import numpy as np import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier

# Paths to the data files

train\_file\_path = '/content/optdigits.tra' test\_file\_path = '/content/optdigits.tes'

# Load the training and testing data

train\_data = pd.read\_csv(train\_file\_path, header=None) test\_data = pd.read\_csv(test\_file\_path, header=None)

# Separate features and labels

X\_train = train\_data.iloc[:, :-1].values # All columns except the last one y\_train = train\_data.iloc[:, -1].values # The last column as labels

X\_test = test\_data.iloc[:, :-1].values # All columns except the last one y\_test = test\_data.iloc[:, -1].values # The last column as labels

# Dictionary to store model names and their accuracies model\_accuracies = {}

# KNN Classifier

# Initialize and train the K-Nearest Neighbors (KNN) classifier knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = knn.predict(X\_test)

model\_accuracies['KNN'] = accuracy\_score(y\_test, y\_pred) print(f'Accuracy: {accuracy\_score(y\_test, y\_pred) \* 100:.2f}%')

Accuracy: 97.83%

# Loop over different values of k for i, k in enumerate(neighbors):

# Setup a k-NN Classifier with k neighbors: knn knn = KNeighborsClassifier(n\_neighbors=k)

# Fit the classifier to the training data knn.fit(X\_train, y\_train)

#Compute accuracy on the training set train\_accuracy[i] = knn.score(X\_train, y\_train)

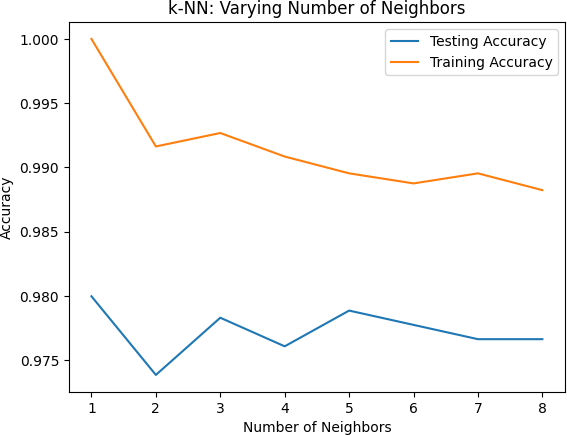
#Compute accuracy on the testing set test\_accuracy[i] = knn.score(X\_test, y\_test)

# Generate plot

import matplotlib.pyplot as plt

plt.title('k-NN: Varying Number of Neighbors') plt.plot(neighbors, test\_accuracy, label = 'Testing Accuracy') plt.plot(neighbors, train\_accuracy, label = 'Training Accuracy') plt.legend()

plt.xlabel('Number of Neighbors') plt.ylabel('Accuracy') plt.show()

# Naive Bayes Classifier

nb\_model = GaussianNB() nb\_model.fit(X\_train, y\_train) y\_pred\_nb = nb\_model.predict(X\_test)

model\_accuracies['Naive Bayes'] = accuracy\_score(y\_test, y\_pred\_nb) print(f'Accuracy: {accuracy\_score(y\_test, y\_pred\_nb) \* 100:.2f}%')

Accuracy: 78.63%

# Decision Tree Classifier

# 2. Decision Tree Classifier

dt\_model = DecisionTreeClassifier(random\_state=0) dt\_model.fit(X\_train, y\_train)

y\_pred\_dt = dt\_model.predict(X\_test)

model\_accuracies['Decision Tree'] = accuracy\_score(y\_test, y\_pred\_dt) print(f'Accuracy: {accuracy\_score(y\_test, y\_pred\_dt) \* 100:.2f}%')

Accuracy: 85.25%

# Random forest Classifier

# 3. Random Forest Classifier

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=0) rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

model\_accuracies['Random Forest'] = accuracy\_score(y\_test, y\_pred\_rf) print(f'Accuracy: {accuracy\_score(y\_test, y\_pred\_rf) \* 100:.2f}%')

Accuracy: 97.05%

# Graphical Representations

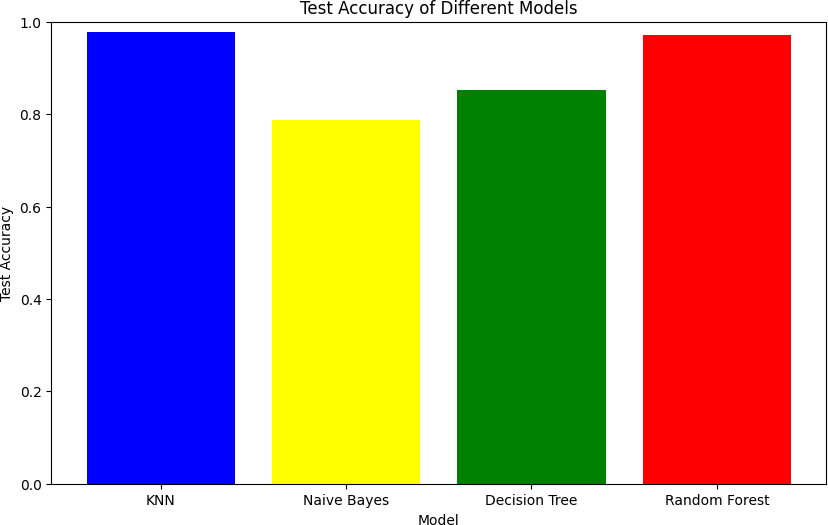
# Plotting the test accuracies plt.figure(figsize=(10, 6))

plt.bar(model\_accuracies.keys(), model\_accuracies.values(), color=['blue', 'yellow', 'green', 'red']) plt.xlabel('Model')

plt.ylabel('Test Accuracy')

plt.title('Test Accuracy of Different Models')

plt.ylim(0, 1) # Scale the y-axis from 0 to 1 for percentage accuracy plt.show()

# List of classifiers to evaluate classifiers = {

"K-Nearest Neighbors": KNeighborsClassifier(n\_neighbors=5), "Naive Bayes": GaussianNB(),

"Decision Tree": DecisionTreeClassifier(random\_state=0),

"Random Forest": RandomForestClassifier(n\_estimators=100, random\_state=0)

}

# Dictionary to store performance metrics for each classifier performance\_data = []

# Evaluate each classifier

for name, model in classifiers.items(): # Train the model model.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Generate classification report

report = classification\_report(y\_test, y\_pred, output\_dict=True) precision = report['weighted avg']['precision']

recall = report['weighted avg']['recall'] f1\_score = report['weighted avg']['f1-score']

# Append the results to the performance data performance\_data.append({

"Classifier": name, "Accuracy": accuracy, "Precision": precision, "Recall": recall,

"F1 Score": f1\_score

})

# Create a DataFrame from the performance data performance\_df = pd.DataFrame(performance\_data)

# Display the performance table print(performance\_df)

Classifier Accuracy Precision Recall F1 Score

0 K-Nearest Neighbors 0.978854 0.979196 0.978854 0.978859

1 Naive Bayes 0.786311 0.830747 0.786311 0.785279

|  |  |  |
| --- | --- | --- |
| 2 | Decision Tree 0.852532 | 0.857132 0.852532 0.853379 |
| 3 | Random Forest 0.970506 | 0.970865 0.970506 0.970541 |

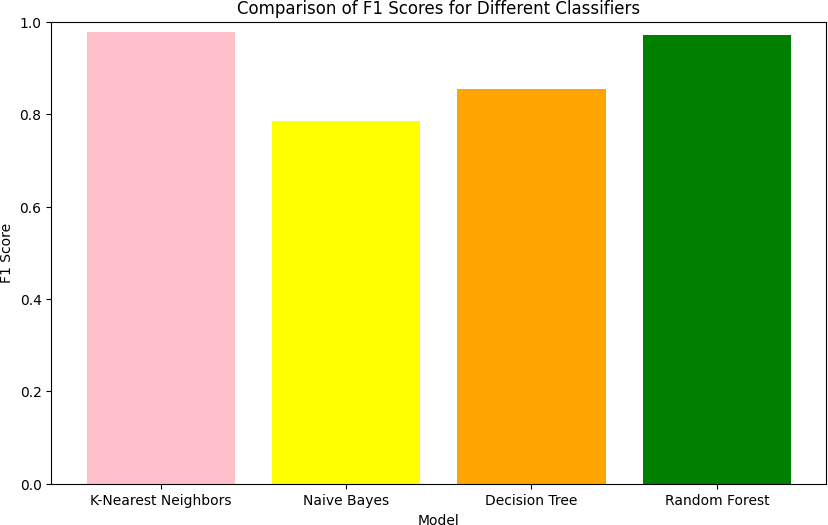
# Plotting the F1 scores plt.figure(figsize=(10, 6))

plt.bar(f1\_scores.keys(), f1\_scores.values(), color=['pink', 'yellow', 'orange', 'green']) plt.xlabel('Model')

plt.ylabel('F1 Score')

plt.title('Comparison of F1 Scores for Different Classifiers') plt.ylim(0, 1) # Scale y-axis from 0 to 1

plt.show()



Start coding or g enerate with AI.