Kyle Tranfaglia

COSC 311

Homework03

23 Apr. 2024

Code and Results

Task 1 Code:

```
'''
Kyle Tranfaglia
COSC311 - Homework03
Last updated 04/23/24
Task 1: Regression on the Computer Hardware Dataset
This program reads in a computer hardware dataset, then measures the
correlation between each attribute and the "ERP" to extract the four
best attributes. With these attributes and the "ERP," the data is randomly
split into 60% training data and 40% testing data. Finally, with
the training data, a multiple linear regression model is built and
evaluated using the testing data. The results show the MAE, MSE, and RMSE.
Task 2: Clustering on Hand-Written Digits
The program reads in the UCI ML hand-written digits dataset, then conducts
a PCA analysis on the dataset and finds m, which is how many
principal components are needed to keep at least 85% variance. Next, the
dataset is transformed from 64 dimensions to m dimensions. With the
dimension-reduced dataset k-means clustering is conducted and the center
of each cluster is displayed. Finally, the learned label is matched
to the true label, and then clustering accuracy is calculated and the
corresponding confusion matrix is displayed.
'''
import stats
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
accuracy_score
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from scipy.stats import mode

# Main

# Task 1: Regression on the Computer Hardware Dataset
hardwareData = pd.read_csv('machine.data')  # Read in data form csv file

# Complete attribute list & statistical attribute list
all_attributes = ['vendor', 'model', 'MYCT', 'MMIN', 'MMAX', 'CACH',
'CHMIN', 'CHMAX', 'PRP']
stat_attributes = all_attributes[2:]
 # Set up a list to store column labels (numbered columns)
hardwareData.columns = [attribute for attribute in all_attributes] +
["ERP"]

correlation_dic = {}
print("\nAttributes correlation coefficent to ERP")
# Get a list and print each attributes correlation coefficent to ERP
for i in stat_attributes:
    correlation = round(stats.correlation(hardwareData['ERP'],
hardwareData[i]), ndigits=6)
    correlation_dic[i] = correlation
    print(i + " Correlation Coefficient to ERP: " + str(correlation))

# Get sorted list of highest correlations to lowest as a key value pair,
sorted by value
sorted_correlation = dict(sorted(correlation_dic.items(), key=lambda item:
item[1], reverse=True))
top_attributes = list(sorted_correlation.keys())[:4]  # Attribute names of
top 4 correlations
print("Top Attributes:", top_attributes)

# Extract features and target data
attribute_data = hardwareData[top_attributes]
target_data = hardwareData['ERP']
```

```
# Split the data into training and testing sets, 60% training, 40% testing
attribute_train, attribute_test, target_train, target_test =
train_test_split(attribute_data, target_data, test_size=0.4,
random_state=7)

# Fit multiple linear regression model
lrModel = LinearRegression()
lrModel.fit(attribute_train, target_train)

# Predict target variable using testing data & generate a comparative
table for target predictions and actual
predictions = lrModel.predict(attribute_test)
comparison = pd.DataFrame({"Prediction":predictions,
"Actual":target_test})

# Calculate evaluation metrics: MAE, MSE, RMSE
mae = round(mean_absolute_error(target_test, predictions), ndigits=6)
mse = round(mean_squared_error(target_test, predictions), ndigits=6)
rmse = round(np.sqrt(mse), ndigits=6)

# Print the evaluation metrics
print("\nEvaluation Matrics")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print(comparison)
```

Task 1 Results:

```
Attributes correlation coefficent to ERP
MYCT Correlation Coefficient to ERP: -0.287806
MMIN Correlation Coefficient to ERP: 0.823113
MMAX Correlation Coefficient to ERP: 0.90418
CACH Correlation Coefficient to ERP: 0.687428
CHMIN Correlation Coefficient to ERP: 0.610094
CHMAX Correlation Coefficient to ERP: 0.606281
PRP Correlation Coefficient to ERP: 0.966423
Top Attributes: ['PRP', 'MMAX', 'MMIN', 'CACH']

Evaluation Matrics
Mean Absolute Error (MAE): 21.968081
Mean Squared Error (MSE): 1271.123984
Root Mean Squared Error (RMSE): 35.652826
     Prediction  Actual
11    56.629422      70
128   91.153224      82
158    9.636640      26
98   -14.458752      15
131   21.122990      46
..          ...     ...
202   10.074498      24
49    25.509812      34
106   -4.989059      18
5    397.615598     381
17    12.325610      22

[84 rows x 2 columns]
```

Task 2 Code:

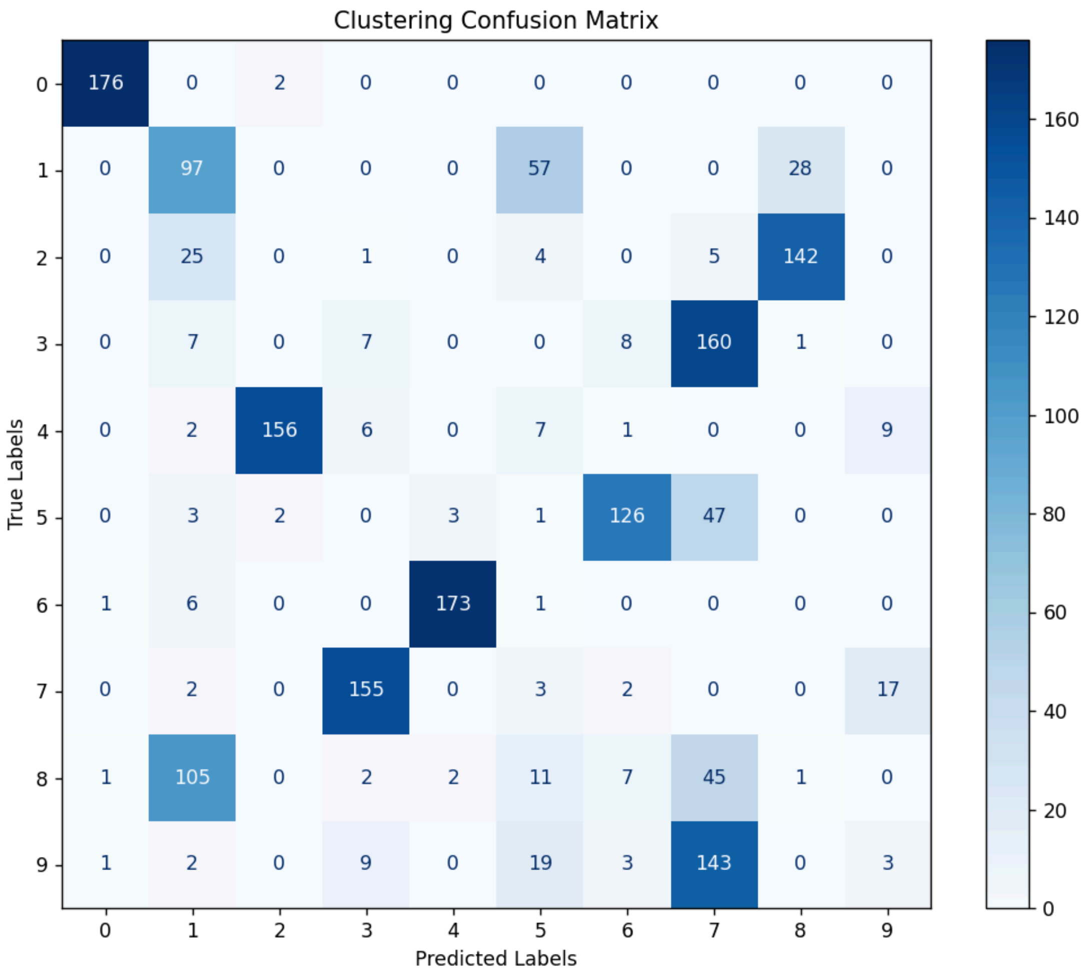
```
# Task 2: Clustering on Hand-Written Digits

# Load the hand-written digits dataset
digits = load_digits()
digits_data = digits.data

# Standardize the matrix
```

```
digits_mean = np.mean(digits_data, axis=0)
digits_std = np.std(digits_data, axis=0)
digits_std[digits_std == 0] = 1e-10  # Change stds of 0 to a small value
to avoid division by zero
digits_normalized = (digits_data - digits_mean) / digits_std

# Perform Principal Component Analysis (PCA) with normalized data
pca_normalized = PCA(n_components=0.85, svd_solver='full')
digits_data_new = pca_normalized.fit_transform(digits_normalized)

# Get the covariance matrix with normalized data
covariance_matrix = pca_normalized.get_covariance()

# Dsiplay covariance matrix with normalized data
print("PCA Analysis with normalized data")
print("Covariance Matrix:")
print(covariance_matrix)

# Number of principal components required to keep at least 85% variance
with normalized data
print("Percentage of variance explained by each component to the total
variance:\n", pca_normalized.explained_variance_ratio_)
print(f"Total explained variance ratio:
{np.sum(pca_normalized.explained_variance_ratio_):.2f}")
print(f"Number of principal components to keep at least 85% variance:
{pca_normalized.n_components_}")

# Perform Principal Component Analysis (PCA) without normalized data
pca_unnormalized  = PCA(n_components=0.85, svd_solver='full')
digits_data_new = pca_unnormalized.fit_transform(digits_data)

# Get the covariance matrix without normalized data
covariance_matrix = pca_unnormalized.get_covariance()

# Dsiplay covariance matrix without normalized data
print("PCA Analysis without normalized data")
print("Covariance Matrix:")
print(covariance_matrix)
```

```python
# Number of principal components required to keep at least 85% variance
without normalized data
print("Percentage of variance explained by each component to the total
variance:\n", pca_unnormalized.explained_variance_ratio_)
print(f"Total explained variance ratio:
{np.sum(pca_unnormalized.explained_variance_ratio_):.2f}")
print(f"Number of principal components to keep at least 85% variance:
{pca_unnormalized.n_components_}")

# Perform PCA
pca_final = PCA(n_components=17)
digits_data_transformed = pca_final.fit_transform(digits_normalized)

# Check the shape of the transformed dataset
print("Shape of the transformed dataset:", digits_data_transformed.shape)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(digits_data_transformed, digits.target, test_size=0.3,
random_state=7)

# Initialize the KNeighborsClassifier object
knn_pca = KNeighborsClassifier(n_neighbors=4)  # Use number of samples in
the training set as neighbors

# Fit the model to the training data using the actual target labels
knn_pca.fit(X_train, y_train)

# Output the center of each cluster
print("Center of each cluster (each cluster represents a digit):",
knn_pca._fit_X)
print(f"Train score after PCA: {knn_pca.score(X_train, y_train):.6f}")
print(f"Test score after PCA: {knn_pca.score(X_test, y_test):.6f}")

# Perform k-means clustering
kmeans = KMeans(n_clusters=10, random_state=7)
cluster_labels = kmeans.fit_predict(digits_data_transformed)

# Output the center of each cluster
print("Center of each cluster (each cluster represents a digit):")
```

```python
for i, center in enumerate(kmeans.cluster_centers_):
    print(f"Cluster {i}: {center}")

# Determine the mapping between cluster labels and true labels
mapped_labels = np.zeros_like(cluster_labels)
for cluster in range(kmeans.n_clusters):
    mask = (cluster_labels == cluster)
    mapped_labels[mask] = mode(digits.target[mask])[0]

# Calculate and print clustering accuracy
print(f"Clustering Accuracy: {accuracy_score(digits.target, 
mapped_labels):.6f}")

# Generate confusion matrix
confusionMatrix = confusion_matrix(digits.target, cluster_labels)

# Visualize the confusion matrix using matplotlib and the sklearn toolset
matrixDisplay = ConfusionMatrixDisplay(confusion_matrix = confusionMatrix)
fig, ax = plt.subplots(figsize=(10, 8))  # Create layout and structure 
figure
matrixDisplay.plot(ax = ax, cmap = 'Blues')  # Create Plot
# Plot labels
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Clustering Confusion Matrix')
plt.show()
```

Task 2 Results:



```
PCA Analysis with normalized data
Covariance Matrix:
[[ 2.32672998e-01 -1.89023298e-18  2.37230314e-19 ... -2.89739575e-18
  -3.45952135e-18 -1.66238638e-18]
 [-1.89023298e-18  1.04601528e+00  5.71522491e-01 ... -2.42299737e-02
   3.69328794e-02 -5.80237064e-02]
 [ 2.37230314e-19  5.71522491e-01  1.06511834e+00 ... -6.01933565e-02
   5.15005840e-02  9.91746019e-02]
 ...
 [-2.89739575e-18 -2.42299737e-02 -6.01933565e-02 ...  1.04197066e+00
   5.98448559e-01  2.79612749e-01]
 [-3.45952135e-18  3.69328794e-02  5.15005840e-02 ...  5.98448559e-01
   1.01599890e+00  6.09960771e-01]
 [-1.66238638e-18 -5.80237064e-02  9.91746019e-02 ...  2.79612749e-01
   6.09960771e-01  9.68420404e-01]]
Percentage of variance explained by each component to the total variance:
 [0.12033916 0.09561054 0.08444415 0.06498408 0.04860155 0.0421412
 0.03942083 0.03389381 0.02998221 0.02932003 0.02781805 0.02577055
 0.02275303 0.0222718  0.02165229 0.01914167 0.01775547 0.01638069
 0.0159646  0.01489191 0.0134797  0.01271931 0.01165837 0.01057647
 0.00975316]
Total explained variance ratio: 0.85
Number of principal components to keep at least 85% variance: 25
PCA Analysis without normalized data
Covariance Matrix:
[[ 3.51466084e+00 -5.78718671e-18 -1.34994464e-18 ... -1.21002648e-16
  -9.91957719e-17 -1.98323628e-17]
 [-5.78718671e-18  3.80642973e+00  2.14257743e+00 ... -1.59024155e-01
   2.46854206e-01  8.01061478e-02]
 [-1.34994464e-18  2.14257743e+00  2.33472276e+01 ... -1.38499476e+00
   1.57421668e+00  8.59342271e-01]
 ...
 [-1.21002648e-16 -1.59024155e-01 -1.38499476e+00 ...  3.41832476e+01
   1.53235375e+01  2.92164014e+00]
 [-9.91957719e-17  2.46854206e-01  1.57421668e+00 ...  1.53235375e+01
   1.57795298e+01  3.25157243e+00]
 [-1.98323628e-17  8.01061478e-02  8.59342271e-01 ...  2.92164014e+00
   3.25157243e+00  4.74865119e+00]]
```

```
Percentage of variance explained by each component to the total variance:
 [0.14890594 0.13618771 0.11794594 0.08409979 0.05782415 0.0491691
 0.04315987 0.03661373 0.03353248 0.03078806 0.02372341 0.02272697
 0.01821863 0.01773855 0.01467101 0.01409716 0.01318589]
Total explained variance ratio: 0.86
Number of principal components to keep at least 85% variance: 17
Shape of the transformed dataset: (1797, 17)
Center of each cluster (each cluster represents a digit): [[-1.07607724  1.02693451 -3.7095651  ...  0.99711544 -0.05738125
  -1.08838584]
 [-0.9057334  -0.26877365 -3.69191049 ...  1.40689326 -0.15838315
  -0.80303456]
 [ 2.79939093  1.78631166  0.68346908 ... -1.39601811 -0.38813397
   0.60478539]
 ...
 [-2.45995488 -3.07434048  4.23014484 ...  2.18492706 -0.86217152
   0.0814921 ]
 [-3.40458428  2.28456177 -4.18285582 ... -1.18905552 -0.69699961
   1.29864589]
 [-3.20722079  2.57850779 -1.01280764 ... -0.56386439 -0.61526342
   0.7281569 ]]
Train score after PCA: 0.980111
Test score after PCA: 0.972222
```

```
Center of each cluster (each cluster represents a digit):
Cluster 0: [ 1.60966986 -1.94770327 -3.05747712  1.68096934 -0.56894269 -0.07060577
 -0.82314102  1.35608319  0.50921416 -0.50327314 -0.42640627 -0.66693033
  0.20020053 -0.34351293 -0.11778329 -0.12672838 -0.0312595 ]
Cluster 1: [-0.3693399   0.73823402  2.472536   -0.25441201 -1.07825773 -0.07497627
 -0.08956277  0.23331786  1.29890451 -0.78825066 -0.17586471  0.15092118
 -0.23552194 -0.31912395  0.2995587  -0.28499649  0.0303407 ]
Cluster 2: [ 4.67361772 -0.7277832   1.17389838  0.87552321 -1.04025634  0.53486127
  0.33862735  0.36177577 -1.2086851   1.15559656 -0.40528439  0.15752555
 -0.10470005  0.49032241  0.01214809  0.09451445 -0.10127997]
Cluster 3: [ 0.49934381  3.48611627  0.52965857  0.33179733 -0.30699327 -1.30832987
 -2.04528831 -0.45888482 -0.5306283   0.69876043  0.35040016  0.28289528
  0.19695147 -0.01935152  0.03329915  0.06655922  0.04600569]
Cluster 4: [ 1.26443313e+00 -3.90997240e+00  1.10866213e+00 -3.75694232e-02
  1.18900523e+00 -4.95481904e-01 -3.37622302e-01 -1.86896315e+00
  9.86591409e-02 -5.31074057e-01  6.66619395e-01 -1.01617187e-03
 -6.29217478e-03  3.82845561e-01 -2.35159778e-01  2.13903541e-02
  2.91584050e-01]
Cluster 5: [ 2.53470634  1.8304086  -0.06855766 -3.08566153 -0.50224337  0.082632
  2.33992335  0.17433677 -0.6479651  -0.13778239  0.73528003 -0.96025027
 -0.02177174 -1.10126042 -0.19332938  0.21132347  0.01419839]
Cluster 6: [-1.74191527  1.80888495  0.24312815  3.02830813  0.59820063 -0.21309834
  2.00390355 -0.38951752 -0.74622616 -0.43572851  0.28677337  0.44386957
  0.61947172 -0.05172179 -0.12809886 -0.13996518 -0.2840894 ]
Cluster 7: [-1.89325678 -0.11633451 -2.21035384 -1.30214649 -0.29858243  0.02795261
  0.24374614 -0.44578581  0.14763479  0.13967612 -0.15045132  0.4394422
 -0.01088541  0.27335446  0.10399118  0.10098093  0.00875286]
Cluster 8: [-3.85026106 -1.34706123  2.00135629 -0.21157078  1.50411192  0.87958113
 -0.35609637  1.76734882 -0.58329538  0.50308176 -0.03210574 -0.65304415
 -0.31639092 -0.03413211  0.02839608  0.00513745  0.02710358]
Cluster 9: [ 5.36687292  5.98373269 -1.29818434 -0.02600644  7.06569256  4.47483588
 -0.96979806 -1.02799094  2.56804714  0.20904576 -1.68808814 -0.82550128
 -0.85574594  0.53095589 -0.94217744  0.69898628 -0.47592506]
\\wsl.localhost\Ubuntu\home\kyletranfaglia\COSC311\Homework03\hardwareRecognition\main.py:161:
`mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change:
l be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or Fals
  mapped_labels[mask] = mode(digits.target[mask])[0]
Clustering Accuracy: 0.705064
```