Python For Applied Machine Learning - Practical 4

Michael Halstead

April 21, 2024

1 Introduction

What will we be covering today?

1. Creating multi-variate Gaussian data;
2. Coding up KMeans clustering;
3. Using the spectral clustering technique from sklearn; and
4. Image normalisation.

2 Data Creation

For our clustering techniques we will create the data using *numpy.random.multivariate_normal*. You will do this in a specific flag from argparse and only run it if that flag is supplied to the command line. Once you have created the data you will save it using the *pickle* library.

You will create three multivariate Gaussian distributions with the following parameters:

$mu_0 =$ [0,0]; $cov_0 =$ [[1,0]
[0,1]]

$mu_1 =$ [4,1]; $cov_1 =$ [[1,0]
[0,1]]

$mu_2 =$ [2,2]; $cov_2 =$ [[0.5,0]
[0,0.5]]

Your training data will come from mu_0 and cov_0, and mu_1 and cov_1, while your evaluation data will be from all three distributions.

1. Create your multivariate Gaussian data for each of the distributions, both train and evaluation. Plot each of the distribution based data in a scatter plot, one for training and one for evaluation.

2. Create the following numpy matrices:

```
data = {}
data['X'] # stack the two training data distributions.
data['Y_0_1'] # stack the mu_0 and mu_1 distributions from the
    evaluation set.
data['Y_0_1_2'] # stack the mu_0, mu_1, and mu_2 distributions from
     the evaluation set.
# now the labels
data['LX'] # store zeros for mu_0 and ones for mu_1 from your
    training dataset (data['X'])
data['LY_0_1'] # store zeros for mu_0 and ones for mu_1 from your
    first evaluation set (data['Y_0_1'])
data['LY_0_1_2'] # store zeros for mu_0, ones for mu_1, and twos
    for mu_2 from your second evaluation set (data['Y_0_1_2']).
```

Once you have created these variables we will then save the dictionary into a pickle file in our data directory.

```
with open(flags.data_name, 'wb') as fid:
  pickle.dump(data, fid)
```

3 KMeans

In this exercise we will create a KMeans class for clustering similar data together. KMeans is based on the Euclidean distance between points in a dataset and N center locations. Each point is assigned to the closest center location. Then we iteratively update the center location based on the mean of the data associated with it. Below is the pseudo-code for the KMeans class:

```
class KMeans():
  def __init__(self, n_clusters, imax=100):
    # store the number of clusters in an object member.
    # store the maximum iterations in an object member.

  def fit(self, data):
    # data should be of the form N,D where N is the number of
    samples and D is the dimensions.
    # Randomly instantiate the centers (C) from the data locations
    as an object member.
    # for i < imax:
      # Assign X to C (Euclidean distance)
      # Calculate new C based on the mean of assigned points of
    data.

  def euclid(self, data, c):
    # calculate the Euclidean distance between the data and the
    supplied center.
    # return the distance as a matrix

  def predict(self, data):
    # instantiate an empty distance matrix
    # calculate the distance between each data point and each
    center.
    # return the associated center location which is the minimum
    distance (argmin).
```

Once you have created the KMeans class you will load the data that we created.

```
with open(flags.data_name, 'rb') as fid:
 data = pickle.load(fid)
```

Next we will create the KMeans object and complete the clustering. As metrics we will use the accuracy score (*from sklearn.metrics import accuracy_score*) and the completeness score (*from sklearn.metrics import completeness_score*). What do you think the problem is with the accuracy score?

```
obj # create KMeans object using a flag (argparse) for the number
    of clusters
# fit the object to data['X']
# predict data['Y_0_1']
# scatter plot the predictions where each center has its own colour
     using labels data['LY_0_1'].
# calculate the completeness_score and the accuracy_score

# predict data['Y_0_1_2']
# scatter plot the predictions where each center has its own colour
     using labels data['LY_0_1_2'].
# calculate the completeness_score and the accuracy_score
```

Once you have done this for 2 clusters try it for 3. What do you notice about the 3 cluster problem for $data[Y\_0\_1\_2]$?

4 Spectral Clustering

The next clustering algorithm we will use is directly from the sklearn package. Spectral clustering is a manner of clustering but does it in a different way to KMeans. You will need to import *from sklearn.cluster import SpectralClustering*.

```
# load the MVG data that we created earlier.
# create a spectral clustering object with the number of clusters
    as a flag. We will use assign_labels="discretize"
# fit predict data['X']
# calculate the centroids based on the output from fit predict and
    data['X']
# calculate the centroid closest to each of the data['Y_0_1']
    points
# calculate the completeness_score and the accuracy_score
```

Do you think this does a better job than KMeans?

5 Image Normalisation

The next task is image normalisation, which you covered in the lectures. There are a number of algorithms for image normalisation but we will concentrate on two, min max normalisation, and mean standard deviation normalisation.

$$X_{minmax} = \frac{X - min}{max - min}$$
$$X_{meanstd} = \frac{X - mu}{std} \tag{1}$$

In this case, X is an image, and min, max, mu, std are vectors with the same number of elements as X has channels (i.e. three for rgb, one for grey). The min max will be normalised on the range $[0, 1]$ as floating point values. The mean and standard deviation version will normalise such that the mean of the pixels is 0 and the output standard deviation is 1. Usually, for the mean and standard deviation we rescale a $[0, 255]$ image by dividing it by 255 first.

For the calculation of the mean and standard deviation you can choose one of two possibilities. You can use a mean of 0.5 (half way between 0 and 1) and a standard deviation of 0.5 also. Or you could calculate the mean and standard deviation of your training set.

1. Read in all the colour snippet red images, don't forget to divide them by 255. Split the images into a training group (90%) and an eval group (10%). Then for the training set calculate the minimum, maximum, mean, and standard deviation of each of the channels in the RGB images.

2. Once you have calculated these values you can normalise your eval set by the values you calculate. To do this you will need to create two functions, one for min max, and one for mean std, call them what you want. Your mean std function should have a default for the mean of 0.5 and the standard deviation of 0.5; meaning that if you don't supply your function with specific values it takes these as standard. Once you have normalised your eval set, display the different minimum and maximum values for your channels.

Why do you think we only use the values calculated from the training set and don't recalculate for our eval set? What do you think the problem is for the whole colour snippets dataset in terms of all of these calculated values? Should you use all the colours to give you a better range?