HT20_AAI_assignment_2_part2

November 20, 2020

In this assignment your task is to implement

1. an univaritare Gaussian Classifier and
2. an bivariate Gaussian Classifier

both using MLE parameter estimation.

Submission:

Submit your code to Canvas no later than **6th of December!**

Comment your code and use reasonable variable names so I am able to follow.

You can make use of basic numpy functions such as exp(), sum(), arange().

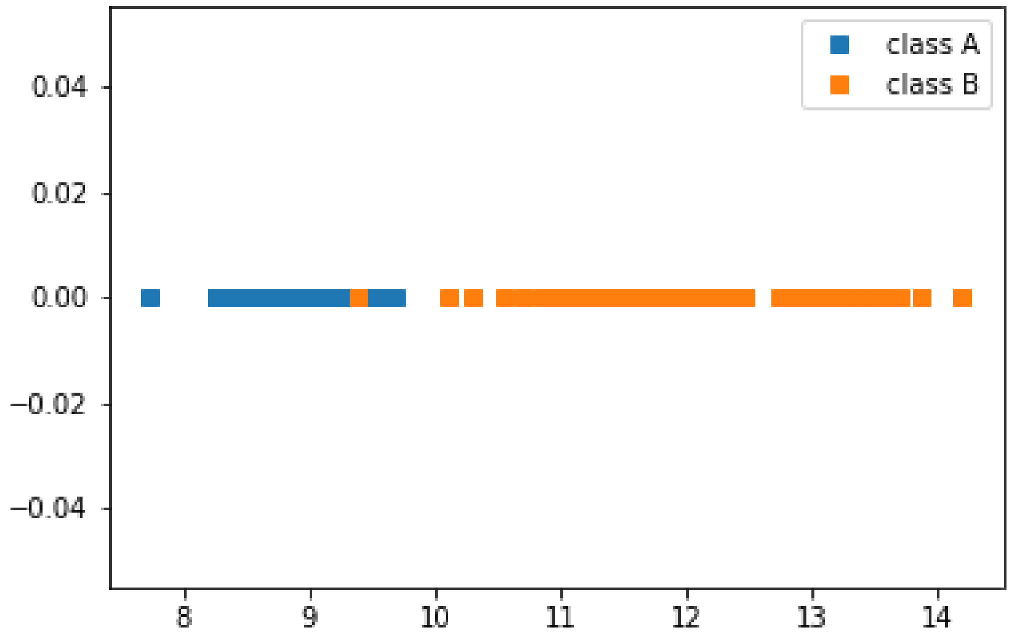
Obviously, you should not make use of build in functions like numpy.std() or numpy.mean().

1 Building a Univariate Gaussian Classifier

Here we create and plot the target data.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

num_samples = 50
x = np.arange(0,20,.01)

mean_classA = 9
mean_classB = 12
var_classA = .5
var_classB = 1

classA = np.vstack((np.random.normal(mean_classA,var_classA,num_samples),np.
 ↪zeros(num_samples)))
classB = np.vstack((np.random.normal(mean_classB,var_classB,num_samples),np.
 ↪ones(num_samples)))

data = np.hstack((classA,classB))

plt.plot(data[0,data[1,:]==0],np.zeros(num_samples), 's', label ="class A")
plt.plot(data[0,data[1,:]==1],np.zeros(num_samples), 's', label ="class B")
```

```
plt.legend()
plt.show()
```



1.1 1.1 Implement the functions to compute the parameters:

[795]:
```
def get_mean_mle(d):
    # your code goes here..
    return pass

def get_sigma_mle(d):
    # your code goes here..
    return pass
```

1.2 1.2. Implement Gaussian

So that you can use it to visualize the pdf of your distributions.

[26]:
```
def gaussian(x,mean,variance):
    # your code goes here..
    return pass
```

1.3 1.3 MLE step

Do the actual parameter computation here:

- split datset according to class labels
- compute means and variances for both classes
- since we know the true parameters you can check if they (roughly) match

[27]:
```
sigma_A = ...
mu_A = ...

sigma_B = ...
mu_A = ...
```

1.4 1.4 Visualize Classifier

Plot the pdfs of both distributions on top of the data.

[ ]:
```
plt.plot(samples_classA,np.zeros(len(samples_classA)), 's', label ="data A")
plt.plot(samples_classB,np.zeros(len(samples_classB)), 's', label ="data B")
#add plottting your gaussians here..


plt.legend()
plt.show()
```

1.5 1.5 Define a classification rule

Define a funtion that returns for a given value of x the class label. Note that this function only needs to decide if the current x belongs to class A - so we return True if x is of class A and false otherwise.

If you prefer [0,1] as outputs that is alright as well. So either [True,False] or [1,0].

Not [A,B] or [classA, classB] or something else.

[38]:
```
def classifyA(x, mu1,sigma1,mu2,sigma2):
    # your code goes here..
    return pass
```

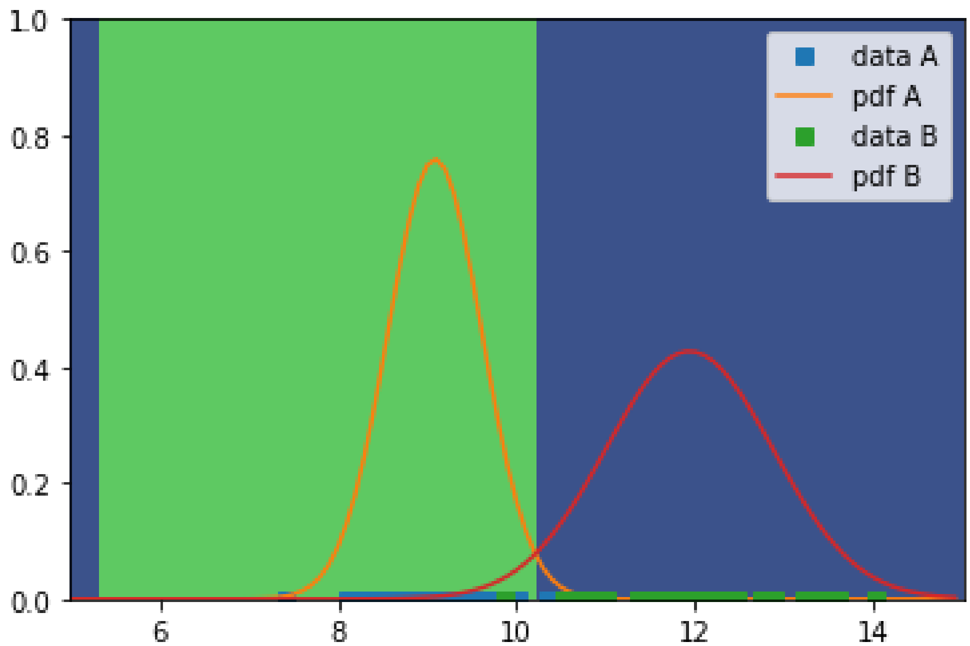
1.6 1.7 Printing Decision Boundary

The following script just uses your classifyA(..) function an colors the background accordingly.

In order for this to work your classifyA function hast to work with vectors for x not just single scalars.

[803]:
```
x = np.arange(0,20,0.05)
y = np.arange(0,1.5,0.1)
X,Y = np.meshgrid(x,y)

Z = classifyA(x, mu_A,sigma_A,mu_B,sigma_B).reshape(-1,1)
z = np.repeat(Z,len(y), axis=1).transpose()
```

```
ax = plt.gca()
ax.plot(samples_classA,np.zeros(len(samples_classA)), 's', label ="data A")
ax.plot(x,gaussian(x,mu_A, sigma_A), label ="pdf A")
ax.plot(samples_classB,np.zeros(len(samples_classB)), 's', label ="data B")
ax.plot(x,gaussian(x,mu_B, sigma_B), label ="pdf B")

ax.contourf(X,Y,z,1)

ax.legend()
ax.set_xlim([5, 15])
ax.set_ylim([0, 1])
plt.show()
```



1.7 1.8 Evaluate Classifier

Here you have to

1. choose a measure to assess the performance of you classifier - argue why it is a suitable choice.
2. can your measure help you improofe your classifier? if so - how?
3. try to improve it.

2 2 Bivariate Gaussian Classifier

In the multivariate case the probability density function of the Gaussian distribution is given by:

$$\mathcal{N}(x \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right) \tag{1}$$

In the two dimensional setting we need to compute the Gaussian parameters $\Theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ (for each of the two classes).

The two dimensional mean $\boldsymbol{\mu} = (\mu_1, \mu_2)$ and Σ the covariance matrix is given by

$$\Sigma = \begin{vmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{vmatrix} \tag{2}$$

we here insist, that Σ is symmetric and non-negative. This mean especially that the terms σ_{12} and σ_{21} are equal.

We will here use the following equations to compute the mean and variances for this classifier

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \tag{3}$$

$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T \tag{4}$$

where we can compute each of the terms in Σ by using

$$\sigma_{ij} = \frac{1}{N} \sum_{n=1}^{N} (x_{ni} - \mu_i)(x_{nj} - \mu_j) \tag{5}$$

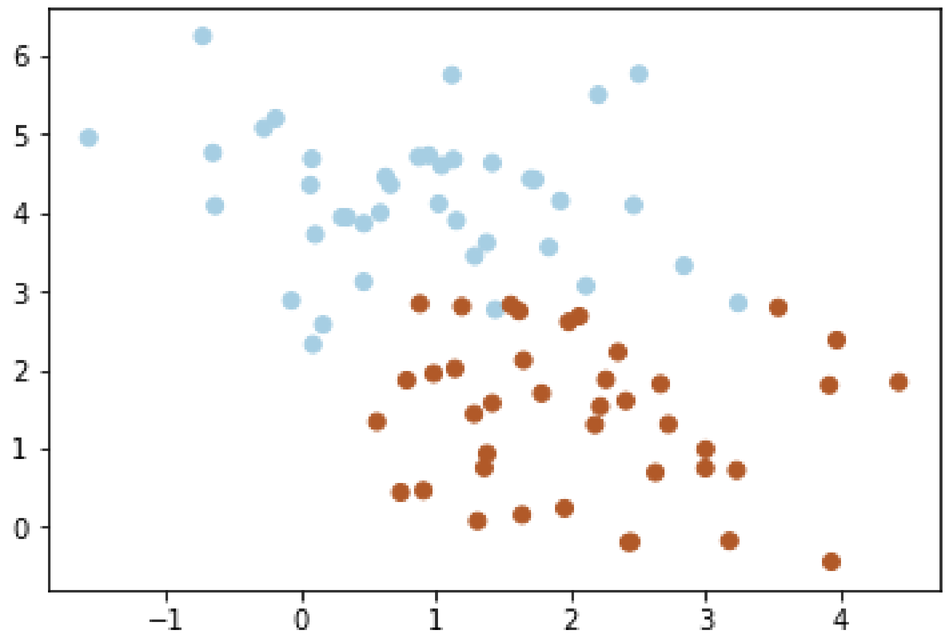
Your task is as in the one dimensional case to:

- compute the parameters
- classify data points
- choose and compute a suitable performance score

First we get some data:

[32]:
```
import sklearn.datasets
data, label = sklearn.datasets.make_blobs(n_samples=80, centers=2,␣
↪n_features=2,random_state=0)
plt.scatter(data[:,0],data[:,1],c=label, cmap = "Paired")
```

[32]:
```
<matplotlib.collections.PathCollection at 0x11e6808d0>
```




```
[13]: import numpy as np
```

2.1 2.1 Implement MLE parameter estimation

Here you have to implement the gievn equations.

```
[14]: def mu(data):
          # your code goes here..
          return pass

      def cov(data):
          # your code goes here..
          return pass
```

```
[33]: mu_A = ...
      cov_A = ...

      mu_B = ...
      cov_B = ...
```

2.2 2.2 Visualize classifier

There should be nothing to do here.

It relies on mu_A,cov_A,mu_B,cov_B beeing implemented correctly. They are directly feed to scipy.stats.multivariate_normal function. This means you do not have to implement the bivariate

Gaussian yourself.

If you are unsure about how they should look like you can find a description here : https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.multivariate_normal.html

The following script plots the data points, the computed distribution and the descision boundary.

[ ]:
```python
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

res = 500
maxs = data.max(axis=0)*1.1
mins = data.min(axis=0)*1.1

x = np.linspace(mins[0],maxs[0],res)
y = np.linspace(mins[1],maxs[1],res)
X,Y = np.meshgrid(x,y)

pos = np.array([X.flatten(),Y.flatten()]).T

distA = multivariate_normal(mu_A, cov_A)
distB = multivariate_normal(mu_B, cov_B)

fig = plt.figure(figsize=(12, 12))
ax = plt.gca()

#data points
plt.plot(samples_A[0,:],samples_A[1,:], '.', label ="data class A")
plt.plot(samples_B[0,:],samples_B[1,:], '.', label ="data class B")

# pdfs
ax.contour(X,Y, distA.pdf(pos).reshape(res,res))
ax.contour(X,Y, distB.pdf(pos).reshape(res,res))

#decision boundary
def classifyBivariate(x,dist1,dist2):
    return dist1.pdf(x) > dist2.pdf(x)
Z = classifyBivariate(pos,distA,distB)
ax.contourf(X,Y,Z.reshape(res,res), cmap='Paired')

plt.legend()
plt.show()
```

2.3 2.2 Questions

1. Does the classifier work as expected?
2. What can you do to improve it?