MLDM2020-HW1-Clust-P1-Frank-Acquaye

April 18, 2020

0.1 Task 1

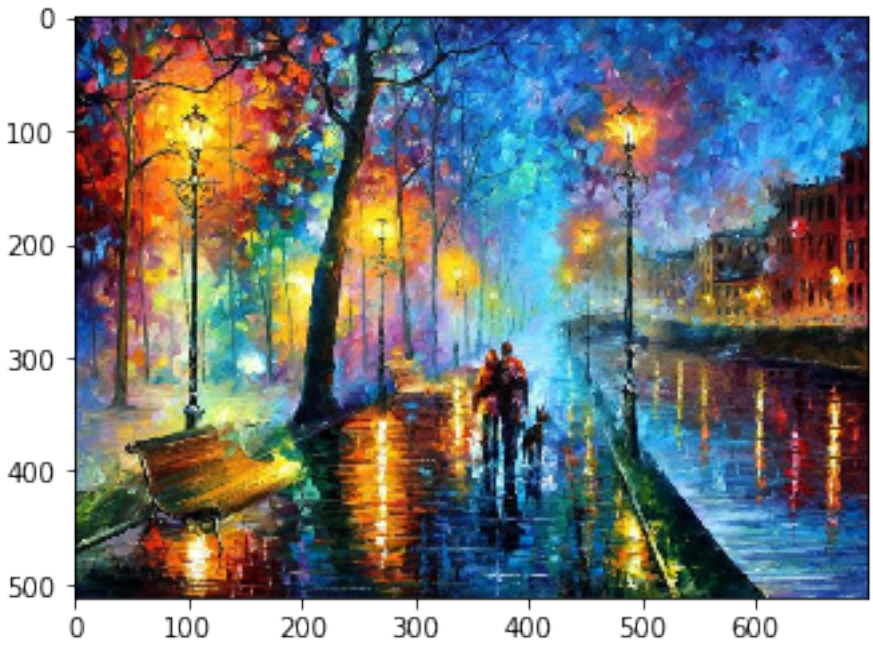
[1]:
```
# Import libraries
from imageio import imread, imwrite
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

[2]:
```
# Read Selected Images
img = imread('landscape6.jpg')
```

[3]:
```
# Getting shape of image
x, y, z = img.shape # You can see, that images are stored as 3-dimensional␣
 ↪arrays, where 3rd dimension contains RGB description of a pixel.
```

[4]:
```
# Plotting obtained image
plt.imshow(img)
```

[4]:
```
<matplotlib.image.AxesImage at 0x10726adf0>
```



[5]:
```
# converting image into 2D Array
features = img.reshape(x*y, z)
features.shape
```

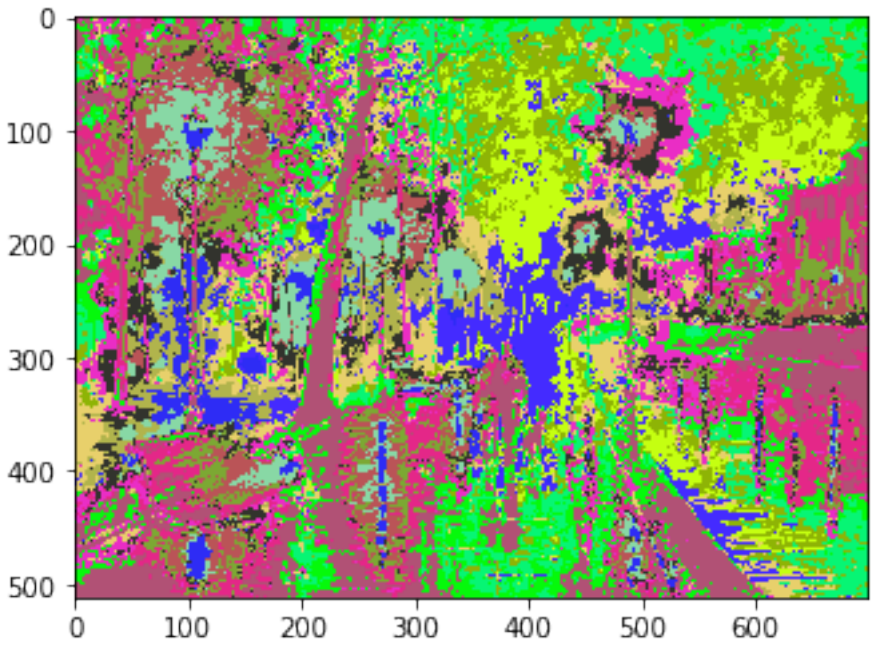
[5]: (358400, 3)

[6]:
```
# importing Kmeans from Scikit Learn
from sklearn.cluster import KMeans
```

[7]:
```
# Clustering image
n_clusters = 15
k_means = KMeans(n_clusters=n_clusters)
k_means.fit(features)
cluster_centers = k_means.cluster_centers_
cluster_labels = k_means.labels_
```

[8]:
```
# https://stackoverflow.com/questions/49643907/
 ↪clipping-input-data-to-the-valid-range-for-imshow-with-rgb-data-0-1-for-floa
compressed_image = ( cluster_centers[cluster_labels].reshape(x, y, z) * 255).
 ↪astype(np.uint8)
plt.imshow(compressed_image)
```

[8]: <matplotlib.image.AxesImage at 0x121d16ee0>



[9]:
```
imwrite('compressed_landscape6.jpg', compressed_image)
```

[10]:
```
import os
compressed_landscape6 = os.path.getsize("compressed_landscape6.jpg")
landscape6 = os.path.getsize("landscape6.jpg")
```

[11]:
```
print("size of compressed landscape is ",
        compressed_landscape6,
        " and actual landscape image is ", landscape6)
```

```
size of compressed landscape is  149930  and actual landscape image is  178310
```

One realises that the size of the image after clustering is smaller than the original image used

0.2 Task 2

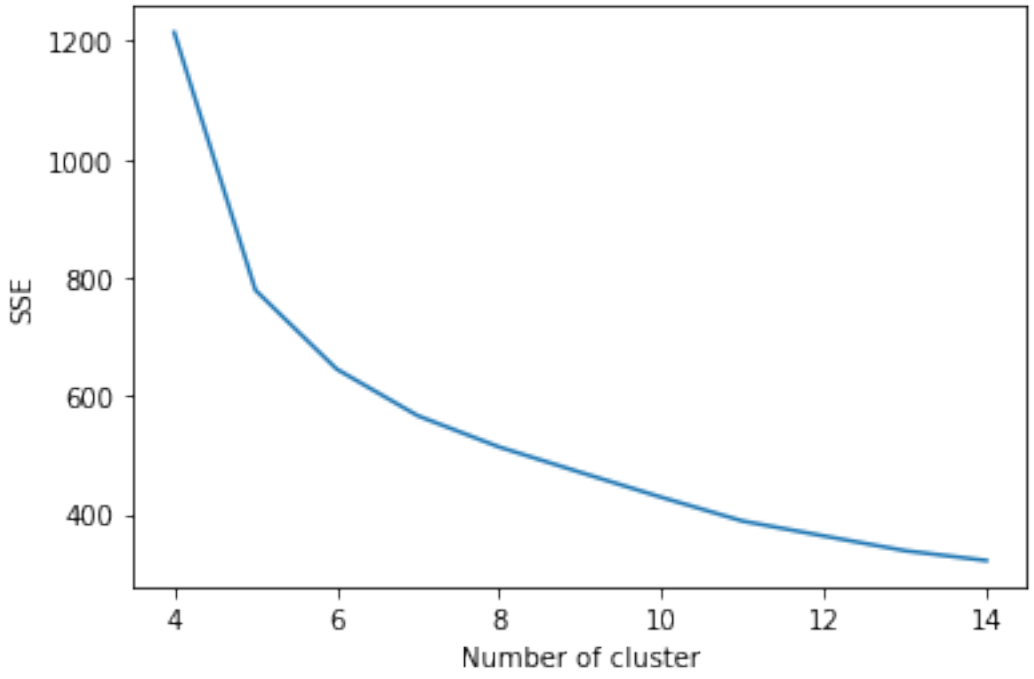
[12]:
```
from numpy import genfromtxt
```

[13]:
```
def conv(x):
    return x.replace(',', '.').encode()

elbow_data = genfromtxt((conv(x) for x in open("elbow.txt")), delimiter='\t')
```

[14]:
```
elbow_data[0]
```

[14]:
```
array([-2.74771081,  3.57568039])
```

[15]:
```
from sklearn.metrics import silhouette_score

sse = {}
for n_clusters in range(4,15):
    #iterating through cluster sizes
    kmeans = KMeans(n_clusters = n_clusters)
    clusters = kmeans.fit(elbow_data)
    sse[n_clusters] = clusters.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```



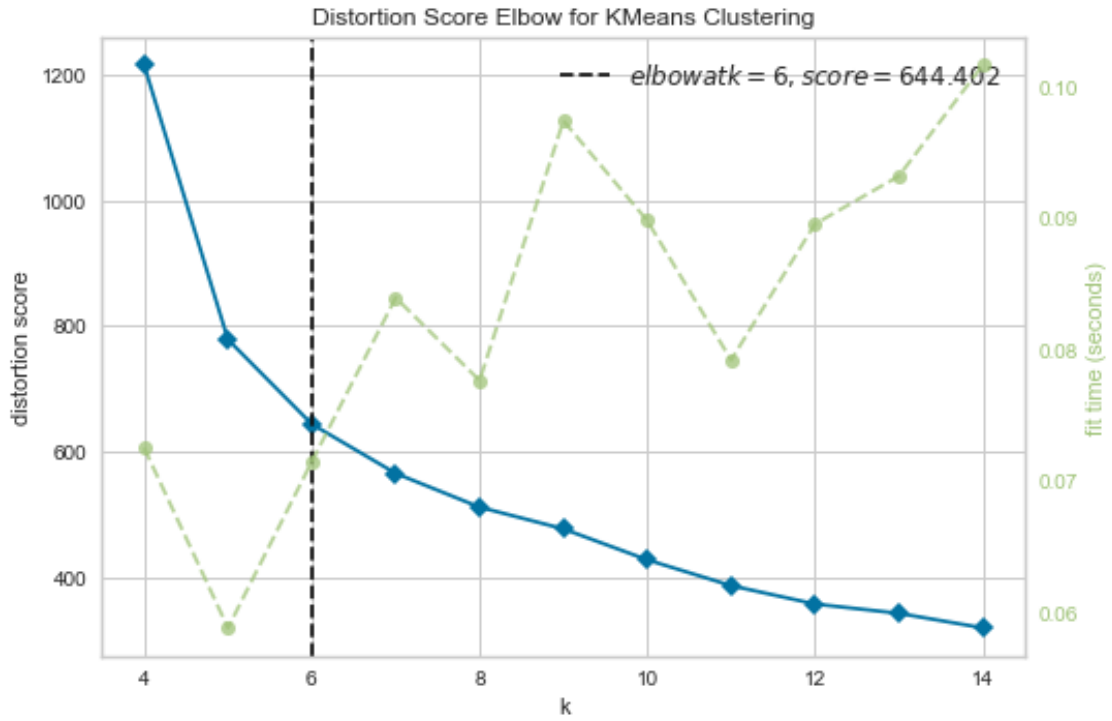
The ideal cluster for elbow.txt is 6. To confirm YellowBricks library will be used as a check

[16]:
```
"""
importing YellowBricks Library
Yellowbrick extends the Scikit-Learn API to make model selection
and hyperparameter tuning easier. Under the hood, it's using Matplotlib.
"""
from yellowbrick.cluster import KElbowVisualizer
```

```
/Users/larry/miniconda3/envs/mldm/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:144: FutureWarning: The
```

```
sklearn.metrics.classification module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.metrics. Anything that cannot be imported from
sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[17]:
```
# Instantiate the clustering model and visualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(4,15))

visualizer.fit(elbow_data)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```



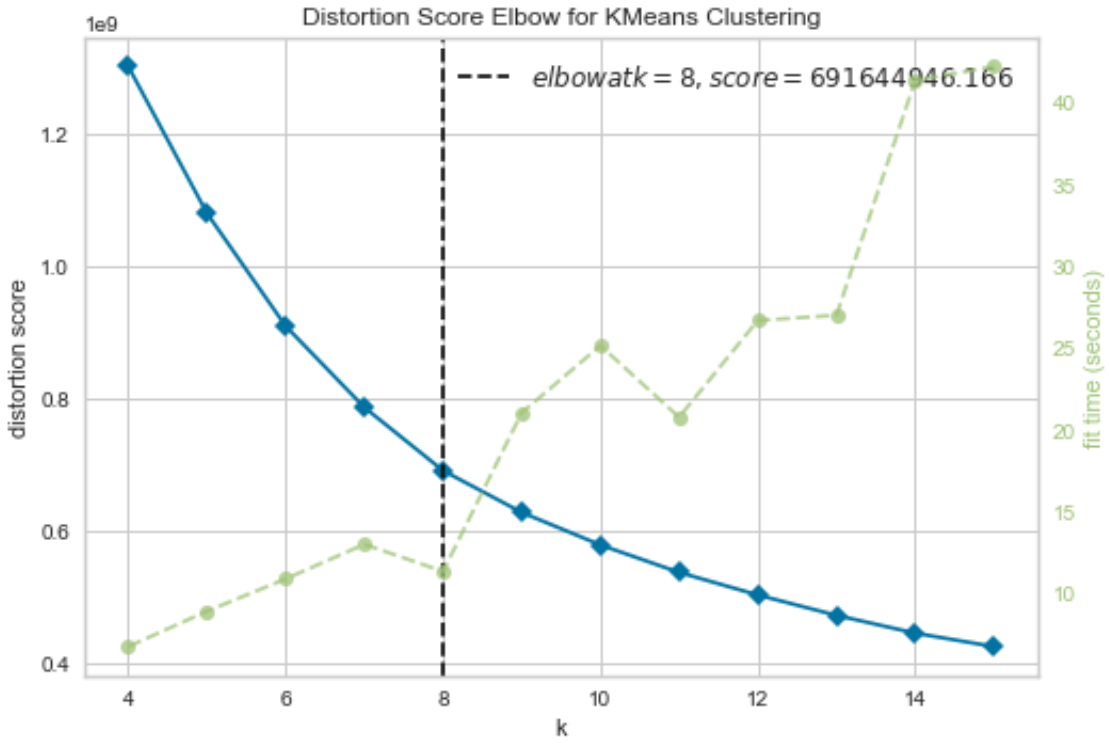
[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11c8659d0>
```

In the cell above it can be observed that the ideal number of clusters is 6

1 Checking Optimal K for Task 1

[18]:
```
# Instantiate the clustering model and visualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(4,16))
```

```
visualizer.fit(features)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

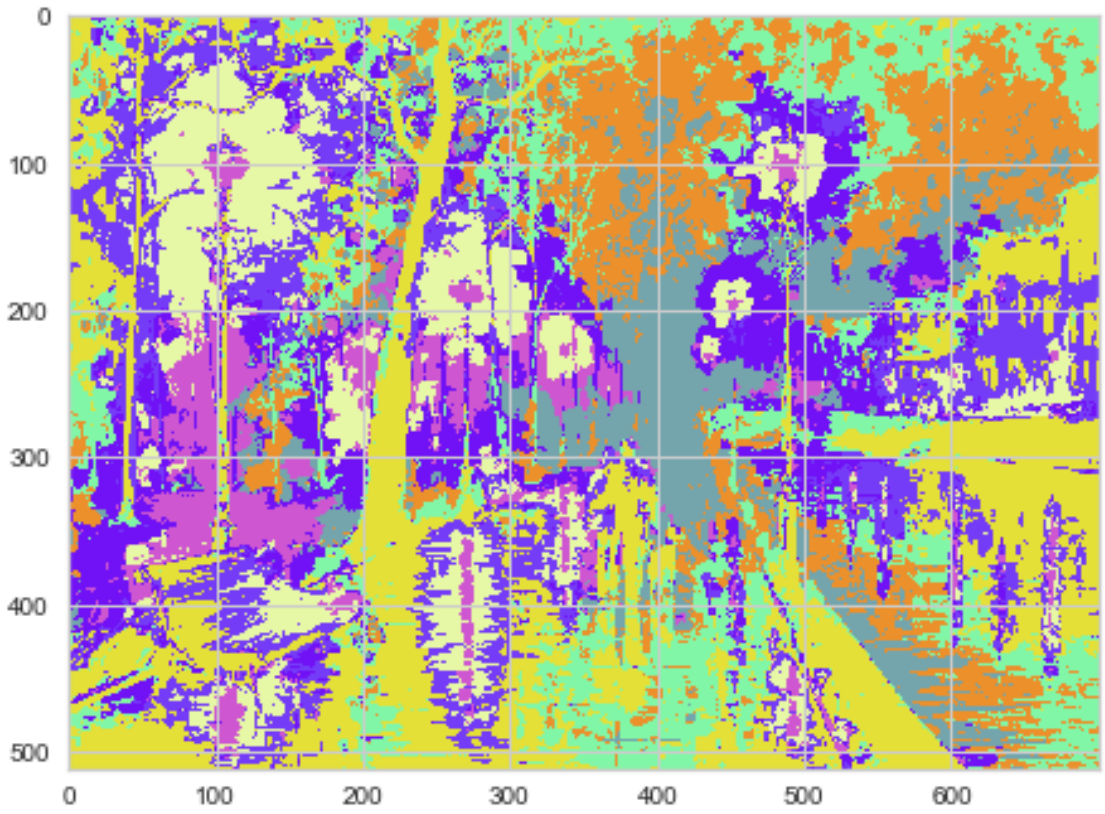


[18]: <matplotlib.axes._subplots.AxesSubplot at 0x11c8652e0>

2 Redoing Task 1 with optimal k=8

[19]:
```
k_means = KMeans(n_clusters=8)
k_means.fit(features)
cluster_centers = k_means.cluster_centers_
cluster_labels = k_means.labels_

compressed_image = ( cluster_centers[cluster_labels].reshape(x, y, z) * 255).
 ↪astype(np.uint8)
plt.imshow(compressed_image)
```

[19]: <matplotlib.image.AxesImage at 0x1193fc4f0>



[20]:
```
imwrite('optimal_k_compressed_landscape6.jpg', compressed_image)
```

[21]:
```
optimal_k_compressed_landscape6 = os.path.
 ↪getsize("optimal_k_compressed_landscape6.jpg")
landscape6 = os.path.getsize("landscape6.jpg")
print("size of optimal compressed landscape is ",
        compressed_landscape6,
        " and actual landscape image is ", landscape6)
```

```
size of optimal compressed landscape is  149930  and actual landscape image is
178310
```

One realises that the size of the image after using a clustering size of 8 is smaller than the original image used

[ ]:
```

```