11849060 谭树杰

Lab 11 –k Nearest Neighbor and Parzen Window

Class Project 9

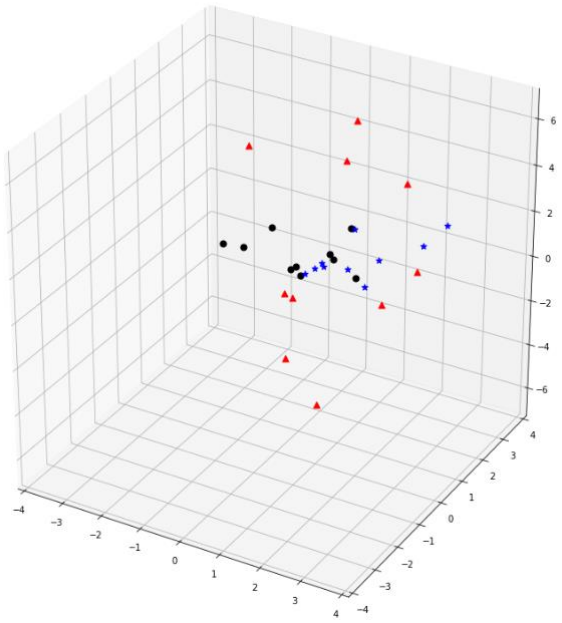
1. Write a C/C++/Java/Matlab program that uses a k-nearest neighbor method to classify input patterns. Use the following table as your training sample.

	ω_1			ω_2			ω_3		
sample	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
1	0.28	1.31	-6.2	0.011	1.03	-0.21	1.36	2.17	0.14
2	0.07	0.58	-0.78	1.27	1.28	0.08	1.41	1.45	-0.38
3	1.54	2.01	-1.63	0.13	3.12	0.16	1.22	0.99	0.69
4	-0.44	1.18	-4.32	-0.21	1.23	-0.11	2.46	2.19	1.31
5	-0.81	0.21	5.73	-2.18	1.39	-0.19	0.68	0.79	0.87
6	1.52	3.16	2.77	0.34	1.96	-0.16	2.51	3.22	1.35
7	2.20	2.42	-0.19	-1.38	0.94	0.45	0.60	2.44	0.92
8	0.91	1.94	6.21	-0.12	0.82	0.17	0.64	0.13	0.97
9	0.65	1.93	4.38	-1.44	2.31	0.14	0.85	0.58	0.99
10	-0.26	0.82	-0.96	0.26	1.94	0.08	0.66	0.51	0.88

- Experiment the program with the following data:
- *k = 3 , x_1 = (0.33, 0.58, - 4.8), x_2 = (0.27, 1.0, - 2.68), x_3 = (- 0.44, 2.8, 6.20)*
- Do the same thing with *k = 11*
- Compare the classification results between *k = 3* and *k = 11*

(use the most dominant class voting scheme amongst the k classes)

We first plot the datasets below as in lab10, class ω_1 is marked red triangle, class ω_2 is marked read circle, class ω_3 is marked blue star:



We calculated the distance of the test pattern to each labeled pattern, for example, the distance of x1 to each labeled pattern is shown below:

In [43]:
```
dist = np.linalg.norm(X - test[:,0][:, np.newaxis], axis=1)
dist
```

Out[43]:
```
array([[ 1.57968351,  4.62302509,  5.29080334],
       [ 4.02839919,  5.01876479,  4.63246155],
       [ 3.68210538,  5.57612769,  5.57676429],
       [ 1.08779594,  4.76552201,  6.66791572],
       [10.59799038,  5.31114865,  5.68467237],
       [ 8.08562923,  4.84087802,  7.03878541],
       [ 5.30420588,  5.53319076,  6.02087203],
       [11.10882982,  4.99609848,  5.79581746],
       [ 9.2842501 ,  5.52534162,  5.81330371],
       [ 3.89246708,  5.06644846,  5.69000879]])
```

We define KNN function with distance matrix and k as input, we calculated the top k minimum distance and corresponding labeled patterns. Then we use the most dominant class voting scheme amongst the k classes:

In [39]:
```
def KNN(dist, k):
    dist_sort = sorted(dist.flatten())
    num_class = np.sum(dist<=dist_sort[k-1],axis=0)
    return np.argmax(num_class)+1
```

The required result is shown below:

In [45]:
```
k = 3
for i in range(3):
    dist = np.linalg.norm(X - test[:,i][:, np.newaxis], axis=1)
    predict = KNN(dist,k)
    print(str(test[:,i]) + " belongs to class " + str(predict))
```

```
[ 0.33  0.58 -4.8 ] belongs to class 1
[ 0.27  1.   -2.68] belongs to class 1
[-0.44  2.8   6.2 ] belongs to class 1
```

In [46]:
```
k = 11
for i in range(3):
    dist = np.linalg.norm(X - test[:,i][:, np.newaxis], axis=1)
    predict = KNN(dist,k)
    print(str(test[:,i]) + " belongs to class " + str(predict))
```

```
[ 0.33  0.58 -4.8 ] belongs to class 1
[ 0.27  1.   -2.68] belongs to class 2
[-0.44  2.8   6.2 ] belongs to class 3
```

We could observe that a small value of k means that noise will have a higher influence on the result and a large value make the result more robust.

Class Project 9

2. The iris data set ("*iris.txt*"), one of the most well known data sets, has been used in pattern recognition literature to evaluate the performance of various classification and clustering algorithms. This data set consists of *150 4D* patterns belonging to three types of iris flowers (setosa, versicolor, and virginica). There are *50* patterns per class. The *4* features correspond to: *sepal length, sepal width, petal length*, and *petal width* (the unit of measurement is cm). The class labels are indicated at the end of every pattern.

   a) Assume that each pattern class has a multivariate Gaussian density with unknown mean vector and a common but unknown covariance matrix. This common covariance matrix could be computed as the average of the three covariance matrices estimated for each class. Design a Bayes classifier for this problem. Use half of the samples for design and the other half for test. Compute the error rate of the classifier.

   b) Now, assume that each pattern class has a multivariate Gaussian density with unknown mean vector and unknown but different covariance matrix. Design a Bayes classifier for this problem. Again, use half of the samples for design and the other half for test. Compute the error rate of the classifier.

   c) Find the non-parametric density estimates of the three classes using Parzen window density estimation method. Assume a 4 dimensional Gaussian kernel when estimating the density and try window widths of 0.01, 0.5, and 10.0. For the Gaussian kernel, assume that the covariance matrix is diagonal. The diagonal entries are the sample variance.

Read data from "iris.txt":

```
data = pd.read_csv('IRISdataSet.txt', sep="[\t]*",engine='python')
data.head()
```

```
/home/joshua/anaconda3/envs/PyTorch/lib/python3.6/site-packages/pandas/i
n-empty pattern match.
  yield pat.split(line.strip())
/home/joshua/anaconda3/envs/PyTorch/lib/python3.6/site-packages/pandas/i
n-empty pattern match.
  yield pat.split(line.strip())
```

	Type	PW	PL	SW	SL
0	Setosa	2	14	33	50
1	Verginica	24	56	31	67
2	Verginica	23	51	31	69
3	Setosa	2	10	36	46
4	Verginica	20	52	30	65

a) We estimate the mean and covariance of three classes.

```
set_mean = np.array([np.mean(B_set), np.mean(C_set), np.mean(D_set), np.mean(E_set)])
ver_mean = np.array([np.mean(B_ver), np.mean(C_ver), np.mean(D_ver), np.mean(E_ver)])
vir_mean = np.array([np.mean(B_vir), np.mean(C_vir), np.mean(D_vir), np.mean(E_vir)])
print(set_mean)
print(ver_mean)
print(vir_mean)
```

```
[ 2.26086957 14.34782609 33.82608696 49.43478261]
[13.25       41.79166667 27.66666667 59.20833333]
[20.35714286 53.85714286 29.57142857 64.46428571]
```

```
set_cov = np.cov(np.vstack([B_set,C_set, D_set, E_set]))
ver_cov = np.cov(np.vstack([B_ver,C_ver, D_ver, E_ver]))
vir_cov = np.cov(np.vstack([B_vir,C_vir, D_vir, E_vir]))
np.cov(np.vstack([B_set,C_set, D_set, E_set]))
```

```
array([[ 0.83794466,  0.99604743,  1.13833992,  1.47233202],
       [ 0.99604743,  3.87351779,  1.79051383,  3.25098814],
       [ 1.13833992,  1.79051383, 11.87747036,  8.66996047],
       [ 1.47233202,  3.25098814,  8.66996047, 13.80237154]])
```

We calculate the common variance as the average of the three covariance matrices estimated in each classs:

```
cov = (set_cov + ver_cov + vir_cov)/3
cov
```

```
array([[ 4.30473472,  5.56744638,  3.72811622,  5.77014625],
       [ 5.56744638, 16.42468421,  5.91061756, 15.59511697],
       [ 3.72811622,  5.91061756, 10.5193899 ,  9.26741187],
       [ 5.77014625, 15.59511697,  9.26741187, 25.73930558]])
```

We calculate the discriminate function use following formulas:

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^t \mathbf{x} + w_{i0}$$

$$\mathbf{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1}$$

$$\mathbf{w}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$$

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^t \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i)$$

```
def decision_boundaries(mu,cov,pwi,x):
    W1 = -0.5 * np.linalg.inv(cov)
    w1 = np.linalg.inv(cov) @ mu
    w10 = -0.5 * mu.T @ np.linalg.inv(cov) @ mu - \
            -0.5 * np.log(np.linalg.det(cov)) + np.log(pwi)
    g = x.T * Matrix(W1) * x + Matrix(w1[:,np.newaxis].T) * x + Matrix([w10])
    return g
```

```
g_set1 = decision_boundaries(set_mean,cov,p_set,X1)
g_ver1 = decision_boundaries(ver_mean,cov,p_ver,X1)
g1 = g_set1 - g_ver1

g_set2 = decision_boundaries(set_mean,cov,p_set,X1)
g_vir1 = decision_boundaries(vir_mean,cov,p_vir,X1)
g2 = g_set2 - g_vir1

g_ver2 = decision_boundaries(ver_mean,cov,p_ver,X1)
g_vir2 = decision_boundaries(vir_mean,cov,p_vir,X1)
g3 = g_ver2 - g_vir2
error = 0
```

The result is in Result_a.txt and error is 0.04

```
--------------number of misclassified points:3
--------------misclassify probability:0.04

```

b) The procedure is similar to a), but we use covariance of each class to calculate Bayes classifier:

```
f = open('Result_b.txt','a+')

g_set1 = decision_boundaries(set_mean,set_cov,p_set,X1)
g_ver1 = decision_boundaries(ver_mean,ver_cov,p_ver,X1)
g1 = g_set1 - g_ver1
g_set2 = decision_boundaries(set_mean,set_cov,p_set,X1)
g_vir1 = decision_boundaries(vir_mean,vir_cov,p_vir,X1)
g2 = g_set2 - g_vir1
g_ver2 = decision_boundaries(ver_mean,ver_cov,p_ver,X1)
g_vir2 = decision_boundaries(vir_mean,vir_cov,p_vir,X1)
g3 = g_ver2 - g_vir2
```

The result is in Result_b.txt and error is 0.093

```
--------------number of misclassified points:7
--------------misclassify probability:0.09333333333333334
```

c) We define parzen to calculate the probabilities of point belong to each class

```
def parzen(w,h,x):
    num_class = len(w)
    r=np.zeros((num_class))
    for i in range(num_class):
      hn=h
      rows, _ = w[i].shape
      for j in range(rows):
          hn=hn/np.sqrt(j+1)
          r[i]=r[i]+np.exp( -np.dot((x-w[i][j,:]), (x-w[i][j,:])) / (2*hn**2)) / (np.sqrt(2*np.pi)*hn)
      r[i]=r[i]/rows
    return r
```

We calculate the covariance whose diagonal entries are the sample variance.

```
f = open('Result_c.txt','a+')
B_var = np.var(B_data)
C_var = np.var(C_data)
D_var = np.var(D_data)
E_var = np.var(E_data)
cov = np.diag([B_var,C_var,D_var,E_var])
cov
```

```
array([[ 59.74222222,   0.        ,   0.        ,   0.        ],
       [  0.        , 285.97226667,   0.        ,   0.        ],
       [  0.        ,   0.        ,  16.27555556,   0.        ],
       [  0.        ,   0.        ,   0.        ,  63.71662222]])
```

We classify by Parzen window method:

```
error = 0
for i in range(A_test.shape[0]):
    r=parzen(X,h,test[i,:])
    num=np.where(r==np.max(r))[0] + 1
    if num == 1:
        s = str(i+1)+ ' belong to set \n'
        f.write(s)
    elif num == 2:
        s = str(i+1)+ ' belong to ver \n'
        f.write(s)
    elif num == 3:
        s = str(i+1)+ ' belong to vir \n'
        f.write(s)

    if num != A_test_new[i]:
        error = error + 1
p_error = error / A_test.shape[0]
s = '--------------number of misclassified points:' + str(error) + '\n'
f.write(s)
s = '--------------misclassify probability:' + str(p_error) + '\n'
f.write(s)
f.close()
```

The result is in Result_c.txt and error is 0.08

```
--------------number of misclassified points:6
--------------misclassify probability:0.08
```