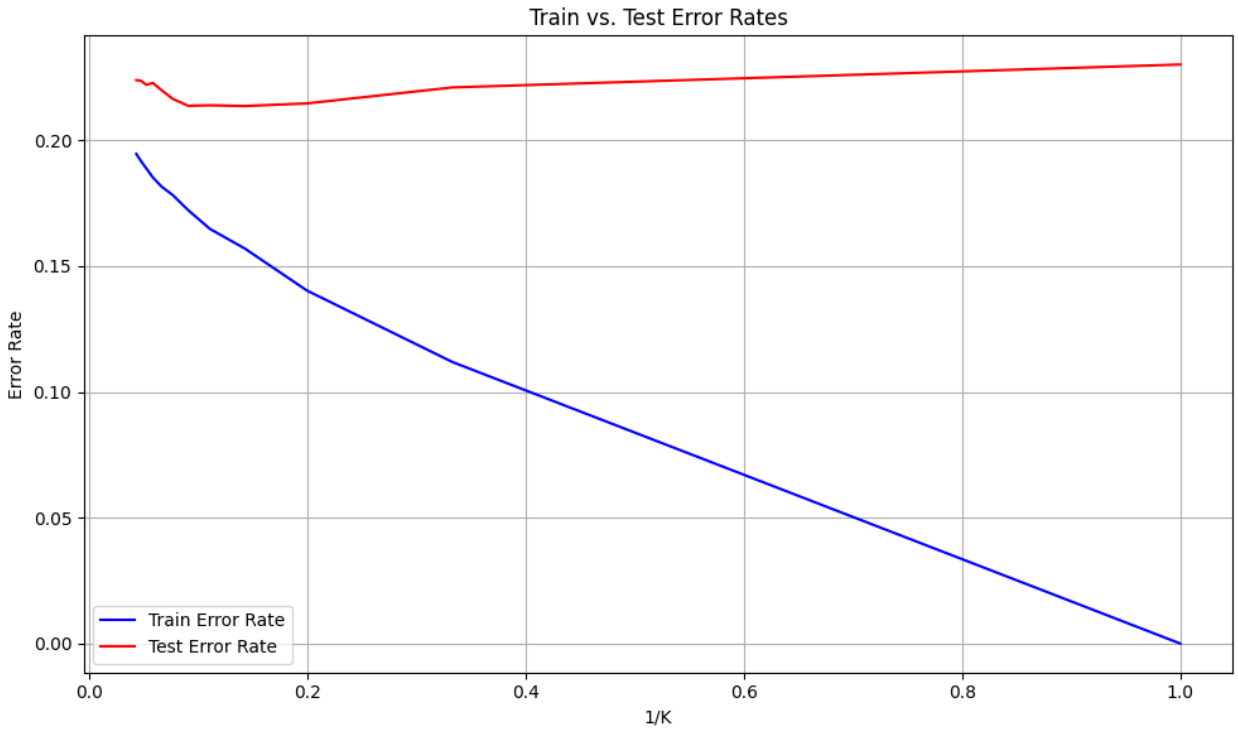
TheDe-bugger

Prediction Accuracy in Training Set:
0.83169

Confusion Matrix for Training Set:

Truth	0	1
Predicted		
0	7965	2230
1	2190	7615

Question 2:



Code:

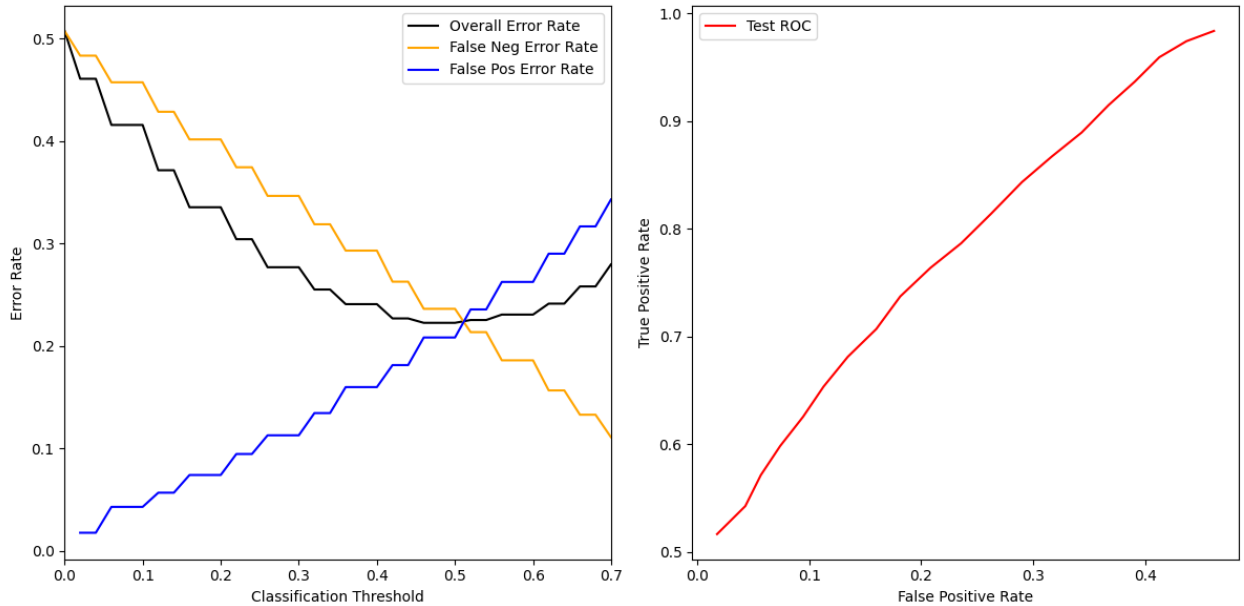
```
error_training_rate = []
error_test_rate = []
k_ratios = []
bestK = 3
minError = error_rate
bestModel = knn
for K in range(1,25,2):
```

```
    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:



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)
```

```python
    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:

I am working on a new project that involves using KNN model to classify whether a car's price will be above $18,400 or below. I am given two datasets with the same features, but one has 100,000 points and one has 500,000. The data includes:
price ($), year, mileage, city, state, make, and model. How should I create a KNN classifier for this data in python to classify as required?

To create a K-Nearest Neighbors (KNN) classifier to predict whether a car's price will be above $18,400 or below, you can follow these steps in Python:

1. **Data Preparation:**
   - Load both datasets (100,000 points and 500,000 points).
   - Combine or concatenate them into a single dataset if they have the same features.
   - Preprocess the data by handling missing values, encoding categorical features (e.g., one-hot encoding), and scaling numerical features. Ensure that the target variable ("above $18,400" or "below $18,400") is properly encoded (e.g., as 0 or 1).
2. **Data Splitting:**
   - Split the combined dataset into training and testing sets. Typically, you would use a portion of the data for training (e.g., 80%) and the rest for testing (e.g., 20%).
3. **Model Training:**
   - Import the scikit-learn library: `from sklearn.neighbors import KNeighborsClassifier`.
   - Create an instance of the KNN classifier, specifying the number of neighbors (K) and other hyperparameters. You may need to experiment with different values of K.
   - Train the classifier on the training data using the `fit()` method.

Regenerate

How can I preprocess the data to use columns that are classes and not continuous integer values

When you have categorical (class) features in your dataset, you need to preprocess them before using them with machine learning algorithms like K-Nearest Neighbors (KNN). The process typically involves encoding categorical features into a numerical format. Here's how you can preprocess categorical features:

1. **Label Encoding:** If a categorical feature has ordinal relationships (meaning there's an inherent order among categories), you can use label encoding. Label encoding assigns a unique integer to each category. Scikit-learn provides the `LabelEncoder` for this purpose.
2. **One-Hot Encoding:** If the categorical feature has no ordinal relationship among categories (nominal), it's better to use one-hot encoding. One-hot encoding creates binary columns for each category, representing the presence or absence of a category. Scikit-learn provides `OneHotEncoder` for this purpose.

Here's an example of how you can preprocess categorical features using one-hot encoding:

python Copy code

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# Load your dataset
# ...

# Select the categorical features
```

When I try loading the data with pd.read_csv, it does not read in the categorical variables. How can I adjust the code to allow this

When you load data using `pd.read_csv`, by default, it may not recognize categorical variables and treat them as regular columns with string values. To work with categorical variables in pandas, you can specify the data type of those columns as "category" using the `dtype` parameter. Here's how to do it:

python Copy code

```python
import pandas as pd

# Load your dataset and specify categorical columns
# Replace 'categorical_columns' with the actual names of your categorical co
categorical_columns = ['city', 'state', 'make', 'model']
df = pd.read_csv('your_dataset.csv', dtype={col: 'category' for col in categ

# Now, these columns will be treated as categorical variables
```

In this code:

1. Replace `'your_dataset.csv'` with the actual path to your dataset.
2. `dtype={col: 'category' for col in categorical_columns}` is a dictionary comprehension that specifies the data type for each column in `categorical_columns` as 'category'. This tells pandas to treat those columns as categorical variables.

After loading the data with this adjustment, you can proceed with one-hot encoding or other preprocessing steps for your categorical variables as needed.

Regenerate