ml-scratch

May 12, 2023

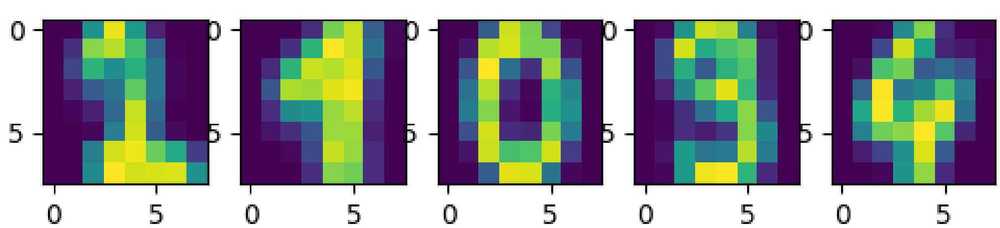
1 KMeans

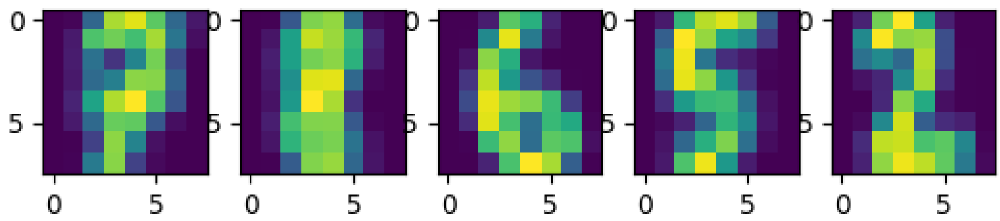
[ ]:
```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import mode
# Ignore this
from warnings import simplefilter
simplefilter(action = "ignore", category = FutureWarning)

digits = load_digits()
X, Y = digits.data, digits.target


def kmeans(X, k, max_iter):

  n = X.shape[0]
  centers = X[np.random.choice(n, k, replace = False)]
  ob_values = []

  for _ in range(max_iter):
    distances = np.linalg.norm(X[:, np.newaxis, :] - centers, axis = -1)
    labels = np.argmin(distances, axis = -1)

    for i in range(len(centers)):
      centers[i] = np.mean(X[labels == i], axis = 0)

    ob_val = 0

    for i in range(n):
      ob_val += (distances[i, labels[i]])**2
    ob_values.append(ob_val)

  # To account for permuted labels and Digits
  lab = np.zeros_like(labels)
```

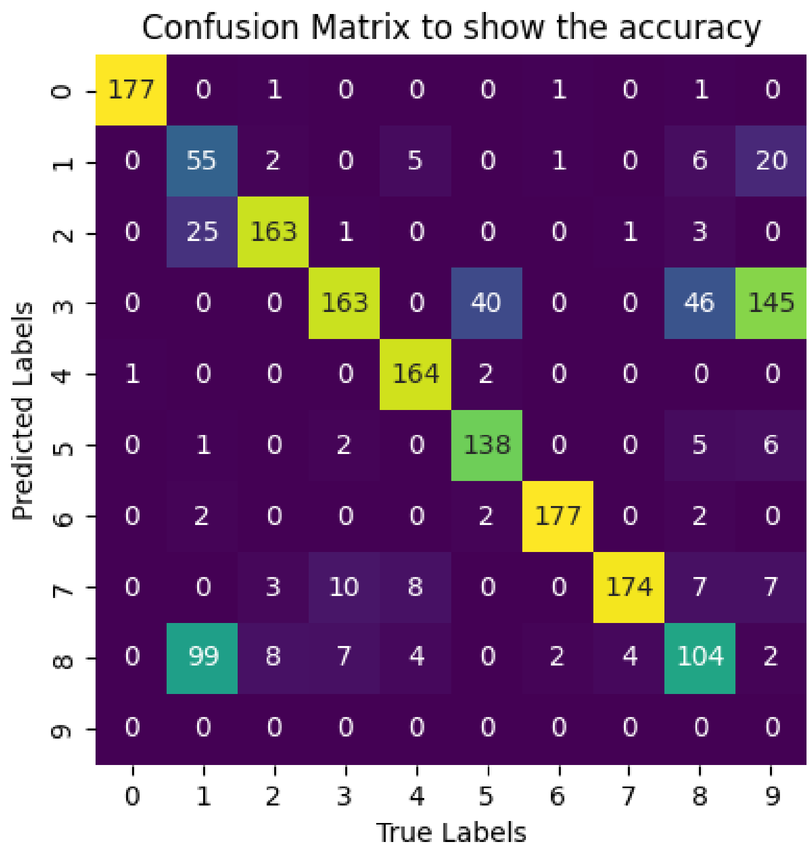
```
  for i in range(10):
    mask = (labels == i)
    lab[mask] = mode(Y[mask])[0]

  return lab, centers, ob_values



labels, centers, obj_values = kmeans(X, 10, 100)
print(obj_values)
print("Predicted Labels:")
print(labels)
print("\nActual Labels:")
print(Y)
print("\nThe Accuracy Score")
print(accuracy_score(labels, Y))

print("The Centers are:")
for i in range(5):
  plt.subplot(1, 5, i + 1)
  plt.imshow(centers[i].reshape((8, 8)))
plt.show()
for i in range(5, 10):
  plt.subplot(1, 5, i - 4)
  plt.imshow(centers[i].reshape((8, 8)))
plt.show()

cm = confusion_matrix(labels, Y)
sns.heatmap(cm, square = True, annot = True, fmt = "d", cbar = False,␣
 ↪xticklabels = digits.target_names, yticklabels = digits.target_names, cmap =␣
 ↪"viridis")
plt.xlabel("True Labels")
plt.ylabel("Predicted Labels")
plt.title("Confusion Matrix to show the accuracy")
plt.show()

for i in range(len(obj_values)):
    if obj_values[i] == obj_values[i + 1]:
        ind = i
        break
itr = np.arange(0, 100)
plt.plot(itr, obj_values, "green")
plt.axvline(x = ind, linestyle = "--", color = "red")
plt.legend(["Plot", f"KMeans becomes stable at iteration {i + 1}"])
plt.xlabel("Number of Iterations")
plt.ylabel("Objective Value")
plt.grid()
```

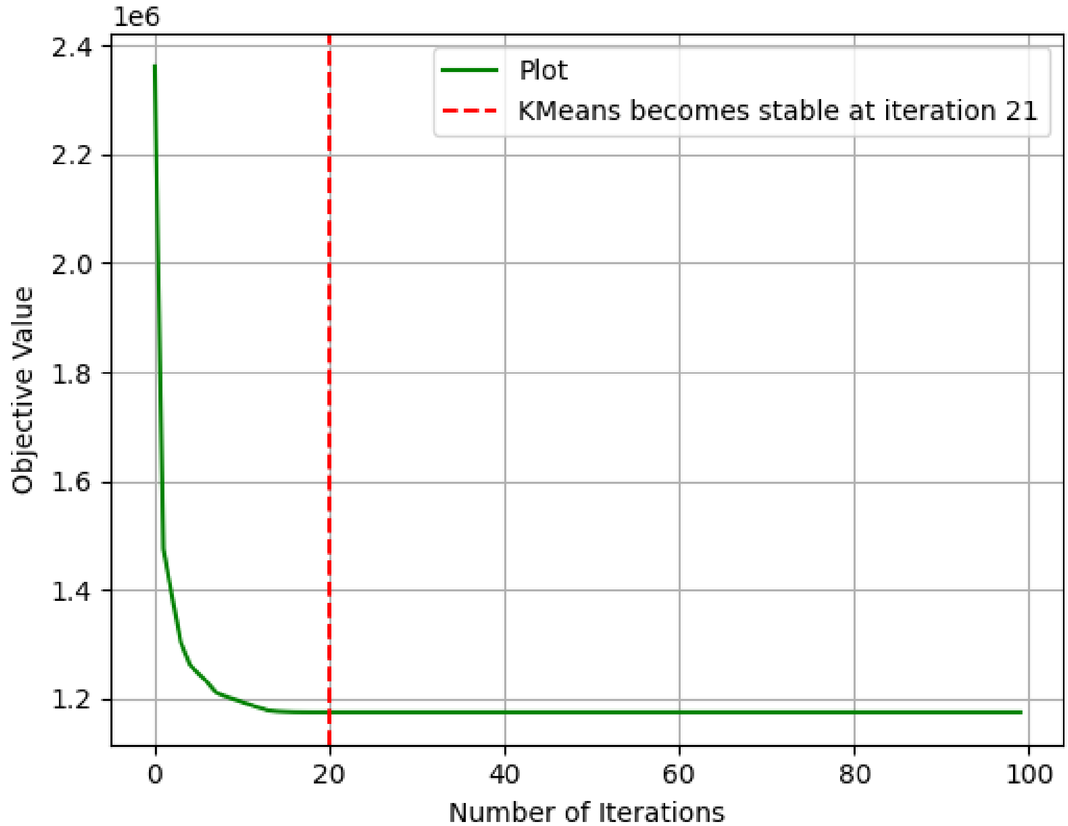
```
plt.show()
```

```
[2362362.0, 1474928.512307769, 1387431.075132158, 1303471.8857646866,
1263198.1141582918, 1245930.6202748832, 1230825.0261610376, 1211530.0066599704,
1204686.3103511613, 1199396.5071347163, 1193830.3471811896, 1188436.7839042586,
1183256.264533338, 1178400.6108346528, 1176725.8096872878, 1175915.4524736998,
1175333.4597189564, 1175189.1330287906, 1175091.2445854363, 1175073.7332125322,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916,
1175065.271077916, 1175065.271077916, 1175065.271077916, 1175065.271077916]
Predicted Labels:
[0 8 8 … 8 3 3]

Actual Labels:
[0 1 2 … 8 9 8]

The Accuracy Score
0.7317751808569839
The Centers are:
```






Confusion Matrix to show the accuracy
Predicted Labels
True Labels
177 0 1 0 0 0 1 0 1 0
0 55 2 0 5 0 1 0 6 20
0 25 163 1 0 0 0 1 3 0
0 0 0 163 0 40 0 0 46 145
1 0 0 0 164 2 0 0 0 0
0 1 0 2 0 138 0 0 5 6
0 2 0 0 0 2 177 0 2 0
0 0 3 10 8 0 0 174 7 7
0 99 8 7 4 0 2 4 104 2
0 0 0 0 0 0 0 0 0 0

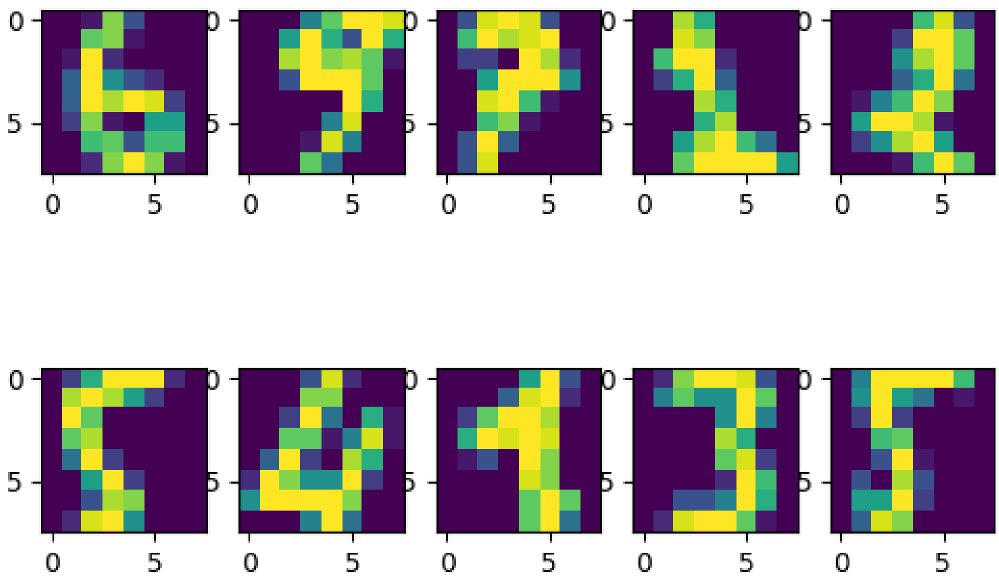


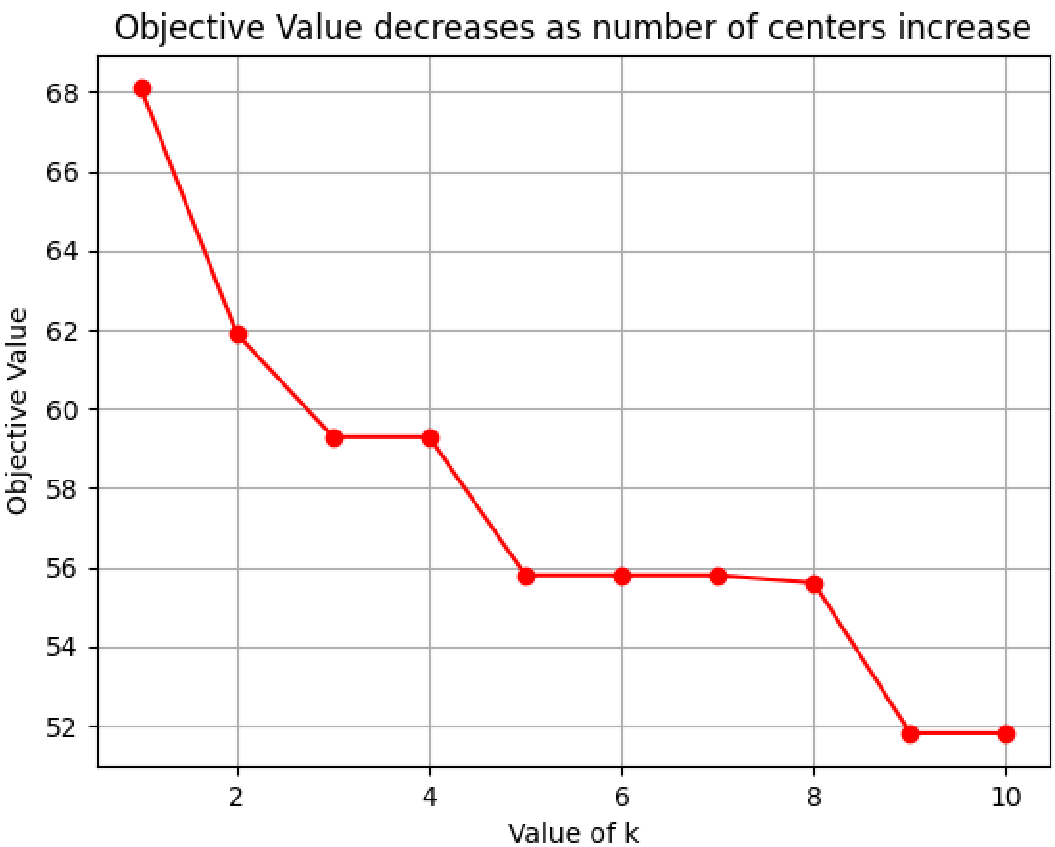
2 KCenter

[ ]:
```
from prettytable import PrettyTable
X, Y = digits.data, digits.target

def cluster(X, centers):
  distances = np.linalg.norm(X[:, np.newaxis, :] - centers, axis = -1)
  labels = np.argmin(distances, axis = 1)
  return distances, labels

def Kcost(X, centers):
  cost = 0
  dist, labels = cluster(X, centers)
  for i in range(len(labels)):
    cost = max(cost, dist[i, labels[i]])
  return cost

def kcenter(X, k):
  n = X.shape[0]
```

```python
  centers = [X[np.random.randint(n)]]
  indices = []
  costs = [Kcost(X, centers)]
  for i in range(k - 1):
    distances = np.linalg.norm(X[:, np.newaxis, :] - centers, axis = -1).
 ↪sum(axis = 1)
    new_center_index = np.argmax(distances, axis = 0)
    indices.append(new_center_index)
    centers.append(X[new_center_index])
    costs.append(Kcost(X, centers))
  return centers, indices, costs


centers, indices, costs = kcenter(X, 10)
print(Y[indices])
print("The Centers are:")
for i in range(5):
  plt.subplot(1, 5, i + 1)
  plt.imshow(centers[i].reshape((8, 8)))
plt.show()
for i in range(5, 10):
  plt.subplot(1, 5, i - 4)
  plt.imshow(centers[i].reshape((8, 8)))
plt.show()


k = np.arange(1, 11)
plt.plot(k, costs, "ro-")
plt.grid()
plt.xlabel("Value of k")
plt.ylabel("Objective Value")
plt.title("Objective Value decreases as number of centers increase")
plt.show()

table = PrettyTable()
table.field_names = ["k value", "Objective Value"]
for i in range(10):
  table.add_row([i +1, costs[i]])
print(table)
```

```
[9 7 1 2 5 4 1 3 5]
The Centers are:
```




Objective Value decreases as number of centers increase
Objective Value
Value of k
68
66
64
62
60
58
56
54
52
2
4
6
8
10

```
+---------+--------------------+
| k value |  Objective Value   |
+---------+--------------------+
|    1    | 68.11754546370561  |
|    2    | 61.89507250177513  |
|    3    | 59.28743543112655  |
|    4    | 59.28743543112655  |
|    5    | 55.794264938253285 |
|    6    | 55.794264938253285 |
|    7    | 55.794264938253285 |
|    8    | 55.59676249567055  |
|    9    | 51.807335387954474 |
|    10   | 51.807335387954474 |
+---------+--------------------+
```

3 K Nearest Neighbour Classifier

[ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from collections import Counter
iris = load_iris()
X, Y = iris.data, iris.target
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, train_size = 0.25)

k = 5
# y_model = []
# for x in Xtest:
#   dist = []
#   for c in Xtrain:
#     dist.append(np.linalg.norm(c - x))
#   indices = np.argsort(dist)[:k]
#   targets = Ytrain[indices]
#   predicted_label = Counter(targets).most_common(1)[0][0]
#   y_model.append(predicted_label)

# A Better Code:
def KNN(Xtrain, Ytrain, Xtest, k):
  dist = np.linalg.norm(Xtest[:, np.newaxis, :] - Xtrain, axis = -1)
  ind = np.argsort(dist, axis = 1)
  ind = ind[:, :k]
  K_classes = np.array([Ytrain[x] for x in ind])
  Y_model = np.array([Counter(x).most_common(1)[0][0] for x in K_classes])
  return Y_model

y_model = KNN(Xtrain, Ytrain, Xtest, k)

print("Predicted Labels:")
```

```python
print(np.array(y_model))
print("\nActual Labels:")
print(Ytest)
print("\nAccuracy Score:")
print(accuracy_score(y_model, Ytest))
```

```
Predicted Labels:
[2 0 2 2 1 1 2 0 1 2 2 2 0 1 1 2 2 0 2 0 0 1 2 2 1 2 1 1 1 0 2 2 0 2 1 1 1
 2 2 1 1 0 2 0 1 2 1 0 1 2 2 1 0 0 1 2 1 2 1 0 1 0 1 0 2 0 2 1 2 2 2 1 0 2
 0 1 0 0 1 0 2 0 0 2 1 0 0 2 0 0 0 0 0 2 2 2 0 1 1 2 1 2 2 0 0 1 0 0 1 1 0
 2 0]

Actual Labels:
[2 0 2 2 1 1 2 0 1 2 2 1 0 2 1 2 2 0 2 0 0 1 2 2 1 2 1 1 1 0 2 2 0 2 1 1 1
 2 2 1 1 0 2 0 1 2 1 0 1 2 2 1 0 0 2 2 1 2 1 0 1 0 1 0 2 0 2 1 2 2 2 1 0 2
 0 1 0 0 1 0 2 0 0 2 1 0 0 2 0 0 0 0 0 2 2 1 0 1 1 2 1 2 2 0 0 1 0 0 1 1 0
 2 0]

Accuracy Score:
0.9646017699115044
```

4 Gaussian Naive Bayes

[ ]:
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_wine

wine = load_wine()
X, Y = wine.data, wine.target

model = GaussianNB()
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, train_size = 0.25)
model.fit(Xtrain, Ytrain)
y_model = model.predict(Xtest)
print("Predicted Labels:")
print(np.array(y_model))
print("\nActual Labels:")
print(Ytest)
print("\nAccuracy Score:")
print(accuracy_score(y_model, Ytest))
```

```
Predicted Labels:
[1 0 0 0 1 1 1 0 0 0 0 1 1 0 1 1 1 2 2 1 1 1 2 2 0 1 0 0 0 1 0 2 1 1 1 2 1
 1 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1 2 0 1 2 2 1 2 2 2 0 1 1 0 1 1 1 2 1 0
 1 0 1 1 1 2 1 0 0 0 2 2 1 1 1 1 1 1 1 1 1 2 0 0 1 2 2 2 1 0 2 2 1 2 0 1 0
 0 2 1 0 2 1 1 1 1 2 0 0 2 2 2 0 0 0 1 1 2 0 1]

Actual Labels:
```

```
[1 0 0 0 1 1 1 0 0 0 0 1 1 0 1 1 1 2 2 1 1 0 2 2 0 1 0 0 0 1 0 2 1 1 1 2 1
 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 2 0 1 2 2 1 2 2 2 0 1 1 0 0 1 1 2 1 0
 1 0 1 1 1 2 0 0 0 0 2 2 0 1 1 1 1 1 1 1 1 2 0 0 1 2 2 2 1 0 2 2 1 2 0 1 0
 0 2 1 0 2 1 1 1 1 2 0 0 2 2 2 0 0 0 1 1 2 0 1]

Accuracy Score:
0.9552238805970149
```

5 Linear Regression

[ ]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
X, Y = diabetes.data, diabetes.target

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, train_size = 0.45)
model = LinearRegression()
model.fit(Xtrain, Ytrain)
y_model = model.predict(Xtest)
print("First 10 Predicted Labels:")
print(np.array(y_model[:10]))
print("\nFirst 10 Actual Labels:")
print(Ytest[:10])
print("\nRMSE:")
print(mean_squared_error(y_model, Ytest)**0.5)
```

```
First 10 Predicted Labels:
[207.06828293 145.74155297 181.70800133 232.18837179 145.15057859
 225.36980872 172.48912217 165.45871961 126.60576782 144.98762215]

First 10 Actual Labels:
[109.  60. 263. 261. 197.  99. 122.  91.  83.  73.]

RMSE:
54.76358134102201
```

6 Principal Component Analysis (PCA)

6.1 Scratch PCA

[ ]:
```python
def My_PCA(X, k):
  means = np.mean(X, axis = 0)
  stds = np.std(X, axis = 0)
  # Scaling the X: same as StandardScaler()
  My_X_scaled = (X - means)/stds
  # Computing the Covariance Matrix of Scaled X
```

```
    cov = np.cov(My_X_scaled.T)
    # Finding EigenValues and EigenVectors of Covariance Matrix
    values, vectors = np.linalg.eig(cov)
    #Adjusting the signs of the EigenVectors
    max_abs_idx = np.argmax(np.abs(vectors), axis=0)
    signs = np.sign(vectors[max_abs_idx, range(vectors.shape[0])])
    vectors = vectors*signs[np.newaxis,:]
    vectors = vectors.T
    # Finding the Indices of the largest EigenValues
    indices = np.argsort(values)[::-1]
    eig = vectors[indices]
    # Taking only the Top k EigenVectors
    W = eig[:k]
    # Finally projection of Original Scaled data using the Top k EigenVectors
    X_proj = My_X_scaled @ W.T
    return X_proj
```

6.2 Built-in PCA

[ ]:
```
X, Y = wine.data, wine.target
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
model = PCA(n_components = 2)
X2D = model.fit_transform(X_scaled)
print(X2D[:10])
print()
X_2D_My = My_PCA(X, 2)
X_2D_My[:,1] = - X_2D_My[:,1]
print(X_2D_My[:10])
X_filtered = model.inverse_transform(X2D)
print()
print(X_filtered[:10])
print()
print(X[:10])
```

```
[[ 3.31675081 -1.44346263]
 [ 2.20946492  0.33339289]
 [ 2.51674015 -1.0311513 ]
 [ 3.75706561 -2.75637191]
 [ 1.00890849 -0.86983082]
 [ 3.05025392 -2.12240111]
 [ 2.44908967 -1.17485013]
 [ 2.05943687 -1.60896307]
 [ 2.5108743  -0.91807096]
 [ 2.75362819 -0.78943767]]
```

```
[[ 3.31675081 -1.44346263]
 [ 2.20946492  0.33339289]
 [ 2.51674015 -1.0311513 ]
 [ 3.75706561 -2.75637191]
 [ 1.00890849 -0.86983082]
 [ 3.05025392 -2.12240111]
 [ 2.44908967 -1.17485013]
 [ 2.05943687 -1.60896307]
 [ 2.5108743  -0.91807096]
 [ 2.75362819 -0.78943767]]

[[ 1.17683758 -0.48854671  0.44943066 -0.80905314  0.90346271  1.40287378
   1.39791791 -0.9486178   1.09629808  0.47110942  0.58106277  1.01020946
   1.47780928]
 [ 0.15764475 -0.61672373 -0.10990684 -0.52523924  0.21383059  0.85030558
   0.93557863 -0.66919329  0.67940854 -0.37249229  0.74867543  0.88597056
   0.51191298]
 [ 0.8619575  -0.3851356   0.32075278 -0.61322768  0.66632506  1.06032437
   1.06095125 -0.72165424  0.8293466   0.32348051  0.45881977  0.77709516
   1.09795087]
 [ 1.87537855 -0.30119251  0.86349723 -0.92833383  1.35937617  1.66203977
   1.57973101 -1.04228148  1.28590532  1.12792641  0.34510016  0.95987296
   2.08315485]
 [ 0.56631018 -0.05171997  0.27285706 -0.25066434  0.40388787  0.45475005
   0.42377954 -0.2761593   0.35040752  0.37160042  0.05647051  0.23643464
   0.60671049]
 [ 1.46674389 -0.27049071  0.66456854 -0.7524653   1.06905533  1.34185572
   1.28292613 -0.84952015  1.03945355  0.85455995  0.31240577  0.79827922
   1.64913728]
 [ 0.92169372 -0.33622623  0.36631025 -0.59855939  0.69977629  1.04297148
   1.03185674 -0.69732275  0.81379056  0.40563523  0.39862112  0.72800935
   1.13098806]
 [ 1.07541476 -0.14304278  0.50431902 -0.50990499  0.77452369  0.91742527
   0.86560067 -0.56850495  0.70872326  0.67024295  0.16178587  0.51002474
   1.17766329]
 [ 0.80641941 -0.40913263  0.28502364 -0.61062629  0.63160944  1.05065466
   1.05885031 -0.72315748  0.82306382  0.26406823  0.48865529  0.79348991
   1.05500549]
 [ 0.77924224 -0.49758648  0.24386877 -0.66735996  0.62753566  1.13809387
   1.16195144 -0.79932956  0.89409453  0.17438109  0.59660284  0.90596569
   1.07767706]]

[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
```

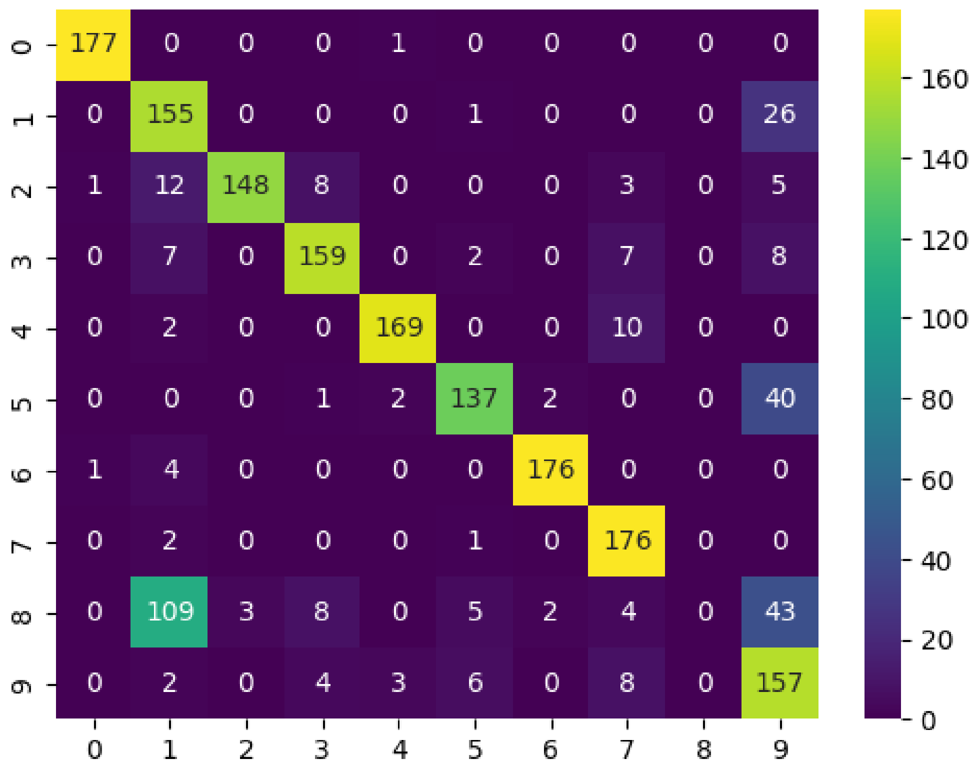
```
 3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
[1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
 2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
[1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
 3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]
[1.420e+01 1.760e+00 2.450e+00 1.520e+01 1.120e+02 3.270e+00 3.390e+00
 3.400e-01 1.970e+00 6.750e+00 1.050e+00 2.850e+00 1.450e+03]
[1.439e+01 1.870e+00 2.450e+00 1.460e+01 9.600e+01 2.500e+00 2.520e+00
 3.000e-01 1.980e+00 5.250e+00 1.020e+00 3.580e+00 1.290e+03]
[1.406e+01 2.150e+00 2.610e+00 1.760e+01 1.210e+02 2.600e+00 2.510e+00
 3.100e-01 1.250e+00 5.050e+00 1.060e+00 3.580e+00 1.295e+03]
[1.483e+01 1.640e+00 2.170e+00 1.400e+01 9.700e+01 2.800e+00 2.980e+00
 2.900e-01 1.980e+00 5.200e+00 1.080e+00 2.850e+00 1.045e+03]
[1.386e+01 1.350e+00 2.270e+00 1.600e+01 9.800e+01 2.980e+00 3.150e+00
 2.200e-01 1.850e+00 7.220e+00 1.010e+00 3.550e+00 1.045e+03]]
```

6.3 PCA as a Noise Filter

6.3.1 KMeans on Original Data

[ ]:
```
from sklearn.cluster import KMeans
X = digits.data
Xnew = np.random.normal(X, 2)

kmeans = KMeans(n_clusters = 10, n_init = "auto", random_state = 42)
kmeans.fit(X)
cluster_labels = kmeans.predict(X)

lab = np.zeros_like(cluster_labels)
for i in range(10):
  mask = cluster_labels == i
  lab[mask] = mode(digits.target[mask])[0]

cm = confusion_matrix(digits.target, lab)
sns.heatmap(cm, cmap = "viridis", annot = True, fmt = ".3g")
```

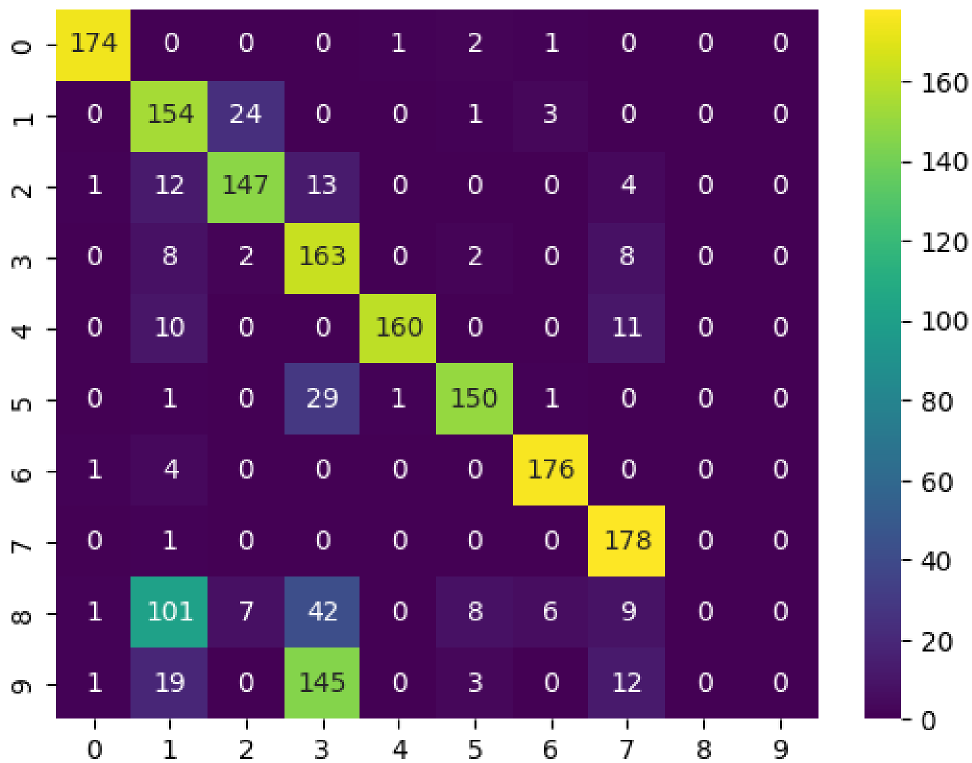
[ ]: <Axes: >



6.3.2 KMeans on Noisy Data

[ ]:
```
kmeans = KMeans(n_clusters = 10, n_init = "auto", random_state = 42)
kmeans.fit(Xnew)
cluster_labels = kmeans.predict(Xnew)

lab = np.zeros_like(cluster_labels)
for i in range(10):
  mask = cluster_labels == i
  lab[mask] = mode(digits.target[mask])[0]

cm = confusion_matrix(digits.target, lab)
sns.heatmap(cm, cmap = "viridis", annot = True, fmt = ".3g")
```

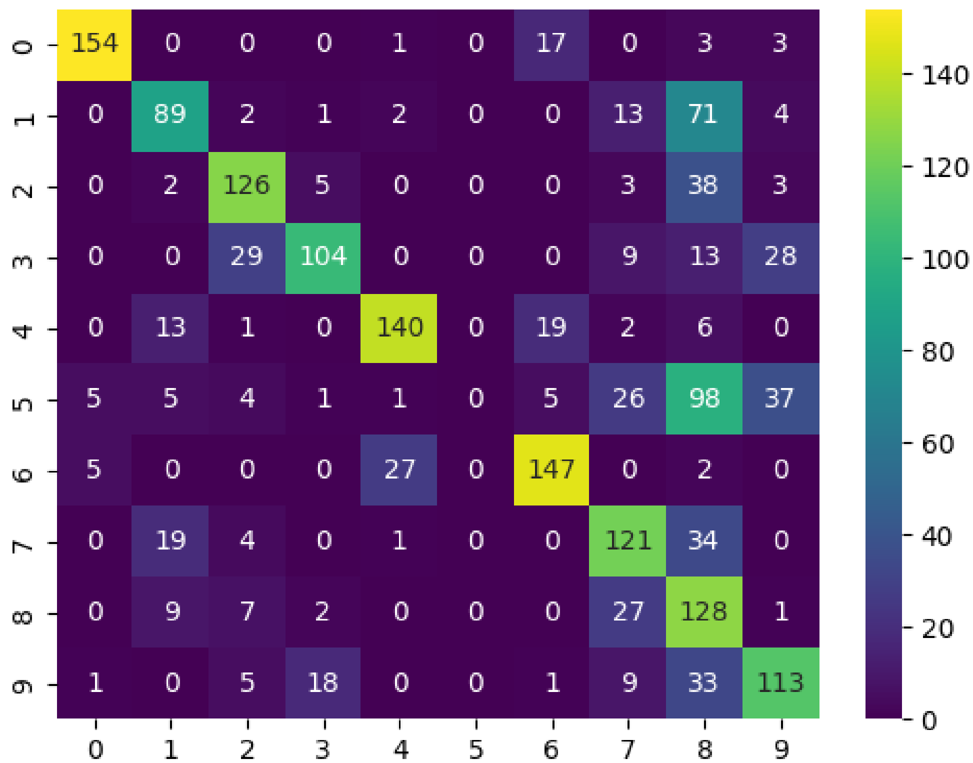
[ ]: <Axes: >



6.3.3 KMeans on Noise Removed using PCA inverse transform

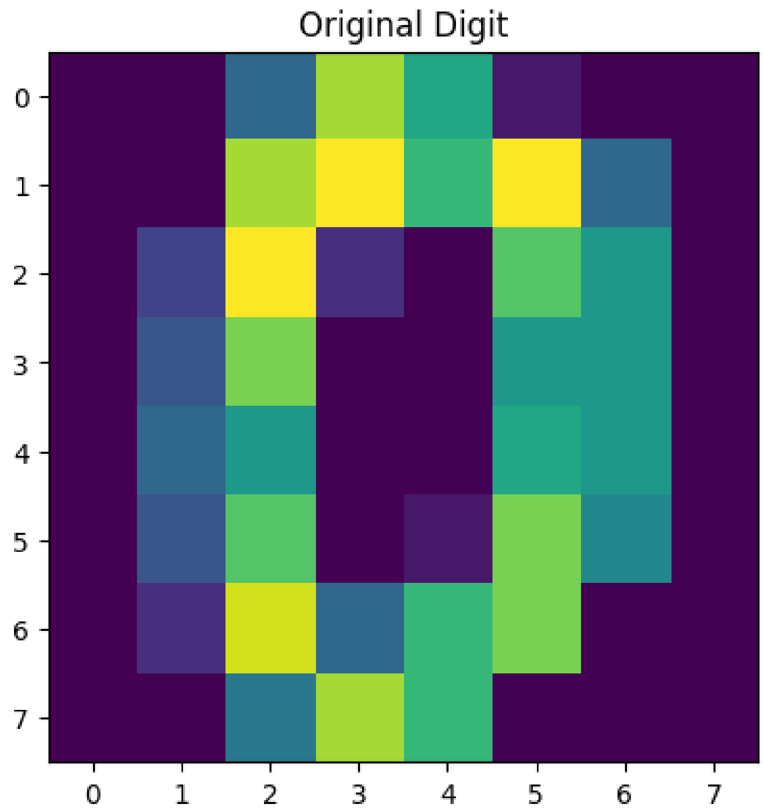
[ ]:
```
model = PCA(n_components = 2, random_state = 42)
X_model = model.fit_transform(Xnew)
filtered = model.inverse_transform(X_model)  # To get back to original␣
 ↪dimensional space
kmeans = KMeans(n_clusters = 10, n_init = "auto", random_state = 42)
kmeans.fit(filtered)
cluster_labels = kmeans.predict(filtered)

lab = np.zeros_like(cluster_labels)
for i in range(10):
  mask = cluster_labels == i
  lab[mask] = mode(digits.target[mask])[0]

cm = confusion_matrix(digits.target, lab)
sns.heatmap(cm, cmap = "viridis", annot = True, fmt = ".3g")
```

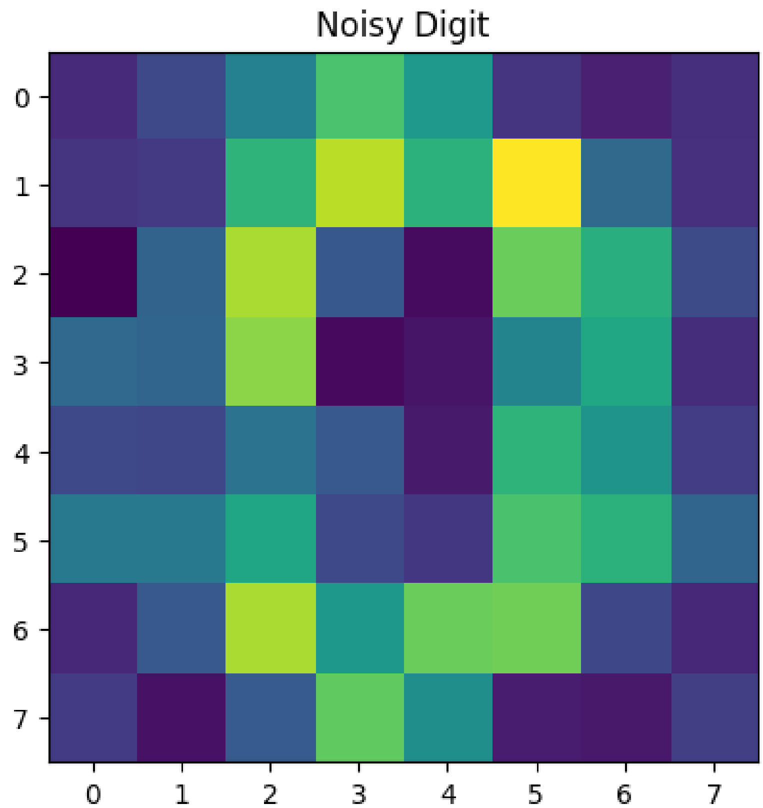
[ ]:
```
<Axes: >
```

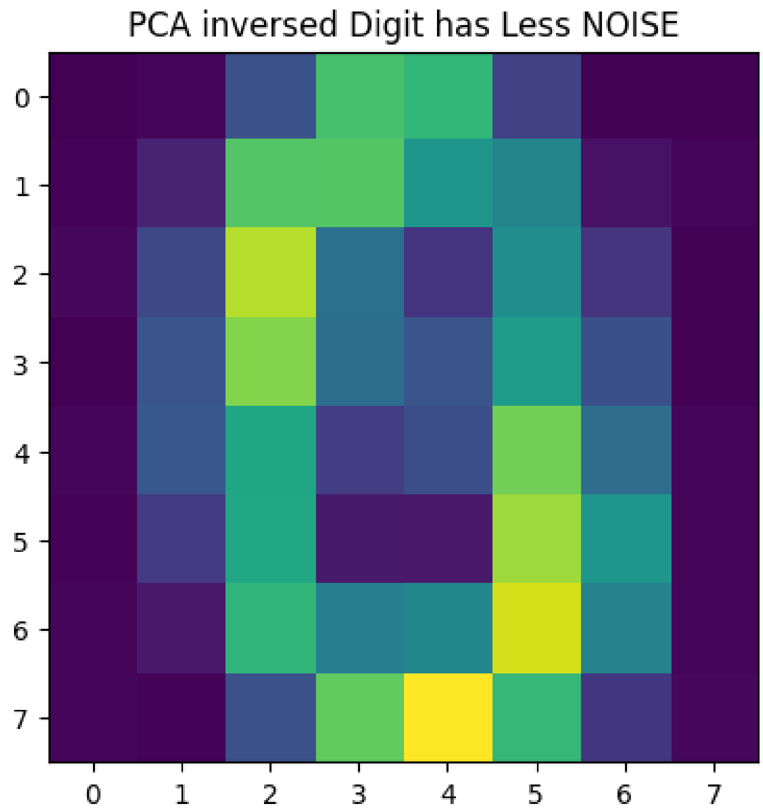


6.3.4 Look at the three digits

[ ]:
```
plt.imshow(X[0].reshape((8, 8)))
plt.title("Original Digit")
plt.show()
plt.imshow(Xnew[0].reshape((8, 8)))
plt.title("Noisy Digit")
plt.show()
plt.imshow(filtered[0].reshape((8, 8)))
plt.title("PCA inversed Digit has Less NOISE")
plt.show()
```


Original Digit
0
1
2
3
4
5
6
7


Noisy Digit
0
1
2
3
4
5
6
7



7 Support Vector Machine (SVM) Classifier

[ ]:
```python
from sklearn.svm import SVC
X, Y = digits.data, digits.target
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, train_size = 0.25)
# SVC has mainly three parameters: kernel, gamma, C
# Main kernels include: Linear(linear), RBF(Radial Basis Function)(rbf),␣
↪Polynomial(poly): These define the nature of manifold boundries surfacing␣
↪the clusters
# By default kernel = "rbf", C = 1.0, gamma = controls the width of Gaussian␣
↪kernels like rbf, ploy, sigmoid
model = SVC(kernel = "rbf", C = 1.0)
model.fit(Xtrain, Ytrain)
y_model = model.predict(Xtest)
print("Predicted Labels:")
print(np.array(y_model))
print("\nActual Labels:")
print(Ytest)
print("\nAccuracy Score:")
print(accuracy_score(y_model, Ytest))
```

```
Predicted Labels:
[9 7 1 … 9 2 7]

Actual Labels:
[9 7 1 … 9 2 7]

Accuracy Score:
0.9695845697329377
```

[ ]:
```python
print(model.support_vectors_.shape)
print(model.support_vectors_)
```

```
(308, 64)
[[ 0.  0.  4. …  0.  0.  0.]
 [ 0.  0.  0. …  9.  0.  0.]
 [ 0.  0.  2. …  9.  1.  0.]
 …
 [ 0.  0. 12. … 15.  6.  0.]
 [ 0.  0. 10. … 10.  0.  0.]
 [ 0.  0.  1. … 13.  0.  0.]]
```

8 Decision Tree Classifier

[ ]:
```python
from sklearn.tree import DecisionTreeClassifier
X, Y = wine.data, wine.target
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, train_size = 0.25)
model = DecisionTreeClassifier()
model.fit(Xtrain, Ytrain)
```

```
y_model = model.predict(Xtest)
print("Predicted Labels:")
print(np.array(y_model))
print("\nActual Labels:")
print(Ytest)
print("\nAccuracy Score:")
print(accuracy_score(y_model, Ytest))
```

```
Predicted Labels:
[0 2 1 2 2 0 0 0 1 2 0 1 0 1 1 1 1 1 0 0 2 1 2 2 1 1 0 1 1 0 0 0 1 2 0 0 2
 0 0 2 1 1 1 1 0 1 1 2 2 0 0 1 1 0 2 2 2 2 1 0 1 1 0 0 1 1 1 2 2 0 2 0 2 2
 0 2 2 1 2 2 0 2 0 1 2 2 0 2 2 0 2 2 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 2 0
 1 2 0 1 1 0 0 2 1 2 0 0 1 0 0 0 0 2 2 1 1 1 2]

Actual Labels:
[0 2 1 2 2 1 0 1 1 2 0 1 0 1 1 0 1 2 1 0 2 1 2 2 1 1 0 1 1 1 0 0 0 2 0 1 2
 1 0 2 1 1 1 1 0 0 1 2 2 0 0 1 1 0 2 2 2 2 1 0 1 0 0 0 1 1 1 2 2 0 2 0 2 2
 0 2 2 1 2 1 0 2 0 1 0 2 0 2 2 0 2 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 2 0
 1 1 0 1 1 0 0 2 1 2 0 0 1 0 0 0 0 2 2 1 1 2 1]

Accuracy Score:
0.8731343283582089
```

8.1 Decision Boundaries

[ ]:

```
def visualize_classifier(model, X, y, ax=None, cmap='rainbow'):
    ax = ax or plt.gca()

    # Plot the training points
    ax.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=cmap,
               clim=(y.min(), y.max()), zorder=3)
    ax.axis('tight')
    ax.axis('off')
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # fit the estimator
    model.fit(X, y)
    xx, yy = np.meshgrid(np.linspace(*xlim, num=200),
                         np.linspace(*ylim, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    # Create a color plot with the results
    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels=np.arange(n_classes + 1) - 0.5,
                           cmap=cmap, clim=(y.min(), y.max()),
```

```
                            zorder=1)

    ax.set(xlim=xlim, ylim=ylim)
```

[ ]:
```
model = PCA(n_components = 2)
X2D = model.fit_transform(X_scaled)
visualize_classifier(DecisionTreeClassifier(), X2D, Y)
```

```
<ipython-input-16-2012e9cc4787>:20: UserWarning: The following kwargs were not
used by contour: 'clim'
  contours = ax.contourf(xx, yy, Z, alpha=0.3,
```