Maximum_Likelihood

January 20, 2021

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

ML Estimates

[ ]:
```python
#Defining mean and covariance of the distribution
mean =[2,-2]
cov = [[0.9,0.2],[0.2,0.3]]

for n_samples in [50, 5000]:
    #generate samples and calculate ML-estimates for the mean and covariance
    X = np.random.multivariate_normal(mean,cov,n_samples)
    mean_est = np.mean(X,axis=0)
    cov_est = np.cov(X.T)

    #print mean and covariance estimates and quality metrics
    print('mean_' + str(n_samples) + ' :')
    print(mean_est)
    print('\nl2_distance_' + str(n_samples) + ' :')
    print(np.linalg.norm(mean-mean_est))

    print('\ncovariance_' + str(n_samples) + ' :')
    print(cov_est)

    print('\nfrob_distance_' + str(n_samples) + ' :')
    print(np.linalg.norm(np.subtract(cov,cov_est)))

    #plot the samples and the true and estimated means
    plt.scatter(X[:,0], X[:,1], label='samples')
    plt.scatter(mean_est[0], mean_est[1], label='mean_est')
    plt.scatter(mean[0], mean[1], label='mean')
    plt.axis('equal')
    plt.title('n = ' + str(n_samples))
    plt.legend()
    plt.show()
```

ML Classification

```
[ ]: mean1 = [0,0,0]
     mean2 = [1,2,2]
     mean3 = [3,3,4]
     cov_diag = 0.8*np.identity(3)                                     #for the␣
      ↪first part of the exercise (diagonal covariance)
     cov_non_diag = np.array([[0.8,0.2,0.1],[0.2,0.8,0.2],[0.1,0.2,0.8]])      #for␣
      ↪the second part of the exercise (non-diagonal covariance)
     n_samples = 999
     titles = ['Test-Data diagonal', 'Test-Data non-diagonal']
     counter = 0
     for cov in [cov_diag, cov_non_diag]:
         #Generating the train and test data for the three classes and merging them␣
      ↪into one dataset for training and one for testing
         Train1 = np.random.multivariate_normal(mean1,cov,int(n_samples/3))
         Train2 = np.random.multivariate_normal(mean2,cov,int(n_samples/3))
         Train3 = np.random.multivariate_normal(mean3,cov,int(n_samples/3))
         Train = np.concatenate((Train1,Train2,Train3), axis=0)

         Test1 = np.random.multivariate_normal(mean1,cov,int(n_samples/3))
         Test2 = np.random.multivariate_normal(mean2,cov,int(n_samples/3))
         Test3 = np.random.multivariate_normal(mean3,cov,int(n_samples/3))
         Test = np.concatenate((Test1,Test2,Test3), axis=0)

         #Plotting the Test-Data to show the difference caused by the different␣
      ↪covariances
         fig = plt.figure()
         ax = Axes3D(fig)
        # ax.view_init(30, 30)
         ax.scatter(Test[:,0], Test[:,1],Test[:,2])
         ax.set_xlabel('X-axis')
         ax.set_ylabel('Y-axis')
         ax.set_zlabel('Z-axis')
         plt.title(titles[counter])
         plt.show()

         counter = counter + 1


         #Calculating ML estimates for mean and covariance
         mean1_est = np.mean(Train1,axis=0)
         mean2_est = np.mean(Train2,axis=0)
         mean3_est = np.mean(Train3,axis=0)
         Cov1_est = np.cov(Train1.T)
         Cov2_est = np.cov(Train2.T)
         Cov3_est = np.cov(Train3.T)
         Cov_est = (Cov1_est + Cov2_est + Cov3_est)/3
```

```python
#Classifying the samples in Test according to the distance to the means of␣
↪the classes
class_label = []
correct_label = 0
for x in Test:
    #calculating the distance of x to each mean
    d1 = np.linalg.norm(x-mean1_est)
    d2 = np.linalg.norm(x-mean2_est)
    d3 = np.linalg.norm(x-mean3_est)

    #finding the smallest distance and saving the according class label to␣
↪the list class_label
    m = min(d1,d2,d3)
    if m == d1:
        class_label.append(1)
    elif m == d2:
        class_label.append(2)
    elif m == d3:
        class_label.append(3)

#Calculating how many samples are labeled correctly (the implementation␣
↪works because the samples in Test are ordered according to classes)
for i in range(len(class_label)):
    if i < 333:
        if class_label[i] == 1:
            correct_label +=1
    elif i >= 333 and  i < 666:
        if class_label[i] == 2:
            correct_label +=1
    elif i >= 666 and  i < 999:
        if class_label[i] == 3:
            correct_label +=1

correct_frac = correct_label/len(Test)
print('fraction of correct assignment: ' + str(correct_frac) + '\n')
```

[ ]: