Advanced Computational Methods for Data Science CS 6301.012

Assignment 5

Sushmitha Mohan Raj sxm144630

Sreesha Nagaraj sxn146630

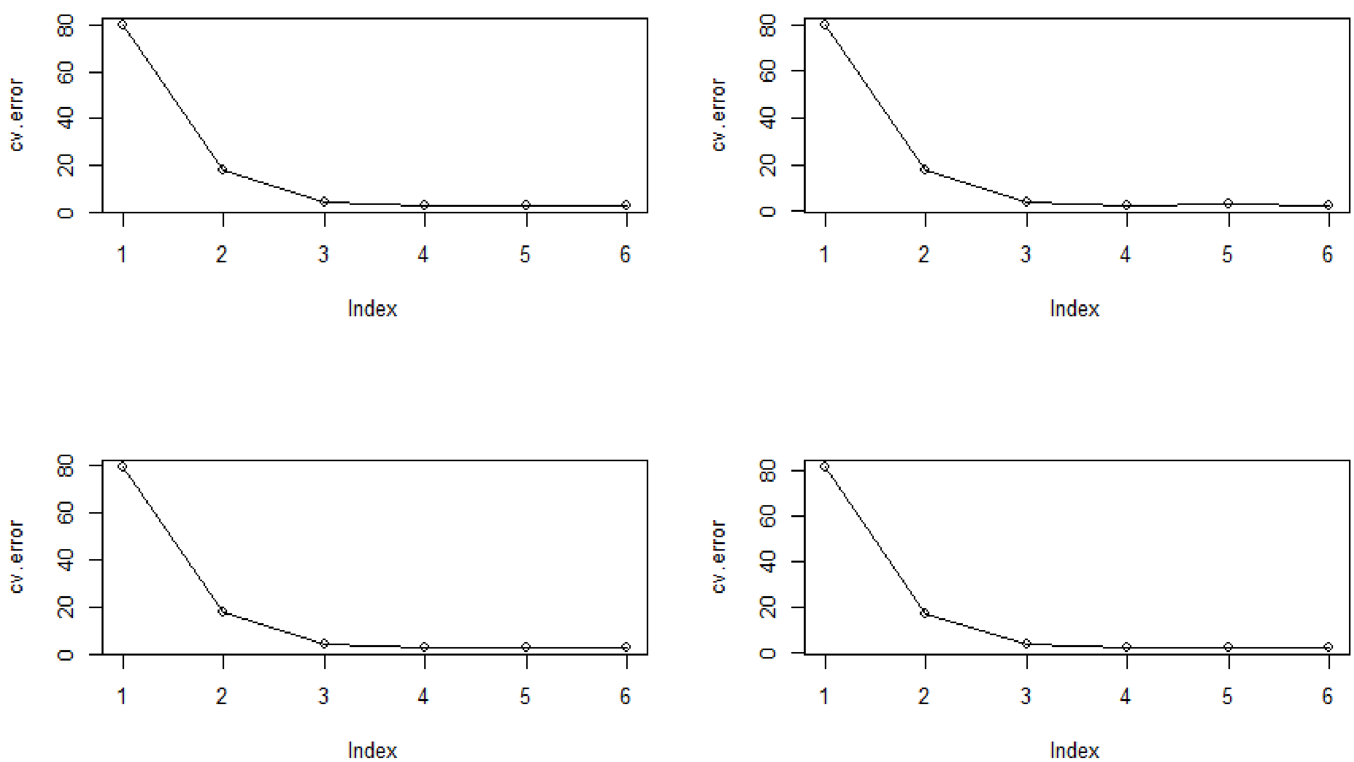
Yacht Hydrodynamics

Below is the R code:

```
attach(yacht_hydrodynamics)

#K=1

cv.error=rep (0,6)

for (i in 1:6){

  glm.fit=glm(V7~poly(V6 ,i),data=yacht_hydrodynamics)

  cv.error[i]=cv.glm (yacht_hydrodynamics ,glm.fit)$delta [1]

}

cv.error

plot(cv.error,type="o",col="black")

#K=2

cv.error=rep (0,6)

for (i in 1:6){

  glm.fit=glm(V7~poly(V6 ,i),data=yacht_hydrodynamics)

  cv.error[i]=cv.glm (yacht_hydrodynamics ,glm.fit,K=2)$delta [1]

}

cv.error

plot(cv.error,type="o",col="black")

#K=5

cv.error=rep (0,6)

for (i in 1:6){

  glm.fit=glm(V7~poly(V6 ,i),data=yacht_hydrodynamics)

  cv.error[i]=cv.glm (yacht_hydrodynamics ,glm.fit,K=5)$delta [1]

}

cv.error
```

```
plot(cv.error,type="o",col="black")

#K=10

cv.error=rep (0,6)

for (i in 1:6){

  glm.fit=glm(V7~poly(V6 ,i),data=yacht_hydrodynamics)

  cv.error[i]=cv.glm (yacht_hydrodynamics ,glm.fit,K=10)$delta [1]

}

cv.error

plot(cv.error,type="o",col="black")
```

Plots for k = 1, 2, 5 and 10 from left to right & top to bottom



for K = 1

cv.error was

80.144679 17.575935 3.917786 2.665399 2.689602

From the above observations, we see that degree 4 polynomial model has the least value. Hence degree 4 model is the model that predicts best.

for K = 2

cv.error was

79.602674 17.406799 3.872209 2.533129 3.067306

From the above observations, we see that degree 4 polynomial model has the least value. Hence degree 4 model is the model that predicts best.

for K = 5

cv.error was

79.558299 17.461399 3.863968 2.574458 2.642308

From the above observations, we see that degree 4 polynomial model has the least value. Hence degree 4 model is the model that predicts best.

for K = 10

cv.error was

81.456069 17.632990 3.969703 2.604383 2.711004

From the above observations, we see that degree 4 polynomial model has the least value. Hence degree 4 model is the model that predicts best.

Banknote Authentication:

#Multi-linear model

```
glm.fit=glm(class~ variance+skewness,data = bankData, family = binomial)

coef(glm.fit)

(Intercept)    variance    skewness    curtosis     entropy

   7.321805   -7.859330   -4.190963   -5.287431   -0.605319

cv.error=cv.glm(bankData,glm.fit)
```

[1] 0.08524051

```
glm.fitML1=glm(class~ skewness+curtosis,data = bankData, family = binomial)
coef(glm.fitML1)

(Intercept)    skewness    curtosis
  0.8777085  -0.4166583  -0.3553684

cv.errorML1=cv.glm(bankData,glm.fitML1)
cv.errorML1$delta[1]
```

[1] 0.166769

```
glm.fitML2=glm(class~ curtosis+entropy,data = bankData, family = binomial)
coef(glm.fitML2)

(Intercept)    curtosis     entropy
-0.44195030  0.08730419 -0.08020583

cv.errorML2=cv.glm(bankData,glm.fitML2)
cv.errorML2$delta[1]
```

[1] 0.240565

The Multi-linear model was built with two predictors. More than two predictors throws warning s.

The Cross-Validation above shows that the linear model with variance and skewness has the lea st MSE

#Linear Models - LOOCV (For all 4 predictors – variance, skewness, curtosis, entropy)

The table below highlights in bold all the least MSE's built using the linear model using each pre dictor. The i- value represents the polynomial degree.

```
for (i in 1:5) {
  glm.fitLM=glm(class~ poly(variance,i),data = bankData, family = binomial)
  cv.errorLM1[i]=cv.glm(bankData,glm.fitLM)$delta[1]
}
```

	Variance	**Skewness**	**Curtosis**	**Entropy**
i=1	0.1091576	0.2023483	0.2415724	0.2475175
i=2	0.1080863	0.2004354	0.2299115	0.2474628
i=3	0.1078766	0.1777716	0.2272558	**0.2472388**
i=4	0.1079556	**0.1719493**	0.2261582	0.2474882
i=5	**0.1078390**	0.1719637	**0.2249622**	0.2473234

#Linear Models – k- Fold CV (For all 4 predictors – variance, skewness, curtosis, entropy)

K = 10

```
set.seed(17)
cv.errorKF=rep(0,10)
for (i in 1:10) {
  glm.fitKF=glm(class~ poly(variance,i),data = bankData, family = binomial)
  cv.errorKF[i]=cv.glm(bankData,glm.fitKF, K = 10)$delta[1]
}
```

	Variance	Skewness	Curtosis	Entropy
i=1	0.1094151	0.2033552	0.2424584	0.2478877
i=2	0.1082462	0.2010359	0.2301815	0.2475096
i=3	**0.1078833**	0.1780798	0.2273132	0.2473215
i=4	0.1079143	0.1719947	0.2263181	0.2473750
i=5	0.1085608	0.1723291	0.2257085	0.2474401
i=6	0.1752029	0.1721942	0.2117189	**0.2434012**
i=7	0.1533065	0.1728757	0.2104414	0.2444057
i=8	0.1088239	0.1716548	**0.2087226**	0.2443718
i=9	0.1531503	**0.1630493**	0.3014444	0.2448072
i=10	0.2252187	0.3456002	0.3752178	0.2443845

#Concrete Strength

#Linear Models – LOOCV

```
cv.errorLM=rep(0,5)

for (i in 1:5) {

  glm.fit=glm(Concrete~ poly(Age,i), data = concreteData)

  cv.errorLM[i]=cv.glm(concreteData,glm.fit)$delta[1]

}
```

	Cement	**Blast.Furnace**	**Fly.Ash**	**Water**	**Superplasticizer**	**Coarse.Aggregate**	**Fine.Aggregate**	**Age**
i=1	210.4948	274.8630	276.7394	256.5072	242.3751	272.3534	272.0300	249.8119
i=2	210.8348	269.2085	274.3826	242.3496	242.6733	270.1058	271.5274	208.3234
i=3	211.2108	267.3118	274.7494	**231.1924**	242.9949	268.0660	**271.2917**	191.4539
i=4	211.3342	265.1630	**273.0629**	226.6225	**243.1103**	267.9624	271.8399	**183.1507**
i=5	**209.9512**	**263.9871**	273.3415	226.8308	243.5011	**267.1689**	271.4130	183.3228

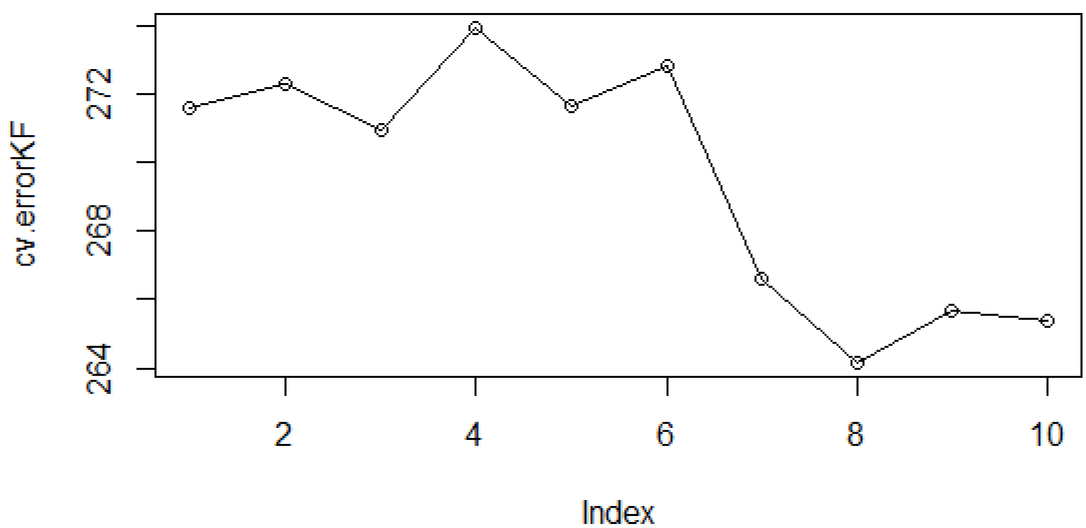
#Linear Models – k- Fold CV

```
set.seed(17)

cv.errorKF=rep(0,10)

for (i in 1:10) {

  glm.fitKF=glm(Concrete~ poly(Fine.Aggregate,i),data = concreteData)

  cv.errorKF[i]=cv.glm(concreteData,glm.fitKF, K = 10)$delta[1]

}
```

K = 10

	Cement	Blast.Furnace	Fly.Ash	Water	Superplasticizer	Coarse.Aggregate	Fine.Aggregate	Age
i=1	210.4637	274.1115	276.1530	255.9863	241.9922	271.8806	271.6151	249.2649
i=2	211.3369	269.4083	274.4802	242.5515	242.8661	270.3379	272.3241	208.6059
i=3	211.3939	267.5871	275.1446	231.7747	243.2920	267.9365	270.9642	191.3563
i=4	211.5884	264.7738	273.6595	226.9832	243.4881	268.0202	273.9573	183.1211
i=5	210.4344	264.9009	273.8430	227.7811	243.5021	267.0089	271.6566	182.9739

i=6	**209.4633**	264.5840	273.9942	224.4368	243.3100	267.2866	272.8459	182.6817
i=7	210.2615	264.5795	273.8260	**223.2994**	242.0661	267.7208	266.6162	182.5806
i=8	210.5778	300.3465	274.5900	228.1008	242.3464	270.6964	**264.1559**	182.2823
i=9	209.9121	**263.2280**	**272.8926**	610.4748	**241.6163**	**259.9246**	265.6554	180.8511
i=10	209.6636	263.4242	273.7718	224.2760	244.3621	261.8349	265.4092	**180.4399**



Plot for predictor Fine Aggregate. The rest can be interpreted from the above table

#Code used in the last two problems

```
bankData=read.csv("C:\\Users\\Sushmitha\\Documents\\Third Semester\\Comp Mthds Data Science\\Assignment V\\Assign_5-Data\\data_banknote_authentication.csv",",",header=TRUE)

head(bankData)

attach(bankData)

library(boot)

#Logistic Regression

names(bankData)

class=factor(class)

is.factor(class)
```

#Multi-linear model

```
glm.fit=glm(class~ variance+skewness,data = bankData, family = binomial)

coef(glm.fit)

cv.error=cv.glm(bankData,glm.fit)

cv.error$delta[1]

glm.fitML1=glm(class~ skewness+curtosis,data = bankData, family = binomial)

coef(glm.fitML1)

cv.errorML1=cv.glm(bankData,glm.fitML1)

cv.errorML1$delta[1]
```

```
glm.fitML2=glm(class~ curtosis+entropy,data = bankData, family = binomial)

coef(glm.fitML2)

cv.errorML2=cv.glm(bankData,glm.fitML2)

cv.errorML2$delta[1]
```

#Linear Model

```
glm.fitLM1=glm(class~ variance,data = bankData, family = binomial)

coef(glm.fitLM1)

cv.errorLM1=cv.glm(bankData,glm.fitLM1)

cv.errorLM1$delta[1]

cv.errorLM=rep(0,5)

for (i in 1:5) {

  glm.fitLM=glm(class~ poly(entropy,i),data = bankData, family = binomial)

  cv.errorLM[i]=cv.glm(bankData,glm.fitLM)$delta[1]

}

  cv.errorLM
```

#k-Fold CV

```
set.seed(17)

cv.errorKF=rep(0,10)

for (i in 1:10) {

  glm.fitKF=glm(class~ poly(variance,i),data = bankData, family = binomial)
```

```
  cv.errorKF[i]=cv.glm(bankData,glm.fitKF, K = 10)$delta[1]
}
cv.errorKF
```

#Concrete Strength

```
concreteData=read.csv("C:\\Users\\Sushmitha\\Documents\\Third Semester\\Comp Mthds Data Science\\Assignment V\\Assign_5-Data\\Concrete_Data.csv",",",header=TRUE)
head(concreteData)
attach(concreteData)
names(concreteData)
glm.fit=glm(Concrete~ Cement,data = concreteData)
cv.errorLM=rep(0,5)
for (i in 1:5) {
  glm.fit=glm(Concrete~ poly(Age,i), data = concreteData)
  cv.errorLM[i]=cv.glm(concreteData,glm.fit)$delta[1]
}
cv.errorLM

set.seed(17)
cv.errorKF=rep(0,10)
for (i in 1:10) {
```

```
  glm.fitKF=glm(Concrete~ poly(Fine.Aggregate,i),data = concreteData)

  cv.errorKF[i]=cv.glm(concreteData,glm.fitKF, K = 10)$delta[1]

}

cv.errorKF
```