```
import numpy as np

# Load the training data and labels
train_data = np.load('X_kannada_MNIST_train.npz')
X_train = train_data['arr_0']

train_labels = np.load('y_kannada_MNIST_train.npz')
y_train = train_labels['arr_0']

# Load the test data and labels
test_data = np.load('X_kannada_MNIST_test.npz')
X_test = test_data['arr_0']

test_labels = np.load('y_kannada_MNIST_test.npz')
y_test = test_labels['arr_0']

# Print the shapes of the loaded data
print("Training data shape:", X_train.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:", X_test.shape)
print("Test labels shape:", y_test.shape)
```

```
Training data shape: (60000, 28, 28)
Training labels shape: (60000,)
Test data shape: (10000, 28, 28)
Test labels sha
```

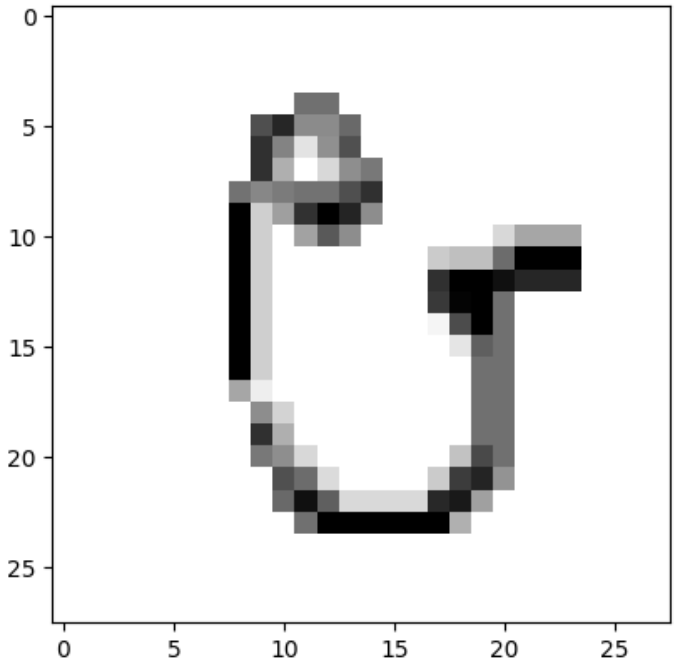
Disk: 27.12 GB/107.72 GB

```
import numpy as np

# Load the npz file
dataset = np.load('X_kannada_MNIST_train.npz')

# Extract the data
x_train = dataset['arr_0']

# Print the shape of the extracted data
print("Training data shape:", x_train.shape)
```

```
Training data shape: (60000, 28, 28)
```

```
import matplotlib.pyplot as plt
import matplotlib


some_digit=X_train[98]


some_digit_img=some_digit.reshape(28,28)


plt.imshow(some_digit_img,cmap=matplotlib.cm.binary,interpolation="nearest")
```

```
<matplotlib.image.AxesImage at 0x7d4e48a57370>
```



```
y_train[98]
```

```
8
```

Disk: 27.12 GB/107.72 GB

Performing PCA to the data

```
from sklearn.decomposition import PCA


X_train_2d = X_train.reshape(X_train.shape[0], -1)
X_test_2d = X_test.reshape(X_test.shape[0], -1)

# Perform PCA with 10 components
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train_2d)
X_test_pca = pca.transform(X_test_2d)


y_train[1].dtype
```

```
dtype('uint8')
```

WRITING A SINGLE FUNCTION TO FIND OUT VARIOUS PARAMETERS

```
from sklearn.metrics import accuracy_score, f1_score, recall_score, roc_auc_score,confusion_matrix

def evaluate_model(model, x_train, y_train, x_test, y_test):
    # Fit the model on the training data
    model.fit(x_train, y_train)

    # Predict labels for the test data
    y_pred = model.predict(x_test)

    # Calculate accuracy, f1-score, and recall
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')

    # Calculate ROC-AUC score
```

```
    y_pred_proba = model.predict_proba(x_test)
    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')

    ## Calculating Confusion Matrix
    conf_matrix=confusion_matrix(y_test,y_pred)

    # Return evaluation metrics
    return accuracy, f1, recall, roc_auc, conf_matrix


from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC


dt_classifier = DecisionTreeClassifier()
dt_accuracy, dt_f1, dt_recall, dt_roc_auc, dt_confusion_matrix = evaluate_model(dt_classifier, X_train_pca,
print("Decision Tree Accuracy:", dt_accuracy)
print("Decision Tree F1-score:", dt_f1)
print("Decision Tree Recall:", dt_recall)
print("Decision Tree ROC-AUC:", dt_roc_auc)
print("Decision Tree Confusion Matrix:")
print(dt_confusion_matrix)
```

```
Decision Tree Accuracy: 0.7993
Decision Tree F1                    8519
Decision Tree R                     99
Decision Tree ROC-AUC: 0.8885
Decision Tree Confusion Matrix:
[[717 150   9  44   7   2   1   8  39  23]
 [ 79 799   5  33  15  16   4   7  30  12]
 [  6   3 920  14   4  20  18   6   7   2]
 [ 22  16   5 774  27  25  31  65  16  19]
 [  4   1   2  38 849  50   9  11  14  22]
 [  9  12   8  15 123 790   8   6  19  10]
 [  8   4   3  46  18  32 769  95   6  19]
 [  9  13   7  63  33   4 137 682  18  34]
 [ 51  22  12   8   8   6   2   5 869  17]
 [ 17  42   3   8  22   7   5  12  60 824]]
```

Disk: 27.12 GB/107.72 GB

```
rf_classifier = RandomForestClassifier()
rf_accuracy, rf_f1, rf_recall, rf_roc_auc, rf_confusion_matrix = evaluate_model(rf_classifier, X_train_pca,
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest F1-score:", rf_f1)
print("Random Forest Recall:", rf_recall)
print("Random Forest ROC-AUC:", rf_roc_auc)
print("Random Forest Confusion Matrix",rf_confusion_matrix)
```

```
Random Forest Accuracy: 0.8758
Random Forest F1-score: 0.8751161035812375
Random Forest Recall: 0.8758000000000001
Random Forest ROC-AUC: 0.9893558277777776
Random Forest Confusion Matrix [[779 150   4  29   5   0   2   4  18   9]
 [ 31 910   1  24   6   2   0   2  15   9]
 [  4   2 980   2   0   7   0   4   1   0]
 [ 10   9   1 886  16  22   8  28  16   4]
 [  0   2   0  29 920  32   2   2   7   6]
 [  0   2   2   8 120 850   3   3   8   4]
 [  2   7   1  48  17  23 831  53   0  18]
 [  5   9   2  41  25   0 128 746   9  35]
 [ 17  13   1   4   3   5   1   0 953   3]
 [  5   9   0   2  16   0   8   6  51 903]]
```

```
nb_classifier = GaussianNB()
nb_accuracy, nb_f1, nb_recall, nb_roc_auc, nb_confusion_matrix = evaluate_model(nb_classifier, X_train_pca,
print("Naive Bayes Accuracy:", nb_accuracy)
print("Naive Bayes F1-score:", nb_f1)
print("Naive Bayes Recall:", nb_recall)
print("Naive Bayes ROC-AUC:", nb_roc_auc)
print("Naive Bayes Confusion Matrix:")
print(nb_confusion_matrix)
```

```
Naive Bayes Accuracy: 0.7725
Naive Bayes F1-score: 0.7711209677576044
Naive Bayes Recall: 0.7725
Naive Bayes ROC-AUC: 0.9722622666666666
Naive Bayes Confusion Matrix:
[[608 229  16  70   8   0   2  13  45   9]
 [ 25 749   9 121   8   2   0   0  66  20]
 [  9   1 967   3   0  15   1   3   1   0]
 [ 13   7   2 844  32  16  21  64   0   1]
 [  1   1   1  38 872  43   1   8  28   7]
 [  0   2  15  47 165 731   6  16  12   6]
 [  5   9  50  30  25  45 729  94   3  10]
 [  9  17   4  80  50   1 176 651   2  10]
 [ 89  36  26   6   4   3   2   4 822   8]
 [ 19  28   1   8  62   0   3  21 106 752]]
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=8)
knn_accuracy, knn_f1, knn_recall, knn_roc_auc, knn_confusion_matrix = evaluate_model(knn_classifier, X_trai
print("K-NN Accuracy
print("K-NN F1-score
print("K-NN Recall:", knn_recall)
print("K-NN ROC-AUC:", knn_roc_auc)
print("K-NN Confusion Matrix")
print(knn_confusion_matrix)
```

Disk: 27.12 GB/107.72 GB

```
K-NN Accuracy: 0.8816
K-NN F1-score: 0.880497476399861
K-NN Recall: 0.8816
K-NN ROC-AUC: 0.9737439388888889
K-NN Confusion Matrix
[[776 169   1  27   2   0   2   3  14   6]
 [ 18 943   0  16   4   1   0   2  11   5]
 [  6   1 979   3   0   7   1   2   1   0]
 [ 10  12   0 887  20  22   4  15  26   4]
 [  0   1   0  14 952  22   1   2   4   4]
 [  0   3   0   8 116 857   3   3   4   6]
 [  0   6   0  41  21  10 843  65   0  14]
 [  7   8   0  52  34   3 166 699   6  25]
 [ 15   9   2   1   3   6   3   0 958   3]
 [  5   7   0   0  25   0   7   4  30 922]]
```

```
svm_classifier = SVC(probability=True)
svm_accuracy, svm_f1, svm_recall, svm_roc_auc, svm_confusion_matrix = evaluate_model(svm_classifier, X_trai
print("SVM Accuracy:", svm_accuracy)
print("SVM F1-score:", svm_f1)
print("SVM Recall:", svm_recall)
print("SVM ROC-AUC:", svm_roc_auc)
print("SVM confusion Matrix")
print(svm_confusion_matrix)
```

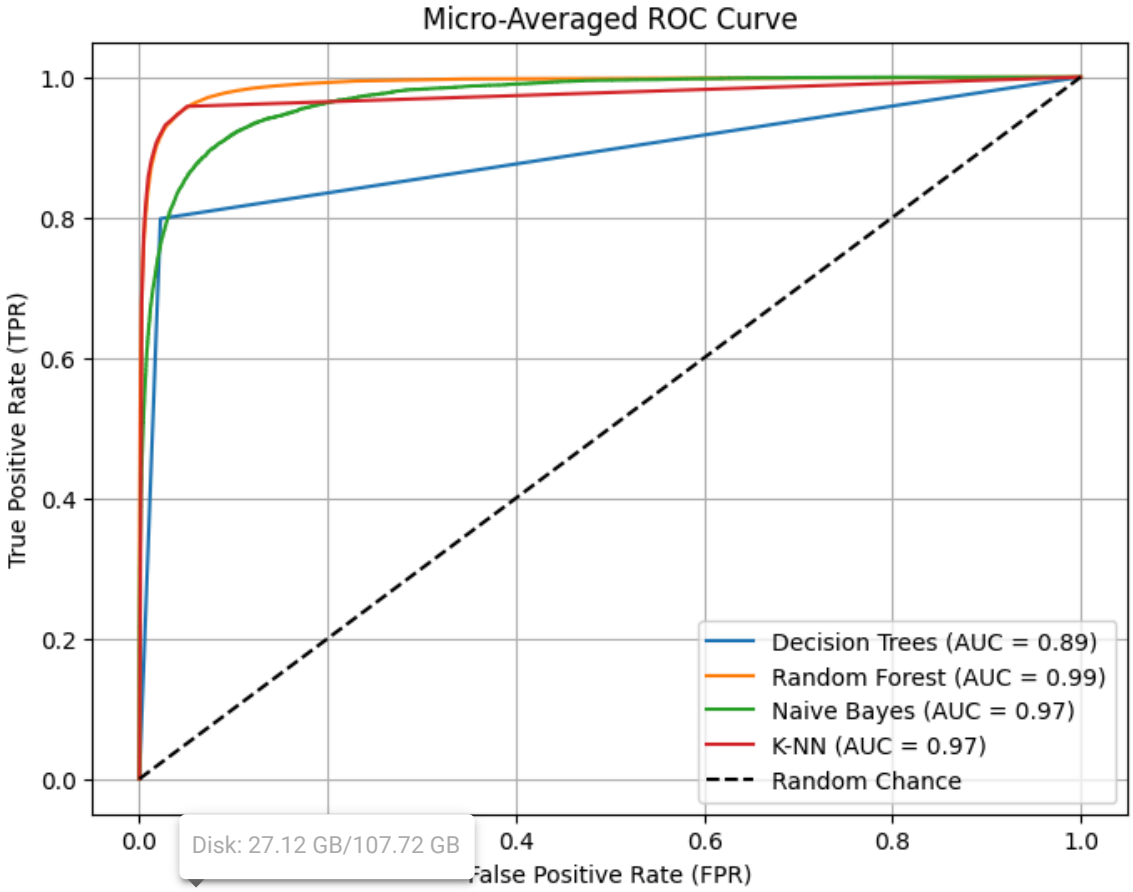
```
SVM Accuracy: 0.8868
SVM F1-score: 0.886272835453126
SVM Recall: 0.8868
SVM ROC-AUC: 0.9924893555555556
SVM confusion Matrix
[[810 140   1  22   6   0   1   2  15   3]
 [ 19 924   0  27   3   3   0   4  12   8]
 [  3   2 984   1   0   5   0   3   1   1]
 [  8  11   0 886  20  22   8  27  18   0]
```

```
[  0   1   0  14 948  22   0   0   5  10]
[  0   4   1   4 124 854   3   3   4   3]
[  0  10   1  50  20  21 845  40   0  13]
[  6  13   0  33  17   4 135 747   7  38]
[  8  14   4   1   4   4   3   0 958   4]
[  2  13   0   0  23   0   6   3  41 912]]
```

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Convert the true labels to one-hot encoded format
y_test_bin = label_binarize(y_test, classes=range(10))

# For Decision Trees
dt_pred_probs = dt_classifier.predict_proba(X_test_pca)
dt_fpr, dt_tpr, _ = roc_curve(y_test_bin.ravel(), dt_pred_probs.ravel())
dt_auc = auc(dt_fpr, dt_tpr)

# For Random Forest
rf_pred_probs = rf_classifier.predict_proba(X_test_pca)
rf_fpr, rf_tpr, _ = roc_curve(y_test_bin.ravel(), rf_pred_probs.ravel())
rf_auc = auc(rf_fpr, rf_tpr)

# For Naive Bayes
nb_pred_probs = nb_classifier.predict_proba(X_test_pca)
nb_fpr, nb_tpr, _ = roc_curve(y_test_bin.ravel(), nb_pred_probs.ravel())
nb_auc = auc(nb_fpr,
```

Disk: 27.12 GB/107.72 GB

```
# For K-NN
knn_pred_probs = knn_classifier.predict_proba(X_test_pca)
knn_fpr, knn_tpr, _ = roc_curve(y_test_bin.ravel(), knn_pred_probs.ravel())
knn_auc = auc(knn_fpr, knn_tpr)


# Plot micro-averaged ROC curves for each classifier
plt.figure(figsize=(8, 6))
plt.plot(dt_fpr, dt_tpr, label='Decision Trees (AUC = {:.2f})'.format(dt_auc))
plt.plot(rf_fpr, rf_tpr, label='Random Forest (AUC = {:.2f})'.format(rf_auc))
plt.plot(nb_fpr, nb_tpr, label='Naive Bayes (AUC = {:.2f})'.format(nb_auc))
plt.plot(knn_fpr, knn_tpr, label='K-NN (AUC = {:.2f})'.format(knn_auc))
#plt.plot(svm_fpr, svm_tpr, label='SVM (AUC = {:.2f})'.format(svm_auc))

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Micro-Averaged ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

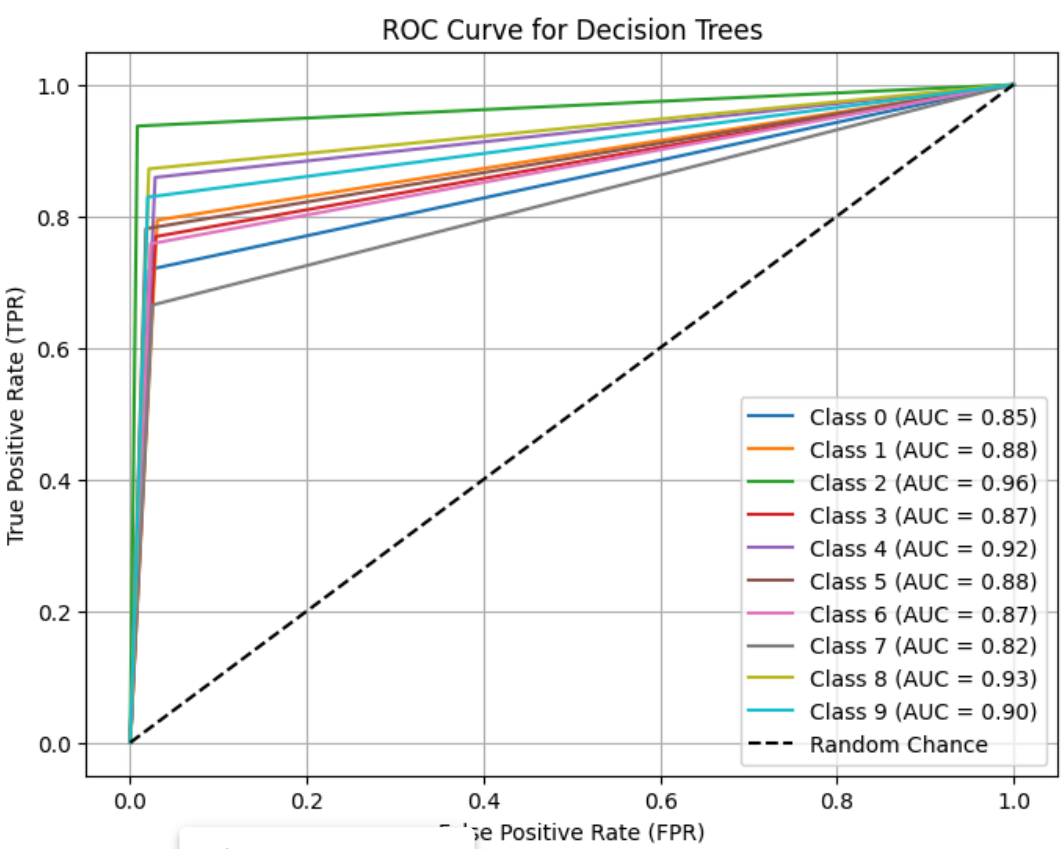

Micro-Averaged ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Decision Trees (AUC = 0.89)
Random Forest (AUC = 0.99)
Naive Bayes (AUC = 0.97)
K-NN (AUC = 0.97)
Random Chance
Disk: 27.12 GB/107.72 GB

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Convert the true labels to one-hot encoded format
y_test_bin = label_binarize(y_test, classes=range(10))

# For Decision Trees
dt_fpr = dict()
dt_tpr = dict()
dt_auc = dict()
for class_idx in range(10):
    dt_pred_probs = dt_classifier.predict_proba(X_test_pca)[:, class_idx]
    dt_fpr[class_idx], dt_tpr[class_idx], _ = roc_curve(y_test_bin[:, class_idx], dt_pred_probs)
    dt_auc[class_idx] = auc(dt_fpr[class_idx], dt_tpr[class_idx])

# For Random Forest
rf_fpr = dict()
rf_tpr = dict()
rf_auc = dict()
for class_idx in range(10):
    rf_pred_probs = rf_classifier.predict_proba(X_test_pca)[:, class_idx]
    rf_fpr[class_idx], rf_tpr[class_idx], _ = roc_curve(y_test_bin[:, class_idx], rf_pred_probs)
    rf_auc[class_idx] = auc(rf_fpr[class_idx], rf_tpr[class_idx])

# For Naive Bayes
nb_fpr = dict()
nb_tpr = dict()
nb_auc = dict()
for class_idx in range(10):
    nb_pred_probs = nb_classifier.predict_proba(X_test_pca)[:, class_idx]
    nb_fpr[class_idx], nb_tpr[class_idx], _ = roc_curve(y_test_bin[:, class_idx], nb_pred_probs)
    nb_auc[class_idx] = auc(nb_fpr[class_idx], nb_tpr[class_idx])

# For K-NN
knn_fpr = dict()
knn_tpr = dict()
knn_auc = dict()
for class_idx in range(10):
    knn_pred_probs = knn_classifier.predict_proba(X_test_pca)[:, class_idx]
    knn_fpr[class_idx], knn_tpr[class_idx], _ = roc_curve(y_test_bin[:, class_idx], knn_pred_probs)
    knn_auc[class_idx] = auc(knn_fpr[class_idx], knn_tpr[class_idx])

# Plot ROC curves for each class for Decision Trees
plt.figure(figsize=(8, 6))
for class_idx in range(10):
    plt.plot(dt_fpr[class_idx], dt_tpr[class_idx], label='Class {} (AUC = {:.2f})'.format(class_idx, dt_auc

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Decision Trees')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# Plot ROC curves for each class for Random Forest
plt.figure(figsize=(8, 6))
for class_idx in range(10):
    plt.plot(rf_fpr[class_idx], rf_tpr[class_idx], label='Class {} (AUC = {:.2f})'.format(class_idx, rf_auc

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Random Forest')
plt.legend(loc='lower right')
```

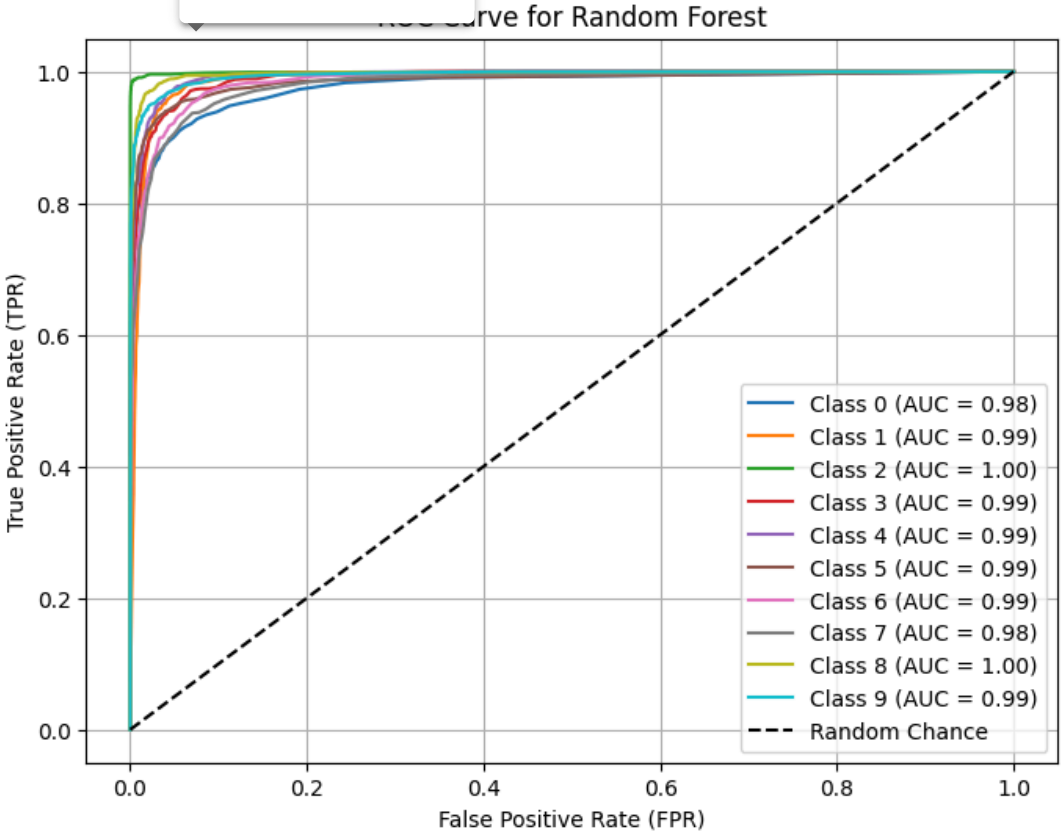
Disk: 27.12 GB/107.72 GB

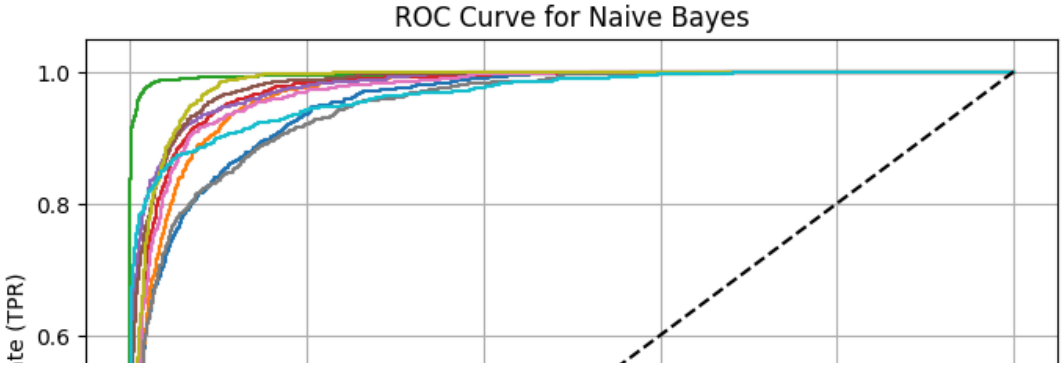
```
plt.grid(True)
plt.show()

# Plot ROC curves for each class for Naive Bayes
plt.figure(figsize=(8, 6))
for class_idx in range(10):
    plt.plot(nb_fpr[class_idx], nb_tpr[class_idx], label='Class {} (AUC = {:.2f})'.format(class_idx, nb_auc

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Naive Bayes')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# Plot ROC curves for each class for K-NN
plt.figure(figsize=(8, 6))
for class_idx in range(10):
    plt.plot(knn_fpr[class_idx], knn_tpr[class_idx], label='Class {} (AUC = {:.2f})'.format(class_idx, knn_

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for K-NN')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

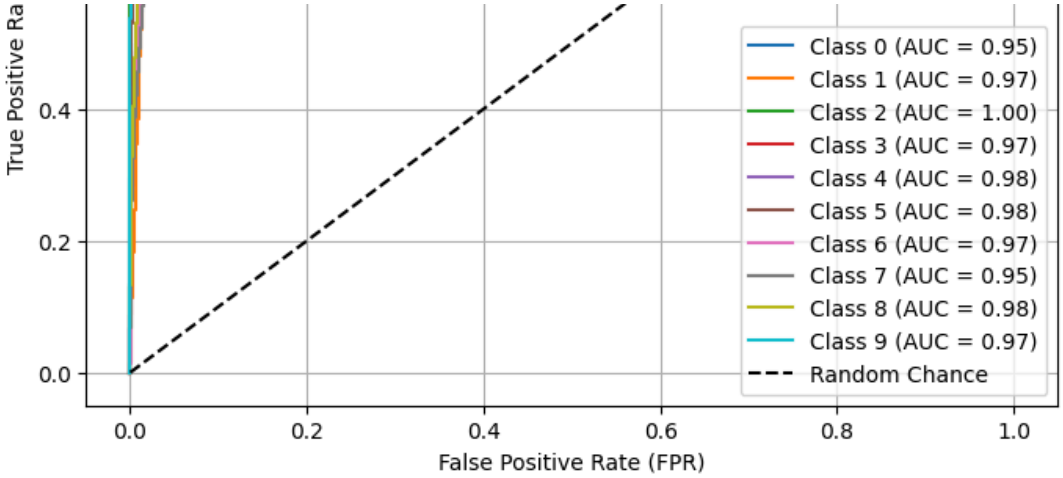
Disk: 27.12 GB/107.72 GB

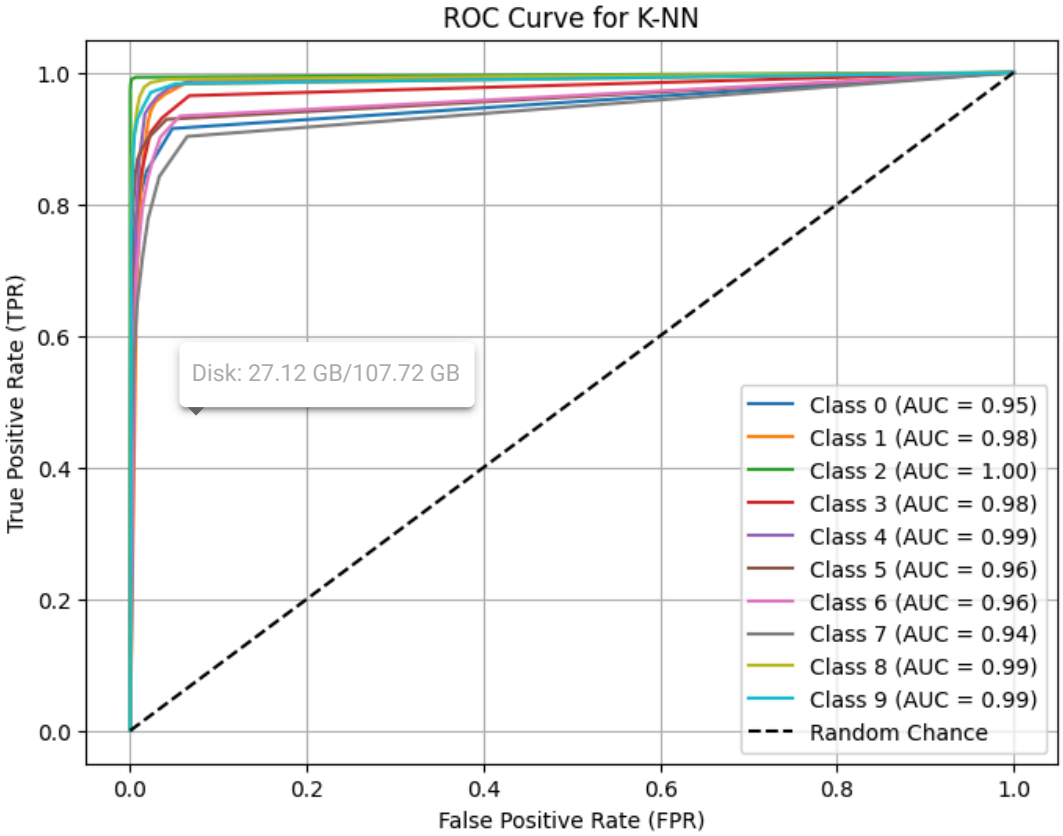


Disk: 27.12 GB/107.72 GB









WHEN the PCA component is 15

```
X_train_2d = X_train.reshape(X_train.shape[0], -1)
X_test_2d = X_test.reshape(X_test.shape[0], -1)

# Perform PCA with 15 components
pca = PCA(n_components=15)
X_train_pca = pca.fit_transform(X_train_2d)
X_test_pca = pca.transform(X_test_2d)
```

```
from sklearn.metrics import accuracy_score, f1_score, recall_score, roc_auc_score

def evaluate_model(model, x_train, y_train, x_test, y_test):
    # Fit the model on the training data
    model.fit(x_train, y_train)

    # Predict labels for the test data
    y_pred = model.predict(x_test)

    # Calculate accuracy, f1-score, and recall
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')

    # Calculate ROC-AUC score
    y_pred_proba = model.predict_proba(x_test)
    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')

    # Return evaluation metrics
    return accuracy, f1, recall, roc_auc


dt_classifier = DecisionTreeClassifier()
dt_accuracy, dt_f1, dt_recall, dt_roc_auc = evaluate_model(dt_classifier, X_train_pca, y_train, X_test_pca,
print("Decision Tree Accuracy:", dt_accuracy)
print("Decision Tree F1-score:", dt_f1)
print("Decision Tree Recall:", dt_recall)
print("Decision Tree
```

Disk: 27.12 GB/107.72 GB

```
Decision Tree Accuracy: 0.8083
Decision Tree F1-score: 0.8073549768477074
Decision Tree Recall: 0.8083
Decision Tree ROC-AUC: 0.8935000000000001
```

```
rf_classifier = RandomForestClassifier()
rf_accuracy, rf_f1, rf_recall, rf_roc_auc = evaluate_model(rf_classifier, X_train_pca, y_train, X_test_pca,
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest F1-score:", rf_f1)
print("Random Forest Recall:", rf_recall)
print("Random Forest ROC-AUC:", rf_roc_auc)
```

```
Random Forest Accuracy: 0.8938
Random Forest F1-score: 0.8932782818582596
Random Forest Recall: 0.8938
Random Forest ROC-AUC: 0.9921343555555555
```

```
nb_classifier = GaussianNB()
nb_accuracy, nb_f1, nb_recall, nb_roc_auc = evaluate_model(nb_classifier, X_train_pca, y_train, X_test_pca,
print("Naive Bayes Accuracy:", nb_accuracy)
print("Naive Bayes F1-score:", nb_f1)
print("Naive Bayes Recall:", nb_recall)
print("Naive Bayes ROC-AUC:", nb_roc_auc)
```

```
Naive Bayes Accuracy: 0.7836
Naive Bayes F1-score: 0.7820186234804182
Naive Bayes Recall: 0.7836000000000001
Naive Bayes ROC-AUC: 0.9747302333333334
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=8)
knn_accuracy, knn_f1, knn_recall, knn_roc_auc = evaluate_model(knn_classifier, X_train_pca, y_train, X_test
print("K-NN Accuracy:", knn_accuracy)
print("K-NN F1-score:", knn_f1)
print("K-NN Recall:", knn_recall)
print("K-NN ROC-AUC:", knn_roc_auc)
```

```
K-NN Accuracy: 0.9073
K-NN F1-score: 0.9067350673770885
K-NN Recall: 0.9073
K-NN ROC-AUC: 0.9823980166666667
```

```
svm_classifier = SVC(probability=True)
svm_accuracy, svm_f1, svm_recall, svm_roc_auc = evaluate_model(svm_classifier, X_train_pca, y_train, X_test
print("SVM Accuracy:", svm_accuracy)
print("SVM F1-score:", svm_f1)
print("SVM Recall:", svm_recall)
print("SVM ROC-AUC:", svm_roc_auc)
```

```
SVM Accuracy: 0.9141
SVM F1-score: 0.9136140969181386
SVM Recall: 0.9141
SVM ROC-AUC: 0.9953097444444443
```

WHEN the PCA component is 20

```
X_train_2d = X_train.reshape(X_train.shape[0], -1)
X_test_2d = X_test.reshape(X_test.shape[0], -1)

# Perform PCA with 20 components
pca = PCA(n_components=20)
X_train_pca = pca.fit_transform(X_train_2d)
X_test_pca = pca.tra
```

Disk: 27.12 GB/107.72 GB

```
dt_classifier = DecisionTreeClassifier()
dt_accuracy, dt_f1, dt_recall, dt_roc_auc = evaluate_model(dt_classifier, X_train_pca, y_train, X_test_pca,
print("Decision Tree Accuracy:", dt_accuracy)
print("Decision Tree F1-score:", dt_f1)
print("Decision Tree Recall:", dt_recall)
print("Decision Tree ROC-AUC:", dt_roc_auc)
```

```
Decision Tree Accuracy: 0.8049
Decision Tree F1-score: 0.8038570656195934
Decision Tree Recall: 0.8049
Decision Tree ROC-AUC: 0.891611111111111
```

```
rf_classifier = RandomForestClassifier()
rf_accuracy, rf_f1, rf_recall, rf_roc_auc = evaluate_model(rf_classifier, X_train_pca, y_train, X_test_pca,
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest F1-score:", rf_f1)
print("Random Forest Recall:", rf_recall)
print("Random Forest ROC-AUC:", rf_roc_auc)
```

```
Random Forest Accuracy: 0.9027
Random Forest F1-score: 0.9021545166474686
Random Forest Recall: 0.9027
Random Forest ROC-AUC: 0.993425588888889
```

```
nb_classifier = GaussianNB()
nb_accuracy, nb_f1, nb_recall, nb_roc_auc = evaluate_model(nb_classifier, X_train_pca, y_train, X_test_pca, y
print("Naive Bayes Accuracy:", nb_accuracy)
print("Naive Bayes F1-score:", nb_f1)
print("Naive Bayes Recall:", nb_recall)
print("Naive Bayes ROC-AUC:", nb_roc_auc)
```

```
Naive Bayes Accuracy: 0.7968
Naive Bayes F1-score: 0.7954118652426857
Naive Bayes Recall: 0.7968
Naive Bayes ROC-AUC: 0.9766295666666667
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=8)
knn_accuracy, knn_f1, knn_recall, knn_roc_auc = evaluate_model(knn_classifier, X_train_pca, y_train, X_test_
print("K-NN Accuracy:", knn_accuracy)
print("K-NN F1-score:", knn_f1)
print("K-NN Recall:", knn_recall)
print("K-NN ROC-AUC:", knn_roc_auc)
```

```
K-NN Accuracy: 0.9207
K-NN F1-score: 0.9204407429116384
K-NN Recall: 0.9206999999999999
K-NN ROC-AUC: 0.9845022888888888
```

```
svm_classifier = SVC(probability=True)
svm_accuracy, svm_f1, svm_recall, svm_roc_auc = evaluate_model(svm_classifier, X_train_pca, y_train, X_test
print("SVM Accuracy:", svm_accuracy)
print("SVM F1-score:", svm_f1)
print("SVM Recall:", svm_recall)
print("SVM ROC-AUC:", svm_roc_auc)
```

```
SVM Accuracy: 0.9309
SVM F1-score: 0.9305465224361292
SVM Recall: 0.9309
SVM ROC-AUC: 0.9968802
```

Disk: 27.12 GB/107.72 GB