University of Waterloo
Econ 424
2023-09-28
Anonymized Usernmae: IJustWannaPass

Prediction Competition # 3

Q1) Due to some data manipulation I was able to get the following prediction accuracy of the training set:

$$\text{Prediciton Accuracy} = 92.308\%$$

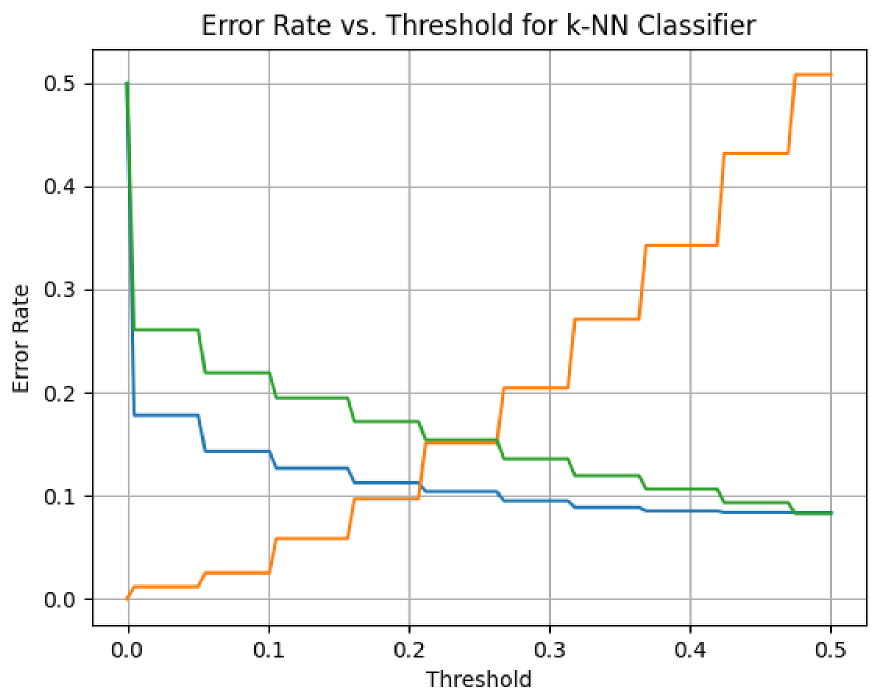
The normalized Confusion Matrix for this model on the training set is as follows (note we ran the 90-10 training test set 10 times):

		Predicted Value		
		0	1	Total
Actual Value	0	233786	19778	253564
	1	18464	227972	246436
	Total	252250	227972	500000

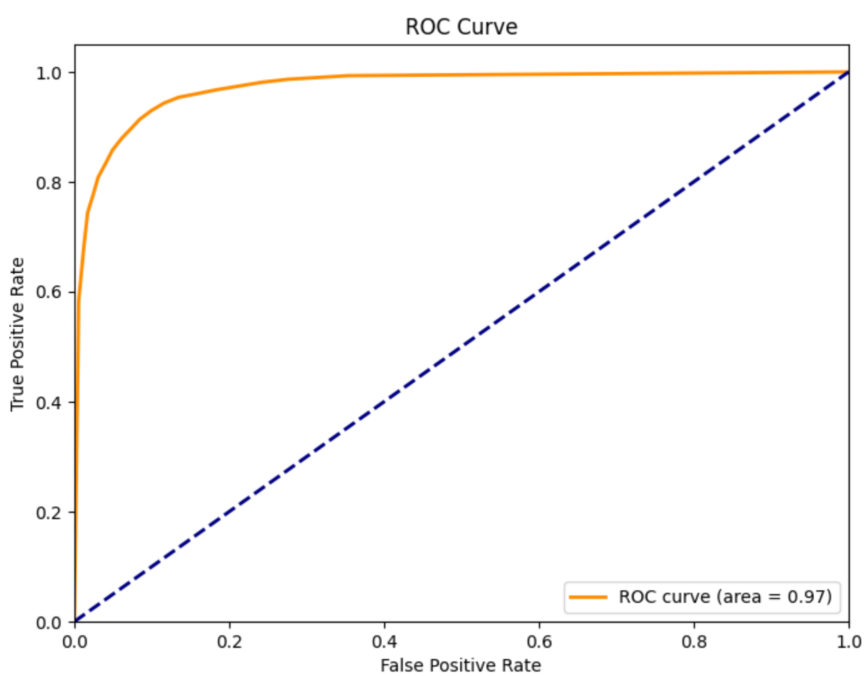
Q2) The first graph showing how 1/k impacts the training and test error can be seen below:

As we can see the training error reduces as k/1 increases, however the test error first decreases and then increases as k/1 gets larger.

Q3) The first graph showing the error rate and the classification curve is seen here:



The second graph showing the ROC curve is seen below:



Q4) To solve Q1 chatGPT was given the prompt:

R given a dataset how should I use knn to classify the data in the most correct ways, without using boosting or bagging how can I increase the accuracy

chatGPT provided the following response:

‹ 3 / 3 ›

To use k-Nearest Neighbors (k-NN) for classification without using boosting or bagging techniques, you can follow these steps to potentially increase the accuracy of your classification:

1. **Data Preprocessing**:
   - Normalize or standardize your dataset: Ensure that all features have the same scale to prevent any feature from dominating the distance calculation.
   - Handle missing values: Decide on a strategy for dealing with missing data, such as imputation or removal.
2. **Feature Selection**:
   - Select relevant features: Identify and retain only the most important features to reduce noise in your dataset. Feature selection techniques like correlation analysis or feature importance scores can be helpful.
3. **Choosing the Right Value for k**:
   - Experiment with different values of k: Try various values of k (the number of neighbors) and use cross-validation to determine which k value provides the best accuracy on your dataset. A small k may lead to overfitting, while a large k may lead to underfitting.
4. **Distance Metric**:
   - Choose an appropriate distance metric: Depending on your dataset and problem, you may want to experiment with different distance metrics, such as Euclidean, Manhattan, or Minkowski distance, to see which one works best.
5. **Cross-Validation**:
   - Employ cross-validation: Use techniques like k-fold cross-validation to assess the performance of your k-NN classifier. This helps you estimate how well your model will generalize to unseen data.
6. **Class Imbalance**:
   - Address class imbalance: If your dataset has imbalanced classes, consider using techniques like oversampling, undersampling, or generating synthetic samples to balance the class distribution. This can prevent the classifier from being biased toward the majority class.
7. **Outlier Handling**:
   - Deal with outliers: Outliers can significantly affect the performance of k-NN. Decide whether to remove outliers or use distance weighting schemes to reduce their impact.
8. **Hyperparameter Tuning**:
   - Optimize hyperparameters: Apart from k, there may be other hyperparameters, such as the weighting scheme (uniform or distance-based) and the distance metric, that you can tune to improve accuracy.
9. **Cross-Feature Scaling**:
   - Consider feature scaling: In some cases, scaling features differently based on their importance can improve accuracy. For example, you might scale important features more than less important ones.

To solve Q2 chatGPT was given the prompt:

given a knn classifer test data set can you create code in python to create and display a confusion matrix

chatGPT provided the following response:

Certainly! To create and display a confusion matrix for a k-NN classifier in Python, you can use libraries like scikit-learn. Here's an example code snippet to help you get started:

python Copy code

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Generate some example data (replace this with your dataset)
# X should be your feature matrix, y should be your target labels
X, y = np.random.rand(100, 2), np.random.randint(0, 2, size=100)

# 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, ran

# Create and train the k-NN classifier
k = 5  # You can choose an appropriate k value
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)

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

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

In this code:

1. We generate some example data (you should replace this with your own dataset).
2. We split the data into training and testing sets.
3. We create a k-NN classifier and train it on the training data.
4. We make predictions on the test data.
5. We calculate the confusion matrix using `sklearn.metrics.confusion_matrix`.
6. We plot the confusion matrix as a heatmap using `matplotlib` and `seaborn`.

Make sure to replace the example data (`X` and `y`) with your actual dataset. This code will display a heatmap of the confusion matrix, showing the true labels on the y-axis and the predicted labels on the x-axis.

To solve Q3 chatGPT was given the prompt:

How would I create a ROC curve in python given a KNN classifier

chatGPT provided the following response:

To create a Receiver Operating Characteristic (ROC) curve in Python for a k-Nearest Neighbors (k-NN) classifier, you can use libraries like scikit-learn. Here's an example of how to do it:

python Copy code

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Generate some example data (replace this with your dataset)
# X should be your feature matrix, y should be your binary target labels (0
X, y = np.random.rand(100, 2), np.random.randint(0, 2, size=100)

# 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, ran

# Create and train the k-NN classifier
k = 5  # You can choose an appropriate k value
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)

# Get the predicted probabilities for the positive class (class 1)
y_scores = knn_classifier.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f}
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

In this code:

1. We generate some example data (you should replace this with your own dataset).
2. We split the data into training and testing sets.
3. We create a k-NN classifier and train it on the training data.

Code for Q1:

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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import ConfusionMatrixDisplay
# Read in the data
data = pd.read_csv("Econ_424_F2023_PC3_training_large.csv", low_memory=False)

# =============== DATA PREPROCESSING ===============

# Modify some of the columns that we wont be using
data.drop(columns=["city", "state", "make"], inplace=True)

# Some models are the same but use weird capitalization or ., this fixes that
data["model"] = data["model"].apply(lambda x: (str(x).lower()).split(".")[0])

# This function will keep a count of each model of car if its below or above 18400
#   ModelDict[model] = (#num cars above 18400, #num cars below 18400)
ModelDict = {}


def classify(price, dep):
    # Cluster cars without considering values after the .
    dep = dep.split(".")[0]

    above = 0
    below = 0
    if price >= 18400:
        above = 1
    else:
        below = 1

    if not dep in ModelDict:
        ModelDict[dep] = (above, below)
    else:
        ModelDict[dep] = (ModelDict[dep][0] + above, ModelDict[dep][1] + below)


data.apply(lambda x: classify(x['price'], x['model']), axis=1)

# This is the factor we use for assigning a numeric value to each car model, since from
#  looking at the data the model has a much greater fit with the price then the milage the
#  difference in model is equal to a value greater than the difference in mileage
diffFactor = 10 ** (int(np.log10(data["mileage"].max())) + 1)

# Since the number of large clusters is dynamic we start at 3 and keep growing
currLargeidx = 3
ModelWeights = {}

for key in ModelDict:
    if ModelDict[key][0] + ModelDict[key][1] < 30:
        # Only < 30 observations, cluster with similar small sample models

        if ModelDict[key][0] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to above cluster (small)
```

```
            ModelWeights[key] = 2 * diffFactor
        elif ModelDict[key][1] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to below cluster (small)
            ModelWeights[key] = 1 * diffFactor
        else:
            # Key belongs to either cluster (small)
            ModelWeights[key] = 0
    else:
        # Key belongs to its own cluster (enough observations)
        ModelWeights[key] = currLargeidx * diffFactor
        currLargeidx += 1

# Change the value of each model using the created dictionary
data["model"] = data["model"].map(lambda x: ModelWeights[x])
# Change the price to be 1 or 0 depending on price
data["price"] = data["price"].apply(lambda x: 1 if x < 18400 else 0)
# Change the year so that it the marginal year difference is worth more
data["year"] = data["year"].map(lambda x: x * (diffFactor / 10))

# =============== DATA TRAINING ===============


X_Train = [0] * 10
Y_Train = [0] * 10
X_Test = [0] * 10
Y_Test = [0] * 10

for i in range(0, 10):
    train, test = train_test_split(data, test_size=0.1)

    Y_Train[i] = train.iloc[:, 0]
    X_Train[i] = train.iloc[:, 1:]

    Y_Test[i] = test.iloc[:, 0]
    X_Test[i] = test.iloc[:, 1:]

neigh = KNeighborsClassifier(n_neighbors=19)

absError = 0
absCorrect = 0

predicted1Actual1 = 0
predicted1Actual0 = 0
predicted0Actual1 = 0
predicted0Actual0 = 0
for i in range(0, 10):
    neigh.fit(X_Train[i], Y_Train[i])
    predict = neigh.predict(X_Test[i])

    for idx in range(0, len(predict)):

        if predict[idx] == 0:
            if (Y_Test[i].iloc[idx] == 0):
                predicted0Actual0 += 1
            else:
                predicted0Actual1 += 1
        else:
            if (Y_Test[i].iloc[idx] == 0):
                predicted1Actual0 += 1
```

```
            else:
                predicted1Actual1 += 1

        # Calculations for Total Error
        if predict[idx] != Y_Test[i].iloc[idx]:
            absError += 1
        else:
            absCorrect += 1

print("Predicted:    0          1")
print("Actual [0]  " + str(round((predicted0Actual0 / (absCorrect + absError)) * 100, 2)) + "% " +
     str(
    round((predicted1Actual0 / (absCorrect + absError)) * 100, 2)) + "%")
print("Actual [1]  " + str(round((predicted0Actual1 / (absCorrect + absError)) * 100, 2)) + "% " +
     str(
    round((predicted1Actual1 / (absCorrect + absError)) * 100, 2)) + "%")



# =============== RUNNING THE PREDICTION ===============

Y_Train = data.iloc[:, 0]
X_Train = data.iloc[:, 1:]

neigh.fit(X_Train, Y_Train)

X_Test = pd.read_csv("Econ_424_F2023_PC3_test_without_response_variable.csv", low_memory=False)
X_Test.drop(columns=["city", "state", "make"], inplace=True)

def castModel(value ):
    if value == "mkxpremiere":
        return ModelWeights["mkxmkx"]
    if "s70" in value:
        return ModelWeights["s704dr"]
    if "lacrosse" in value:
        return ModelWeights["lacrossefwd"]
    if "routansel" in value:
        return ModelWeights["routan4dr"]
    if "caliber" in value:
        return 0
    if "eclipse" in value:
        return ModelWeights["eclipse2dr"]
    if "equinox" in value:
        return ModelWeights["equinoxawd"]
    if value in ModelWeights:
        return ModelWeights[value]
    else:
        # Assume it's in the pool of small sample observations
        return 0


X_Test["model"] = X_Test["model"].apply(lambda x: (str(x).lower()).split(".")[0])
X_Test["model"] = X_Test["model"].map(lambda x: castModel(x))
X_Test["year"] = X_Test["year"].map(lambda x: x * (diffFactor / 10))

estimates = neigh.predict(X_Test)

# Write the output
f = open('predictions.csv', 'w')
```

```python
for estimate in estimates:
    f.writelines(str(estimate) + ",\n")
```

Code for Q2:

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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
# Read in the data
data = pd.read_csv("Econ_424_F2023_PC3_training_small.csv", low_memory=False)

# =============== DATA PREPROCESSING ===============

# Modify some of the columns that we wont be using
data.drop(columns=["city", "state", "make"], inplace=True)

# Some models are the same but use weird capitalization or ., this fixes that
data["model"] = data["model"].apply(lambda x: (str(x).lower()).split(".")[0])

# This function will keep a count of each model of car if its below or above 18400
#   ModelDict[model] = (#num cars above 18400, #num cars below 18400)
ModelDict = {}


def classify(price, dep):
    # Cluster cars without considering values after the .
    dep = dep.split(".")[0]

    above = 0
    below = 0
    if price >= 18400:
        above = 1
    else:
        below = 1

    if not dep in ModelDict:
        ModelDict[dep] = (above, below)
    else:
        ModelDict[dep] = (ModelDict[dep][0] + above, ModelDict[dep][1] + below)


data.apply(lambda x: classify(x['price'], x['model']), axis=1)

# This is the factor we use for assigning a numeric value to each car model, since from
#   looking at the data the model has a much greater fit with the price then the milage the
#   difference in model is equal to a value greater than the difference in mileage
diffFactor = 10 ** (int(np.log10(data["mileage"].max())) + 1)

# Since the number of large clusters is dynamic we start at 3 and keep growing
currLargeidx = 3
ModelWeights = {}

for key in ModelDict:
    if ModelDict[key][0] + ModelDict[key][1] < 30:
        # Only < 30 observations, cluster with similar small sample models

        if ModelDict[key][0] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to above cluster (small)
            ModelWeights[key] = 2 * diffFactor
```

```
        elif ModelDict[key][1] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to below cluster (small)
            ModelWeights[key] = 1 * diffFactor
        else:
            # Key belongs to either cluster (small)
            ModelWeights[key] = 0
    else:
        # Key belongs to its own cluster (enough observations)
        ModelWeights[key] = currLargeidx * diffFactor
        currLargeidx += 1

# Change the value of each model using the created dictionary
data["model"] = data["model"].map(lambda x: ModelWeights[x])
# Change the price to be 1 or 0 depending on price
data["price"] = data["price"].apply(lambda x: 1 if x < 18400 else 0)
# Change the year so that it the marginal year difference is worth more
data["year"] = data["year"].map(lambda x: x * (diffFactor / 10))

# =============== DATA TRAINING ===============

trainingResults = []
testResults = []
valsOverK = []
for k in range(1, 100, 1):
    neigh = KNeighborsClassifier(n_neighbors=k,n_jobs=-1)

    train, test = train_test_split(data, test_size=0.1)

    Y_Train = train.iloc[:, 0]
    X_Train = train.iloc[:, 1:]

    Y_Test = test.iloc[:, 0]
    X_Test = test.iloc[:, 1:]

    neigh.fit(X_Train, Y_Train)

    testPass = 0
    testError = 0

    trainingPass = 0
    trainingError = 0

    # Run tests of training data
    trainingPredict = neigh.predict(X_Train)
    for idx in range(0, len(trainingPredict)):
        if trainingPredict[idx] != Y_Train.iloc[idx]:
            trainingError += 1
        else:
            trainingPass += 1

    # Run tests of test data
    testPredict = neigh.predict(X_Test)
    for idx in range(0, len(testPredict)):
        if testPredict[idx] != Y_Test.iloc[idx]:
            testError += 1
        else:
            testPass += 1

    # Push the results onto an array to be printed later
```

```
    trainingResults.append(trainingError/(trainingPass+trainingError))
    testResults.append(testError/(testPass+testError))
    valsOverK.append(1 / k)

    print("====" + str(k) + "====")
    print("Training Results: " + str(trainingError/(trainingPass+trainingError)))
    print("Test Results: " + str(testError/(testPass+testError)))

plt.plot(valsOverK, trainingResults, label="Training Error")
plt.plot(valsOverK, testResults, label="Test Error")
plt.xlabel("1/K")
plt.ylabel("Error Rate")
plt.title("Effect of K on Test and Training Error Rate")
plt.legend()
plt.show()
```

Code for Q3:

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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix
# Read in the data
data = pd.read_csv("Econ_424_F2023_PC3_training_small.csv", low_memory=False)

# =============== DATA PREPROCESSING ===============

# Modify some of the columns that we wont be using
data.drop(columns=["city", "state", "make"], inplace=True)

# Some models are the same but use weird capitalization or ., this fixes that
data["model"] = data["model"].apply(lambda x: (str(x).lower()).split(".")[0])

# This function will keep a count of each model of car if its below or above 18400
#   ModelDict[model] = (#num cars above 18400, #num cars below 18400)
ModelDict = {}


def classify(price, dep):
    # Cluster cars without considering values after the .
    dep = dep.split(".")[0]

    above = 0
    below = 0
    if price >= 18400:
        above = 1
    else:
        below = 1

    if not dep in ModelDict:
        ModelDict[dep] = (above, below)
    else:
        ModelDict[dep] = (ModelDict[dep][0] + above, ModelDict[dep][1] + below)


data.apply(lambda x: classify(x['price'], x['model']), axis=1)

# This is the factor we use for assigning a numeric value to each car model, since from
#  looking at the data the model has a much greater fit with the price then the milage the
#  difference in model is equal to a value greater than the difference in mileage
diffFactor = 10 ** (int(np.log10(data["mileage"].max())) + 1)

# Since the number of large clusters is dynamic we start at 3 and keep growing
currLargeidx = 3
ModelWeights = {}

for key in ModelDict:
    if ModelDict[key][0] + ModelDict[key][1] < 30:
        # Only < 30 observations, cluster with similar small sample models

        if ModelDict[key][0] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to above cluster (small)
```

```
            ModelWeights[key] = 2 * diffFactor
        elif ModelDict[key][1] / (ModelDict[key][0] + ModelDict[key][1]) > 0.9:
            # Key belongs to below cluster (small)
            ModelWeights[key] = 1 * diffFactor
        else:
            # Key belongs to either cluster (small)
            ModelWeights[key] = 0
    else:
        # Key belongs to its own cluster (enough observations)
        ModelWeights[key] = currLargeidx * diffFactor
        currLargeidx += 1

# Change the value of each model using the created dictionary
data["model"] = data["model"].map(lambda x: ModelWeights[x])
# Change the price to be 1 or 0 depending on price
data["price"] = data["price"].apply(lambda x: 1 if x < 18400 else 0)
# Change the year so that it the marginal year difference is worth more
data["year"] = data["year"].map(lambda x: x * (diffFactor / 10))

# =============== DATA TRAINING ===============

trainingResults = []
testResults = []
valsOverK = []
neigh = KNeighborsClassifier(n_neighbors=19,n_jobs=-1)

train, test = train_test_split(data, test_size=0.1)

Y_Train = train.iloc[:, 0]
X_Train = train.iloc[:, 1:]

Y_Test = test.iloc[:, 0]
X_Test = test.iloc[:, 1:]

neigh.fit(X_Train, Y_Train)

y_scores = neigh.predict_proba(X_Test)[:,1]
# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(Y_Test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()

# Define a range of thresholds to test
thresholds = np.linspace(0.0, 0.5, 100)

# Initialize lists to store error rates
total_error = []
false_negative = []
```

```
false_positive = []

# Calculate error rates for different thresholds
for threshold in thresholds:
    y_pred = (neigh.predict_proba(X_Test)[:, 1] >= threshold).astype(int)
    error_rate = 1 - accuracy_score(Y_Test, y_pred)
    total_error.append(error_rate)

    tn, fp, fn, tp = confusion_matrix(Y_Test, y_pred).ravel()
    false_negative_rate = fn / (tp)
    false_postive_rate = fp / (fp + tn)

    false_negative.append(false_negative_rate)
    false_positive.append(false_postive_rate)

# Plot the relationship between threshold and error rate
plt.plot(thresholds, total_error)
plt.plot(thresholds, false_negative)
plt.plot(thresholds, false_positive)
plt.xlabel('Threshold')
plt.ylabel('Error Rate')
plt.title('Error Rate vs. Threshold for k-NN Classifier')
plt.grid(True)
plt.show()
```