Data Science Fundamentals 5

Basic introduction on how to perform typical machine learning tasks with Python.

Prepared by Mykhailo Vladymyrov & Aris Marcolongo, Science IT Support, University Of Bern, 2020

Solutions to Part 3.

In [0]:
```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.mixture import GaussianMixture

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from matplotlib import  pyplot as plt
import numpy as np
import pandas as pd
from imageio import imread
from time import time as timer
import os

import tensorflow as tf

%matplotlib inline
from matplotlib import animation
from IPython.display import HTML

import umap
from scipy.stats import entropy
```

In [2]:
```
if not os.path.exists('data'):
    path = os.path.abspath('.')+'/colab_material.tgz'
    tf.keras.utils.get_file(path, 'https://github.com/neworldemancer/DSF
5/raw/master/colab_material.tgz')
    !tar -xvzf colab_material.tgz > /dev/null 2>&1
```

```
Downloading data from https://github.com/neworldemancer/DSF5/raw/master/c
olab_material.tgz
98304/96847 [==============================] - 0s 0us/step
```

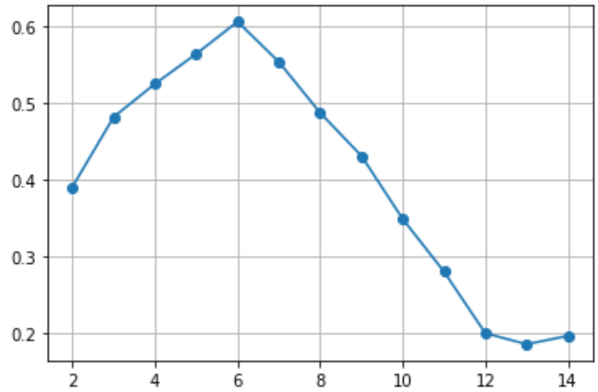
In [0]:
```
from utils.routines import *
```

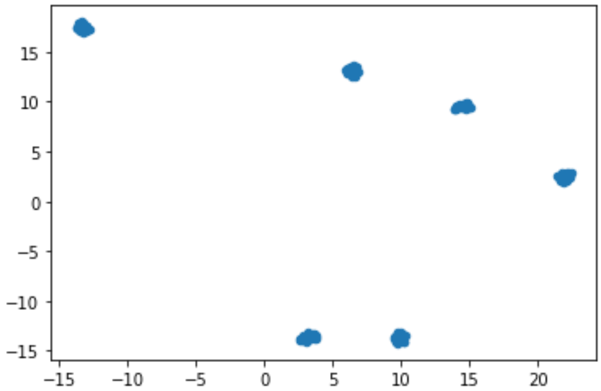
EXERCISE 1: Discover the number of Gaussians

In [4]:
```
### In this exercise you are given the dataset points, consisting of hig
h-dimensional data. It was built taking random
#samples from a number k of multimensional gaussians. The data is theref
ore made of k clusters but, being
#very high dimensional, you cannot visualize it. Your task it too use K-
means combined with the Silouhette
#score to find the number of k.

# 1. Load the data using the function load_ex1_data_clust() , check the
dimensionality of the data.

points=load_ex1_data_clust()

# 2. Fix a number of clusters k and define a KMeans clusterer object. Pe
rform the fitting and compute the Silhouette score.
# Save the results on a list.

plt.figure()
scores=[]
for itrial in range(2,15):
    print('Number of clusters considered: ',itrial)
    clusterer = KMeans(n_clusters=itrial, random_state=10)
    cluster_labels = clusterer.fit_predict(points)
    score=silhouette_score(points,cluster_labels)
    scores.append(score)

# 3. Plot the Silhouette scores as a function ok k? What is the number o
f clusters ?
plt.grid()
plt.plot(np.arange(len(scores))+2,np.array(scores),'-o')

# 4. Optional. Check the result that you found via umap.
plt.figure()
umap_model = umap.UMAP(random_state=1711)
umap_gs = umap_model.fit_transform(points)
plt.scatter(umap_gs[:, 0], umap_gs [:, 1], s=20)
```

```
Number of clusters considered:  2
Number of clusters considered:  3
Number of clusters considered:  4
Number of clusters considered:  5
Number of clusters considered:  6
Number of clusters considered:  7
Number of clusters considered:  8
Number of clusters considered:  9
Number of clusters considered:  10
Number of clusters considered:  11
Number of clusters considered:  12
Number of clusters considered:  13
Number of clusters considered:  14
```

```
Out[4]: <matplotlib.collections.PathCollection at 0x7fb682fd9ac8>
```





EXERCISE 2: Predict the good using K-Means

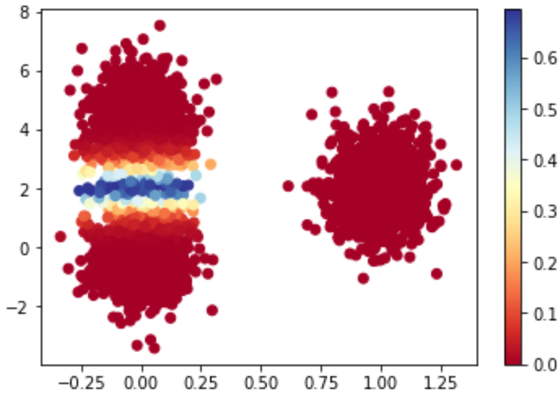
In [5]:
```
#In this exercise you are asked to use the clustering performed by K-mea
ns to predict the good in the f-mnist dataset.
#Here we are using the clustering as a preprocessing for a supervised ta
sk. We need therefore the correct labels
#on a training set and #o test the result on a test set:

# 1. Load the dataset.

fmnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fmnist.load_d
ata()

X_train=train_images[:5000,:].reshape(5000,-1)
y_train=train_labels[:5000]

X_test=test_images[:1000,:].reshape(1000,-1)
y_test=test_labels[:1000]

# 2. FITTING STEP: The fitting step consists first here in the computati
on of the cluster center, which was done during
# the presentation. Second, to each cluster center we need than to assig
n a good-label, which will be given by the
# majority class of the sample belonging to that cluster.

def most_common(nclusters, supervised_labels, cluster_labels):

    """
    Args:
    - nclusters : the number of clusteres
    - supervised_labels : for each sample, the labelling provided by the
training data ( e.g. in y_train or y_test)
    - cluster_labels : for each good, the cluster it was assigned by K-M
eans using the predict method of the Kmeans object

    Returns:
    - a list "assignement" of lengths nclusters, where assignement[i] is
the majority class of the i-cluster
    """

    assignement=[]
    for icluster in range(nclusters):
        indices=list(supervised_labels[cluster_labels==icluster])
        try:
            chosen= max(set(indices), key=indices.count)
        except ValueError :
            print('Em')
            chosen=1
        assignement.append(chosen)

    return assignement


clusterer = KMeans(n_clusters=10, random_state=10)

clusterer.fit(X_train)

cluster_labels = clusterer.predict(X_train)
assignement=most_common(10, y_train, cluster_labels)

print(assignement)

print('Training set')

cluster_labels = clusterer.predict(X_train)

new_labels=[assignement[i] for i in cluster_labels]

cm=metrics.confusion_matrix( y_train, new_labels)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
[4, 6, 9, 3, 7, 0, 8, 8, 9, 1]
Training set
0.5492
[[248   6   0  62   6   0 131   1   3   0]
 [  1 482   0  37   8   0  28   0   0   0]
 [  5   0   0  14 324   0 153   1   7   0]
 [  7 165   0 223   5   0 101   0   0   0]
 [  2   2   0  77 333   0  68   0   6   0]
 [  0   0   0   0   0   0 111 291   2  89]
 [ 68   3   0  52 181   0 178   1   9   1]
 [  0   0   0   0   0   0   0 410   0 102]
 [  0   0   0   7  16   0  46  25 378  18]
 [  0   0   0   0   0   0  10   2   0 494]]

Test set
0.558
[[60  0  0 12  4  0 31  0  0  0]
 [ 0 94  0  6  0  0  5  0  0  0]
 [ 2  0  0  2 71  0 35  0  1  0]
 [ 2 25  0 48  1  0 17  0  0  0]
 [ 0  1  0 23 79  0 12  0  0  0]
 [ 0  0  0  0  0  0 17 55  0 15]
 [12  1  0  9 32  0 41  0  2  0]
 [ 0  0  0  0  0  0  0 78  0 17]
 [ 0  0  0  1  5  0 14  3 67  5]
 [ 0  0  0  0  0  0  1  3  0 91]]
[4, 6, 9, 3, 7, 0, 8, 8, 9, 1]
Test set with 10 clusters
0.558
[[60  0  0 12  4  0 31  0  0  0]
 [ 0 94  0  6  0  0  5  0  0  0]
 [ 2  0  0  2 71  0 35  0  1  0]
 [ 2 25  0 48  1  0 17  0  0  0]
 [ 0  1  0 23 79  0 12  0  0  0]
 [ 0  0  0  0  0  0 17 55  0 15]
 [12  1  0  9 32  0 41  0  2  0]
 [ 0  0  0  0  0  0  0 78  0 17]
 [ 0  0  0  1  5  0 14  3 67  5]
 [ 0  0  0  0  0  0  1  3  0 91]]
[2, 3, 5, 1, 0, 5, 1, 4, 2, 9, 2, 3, 9, 2, 8, 8, 9, 8, 7, 0]
Test set with 20 clusters
0.648
[[87  0  6 11  2  0  0  0  1  0]
 [ 5 93  1  6  0  0  0  0  0  0]
 [15  0 85  1 10  0  0  0  0  0]
 [18 16  3 55  1  0  0  0  0  0]
 [ 6  0 57 17 35  0  0  0  0  0]
 [ 0  0  0  0  0 73  0  4  0 10]
 [34  0 36  9 16  2  0  0  0  0]
 [ 0  0  0  0  0 14  0 59  0 22]
 [ 6  0  7  1  1  8  0  0 69  3]
 [ 0  0  0  0  0  3  0  0  0 92]]
[6, 0, 7, 3, 9, 1, 4, 9, 5, 4, 2, 8, 8, 1, 1, 9, 0, 2, 3, 4, 5, 7, 3, 8,
6, 5, 3, 9, 5, 2]
Test set with 30 clusters
0.652
```

EXERCISE 3 : Find the prediction uncertainty

In [6]:
```
#In this exercise you need to load the dataset used to present K-means (
def km_load_th1() ) or the one used to discuss
# the Gaussian mixtures model ( def km_load_th1() ).
#As discussed, applying a fitting based on gaussian mixtures you can not
only predict the cluster label for each point,
#but also a probability distribution over the clusters.

#From this probability distribution, you can compute for each point the
entropy of the corresponging
#distribution (using for example scipy.stats.entropy) as an estimation o
f the undertainty of the prediction.
#Your task is to plot the data-cloud with a color proportional to the un
certainty of the cluster assignement.

# In detail you shoud:
# 1. Instantiate a GaussianMixture object with the number of clusters th
at you expect
# 2. fit the object on the dataset with the fit method

from scipy.stats import entropy
points=gm_load_th1()

plt.figure()
clf = GaussianMixture(n_components=3, covariance_type='full')

clf.fit(points)

# 3. compute the cluster probabilities using the method predict_proba. T
his will return a matrix of
# dimension npoints x nclusters
# 4. use the entropy function ( from scipy.stats import entropy ) to eva
luate for each point the uncertainty of the
#prediction

cluster_labels_prob=clf.predict_proba(points)

entropies=[]
for point in range(len(cluster_labels_prob)):
    entropies.append(entropy(cluster_labels_prob[point]))

# 5. Plot the points colored accordingly to their uncertanty.

cm = plt.cm.get_cmap('RdYlBu')
sc = plt.scatter(points[:,0], points[:,1], c=entropies, cmap=cm)
plt.colorbar(sc)
```

Out[6]: <matplotlib.colorbar.Colorbar at 0x7fb680f58e48>



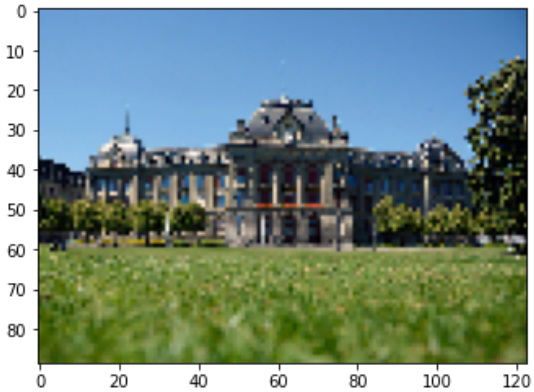
EXERCISE 4.

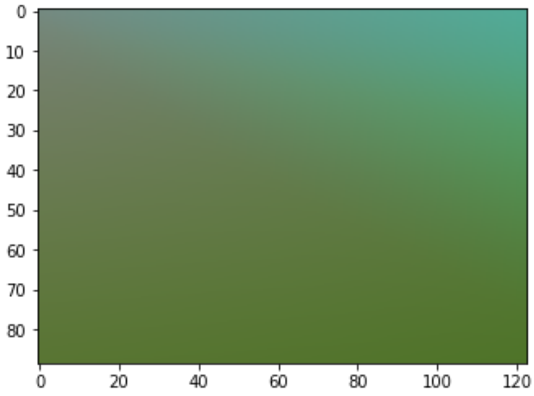
Load some image, downscale to a similar resolution, and train a deeper model, for example 5 layers, more parameters in widest layers.

In [7]:
```
# solution

# 1. Load your image
image_big = imread('https://www.unibe.ch/unibe/portal/content/carousel/s
howitem940548/UniBE_Coronavirus_612p_eng.jpg')
image_big = image_big[...,0:3]/255
plt.imshow(image_big)

image = image_big[::5, ::5]
plt.imshow(image)
plt.show()

h, w, c = image.shape
X = np.meshgrid(np.linspace(0, 1, w), np.linspace(0, 1, h))
X = np.stack(X, axis=-1).reshape((-1, 2))

Y = image.reshape((-1, c))
X.shape, Y.shape


# 2. build a deeper model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(2,)),
  tf.keras.layers.Dense(32, activation=tf.keras.layers.LeakyReLU()),
  tf.keras.layers.Dense(512, activation=tf.keras.layers.LeakyReLU()),
  tf.keras.layers.Dense(64, activation=tf.keras.layers.LeakyReLU()),
  tf.keras.layers.Dense(16, activation=tf.keras.layers.LeakyReLU()),
  tf.keras.layers.Dense(8, activation=tf.keras.layers.LeakyReLU()),
  tf.keras.layers.Dense(c, activation='sigmoid'),
])
model.compile(optimizer='adam',
              loss='mae',
              metrics=['mse'])
model.summary()

# 3. inspect the evolution

ims = []
n_ep_tot = 0
for i in range(200):
  if i % 10 == 0:
    print(f'epoch {i}', end='\n')
  ne = (2 if (i<50) else (20 if (i<100) else (200 if (i<150) else 100
0)))
  model.fit(X, Y, epochs=ne, batch_size=1*2048, verbose=0)

  Y_p = model.predict(X)
  Y_p = Y_p.reshape((h, w, c))
  ims.append(Y_p)
  n_ep_tot += ne

print(f'total numer of epochs trained:{n_ep_tot}')

plt.rcParams["animation.html"] = "jshtml"  # for matplotlib 2.1 and abov
e, uses JavaScript
fig = plt.figure()
im = plt.imshow(ims[0])

def animate(i):
    img = ims[i]
    im.set_data(img)
    return im

ani = animation.FuncAnimation(fig, animate, frames=len(ims))

ani
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 2)                 0         
_________________________________________________________________
dense (Dense)                (None, 32)                96        
_________________________________________________________________
dense_1 (Dense)              (None, 512)               16896     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                32832     
_________________________________________________________________
dense_3 (Dense)              (None, 16)                1040      
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136       
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 27        
=================================================================
Total params: 51,027
Trainable params: 51,027
Non-trainable params: 0
_________________________________________________________________
epoch 0
epoch 10
epoch 20
epoch 30
epoch 40
epoch 50
epoch 60
epoch 70
epoch 80
epoch 90
epoch 100
epoch 110
epoch 120
epoch 130
epoch 140
epoch 150
epoch 160
epoch 170
epoch 180
epoch 190
total numer of epochs trained:61100
```

Out[7]: <matplotlib.animation.FuncAnimation at 0x7fb686a04198>





In [17]:

Out[17]: