TheDe-bugger

Prediction Accuracy in Training Set:

0.83169

Confusion Matrix for Training Set:

Truth 0 1

Predicted

0 7965 2230

1 2190 7615

Question 2:

A graph with a line and a blue line

Description automatically generated

Code:

error\_training\_rate = []

error\_test\_rate = []

k\_ratios = []

bestK = 3

minError = error\_rate

bestModel = knn

for K in range(1,25,2):

knn = KNeighborsClassifier(n\_neighbors=K)

# Test Predictions

knn\_pred = knn.fit(X\_train, y\_train).predict(X\_test)

C = confusion\_table(knn\_pred,y\_test)

total\_predictions = C[0][0] + C[0][1] + C[1][0] + C[1][1]

false\_positives = C[0][1]

false\_negatives = C[1][0]

curr\_error = (false\_positives + false\_negatives) / total\_predictions

error\_test\_rate.append(curr\_error)

if (curr\_error < minError):

minError = curr\_error

bestK = K

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

# Training Predictions

knn\_pred\_train = knn.fit(X\_train, y\_train).predict(X\_train)

C = confusion\_table(knn\_pred\_train,y\_train)

total\_predictions = C[0][0] + C[0][1] + C[1][0] + C[1][1]

false\_positives = C[0][1]

false\_negatives = C[1][0]

error\_training\_rate.append((false\_positives + false\_negatives) / total\_predictions)

k\_ratios.append(1/K)

print("Best K:" + str(bestK))

print("Error Rate: " + str(minError))

# Create a double line graph

plt.figure(figsize=(10, 6))

# Plot train error rates

plt.plot(k\_ratios, error\_training\_rate, label='Train Error Rate', color='blue')

# Plot test error rates

plt.plot(k\_ratios, error\_test\_rate, label='Test Error Rate', color='red')

# custom\_ticks = [0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.6, 0.9, 1.0]

# plt.xticks(custom\_ticks, custom\_ticks)

# plt.rcParams["figure.autolayout"] = True

# Add labels and legend

plt.xlabel('1/K')

plt.ylabel('Error Rate')

plt.title('Train vs. Test Error Rates')

plt.legend()

# Show the plot

plt.grid()

plt.show()

Question 3:

A graph of a graph of a number of error

Description automatically generated with medium confidence

Code:

# Question 3)

# Using training data only, draw a figure with:

# 1 and 2 types of error rates as a function of classification threshold

# 3) the overall error rate as a function of the classification threshold

# Replicate figure 4.7 for training set

thresholds = np.arange(0, 1.02, 0.02)

knn = KNeighborsClassifier(n\_neighbors=20)

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

tpr\_test = []

fpr\_test = []

fnr\_test = []

error\_rate\_test = []

y\_test\_probs = knn.predict\_proba(X\_test)[:, 1]

for threshold in thresholds:

# Apply threshold to probabilities to get binary predictions

y\_test\_pred = (y\_test\_probs >= threshold).astype(int)

# Calculate confusion matrices

cm\_test = confusion\_table(y\_test, y\_test\_pred)

# Calculate TPR, FPR, and error rates

tn\_test, fp\_test, fn\_test, tp\_test = [cm\_test[0][0], cm\_test[0][1], cm\_test[1][0], cm\_test[1][1]]

tpr\_test.append(tp\_test / (tp\_test + fn\_test))

fpr\_test.append(fp\_test / (fp\_test + tn\_test))

fnr\_test.append(fn\_test / (fn\_test + tp\_test))

error\_rate\_test.append((fp\_test + fn\_test) / len(y\_test))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(thresholds, error\_rate\_test, label='Overall Error Rate', color='black')

plt.plot(thresholds, fnr\_test, label='False Neg Error Rate', color='orange')

plt.plot(thresholds, fpr\_test, label='False Pos Error Rate', color='blue')

plt.xlim(0, 0.7)

plt.xlabel('Classification Threshold')

plt.ylabel('Error Rate')

plt.legend()

# Plot the ROC curve:

plt.subplot(1, 2, 2)

plt.plot(fpr\_test, tpr\_test, label='Test ROC', color='red')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend()

plt.show()

Code for Questions 1 and 2)

# %%

# Import Statements for everything we use

import numpy as np

import pandas as pd

from matplotlib.pyplot import subplots

import statsmodels.api as sm

from ISLP import load\_data

from ISLP.models import (ModelSpec as MS, summarize)

#

from ISLP import confusion\_table

from ISLP.models import contrast

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from matplotlib.pyplot import subplots

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc, classification\_report, confusion\_matrix

from sklearn.preprocessing import LabelEncoder

# %%

# utilize either training set to predict whether car price < 18,400 = 1 or 0 otherwise

categorical\_columns = ['city', 'state', 'make', 'model']

carLarge = pd.read\_csv('./Econ\_424\_F2023\_PC3\_training\_small.csv', dtype={col: 'category' for col in categorical\_columns})

carLarge

# %%

# label encoder to encode the different categorical features

label\_encoder = LabelEncoder()

for category in categorical\_columns:

carLarge[category] = label\_encoder.fit\_transform(carLarge[category])

# %%

# create the binary variable

carLarge['target'] = carLarge['price'].apply(lambda x: 1 if x < 18400 else 0)

carLarge

y = carLarge['target'].values

# %%

# Features

# 'city','make','model','state'

features = carLarge.drop(columns=['price','target'])

print(features)

scaler = StandardScaler(with\_mean=True, with\_std=True,copy=True)

scaler.fit(features)

X\_std = scaler.transform(features)

feature\_std = pd.DataFrame( X\_std ,columns=features.columns)

print("feature std:")

print(feature\_std.std())

print(feature\_std)

# regular

X = features.values

print("printing regular features")

print(X)

# %%

# Split data

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(feature\_std, y, test\_size=0.2, random\_state=42)

# %%

# Create and Train KNN

knn = KNeighborsClassifier(n\_neighbors=3)

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

# %%

# Predictions

y\_pred = knn.predict(X\_test)

# %%

# Model accuracy:

c = confusion\_table(y\_pred,y\_test)

print(c)

# %%

from sklearn.metrics import accuracy\_score

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

false\_positives = c[0][1]

false\_negatives = c[1][0]

error\_rate =(false\_positives + false\_negatives) / len(y\_test)

print(error\_rate)

# %% [markdown]

#

# %%

# Question 2)

# using the training data (large or small), draw a graph that replicates the pattern in 2.17

error\_training\_rate = []

error\_test\_rate = []

k\_ratios = []

bestK = 3

minError = error\_rate

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

for K in range(1,25,2):

knn = KNeighborsClassifier(n\_neighbors=K)

# Test Predictions

knn\_pred = knn.fit(X\_train, y\_train).predict(X\_test)

C = confusion\_table(knn\_pred,y\_test)

total\_predictions = C[0][0] + C[0][1] + C[1][0] + C[1][1]

false\_positives = C[0][1]

false\_negatives = C[1][0]

curr\_error = (false\_positives + false\_negatives) / total\_predictions

error\_test\_rate.append(curr\_error)

if (curr\_error < minError):

minError = curr\_error

bestK = K

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

# Training Predictions

knn\_pred\_train = knn.fit(X\_train, y\_train).predict(X\_train)

C = confusion\_table(knn\_pred\_train,y\_train)

total\_predictions = C[0][0] + C[0][1] + C[1][0] + C[1][1]

false\_positives = C[0][1]

false\_negatives = C[1][0]

error\_training\_rate.append((false\_positives + false\_negatives) / total\_predictions)

k\_ratios.append(1/K)

print("Best K:" + str(bestK))

print("Error Rate: " + str(minError))

# %%

# %%

# Show that KNN estimation has following pattern:

# 1) training error decreases as 1/K increases

# 2) test error first decreases and then increases as 1/K (flexibility) increases

# Create a double line graph

plt.figure(figsize=(10, 6))

# Plot train error rates

plt.plot(k\_ratios, error\_training\_rate, label='Train Error Rate', color='blue')

# Plot test error rates

plt.plot(k\_ratios, error\_test\_rate, label='Test Error Rate', color='red')

plt.xlabel('1/K')

plt.ylabel('Error Rate')

plt.title('Train vs. Test Error Rates')

plt.legend()

# Show the plot

plt.grid()

plt.show()

# %%

# Read in test dataset

testData = pd.read\_csv('./Econ\_424\_F2023\_PC3\_test\_without\_response\_variable.csv')

# label encoder to encode the different categorical features

label\_encoder = LabelEncoder()

for category in categorical\_columns:

testData[category] = label\_encoder.fit\_transform(testData[category])

featuresNew = testData

print(featuresNew)

# Scaler stuff

scalerNew = StandardScaler(with\_mean=True, with\_std=True,copy=True)

scalerNew.fit(featuresNew)

X\_stdNew = scalerNew.transform(featuresNew)

feature\_std\_new = pd.DataFrame(X\_stdNew ,columns=featuresNew.columns)

print("feature std:")

print(feature\_std\_new.std())

print(feature\_std\_new)

# regular

print("printing regular features")

print(X)

# Make Predictions on the Test Set

Y\_Test = bestModel.predict(feature\_std\_new)

# %%

print(Y\_Test)

print(len(Y\_Test))

# %%

# output prediction results in csv

# Specify the file path where you want to save the CSV

csv\_file = "./output.csv"

# Use numpy.savetxt to save the array as a CSV file

np.savetxt(csv\_file, Y\_Test, delimiter="\n", fmt="%1d")

# %%

Y\_Train = bestModel.predict(feature\_std)

print(accuracy\_score(Y\_Train,y))

C = confusion\_table(Y\_Train,y)

total\_predictions = C[0][0] + C[0][1] + C[1][0] + C[1][1]

false\_positives = C[0][1]

false\_negatives = C[1][0]

curr\_error = (false\_positives + false\_negatives) / total\_predictions

error\_test\_rate.append(curr\_error)

# %%

print(c)

Question 4:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated