Faculty of Technology, Policy and Management

SPM 4450 Fundamentals of Data Analytics

Verifying Bias-Variance Dilemma

With assumed “true” function

Authors	:	Paraskevi Kokosia	4417062
		Bramka Arga Jafino	4516516
		Rhythima Shinde	4516389
Professor	:	Prof. Jan Van Der Berg	

Motivation

One of the few things that lurks as much challenge in data analysis is the bias-variance dilemma. In order to improve the data fitting process and to avoid the mistake of over- or under-fitting we need to understand how different sources of error lead to bias and variance. The bias-variance dilemma refers to a trade-off between a model's ability to minimize bias and variance.

In this report an effort will be made to understand these two type of errors and the behaviour that each error performs in relation with the linear, cubic and quadratic regression models using sample sizes of 10,100 and 1000 (and even 5, and 5000 to check expectations).

Analysis

Based on the prediction error formula, we have repeated the bias variance experiment. We begin by defining our own function f(x) where x is distributed uniformly between the interval (1, 10) with the interval size of 1/9, 1/99, 1/999 to get the sample size of 10, 100, and 1000. The function is then distributed normally for every value of x with the variance of 1, such that Y=N (μ=5+x*cos(x), σ^2=1). The main function here is Y=5+x*cos(x). The expansion of cos x gives a polynomial function as shown in equation 1 below.

$$\cos x = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 - \ldots \text{ for } -\infty < x < \infty \qquad \text{[Equation 1]}$$

In order to come up with the linear, cubic and quadratic regression functions we used least squares estimation to estimate the parameters. In the data processing phase we analysed the model first for the linear approximations by generating line graphs for each sample size for all simulations. The same procedure was followed in order to generate the graphs for the cubic and quadratic approximation. In the error calculation phase based on the equation 2 we calculated the mean squared error for each of the three approximations. Then the conclusions of the simulation analysis are presented and the python code can be found in appendix and on GitHub (https://github.com/rhythimashinde/FDA/blob/master/Final.py).

The prediction error can be explained by the following equation:

$$\mathrm{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2 \qquad \text{[Equation 2]}$$

Where:

$$\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)\right] - f(x)$$

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\right)^2\right]$$

And σ^2 is the noise that cannot be reduced by any model. (Berthold & Hand, 2003)

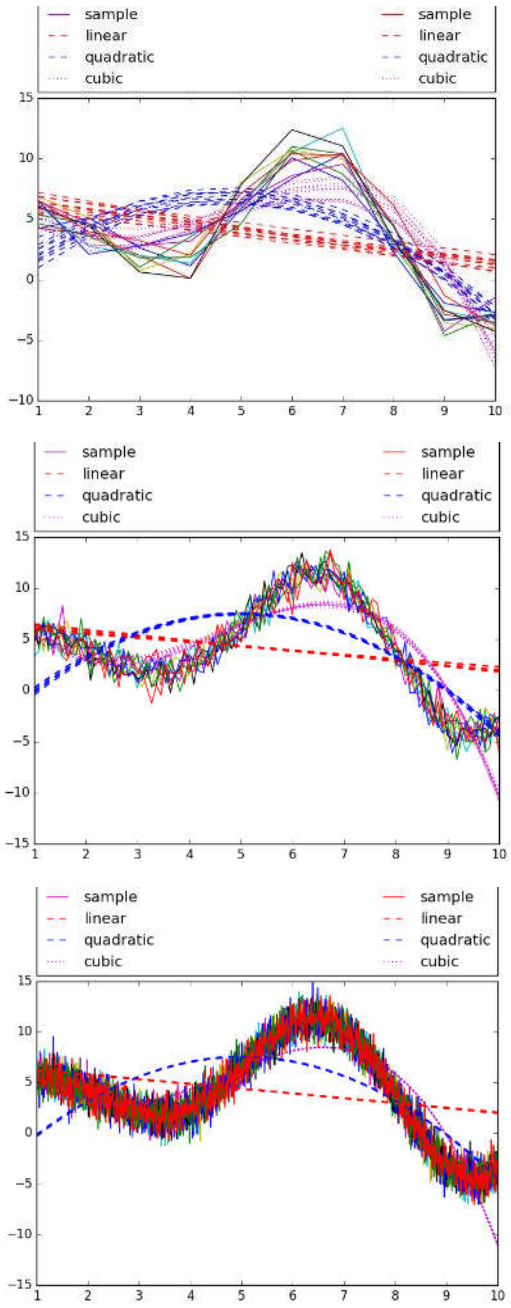

Figure 1: The linear, quadratic and cubic regression fits for 10, 100, 1000 samples generated respectively

Table 1: The bias, variance and mean square error for linear, quadratic and cubic regressions (sample size: 10, 100, 1000)

	Bias			Variance			Mean Square error		
Column	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
m	10	100	1000	10	100	1000	10	100	1000
Linear	17.927	18.912	18.953	0.286	0.019	0.001	18.213	18.930	18.954
Quadratic	10.690	10.506	10.445	0.387	0.028	0.002	11.077	10.534	10.447
Cubic	6.037	4.472	4.245	0.442	0.040	0.003	6.479	4.512	4.248

Conclusions

Observation on Bias:

From Table 1, we will try to deduce the performance of various models. Considering the bias, linear regression model always shows highest bias and this bias does not reduce with increasing sample size (around the range of 18). This implies that for linear models, there is irreducible bias error. Similarly, the bias of quadratic and cubic model can be seen to be very low.

Observation on Variance:

The variance always decreases with sample size significantly whichever the model is, which shows that the variance error is reducible. But the variance increases for small sample size significantly as the model starts becoming complex (check column 4, n=10). Similarly the variance increases for larger sample size but not as significantly (check column 6, n=1000).

Conclusions on comparing models:

As a comparison of all the models, for small sample size (say 10) linear model is much better than quadratic and cubic model (despite high bias, as it has relatively low variance). At the same time, for larger sample size, quadratic and cubic performs much better (as the bias is low and the variance have negligible difference – as can be seen for the sample of 1000). Cubic and quadratic needs more data to have lowest prediction error (prediction error decreases with increasing sample size), provided enough sample size (data) is available.

The expected behaviour of mean square error is that, for higher model complexity, it decreases as the sample size gets larger, but it increases for the smaller sized models. Here, in conclusion, for larger sample sizes, complex models fit better (seeing last column, n=1000). But, as the function here is ongoing polynomial as shown in equation 2, the models (linear, quadratic and cubic) show a very high bias as compared to the variance and thus the actual expected behaviour of the mean square error cannot be seen for the smaller size samples. But it can be seen that the difference in models (differences in mean square error) for small size (column 7, n=10) does not decrease as significantly as for large size (last column, n=1000). This concludes that for such complex functions, complex models like cubic fits the best. Even checking for the model as low as 5 and as large as 5000 delivered similar trends (increasing variance and decreasing bias, but due to high value of bias, the mean square error is highest for linear and lowest for cubic).

The reason behind this can be validated by the principle of overfitting of the model that takes place for bigger samples (like n=1000) which increases the variance with increasing complexity of the model. The bias reduces irrespective of the sample size as the complex models fit better for this complex polynomial function.

References

Berthold, M., & Hand, D. (2003). *Intelligent data analysis: an introduction.* Springer Science & Business Media.

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import csv
#import panda as pd
from scipy.interpolate import *
f=[]
a1=[]
a2=[]
a3=[]
samplesize=5000
i=0
trainingsets = 10
while i<trainingsets:
        x=np.linspace(1,10,samplesize)
        #y=x**2
        y=5+x*np.cos(x)
        s=np.random.normal(y,1,samplesize)
        f.append(s)
        f.append(x)
        b = open('test1000final.csv', 'w')
        a = csv.writer(b)
        #print f
        a.writerows(f)
        p1=np.polyfit(x,s,1)
        p2=np.polyfit(x,s,2)
        p3=np.polyfit(x,s,3)
        plt.plot(x,s)
        plt.plot(x,np.polyval(p1,x),'r--')
        plt.plot(x,np.polyval(p2,x),'b--')
        plt.plot(x,np.polyval(p3,x),'m:')
        #print p1
        #print p2
        #print p3
        a1.append(p1)
        a.writerows(a1)
        a2.append(p2)
        a.writerows(a2)
        a3.append(p3)
        a.writerows(a3)
        i +=1
plt.show()
x=np.linspace(1,10,samplesize)

i=0
f0=[]

while i < 10:
    y=a1[i][0] * x + a1[i][1]
    f0.append(y)
    i = i + 1

##linear approximation f(x|T)
x=np.linspace(1,10,samplesize)

i=0
flinear=[]

while i < trainingsets:
    y=a1[i][0] * x + a1[i][1]
    flinear.append(y)
    i = i + 1

##the E[f(x|T)]
j = 0
expectedflinearlist = []
```

```
while j < samplesize:
    expectedflinear = sum(row[j] for row in flinear) / len(flinear)
    expectedflinearlist.append(expectedflinear)
    j = j + 1

##the f(x)
x=np.linspace(1,10,samplesize)
y=5+x*np.cos(x)

##the squared bias (f(x) - E[f(x|T)])^2
squaredbiaslist = list((np.array(y) - np.array(expectedflinearlist))**2)
squaredbiaslinear = reduce(lambda x, y: x + y, squaredbiaslist) /
len(squaredbiaslist)
squaredbiaslinear

##the variance
variancelinear = []
i=0
while i < trainingsets:
    h=list((np.array(flinear[i]) - np.array(expectedflinearlist))**2)
    variancelinear.append(h)
    i = i + 1

expectedvariancelinearlist = []
j=0
while j < samplesize:
    expectedvariancelinear = sum(row[j] for row in variancelinear) /
len(variancelinear)
    expectedvariancelinearlist.append(expectedvariancelinear)
    j = j + 1

finalvariancelinear = reduce(lambda x, y: x + y, expectedvariancelinearlist) /
len(expectedvariancelinearlist)

print squaredbiaslinear
print finalvariancelinear

##quadratic approximation f(x|T)
x=np.linspace(1,10,samplesize)

i=0
fquadratic=[]

while i < trainingsets:
    y=a2[i][0] * (x)**2 + a2[i][1] * x + a2[i][2]
    fquadratic.append(y)
    i = i + 1

##the E[f(x|T)]
j = 0
expectedfquadraticlist = []

while j < samplesize:
    expectedfquadratic = sum(row[j] for row in fquadratic) / len(fquadratic)
    expectedfquadraticlist.append(expectedfquadratic)
    j = j + 1

##the f(x)
x=np.linspace(1,10,samplesize)
y=5+x*np.cos(x)

##the squared bias (f(x) - E[f(x|T)])^2
squaredbiaslist = list((np.array(y) - np.array(expectedfquadraticlist))**2)
squaredbiasquadratic = reduce(lambda x, y: x + y, squaredbiaslist) /
len(squaredbiaslist)

##the variance
variancequadratic = []
```

```
i=0
while i < trainingsets:
    h=list((np.array(fquadratic[i]) - np.array(expectedfquadraticlist))**2)
    variancequadratic.append(h)
    i = i + 1

expectedvariancequadraticlist = []
j=0
while j < samplesize:
    expectedvariancequadratic = sum(row[j] for row in variancequadratic) /
len(variancequadratic)
    expectedvariancequadraticlist.append(expectedvariancequadratic)
    j = j + 1

finalvariancequadratic = reduce(lambda x, y: x + y, expectedvariancequadraticlist)
/ len(expectedvariancequadraticlist)

print squaredbiasquadratic
print finalvariancequadratic

##cubic approximation f(x|T)
x=np.linspace(1,10,samplesize)

i=0
fcubic=[]

while i < trainingsets:
    y=a3[i][0] * (x)**3 + a3[i][1] * (x)**2 + a3[i][2] * x + a3[i][3]
    fcubic.append(y)
    i = i + 1

##the E[f(x|T)]
j = 0
expectedfcubiclist = []

while j < samplesize:
    expectedfcubic = sum(row[j] for row in fcubic) / len(fcubic)
    expectedfcubiclist.append(expectedfcubic)
    j = j + 1

##the f(x)
x=np.linspace(1,10,samplesize)
y=5+x*np.cos(x)

##the f(x|T)

##the squared bias (f(x) - E[f(x|T)])^2
squaredbiaslist = list((np.array(y) - np.array(expectedfcubiclist))**2)
squaredbiascubic = reduce(lambda x, y: x + y, squaredbiaslist) /
len(squaredbiaslist)
squaredbiascubic

##the variance
variancecubic = []
i=0
while i < trainingsets:
    h=list((np.array(fcubic[i]) - np.array(expectedfcubiclist))**2)
    variancecubic.append(h)
    i = i + 1

expectedvariancecubiclist = []
j=0
while j < samplesize:
    expectedvariancecubic = sum(row[j] for row in variancecubic) /
len(variancecubic)
    expectedvariancecubiclist.append(expectedvariancecubic)
    j = j + 1
```

```
finalvariancecubic = reduce(lambda x, y: x + y, expectedvariancecubiclist) /
len(expectedvariancecubiclist)

print squaredbiascubic
print finalvariancecubic
```