

# Split 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)

# Initialize and train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probability estimates for␣
↪ROC curve

# Performance metrics
```

```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)

print("Performance Metrics:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"ROC AUC: {roc_auc:.2f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Plotting ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# Plotting the predictions
plt.figure(figsize=(8, 6))

# Plot true labels in the test set
plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], color='blue',␣
 ↪label='True Class 0', marker='o')
plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1],␣
 ↪color='green', label='True Class 1', marker='o')

# Overlay predicted labels in the test set
plt.scatter(X_test[y_pred == 0][:, 0], X_test[y_pred == 0][:, 1], color='blue',␣
 ↪marker='x', label='Predicted Class 0')
plt.scatter(X_test[y_pred == 1][:, 0], X_test[y_pred == 1][:, 1],␣
 ↪color='green', marker='x', label='Predicted Class 1')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Logistic Regression Predictions')
plt.legend(loc="best")
```

```
plt.show()
```

```
Performance Metrics:
Accuracy: 0.87
Precision: 0.73
Recall: 0.85
ROC AUC: 0.91

Confusion Matrix:
[[28  4]
 [ 2 11]]
```




Logistic Regression Predictions
True Class 0
True Class 1
Predicted Class 0
Predicted Class 1
Feature 1
Feature 2
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.5
3.0
3.5
4.0
4.5

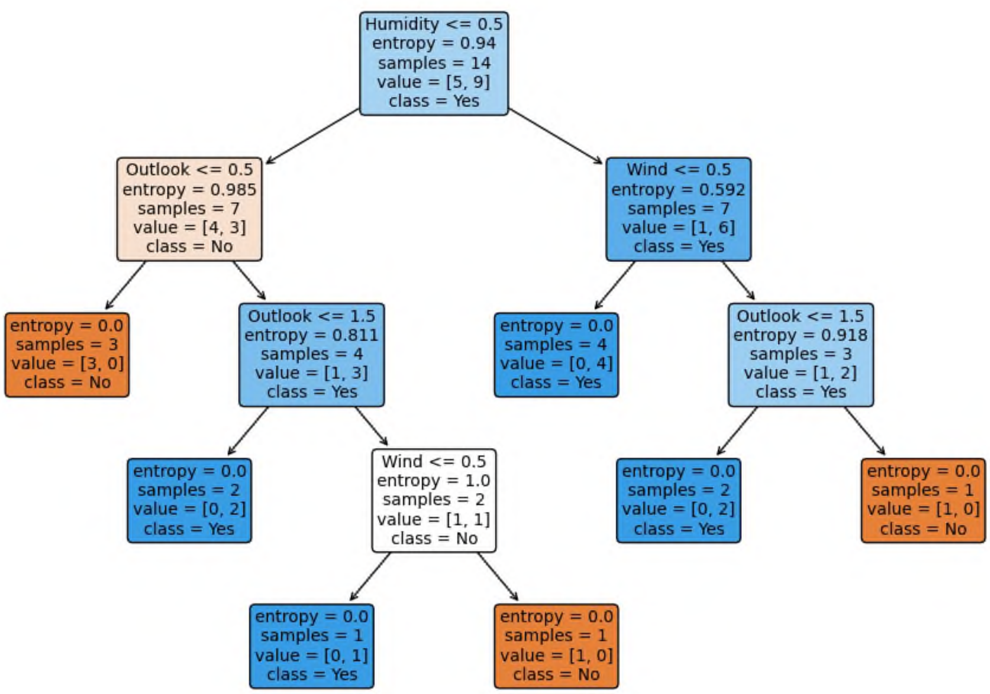
[ ]:

3-Decsion Tree Classifier

November 11, 2024

[7]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Data Preparation
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy',␣
 ↪'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast',␣
 ↪'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',␣
 ↪'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',␣
 ↪'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',␣
 ↪'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',␣
 ↪'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

# Convert categorical data to numeric using factorization
df_encoded = df.apply(lambda x: pd.factorize(x)[0])

# Separate features and target
X = df_encoded.drop('PlayTennis', axis=1)
y = df_encoded['PlayTennis']

# Build the Decision Tree using sklearn
clf = DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X, y)

# Plot the Decision Tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf,
```

```python
                feature_names=df.columns[:-1].tolist(),  # Convert feature names␣
 ↪to list
                class_names=['No', 'Yes'],
                filled=True,
                rounded=True,
                fontsize=10)
plt.show()

# A function to predict the outcome using the decision tree
def predict(query, clf, feature_names):
    query_encoded = [pd.factorize(df[feature])[0][df[feature] ==␣
 ↪query[feature]].tolist()[0] for feature in feature_names]
    prediction = clf.predict([query_encoded])
    return 'Yes' if prediction == 1 else 'No'

# Sample query
query = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High', 'Wind':␣
 ↪'Strong'}
prediction = predict(query, clf, df.columns[:-1].tolist())
print(f"Prediction for {query}: {prediction}")
```




```
Prediction for {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High',
'Wind': 'Strong'}: No
```

```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:464: UserWarning: X does not have valid feature names,
but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

[8]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Define the data
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy',␣
 ↪'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast',␣
 ↪'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',␣
 ↪'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',␣
 ↪'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',␣
 ↪'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',␣
 ↪'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Create a DataFrame from the data
df = pd.DataFrame(data)

# Split the data into features (X) and target (y)
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']

# Use OneHotEncoder to encode categorical features
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_encoded = encoder.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.
 ↪2, random_state=42)

# Create a Decision Tree classifier
clf = DecisionTreeClassifier()

# Fit the classifier to the training data
clf.fit(X_train, y_train)
```

```python
# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Sample input data for prediction
sample_input = {
    'Outlook': ['Sunny'],
    'Temperature': ['Cool'],
    'Humidity': ['High'],
    'Wind': ['Weak']
}

# Encode the sample input using the same encoder
sample_input_encoded = encoder.transform(pd.DataFrame(sample_input))

# Make predictions for the sample input
sample_prediction = clf.predict(sample_input_encoded)

# Print the prediction
if sample_prediction[0] == 'No':
    print("Prediction: No, don't play tennis.")
else:
    print("Prediction: Yes, play tennis.")
```

```
Accuracy: 1.0
Prediction: No, don't play tennis.
```

```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[ ]:

```python

```

4-k-Nearest Neighbour

November 11, 2024

Case Study: Iris Flower Classification using k-Nearest Neighbors (k-NN)

Background: You are a data scientist working on a project to develop a machine learning model for classifying iris flowers into three species based on their sepal length and sepal width. The Iris dataset is a well-known dataset in the field of machine learning, and you decide to implement the k-Nearest Neighbors (k-NN) algorithm to perform this classification task.

Dataset: The Iris dataset contains measurements of four features (sepal length, sepal width, petal length, and petal width) for 150 iris flowers, each belonging to one of three species: Setosa, Versicolor, and Virginica.

Tasks for you:

Your task is to create a Python program that accomplishes the following:

Load the Iris dataset and select only the first two features (sepal length and sepal width) for simplicity.

Split the dataset into a training set and a testing set, where 70% of the data is used for training and 30% for testing. Use a random seed of 42 for consistency.

Initialize a k-NN classifier with a chosen value of k (you can experiment with different values of k).

Train the k-NN classifier using the training data.

Make predictions on the test data using the trained classifier.

Evaluate the performance of the classifier by displaying the confusion matrix and classification report (precision, recall, F1-score, etc.).

Visualize the dataset and decision boundaries using a scatter plot. Plot the training data, testing data, and decision boundaries on the same graph.

Add a hard-coded new sample with sepal length and sepal width values for testing purposes. Classify and visualize this new sample using the k-NN model.

Note: You can use libraries such as NumPy, pandas, scikit-learn, and Matplotlib to complete the tasks.

Question:

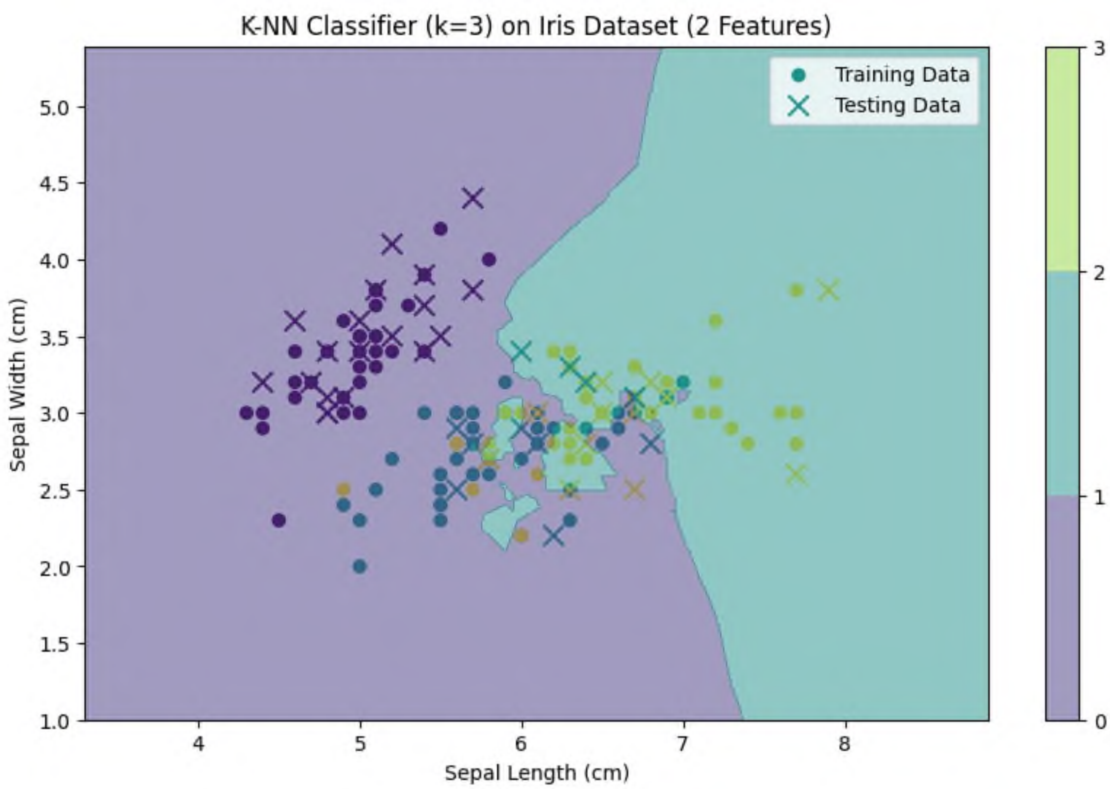
After completing the tasks above, answer the following question:

Explain the significance of the k value in the k-NN algorithm and how changing this value can impact the model's performance.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Select only the first two features (sepal length and␣
↪sepal width)
y = iris.target

# Split the dataset 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)

# Initialize k-NN classifier (You can adjust 'n_neighbors' for different values␣
↪of k)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Fit the classifier to the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier's performance
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualize the dataset and decision boundaries
plt.figure(figsize=(10, 6))

# Plot the training data points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis',␣
↪label='Training Data')

# Plot the testing data points
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='x',␣
↪s=100, label='Testing Data')

# Plot decision boundaries
```

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.
 ↪01))
Z = knn_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap='viridis', alpha=0.5, levels=range(4))
plt.colorbar()

plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title(f'K-NN Classifier (k={k}) on Iris Dataset (2 Features)')
plt.legend()
plt.show()
```

```
Confusion Matrix:
[[19  0  0]
 [ 0  7  6]
 [ 0  5  8]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.58      0.54      0.56        13
           2       0.57      0.62      0.59        13

    accuracy                           0.76        45
   macro avg       0.72      0.72      0.72        45
weighted avg       0.76      0.76      0.76        45
```



K-NN Classifier (k=3) on Iris Dataset (2 Features)
Training Data
Testing Data
Sepal Width (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8
Sepal Length (cm)
3
2
1
0


```
[ ]:
```

```
[ ]:
```

5-K-means

November 11, 2024

1 Implementing K-means Algorithm

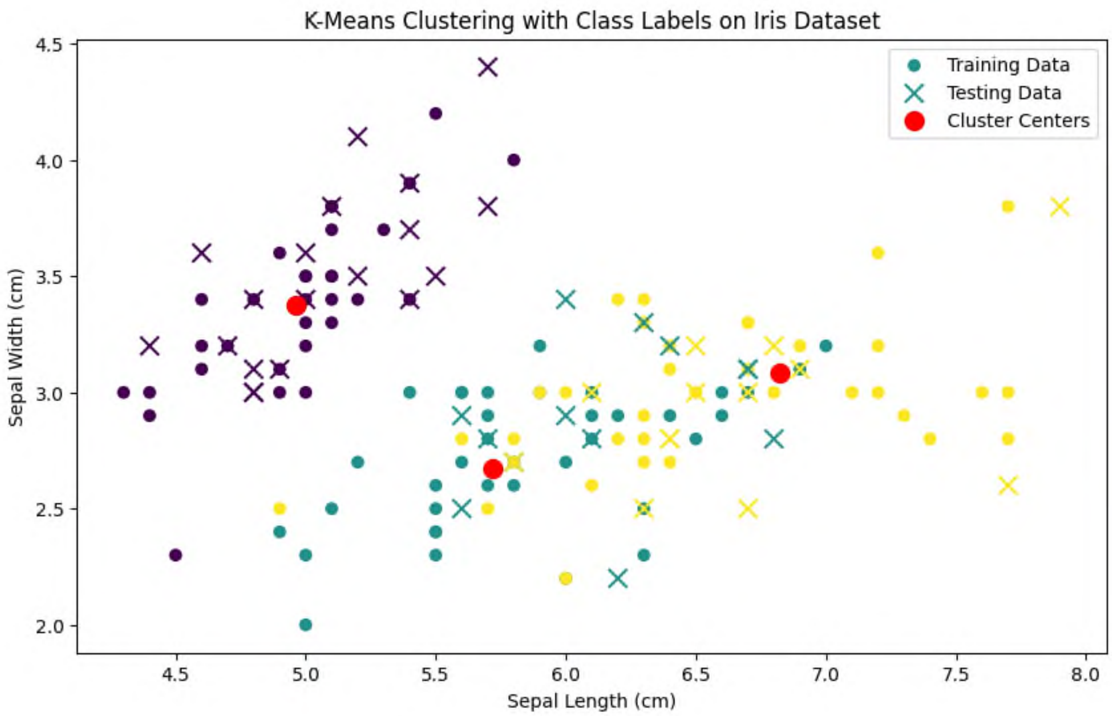
[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Select only the first two features (sepal length and␣
 ↪sepal width)
y = iris.target

# Split the dataset 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)

# Initialize K-Means clustering with the number of clusters equal to the number␣
 ↪of classes
n_clusters = len(np.unique(y))
kmeans = KMeans(n_clusters=n_clusters, random_state=42)

# Fit K-Means clustering to the training data
kmeans.fit(X_train)

# Assign cluster labels to data points in the test set
cluster_labels = kmeans.predict(X_test)

# Assign class labels to clusters based on the most frequent class label in␣
 ↪each cluster
cluster_class_labels = []
for i in range(n_clusters):
    cluster_indices = np.where(cluster_labels == i)[0]
    cluster_class_labels.append(np.bincount(y_test[cluster_indices]).argmax())
```

```python
# Assign cluster class labels to data points in the test set
y_pred = np.array([cluster_class_labels[cluster_labels[i]] for i in␣
 ↪range(len(X_test))])

# Evaluate the classifier's performance
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualize the dataset and cluster centers
plt.figure(figsize=(10, 6))

# Plot the training data points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis',␣
 ↪label='Training Data')

# Plot the testing data points
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='x',␣
 ↪s=100, label='Testing Data')

# Plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪c='red', marker='o', s=100, label='Cluster Centers')

plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('K-Means Clustering with Class Labels on Iris Dataset')
plt.legend()
plt.show()
```

```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
Confusion Matrix:
[[19  0  0]
 [ 0  8  5]
 [ 0  3 10]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.73      0.62      0.67        13
```

```
           2       0.67      0.77      0.71        13

    accuracy                           0.82        45
   macro avg       0.80      0.79      0.79        45
weighted avg       0.82      0.82      0.82        45
```



[ ]:

6-Random Forest

November 11, 2024

[ ]:

```

```

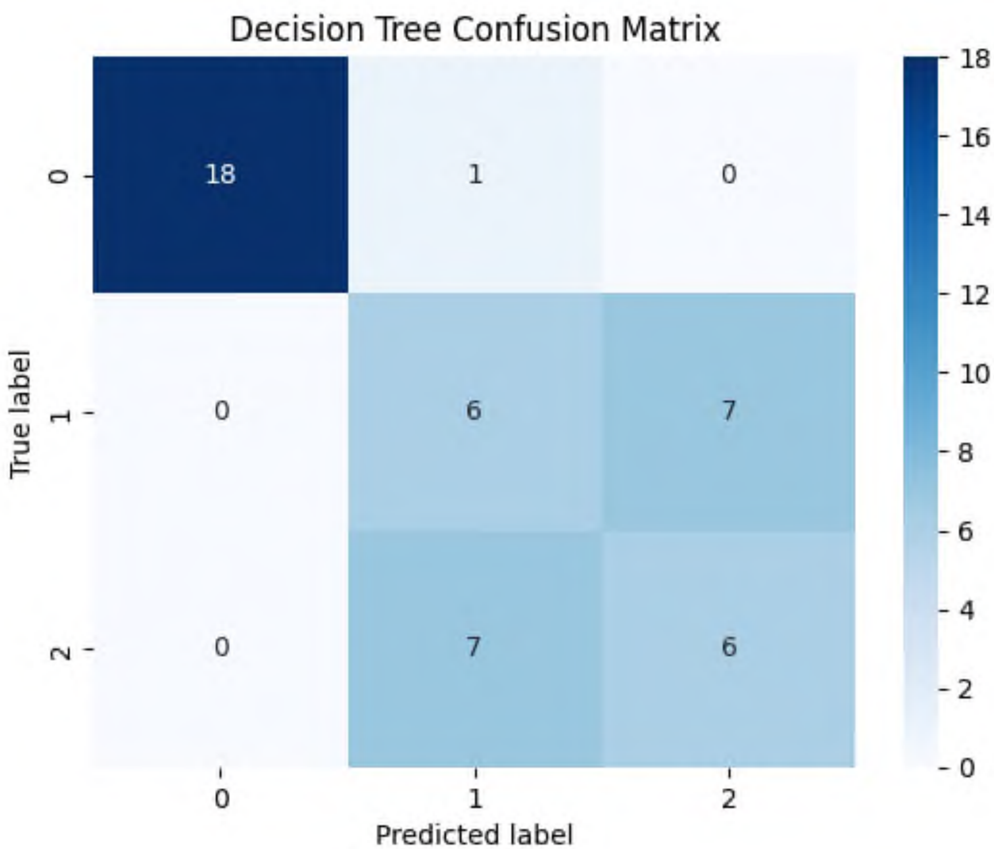
[2]:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the dataset and choose only two features for visualization (we'll use␣
 ↪the first two)
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names).iloc[:, :2]  # Only␣
 ↪first two features
y = pd.DataFrame(iris.target, columns=['species'])

# Split the dataset 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)

# Function to plot decision boundary
def plot_decision_boundary(clf, X, y, title):
    # Create a meshgrid
    x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
    y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict for the entire grid
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
```

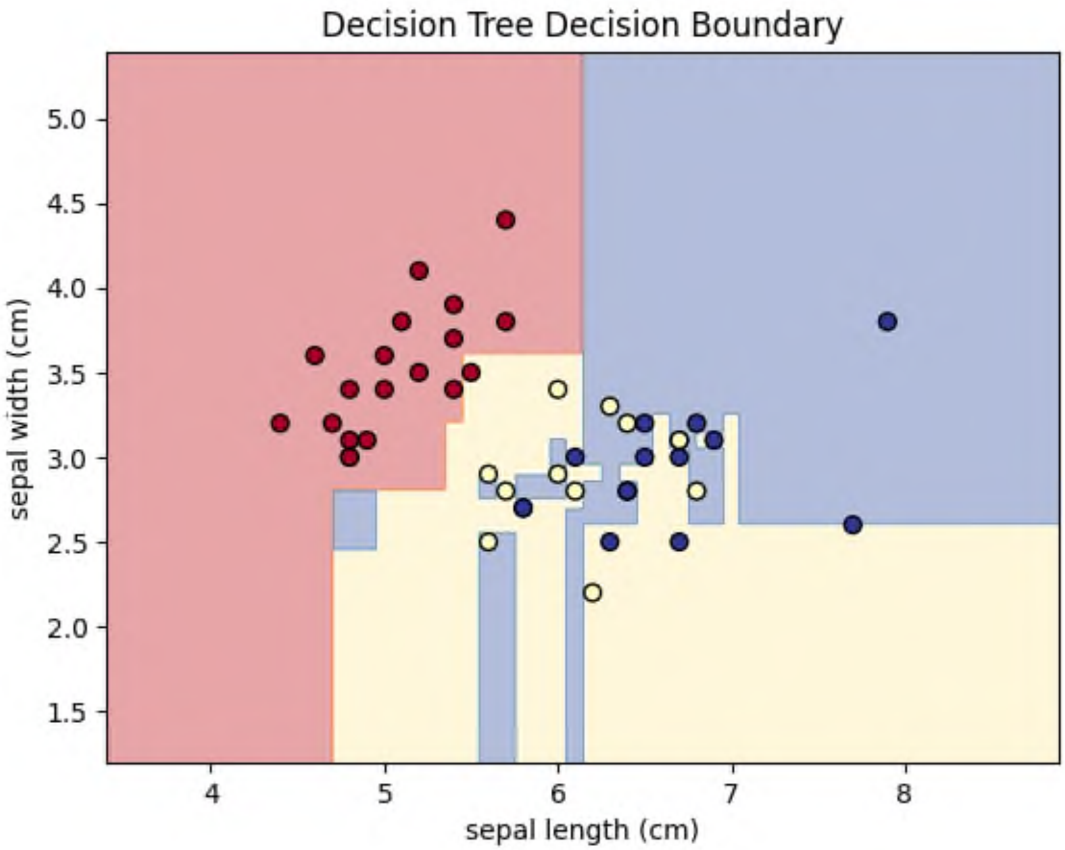
```
    # Plot the contour and training points
    plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y.values.ravel(), s=40,␣
 ↪edgecolor='k', cmap=plt.cm.RdYlBu)
    plt.title(title)
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
    plt.show()

# Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions with Decision Tree
dt_predictions = dt_model.predict(X_test)

# Decision Tree Accuracy and Confusion Matrix
dt_accuracy = accuracy_score(y_test, dt_predictions)
dt_confusion_matrix = confusion_matrix(y_test, dt_predictions)

print(f"Decision Tree Accuracy: {dt_accuracy}")
print("Decision Tree Classification Report:")
print(classification_report(y_test, dt_predictions))

# Plot Confusion Matrix for Decision Tree
sns.heatmap(dt_confusion_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Decision Tree Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Plot Decision Boundary for Decision Tree
plot_decision_boundary(dt_model, X_test, y_test, "Decision Tree Decision␣
 ↪Boundary")

# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train.values.ravel())

# Make predictions with Random Forest
rf_predictions = rf_model.predict(X_test)

# Random Forest Accuracy and Confusion Matrix
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_confusion_matrix = confusion_matrix(y_test, rf_predictions)

print(f"Random Forest Accuracy: {rf_accuracy}")
```

```python
print("Random Forest Classification Report:")
print(classification_report(y_test, rf_predictions))

# Plot Confusion Matrix for Random Forest
sns.heatmap(rf_confusion_matrix, annot=True, fmt='d', cmap='Greens')
plt.title('Random Forest Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Plot Decision Boundary for Random Forest
plot_decision_boundary(rf_model, X_test, y_test, "Random Forest Decision␣
↪Boundary")
```

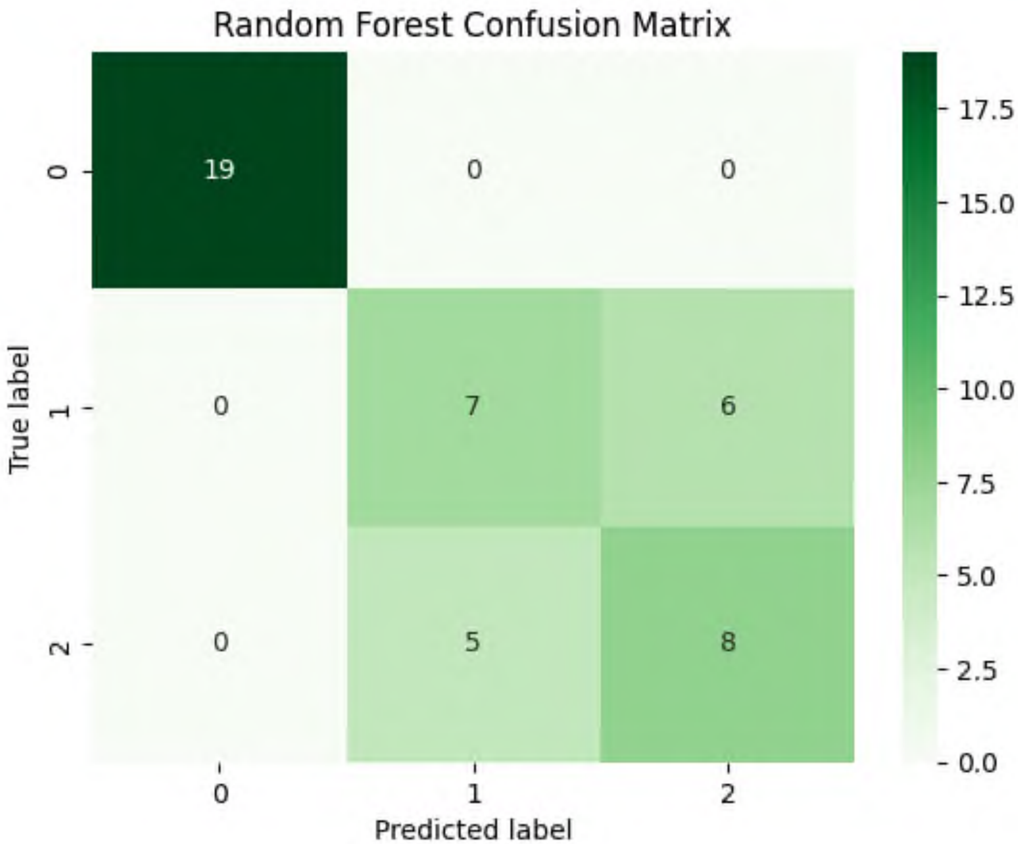
```
Decision Tree Accuracy: 0.6666666666666666
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        19
           1       0.43      0.46      0.44        13
           2       0.46      0.46      0.46        13

    accuracy                           0.67        45
   macro avg       0.63      0.62      0.63        45
weighted avg       0.68      0.67      0.67        45
```



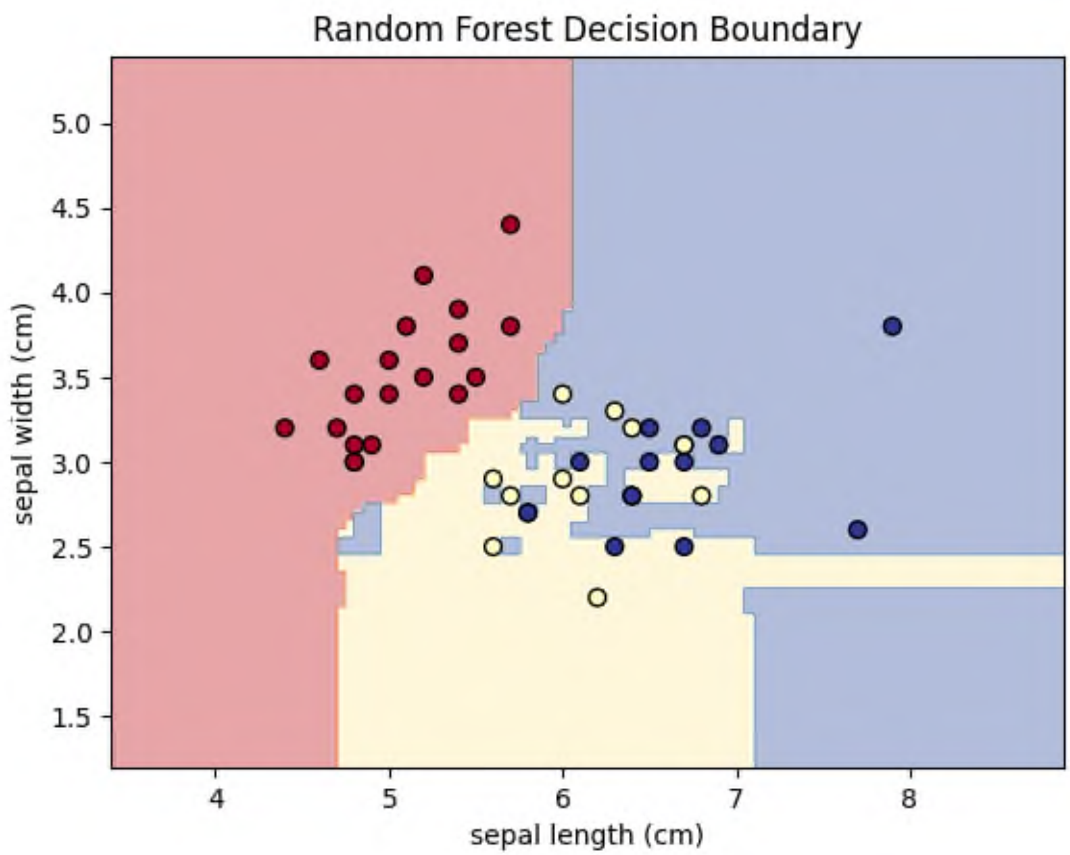

```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:464: UserWarning: X does not have valid feature names,
but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```




```
Random Forest Accuracy: 0.7555555555555555
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.58      0.54      0.56        13
           2       0.57      0.62      0.59        13

    accuracy                           0.76        45
   macro avg       0.72      0.72      0.72        45
weighted avg       0.76      0.76      0.76        45
```




```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:464: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```


Random Forest Decision Boundary
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8

[ ]:

[ ]:

[ ]:

7-Clustering

November 11, 2024

[ ]:

```

```

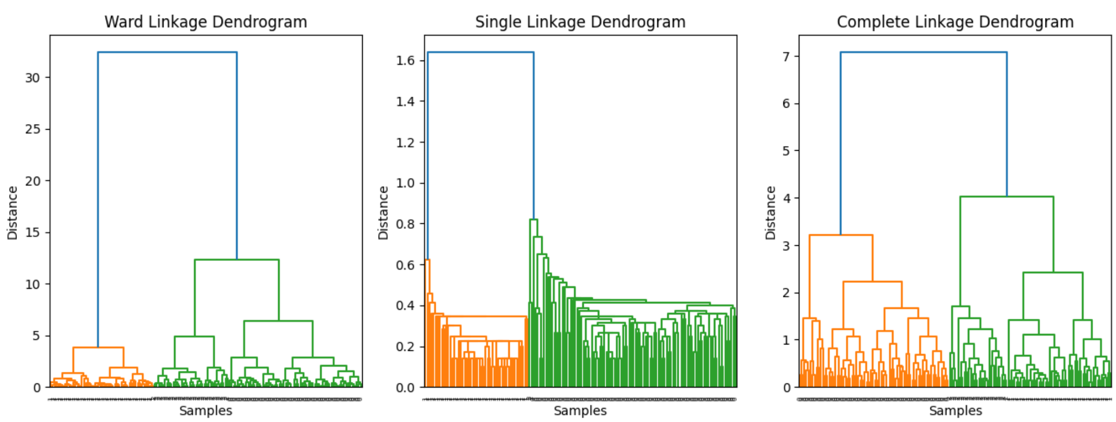
[1]:

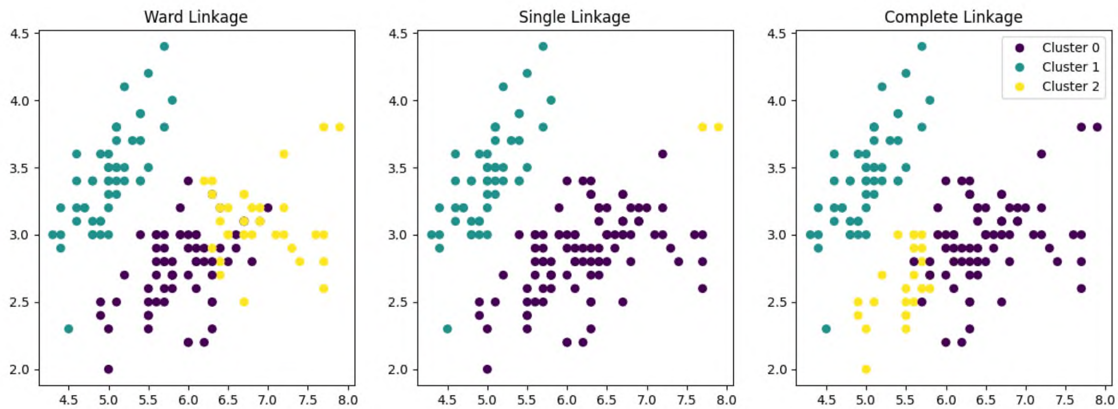
```
import pandas as pd
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

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

# Step 1: Hierarchical Clustering with Different Linkage Methods and Draw␣
↪Dendrograms
n_clusters = 3  # Number of clusters
linkage_methods = ['ward', 'single', 'complete']  # Different linkage methods
cluster_labels = []

# Define figure and axes for dendrograms
plt.figure(figsize=(15, 5))
dendrogram_axes = []

for i, linkage_method in enumerate(linkage_methods):
    labels = AgglomerativeClustering(n_clusters=n_clusters,␣
↪linkage=linkage_method).fit_predict(X)
    cluster_labels.append(labels)

    # Create a dendrogram for the current linkage method
    dendrogram_data = linkage(X, method=linkage_method)
    dendrogram_axes.append(plt.subplot(1, len(linkage_methods), i + 1))
    dendrogram(dendrogram_data, orientation='top', labels=labels)
```

```python
    plt.title(f"{linkage_method.capitalize()} Linkage Dendrogram")
    plt.xlabel('Samples')
    plt.ylabel('Distance')

# Plot the clustering results for different linkage methods
plt.figure(figsize=(15, 5))
for i, linkage_method in enumerate(linkage_methods):
    plt.subplot(1, len(linkage_methods), i + 1)
    scatter = plt.scatter(X[:, 0], X[:, 1], c=cluster_labels[i], cmap='viridis',
                          label=f'Clusters ({linkage_method.capitalize()}␣
 ↪Linkage)')
    plt.title(f"{linkage_method.capitalize()} Linkage")

# Add a legend to the scatter plots
plt.legend(handles=scatter.legend_elements()[0], labels=[f'Cluster {i}' for i␣
 ↪in range(n_clusters)])

# Step 2: Feature Engineering (Using cluster assignment as a feature)
X_with_cluster = np.column_stack((X, cluster_labels[-1]))  # Using complete␣
 ↪linkage

# Step 3: Classification
X_train, X_test, y_train, y_test = train_test_split(X_with_cluster, y,␣
 ↪test_size=0.2, random_state=42)
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train, y_train)

# Step 4: Prediction
y_pred = classifier.predict(X_test)

# Step 5: Test Score and Confusion Matrix
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Generate classification report with zero_division parameter
classification_rep = classification_report(y_test, y_pred, zero_division=0)

# Print cluster descriptions
cluster_descriptions = {
    'ward': 'Clusters based on Ward linkage interpretation.',
    'single': 'Clusters based on Single linkage interpretation.',
    'complete': 'Clusters based on Complete linkage interpretation.'
}

for method in linkage_methods:
    print(f"Cluster Descriptions ({method.capitalize()} Linkage):")
```

```
    print(cluster_descriptions[method.lower()])  # Convert to lowercase for␣
 ↪dictionary access

# Print accuracy, confusion matrix, and classification report
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)

plt.show()
```

```
Cluster Descriptions (Ward Linkage):
Clusters based on Ward linkage interpretation.
Cluster Descriptions (Single Linkage):
Clusters based on Single linkage interpretation.
Cluster Descriptions (Complete Linkage):
Clusters based on Complete linkage interpretation.
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       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
```





Ward Linkage
Single Linkage
Complete Linkage
Cluster 0
Cluster 1
Cluster 2

[ ]:

8-naïve Bayesian Classifier

November 11, 2024

[ ]:

```
```

[2]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Example dataset
documents = [
    "This is a positive document",
    "Negative sentiment in this text",
    "A very positive review",
    "Review with a negative tone",
    "Neutral document here"
]

labels = ["positive", "negative", "positive", "negative", "neutral"]

# Create a vectorizer to convert text to numerical features
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,
 ↪random_state=42)

# Create and train a Multinomial Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Predict the classes for the test data
y_pred = classifier.predict(X_test)

# Calculate and print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

```
# Print actual and predicted classes for the test data
print("Test Data:")
for actual, predicted in zip(y_test, y_pred):
    print(f"Actual: {actual}, Predicted: {predicted}")


print(f"\nAccuracy: {accuracy:.2f}")
print(report)
```

```
Test Data:
Actual: negative, Predicted: negative

Accuracy: 1.00
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1
```

[ ]:

```

```

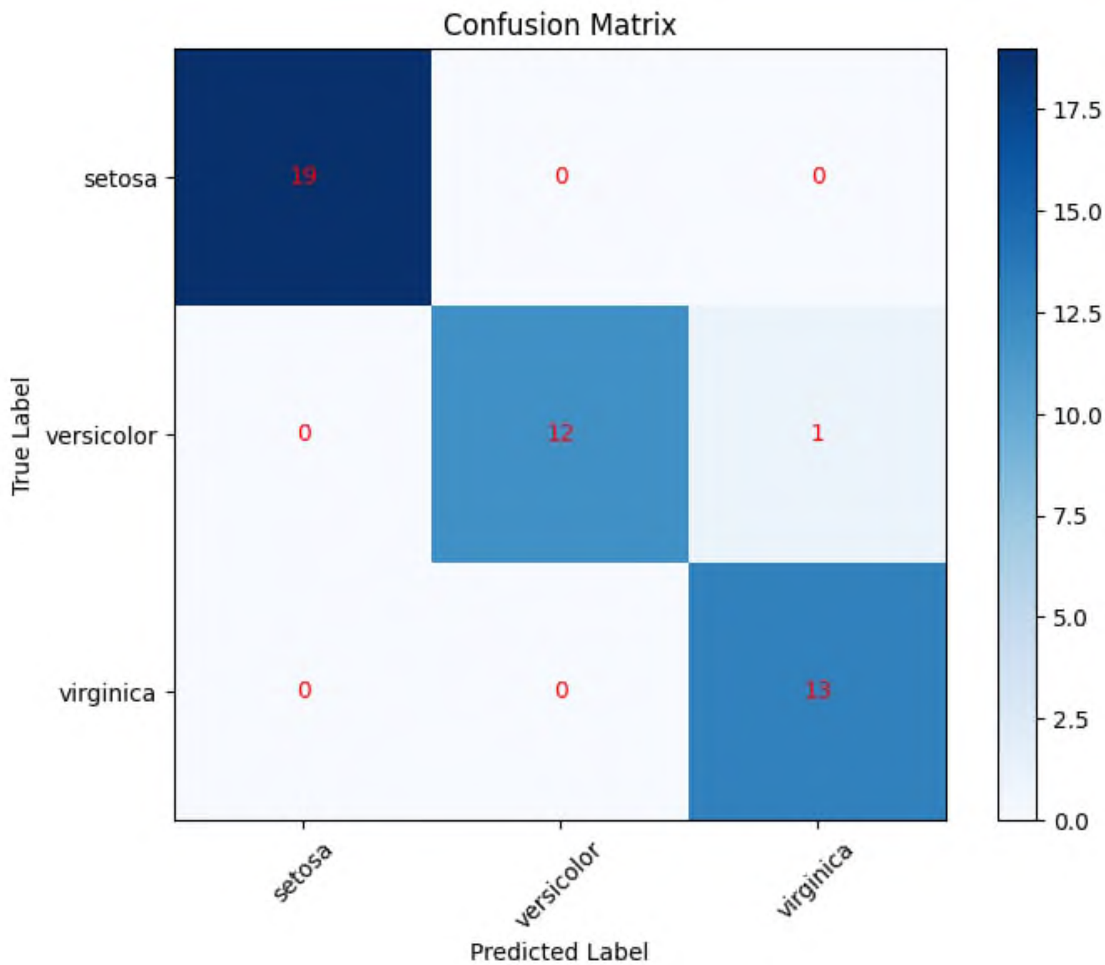
[2]:

```
# Naive Bayes classifier for Numerical data
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪confusion_matrix, classification_report

# Load Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split 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)

# Initialize and train Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)
```

```python
# Predict on test set
y_pred = model.predict(X_test)

# Performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Performance Metrics:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Predict on new sample
new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])  # Example input (sepal length,␣
 ↪sepal width, petal length, petal width)
predicted_class = model.predict(new_sample)
print("\nPrediction on new sample:")
print(f"Input: {new_sample[0]}")
print(f"Predicted class: {data.target_names[predicted_class][0]}")

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
tick_marks = np.arange(len(data.target_names))
plt.xticks(tick_marks, data.target_names, rotation=45)
plt.yticks(tick_marks, data.target_names)

# Adding text annotations
for i in range(len(data.target_names)):
    for j in range(len(data.target_names)):
        plt.text(j, i, conf_matrix[i, j], ha="center", va="center", color="red")

plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Performance Metrics:
Accuracy: 0.98
```

```
Precision: 0.98
Recall: 0.98

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45


Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]

Prediction on new sample:
Input: [5.1 3.5 1.4 0.2]
Predicted class: setosa
```


Confusion Matrix
setosa
versicolor
virginica
True Label
19
0
0
0
12
1
0
0
13
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
setosa
versicolor
virginica
Predicted Label

[ ]:

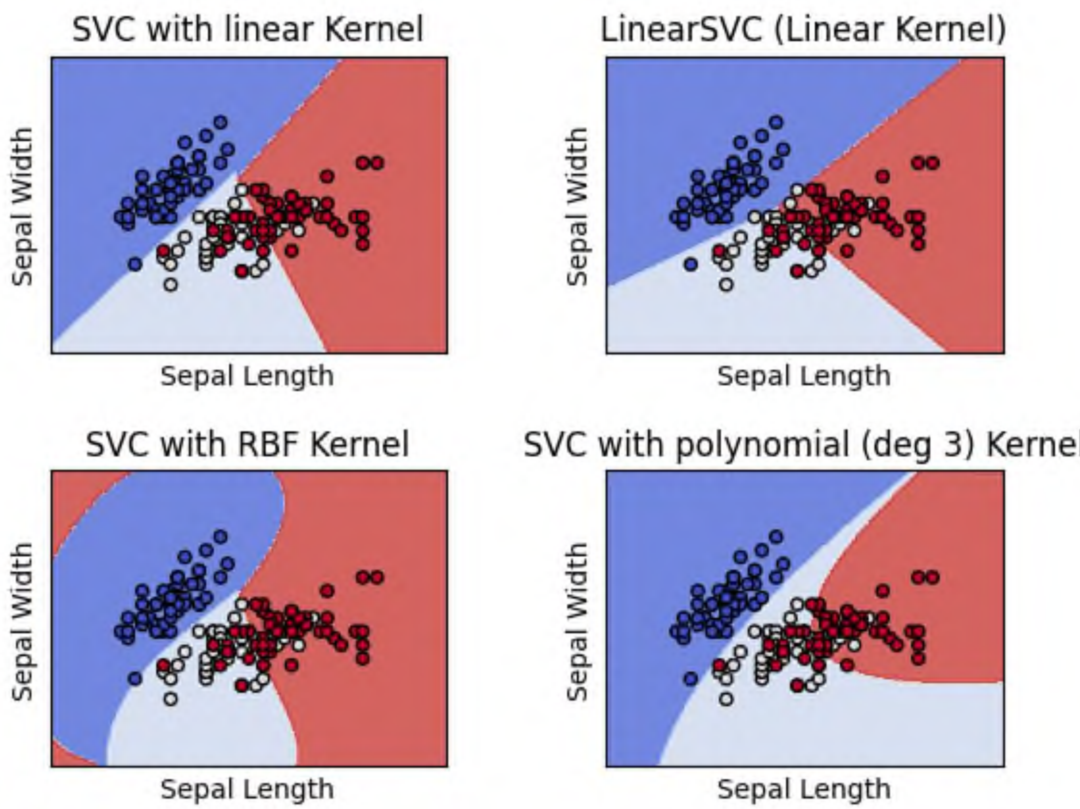
9-SVMClassifier

November 11, 2024

[1]:
```python
#  SVM Classfier
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

def make_meshgrid(x,y,h=0.02):

    x_min, x_max=x.min()-1,x.max()+1
    y_min, y_max=y.min()-1,y.max()+1
    xx,yy=np.meshgrid(np.arange(x_min,x_max,h),np.arange(y_min,y_max,h))
    return xx,yy

def plot_contours(ax,clf,xx,yy,**params):
    Z=clf.predict(np.c_[xx.ravel(),yy.ravel()])
    Z=Z.reshape(xx.shape)
    out=ax.contourf(xx,yy,Z,**params)
    return out


# import the data
iris=datasets.load_iris()
X=iris.data[:,:2]
y=iris.target

C=1.0 # regularization parameter

models=(
    svm.SVC(kernel="linear",C=C),
    svm.LinearSVC(C=C,max_iter=10000),
    svm.SVC(kernel='rbf',gamma=0.7,C=C),
    svm.SVC(kernel='poly',degree=3, gamma='auto',C=C),

)
models=(clf.fit(X,y) for clf in models)

# title of the plots
title=(
```

```
    "SVC with linear Kernel",
    "LinearSVC (Linear Kernel)",
    "SVC with RBF Kernel",
    "SVC with polynomial (deg 3) Kernel",
)

# set up 2x2 grid
fig, sub=plt.subplots(2,2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1=X[:,0], X[:,1]
xx, yy=make_meshgrid(X0,X1)

for clf, title, ax in zip(models, title, sub.flatten()):
    plot_contours(ax,clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0,X1,c=y,cmap=plt.cm.coolwarm, s=20, edgecolor='k')
    ax.set_xlim(xx.min(),xx.max())
    ax.set_ylim(yy.min(),yy.max())
    ax.set_xlabel("Sepal Length")
    ax.set_ylabel("Sepal Width")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```

```
C:\Users\rkmau\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
```


SVC with linear Kernel
Sepal Width
Sepal Length
LinearSVC (Linear Kernel)
Sepal Width
Sepal Length
SVC with RBF Kernel
Sepal Width
Sepal Length
SVC with polynomial (deg 3) Kernel
Sepal Width
Sepal Length

```
[ ]:
```

10-ANN-Backpropogation

November 11, 2024

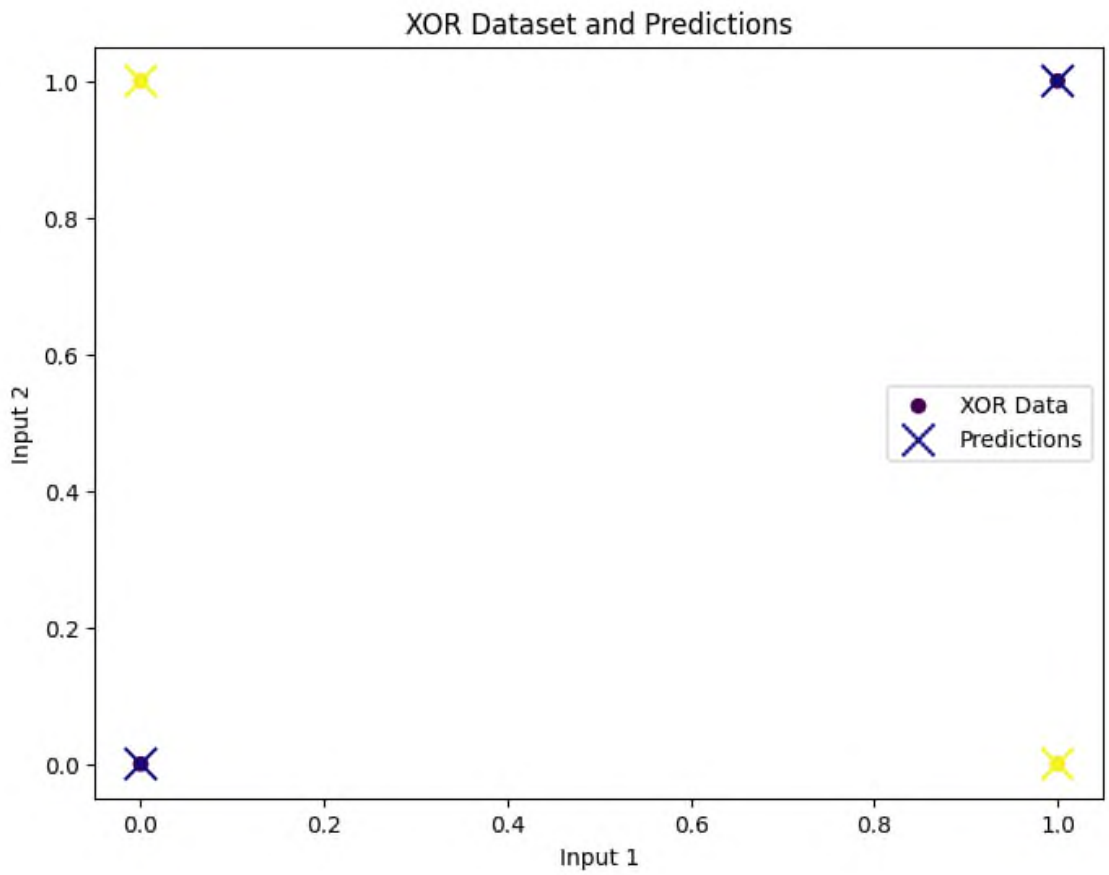
[ ]:

```

```

[1]:

```
import numpy as np
import matplotlib.pyplot as plt

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Define the neural network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights with random values
        self.weights_input_hidden = np.random.uniform(size=(input_size,␣
↪hidden_size))
        self.weights_hidden_output = np.random.uniform(size=(hidden_size,␣
↪output_size))

    def forward(self, X):
        # Forward propagation
        self.hidden_input = np.dot(X, self.weights_input_hidden)
        self.hidden_output = sigmoid(self.hidden_input)
        self.output = sigmoid(np.dot(self.hidden_output, self.
↪weights_hidden_output))
        return self.output

    def backward(self, X, y, learning_rate):
        # Backpropagation
        error_output = y - self.output
        delta_output = error_output * sigmoid_derivative(self.output)

        error_hidden = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = error_hidden * sigmoid_derivative(self.hidden_output)
```

```python
        self.weights_hidden_output += self.hidden_output.T.dot(delta_output) *␣
↪learning_rate
        self.weights_input_hidden += X.T.dot(delta_hidden) * learning_rate

    def train(self, X, y, learning_rate, epochs):
        for _ in range(epochs):
            output = self.forward(X)
            self.backward(X, y, learning_rate)

    def predict(self, X):
        return self.forward(X)

# XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Initialize and train the neural network
input_size = 2
hidden_size = 4
output_size = 1
learning_rate = 0.1
epochs = 10000

nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(X, y, learning_rate, epochs)

# Make predictions
predictions = nn.predict(X)

# Plot the XOR dataset and predictions
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', label='XOR Data')
plt.scatter(X[:, 0], X[:, 1], c=np.round(predictions), cmap='plasma',␣
↪marker='x', s=200, label='Predictions')
plt.title('XOR Dataset and Predictions')
plt.xlabel('Input 1')
plt.ylabel('Input 2')
plt.legend()
plt.show()

# Print predictions and actual values
for i in range(len(X)):
    print(f"Input: {X[i]}, Actual: {y[i]}, Predicted: {np.
↪round(predictions[i])}")
```




```
Input: [0 0], Actual: [0], Predicted: [0.]
Input: [0 1], Actual: [1], Predicted: [1.]
Input: [1 0], Actual: [1], Predicted: [1.]
Input: [1 1], Actual: [0], Predicted: [0.]
```

```
[ ]:
```