EE20BTECH11047

assignment4

February 14, 2023

[83]:
```
import numpy as np
import scipy.stats as stats
from scipy.stats import norm
import matplotlib.pyplot as plt
import pandas as pd
from ctypes import sizeof
import scipy.optimize
```

[100]:
```
data = pd.read_csv('A4_input.txt',sep= ' ')
data =pd.DataFrame(data)
print(data)
```

```
           x         y  sigma_y
0   0.417022  0.121328      0.1
1   0.720324  0.849527      0.1
2   0.000114 -1.017014      0.1
3   0.302333 -0.391716      0.1
4   0.146756 -0.680730      0.1
5   0.092339 -0.748515      0.1
6   0.186260 -0.702849      0.1
7   0.345561 -0.074994      0.1
8   0.396767  0.041118      0.1
9   0.538817  0.418206      0.1
10  0.419195  0.104199      0.1
11  0.685220  0.771592      0.1
12  0.204452 -0.561584      0.1
13  0.878117  1.433748      0.1
14  0.027388 -0.971264      0.1
15  0.670468  0.843497      0.1
16  0.417305 -0.060413      0.1
17  0.558690  0.389839      0.1
18  0.140387 -0.768235      0.1
19  0.198101 -0.649073      0.1
```

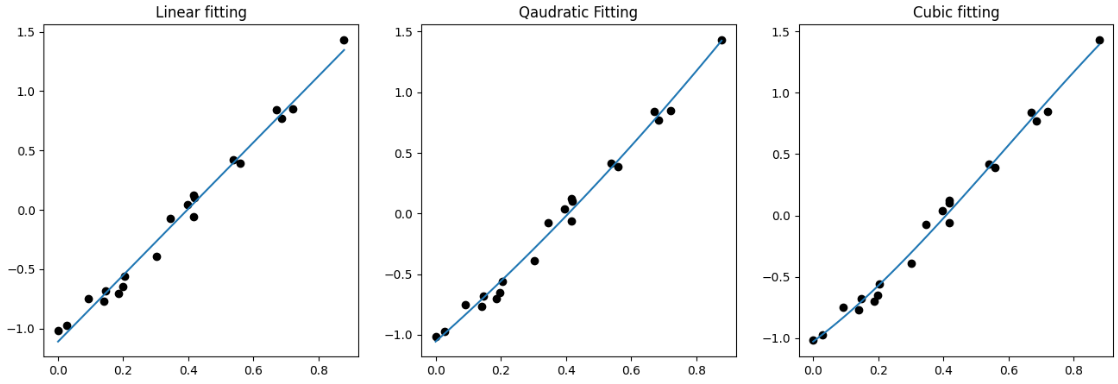
[101]:
```
x=data['x']
y=data['y']
sigma=data['sigma_y']
```

[102]:
```
def linear_fit(x,m,c):
    return m*x+c

def quadratic_fit(x,a,b,c):
    return a + b*x + c*(x**2)

def cubic_fit(x,a,b,c,d):
    return a + b*x + c*(x**2) + d*(x**3)

linear_fit_params,linear_cov = scipy.optimize.curve_fit(linear_fit,x,y,sigma =␣
 ↪sigma)
print(f"Best fit line for is y = {linear_fit_params[0]}x +␣
 ↪{linear_fit_params[1]}")

quadratic_fit_params,linear_cov = scipy.optimize.
 ↪curve_fit(quadratic_fit,x,y,sigma = sigma)
print(f"Best fit quadratic eq for is y = {quadratic_fit_params[0]} +␣
 ↪{quadratic_fit_params[1]}x + {quadratic_fit_params[2]}x**2")

cubic_fit_params,linear_cov = scipy.optimize.curve_fit(cubic_fit,x,y,sigma =␣
 ↪sigma)
print(f"Best fit cubic eq for is y = {cubic_fit_params[0]} +␣
 ↪{cubic_fit_params[1]}x + {cubic_fit_params[2]}x**2 +␣
 ↪{cubic_fit_params[3]}x**3")
```

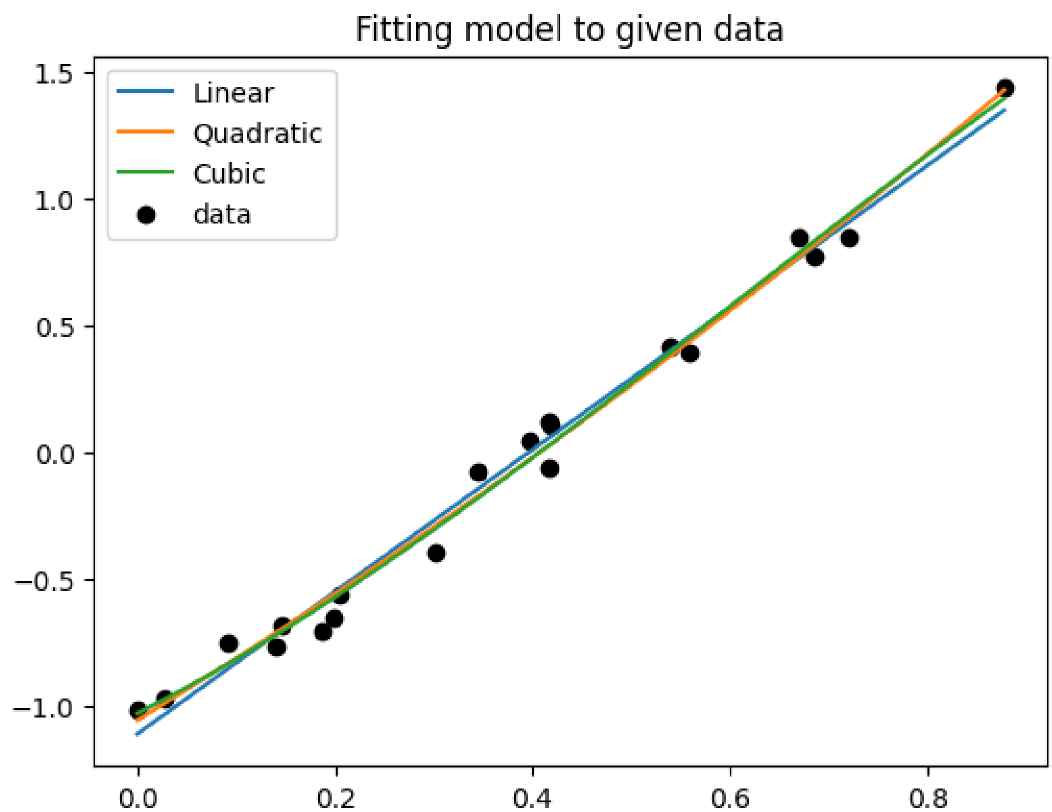
```
Best fit line for is y = 2.7978986063937183x + -1.1102808170221141
Best fit quadratic eq for is y = -1.055789148253146 + 2.384751844139253x +
0.502612970397154x**2
Best fit cubic eq for is y = -1.0291046143544065 + 1.971840396239574x +
1.744513808670288x**2 + -0.9672503050014959x**3
```

[129]:
```
x_axis= np.linspace(0,max(x),100)
plt.figure(figsize=(16,5))
plt.subplot(1,3,1)
plt.scatter(x,y,color = 'black')
plt.plot(x_axis,linear_fit(x_axis,linear_fit_params[0],linear_fit_params[1]))
plt.title("Linear fitting")
plt.subplot(1,3,2)
plt.scatter(x,y,color = 'black')
plt.
 ↪plot(x_axis,quadratic_fit(x_axis,quadratic_fit_params[0],quadratic_fit_params[1],quadratic_
plt.title("Qaudratic Fitting")
plt.subplot(1,3,3)
plt.scatter(x,y,color = 'black')
plt.
 ↪plot(x_axis,cubic_fit(x_axis,cubic_fit_params[0],cubic_fit_params[1],cubic_fit_params[2],cu
plt.title("Cubic fitting")
```

```
plt.show()
```



[123]:
```
plt.plot(x_axis,linear_fit(x_axis,linear_fit_params[0],linear_fit_params[1]))
plt.
 ↪plot(x_axis,quadratic_fit(x_axis,quadratic_fit_params[0],quadratic_fit_params[1],quadratic_
plt.
 ↪plot(x_axis,cubic_fit(x_axis,cubic_fit_params[0],cubic_fit_params[1],cubic_fit_params[2],cu
plt.scatter(x,y,color = 'black')
plt.legend(['Linear' , 'Quadratic' , 'Cubic' , 'data'])
plt.title("Fitting model to given data")
plt.show()
```



[104]:
```
def chi2_likelihood(degree,params,x,y,sigma):
    if degree==1:
        y_dash = linear_fit(x,params[0],params[1])
    if degree==2:
        y_dash = quadratic_fit(x,params[0],params[1],params[2])
    if degree==3:
        y_dash = cubic_fit(x,params[0],params[1],params[2],params[3])
    error_fitting = np.sum(((y-y_dash)/sigma)**2)
    return stats.chi2.pdf(error_fitting,len(x)-1-degree),error_fitting
```

[106]:
```
linear_chi2_liklihood,linear_error_fitting =␣
 ↪chi2_likelihood(1,linear_fit_params,x,y,sigma)
print("chi2 liklihood for linear fitting is : ",linear_chi2_liklihood)
quadratic_chi2_liklihood,quadratic_error_fitting =␣
 ↪chi2_likelihood(2,quadratic_fit_params,x,y,sigma)
print("chi2 liklihood for quadratic fitting is : ",quadratic_chi2_liklihood)
cubic_chi2_liklihood,cubic_error_fitting =␣
 ↪chi2_likelihood(3,cubic_fit_params,x,y,sigma)
print("chi2 liklihood for linear fitting is : ",cubic_chi2_liklihood)
print(" ")
```

```
print('p-value quadratic fit wrt linear fit: {:.5f}'.format(stats.chi2.
 ↪sf(quadratic_error_fitting,len(x)-2-1)))
print('p-value cubic fit wrt linear fit: {:.5f}'.format(stats.chi2.
 ↪sf(cubic_error_fitting,len(x)-3-1)))
```

```
chi2 liklihood for linear fitting is :  0.045383795585918596
chi2 liklihood for quadratic fitting is :  0.036608447550140304
chi2 liklihood for linear fitting is :  0.04215280601005979

p-value quadratic fit wrt linear fit: 0.92340
p-value cubic fit wrt linear fit: 0.90987
```

From the liklihood values we can see that chi2 value is higher for linear fitting therefore best fitting model is linear fitting

[90]:
```
def AIC(degree,params,x,y,sigma):
    if degree == 1:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,linear_fit(x,params[0],params[1]),sigma))+2*(degree + 1)
    if degree == 2:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,quadratic_fit(x,params[0],params[1],params[2]),sigma)) + 2*(degree␣
 ↪+ 1)
    if degree == 3:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,cubic_fit(x,params[0],params[1],params[2],params[3]),sigma)) +␣
 ↪2*(degree + 1)
    return log_liklihood

def BIC(degree,params,x,y,sigma):
    if degree == 1:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,linear_fit(x,params[0],params[1]),sigma))+np.log(len(x))*(degree +␣
 ↪1)
    if degree == 2:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,quadratic_fit(x,params[0],params[1],params[2]),sigma)) + np.
 ↪log(len(x))*(degree + 1)
    if degree == 3:
        log_liklihood  = -2*np.sum(stats.norm.
 ↪logpdf(y,cubic_fit(x,params[0],params[1],params[2],params[3]),sigma)) + np.
 ↪log(len(x))*(degree + 1)
    return log_liklihood

def AICc(degree,params,x,y,sigma):
    k = degree+1
    N = len(x)
```

```
    return AIC(degree,params,x,y,sigma) + 2*k*(k+1)/(N-k-1)
```

[91]:
```
print(f"Values for linear fitting are : ")
print("AIC  =", AIC(1,linear_fit_params,x,y,sigma))
print("BIC  =", BIC(1,linear_fit_params,x,y,sigma))
print("AICc =" ,AICc(1,linear_fit_params,x,y,sigma))
print(" ")
print(f"Values for quadratic fitting are : ")
print("AIC  =", AIC(2,quadratic_fit_params,x,y,sigma))
print("BIC  =", BIC(2,quadratic_fit_params,x,y,sigma))
print("AICc =" ,AICc(2,quadratic_fit_params,x,y,sigma))
print(" ")
print(f"Values for cubic fitting are : ")
print("AIC  =", AIC(3,cubic_fit_params,x,y,sigma))
print("BIC  =", BIC(3,cubic_fit_params,x,y,sigma))
print("AICc =" ,AICc(3,cubic_fit_params,x,y,sigma))
```

```
Values for linear fitting are :
AIC  = -40.03668681607269
BIC  = -38.04522226896471
AICc = -39.330804463131514

Values for quadratic fitting are :
AIC  = -39.849820624005616
BIC  = -36.862623803343645
AICc = -38.349820624005616

Values for cubic fitting are :
AIC  = -38.26081851760257
BIC  = -34.27788942338661
AICc = -35.59415185093591
```

Here we can see that values of all three AIC, BIC, AICc are least for linear fitting. Therefore we can conclude that Linear fitting is the best model for the given dataset

0.1 Question 2

[92]:
```
data = np.array([[ 0.42,  0.72,  0.  ,  0.3 ,  0.15,
                   0.09,  0.19,  0.35,  0.4 ,  0.54,
                   0.42,  0.69,  0.2 ,  0.88,  0.03,
                   0.67,  0.42,  0.56,  0.14,  0.2  ],
                 [ 0.33,  0.41, -0.22,  0.01, -0.05,
                  -0.05, -0.12,  0.26,  0.29,  0.39,
                   0.31,  0.42, -0.01,  0.58, -0.2 ,
                   0.52,  0.15,  0.32, -0.13, -0.09 ],
                 [ 0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
```

```
                    0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1  ]])
x, y, sigma_y = data
```

[93]:
```
linear_params_jvdp,linear_cov_params = scipy.optimize.
 ↪curve_fit(linear_fit,x,y,sigma =sigma_y)
quadratic_params_jvdp,quadraic_cov_jvdp =scipy.optimize.
 ↪curve_fit(quadratic_fit,x,y,sigma=sigma_y)
```

[94]:
```
print("Compairing AIC values for linear and quadrating fitting : ")
AIC_linear = AIC(1,linear_params_jvdp,x,y,sigma)
print("Linear fitting : ",AIC_linear)
AIC_quadratic = AIC(2,quadratic_params_jvdp,x,y,sigma)
print("Quadratic fitting : ",AIC_quadratic)
```

```
Compairing AIC values for linear and quadrating fitting :
Linear fitting :  -40.02173401322526
Quadratic fitting :  -39.8830271730082
```

Here value of AIC is lesser for linear fitting.

Therefore best fitting model is Linear

[95]:
```
print("Compairing BIC values for linear and quadrating fitting : ")
BIC_linear = BIC(1,linear_params_jvdp,x,y,sigma)
print("Linear fitting : ",BIC_linear)
BIC_quadratic = BIC(2,quadratic_params_jvdp,x,y,sigma)
print("Quadratic fitting : ",BIC_quadratic)
```

```
Compairing BIC values for linear and quadrating fitting :
Linear fitting :  -38.03026946611728
Quadratic fitting :  -36.89583035234623
```

Here value of BIC is lesser for linear fitting.

Therefore best fitting model is Linear

[96]:
```
print("Compairing AICc values for linear and quadrating fitting : ")
AICc_linear = AICc(1,linear_params_jvdp,x,y,sigma)
print("Linear fitting : ",AICc_linear)
AICc_quadratic = AICc(2,quadratic_params_jvdp,x,y,sigma)
print("Quadratic fitting : ",AICc_quadratic)
```

```
Compairing AICc values for linear and quadrating fitting :
Linear fitting :  -39.31585166028409
Quadratic fitting :  -38.3830271730082
```

Here value of AICc is lesser for linear fitting.

Therefore best fitting model is Linear

These results agree with the frequentist model comparison results shown on the blog

0.2 Question 3

Title :A monitoring framework for deployed machine learning models with supply chain examples pulsars

Link : https://arxiv.org/pdf/2211.06239.pdf

In this paper author describes about a framework for a big data supply chain application and its implementation for a big data supply chain application. This implementation is used to study drift in model features,predictions and performance

According to the Penn-state website,the correct uses of the K-S test are:

1) Application in 1 dimension.

2) The model should not be derived from the dataset.

3) KS test is most sensitive when the EDFs differ in a global fashion near the center of the distribution.

In this paper the data is in one dimensional. The value of D is 0.005227.This small value of D signifies that our model gives best prediction values.

Hence K-S test is applied correctly as per the warnings of Penn State

0.3 Question 4

[113]:
```
Higgs_p_value = np.array([10**-1,10**-2,10**-3,10**-5,10**-7,10**-9])
Higgs_significance = stats.norm.isf(Higgs_p_value)
print("Significance in terms of number of number of sigmas of the Higgs boson␣
↪discovery: ", Higgs_significance )
```

```
Significance in terms of number of number of sigmas of the Higgs boson
discovery:  [1.28155157 2.32634787 3.09023231 4.26489079 5.19933758 5.99780702]
```

[114]:
```
ligo_p_value = 2 * (10 ** -7)
ligo_significance = stats.norm.isf(ligo_p_value)
print(f"Significance for LIGO discovery: {ligo_significance} sigmas\n")
```

```
Significance for LIGO discovery: 5.068957749717791 sigmas
```

[112]:
```
chi2_super_K = 65.2
dof_super_K = 67
chi2_gof_super_K = 1 - stats.chi2(dof_super_K).cdf(chi2_super_K)
print(f"The Chi-Squared goodness of fit for Super-K discovery:␣
↪{chi2_gof_super_K}")
```

```
The Chi-Squared goodness of fit for Super-K discovery: 0.5394901931099038
```

[ ]:
```

```