HomeWork 05

Deep Dand

May 15, 2018

Abstract

Kmeans and MLP experiments

1 Problem 1

1.1 e

The code below was run for different values of n_colors between $2 - 64$.
Code Begins

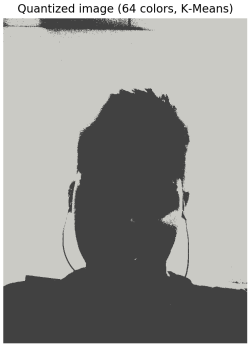
```
# Modified by: Pablo Rivas <Pablo.Rivas@Marist.edu>
# on Nov/3/2016
# Original Code:
# http://scikit-learn.org/stable/_downloads/
    ↪ plot_color_quantization.py
# Original Authors:
# Robert Layton <robertlayton@gmail.com>
# Olivier Grisel <olivier.grisel@ensta.org>
# Mathieu Blondel <mathieu@mblondel.org>
#
# License: BSD 3 clause

print(__doc__)
from scipy import misc
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin
from sklearn.datasets import load_sample_image
from sklearn.utils import shuffle
from time import time

# Specify the number of colors here
n_colors = 15
```

```
# Load your photo here
img = misc.imread('IMG_20180427_093834.jpg')

# Convert to floats instead of the default 8 bits integer coding.
    ↪ Dividing by
# 255 is important so that plt.imshow behaves works well on float
    ↪ data (need to
# be in the range [0-1])
img = np.array(img, dtype=np.float64) / 255

# Load Image and transform to a 2D numpy array.
w, h, d = original_shape = tuple(img.shape)
assert d == 3
image_array = np.reshape(img, (w * h, d))

print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0)[:1000]
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(
    ↪ image_array_sample)
print("done in %0.3fs." % (time() - t0))

# Get labels for all points
print("Predicting color indices on the full image (k-means)")
t0 = time()
labels = kmeans.predict(image_array)
print("done in %0.3fs." % (time() - t0))

def recreate_image(codebook, labels, w, h):
    """Recreate the (compressed) image from the code book & labels
        ↪ """
    d = codebook.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = codebook[labels[label_idx]]
            label_idx += 1
    return image

# Display all results, alongside original image
plt.figure(1)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Original image (96,615 colors)')
```

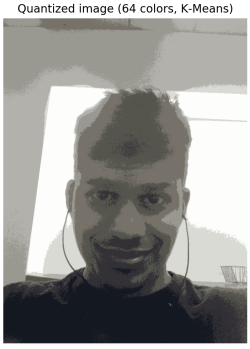
```
plt.imshow(img)

plt.figure(2)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Quantized image (64 colors, K-Means)')
plt.imshow(recreate_image(kmeans.cluster_centers_, labels, w, h))
plt.savefig('n_colors_%s' % (str(n_colors)), dpi=200, bbox_inches
    ↪ ='tight')
plt.show()
```

Code ends
Output
As it is seen in the images above, as we get close to the colors value 2, the image starts to look more and more funny. As the *n_colors* value is closer to 64, the image is reproduced well with more colors.

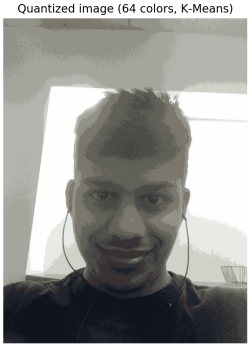
In the images below, until values of n_color =15, 16 are the last to show any signs of color and after that as values are reduces, the image gets more and more funny.



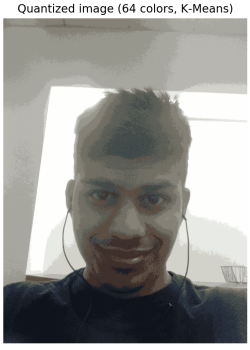
(a) n_color=2



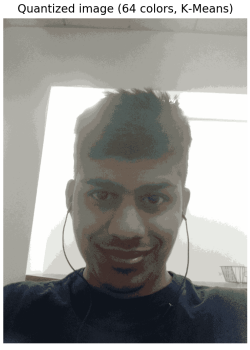
(b) n_color=8



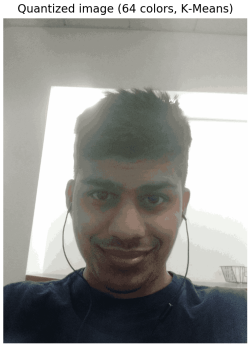
(c) n_color=12



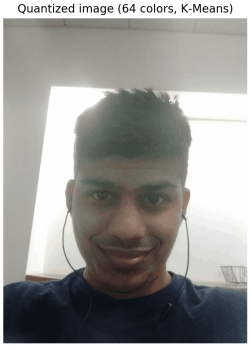
(d) n_color=15



(e) n_color=16



(f) n_color=32



(g) n_color=64

Figure 1: 3 x 3

Ans ii.

The other possible applications could be, image compression where we can choose value of n_color to a value which can represent the image almost equal quality but will eventually take really less pixels to draw it.

Ans iii.

The resulting picture was funny at the end because the less number of colors meaning that pixels in the original picture were replaced by shades of black and white and it makes white brighter in the picture and black darker. All other colors are converted in one or the other shade of gray depending on the n_colors value. For e.g, if it is 2(the first image in the grid), the image will only be converted in black and white. If the nvalue = 15, it the algorithm has 13 shades of gray and white and black color to represent the whole picture.

2 Problem 2

MLP Program.

2.1 c

The code below is run for 1000 samples and the best value of neurons and η is noted.

```
import hw5gendata
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold

# number of samples
N = 10000

# generate data & split it into X (training input) and y (target
    ↪ output)
X, y = hw5gendata.genDataSet(N)

# linear regression solution
w=np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y)


#neurons <- number of neurons in the hidden layer
#eta <- the learning rate parameter

bestNeurons=0
bestEta=0
bestScore=float('-inf')
score=0
for neurons in range(1,101,1):
  for eta in range(1,11,1):
    eta=eta/10.0
    kf = KFold(n_splits=10)
    cvscore=[]
    for train, validation in kf.split(X):
      X_train, X_validation, y_train, y_validation = X[train, :],
          ↪ X[validation, :], y[train], y[validation]
      # here we create the MLP regressor
      mlp = MLPRegressor(hidden_layer_sizes=(neurons,), verbose=
          ↪ False, learning_rate_init=eta)
      # here we train the MLP
      mlp.fit(X_train, y_train)
```

```
      # now we get E_out for validation set
      score=mlp.score(X_validation, y_validation)
      cvscore.append(score)

    # average CV score
    score=sum(cvscore)/len(cvscore)
    if (score > bestScore):
      bestScore=score
      bestNeurons=neurons
      bestEta=eta
      print("Neurons " + str(neurons) + ", eta " + str(eta) + ".
          ↪ Testing set CV score: %f" % score)

# here we get a new training dataset
X, y = hw5gendata.genDataSet(N)
# here we create the final MLP regressor
mlp = MLPRegressor(hidden_layer_sizes=(bestNeurons,), verbose=
    ↪ True, learning_rate_init=bestEta)
# here we train the final MLP
mlp.fit(X, y)
# E_out in training
print("Training set score: %f" % mlp.score(X, y))
# here we get a new testing dataset
X, y = hw5gendata.genDataSet(N)
# here test the final MLP regressor and get E_out for testing set
ypred=mlp.predict(X)
score=mlp.score(X, y)
print("Testing set score: %f" % score)
plt.plot(X[:, 0], X[:, 1], '.')
plt.plot(X[:, 0], y, 'rx')
plt.plot(X[:, 0], ypred, '-k')
ypredLR=X.dot(w)
plt.plot(X[:, 0], ypredLR, '--g')
plt.show()
```

Code ends.
Output
The screen-shot below shows the result with

$$Number\ of\ neurons\ = 23$$

$$\eta = 0.1$$

$$Training\ set\ score\ = 0.8345$$

$$Testing\ set\ score = 0.8397$$

```
C:\Users\deepd\Documents\GitHub\TheMachineLearningS18\HW05>python hw5.MLP.sol.py
Neurons 1, eta 0.1. Testing set CV score: -7.829672
Neurons 23, eta 0.2. Testing set CV score: -3.045104
Iteration 1, loss = 1.21697029
Iteration 2, loss = 0.24213331
Iteration 3, loss = 0.14758218
Iteration 4, loss = 0.08270514
Iteration 5, loss = 0.05587305
Iteration 6, loss = 0.04835602
Iteration 7, loss = 0.02798825
Iteration 8, loss = 0.02704120
Iteration 9, loss = 0.02480184
Iteration 10, loss = 0.01906120
Iteration 11, loss = 0.01782004
Iteration 12, loss = 0.01933918
Iteration 13, loss = 0.02055518
Iteration 14, loss = 0.01959885
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
Training set score: 0.834563
Testing set score: 0.839756
```

The screen-shot below shows the result with

$$Number\ of\ neurons\ = 74$$

$$\eta = 0.1$$

$$Training\ set\ score\ = 0.9012$$

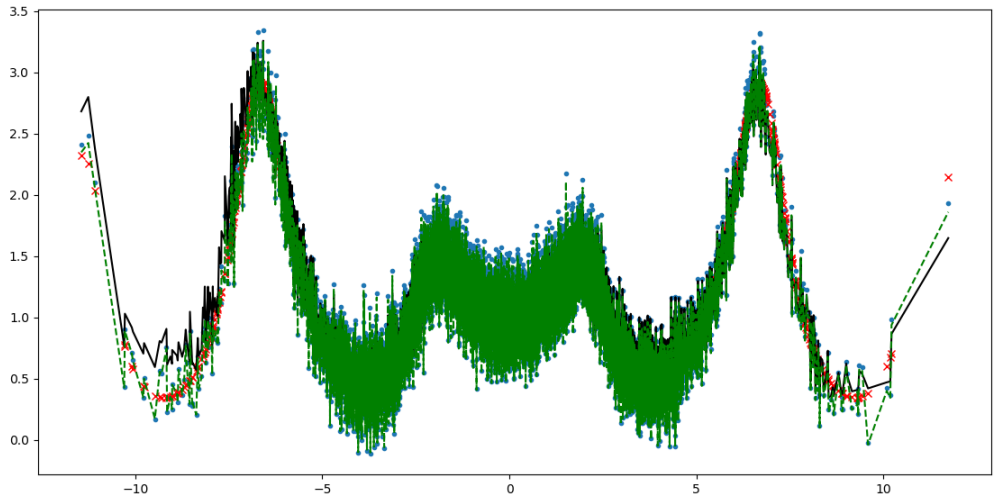
$$Testing\ set\ score\ = 0.8813$$

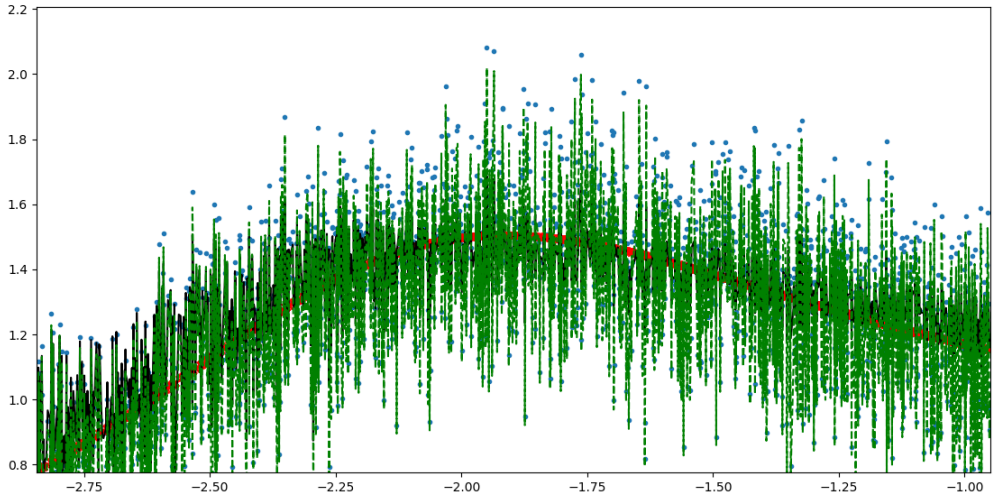
```
C:\Users\deepd\Documents\GitHub\TheMachineLearningS18\HW05>python hw5.MLP.sol.py
Neurons 1, eta 0.1. Testing set CV score: -14.820856
Neurons 1, eta 0.3. Testing set CV score: -13.828293
Neurons 1, eta 0.4. Testing set CV score: -5.206453
Neurons 1, eta 0.5. Testing set CV score: -4.641689
Neurons 1, eta 0.9. Testing set CV score: -4.490173
Neurons 4, eta 0.5. Testing set CV score: -2.931489
Neurons 5, eta 0.4. Testing set CV score: -2.929142
Neurons 74, eta 0.1. Testing set CV score: -2.760011
Iteration 1, loss = 1.20584943
Iteration 2, loss = 0.28301453
Iteration 3, loss = 0.06690046
Iteration 4, loss = 0.06983276
Iteration 5, loss = 0.04064526
Iteration 6, loss = 0.02982603
Iteration 7, loss = 0.01992784
Iteration 8, loss = 0.01913576
Iteration 9, loss = 0.01435669
Iteration 10, loss = 0.01255075
Iteration 11, loss = 0.01201104
Iteration 12, loss = 0.01116562
Iteration 13, loss = 0.01106893
Iteration 14, loss = 0.01036030
Iteration 15, loss = 0.01136067
Iteration 16, loss = 0.01028602
Iteration 17, loss = 0.00989304
Iteration 18, loss = 0.00983979
Iteration 19, loss = 0.01044231
Iteration 20, loss = 0.00970735
Iteration 21, loss = 0.01031942
Iteration 22, loss = 0.00982699
Iteration 23, loss = 0.01007538
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
Training set score: 0.901293
Testing set score: 0.881393
```

As seen above, I tried running program two times to see the difference in results and also understand the relationship between number of neurons. The program is run again with 10000 data points and below is the result for the same. The output is,

```
C:\Users\deepd\Documents\GitHub\TheMachineLearningS18\HW05>python hw5.MLP.sol.py
Neurons 1, eta 0.1. Testing set CV score: -9.886750
Neurons 1, eta 0.2. Testing set CV score: -8.904145
Neurons 1, eta 0.4. Testing set CV score: -4.548810
Neurons 1, eta 1.0. Testing set CV score: -4.046935
Neurons 14, eta 0.1. Testing set CV score: -3.611722
Neurons 22, eta 0.2. Testing set CV score: -2.033671
Neurons 57, eta 0.2. Testing set CV score: -2.031696
Neurons 62, eta 0.2. Testing set CV score: -1.469766
Neurons 70, eta 0.2. Testing set CV score: -1.317948
Iteration 1, loss = 0.86553644
Iteration 2, loss = 0.02568616
Iteration 3, loss = 0.01599589
Iteration 4, loss = 0.01509835
Iteration 5, loss = 0.01548558
Iteration 6, loss = 0.01213069
Iteration 7, loss = 0.02525189
Iteration 8, loss = 0.01622058
Iteration 9, loss = 0.01169560
Iteration 10, loss = 0.01514116
Iteration 11, loss = 0.01104734
Iteration 12, loss = 0.01126189
Iteration 13, loss = 0.01003569
Iteration 14, loss = 0.01026281
Iteration 15, loss = 0.00899383
Iteration 16, loss = 0.01578172
Iteration 17, loss = 0.01585325
Iteration 18, loss = 0.01243698
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
Training set score: 0.893407
Testing set score: 0.895793
```

The output graph looks like,





Question- What do you think is happening? What is your interpretation of the number of neurons with respect to the performance of the network?
Answer

- I think that that the program with 1000 data points and 10000 data points have maximum $\eta = 0.2$ and it produces pretty good results.
- For this network any value above 0.2 for η is large and so it stops iterating.
- As far as number of neurons go, it the two results for 1000 data points suggest neurons count 23 and 74, there is one more result that I didn't include because it also had neurons count to be 79 with $\eta = 0.1$.
- With 10000 data points, it was surprising to see that number of neurons are 70 which states that for $1000-10000$ data points network performance is almost the same given the number of neurons are in range of 70.