V2A3_regression_airfoilnoise

December 17, 2018

In [3]:
```
#!/usr/bin/env python
# V2A3_regression_airfoilnoise.py
# Programmgeruest zu Versuch 2, Aufgabe 3
# to log outputs start with: python V2A3_regression_airfoilnoise.py >V2A3_regression_a

import numpy as np
import pandas as pd

from V2A2_Regression import *


# ***** MAIN PROGRAM ********
# (I) Hyper-Parameters
S=3;                # S-fold cross-validation
lmbda=1;            # regularization parameter (lambda>0 avoids also singularities)
K=13;                # K for K-Nearest Neighbors
flagKLinReg = 1;    # if flag==1 and K>=D then do a linear regression of the KNNs to ma
deg=5;              # degree of basis function polynomials
flagSTD=1;          # if >0 then standardize data before training (i.e., scale X to mea
N_pred=5;           # number of predictions on the training set for testing
x_test_1 = [1250,11,0.2,69.2,0.0051];   # REPLACE dummy code: define test vector 1
x_test_2 = [1305,8,0.1,57.7,0.0048];   # REPLACE dummy code: define test vector 2
```

In [4]:
```
# (II) Load data
fname='./AirfoilSelfNoise/airfoil_self_noise.xls'
airfoil_data = pd.read_excel(fname,0); # load data as pandas data frame
T = airfoil_data.values[:,5]           # target values = noise load (= column 5 of dat
X = airfoil_data.values[:,:5]          # feature vectors (= column 0-4 of data table)
N,D=X.shape                            # size and dimensionality of data set
idx_perm = np.random.permutation(N)    # get random permutation for selection of test
print("Data set ",fname," has size N=", N, " and dimensionality D=",D)
print("X=",X)
print("T=",T)
print("x_test_1=",x_test_1)
print("x_test_2=",x_test_2)
print("number of basis functions M=", len(phi_polynomial(X[1],deg)))
```

```
Data set  ./AirfoilSelfNoise/airfoil_self_noise.xls  has size N= 1502  and dimensionality D= 5
```

```
X= [[1.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 [1.25000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 [1.60000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 ...
 [4.00000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]
 [5.00000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]
 [6.30000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]]
T= [125.201 125.951 127.591 ... 106.604 106.224 104.204]
x_test_1= [1250, 11, 0.2, 69.2, 0.0051]
x_test_2= [1305, 8, 0.1, 57.7, 0.0048]
number of basis functions M= 252
```

```
In [5]: # (III) Do least-squares regression with regularization
        print("\n#### Least Squares Regression with regularization lambda=", lmbda, " ####")
        lsr = LSRRegressifier(lmbda=lmbda,phi=lambda x: phi_polynomial(x,deg),flagSTD=flagSTD)
        lsr.fit(X,T)
        print("lsr.W_LSR=",lsr.W_LSR)
        print("III.1) Some predictions on the training data:")
        for i in range(N_pred):
            n=idx_perm[i]
            print("Prediction for X[",n,"]=",X[n]," is y=",lsr.predict(X[n]),", whereas true va
        print("III.2) Some predicitions for new test vectors:")
        print("Prediction for x_test_1 is y=", lsr.predict(x_test_1))    # REPLACE dummy code:
        print("Prediction for x_test_2 is y=", lsr.predict(x_test_2))    # REPLACE dummy code:
        print("III.3) S=",S,"fold Cross Validation:")
        err_abs,err_rel = lsr.crossvalidate(S,X,T)                  # REPLACE dummy code: do c
        print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min,max)=",
```

```
#### Least Squares Regression with regularization lambda= 1  ####
lsr.W_LSR= [-3.94646520e-01 -1.80155527e+00 -3.96662974e-01 -8.64281233e-01
  3.26438812e-01 -6.21770966e-01  6.48905815e-01 -7.71134930e-01
  2.95779376e-01 -8.67132337e-02  3.61560493e-01 -2.81989365e-02
 -3.64074945e-02  1.43998983e-01  4.77503076e-02 -3.63380016e-02
 -9.73944497e-02 -3.71505203e-02 -1.53013579e-01 -2.47759041e-01
  1.29242314e-01  3.13172340e-01  1.50772922e-01  8.40072758e-01
 -8.44083280e-02  1.32532606e+00  2.52907215e-01  2.08504016e-01
 -2.03801432e-01  5.74149325e-01  3.06551563e-01 -2.34531355e-01
  4.66271349e-01 -2.43145107e-01  1.06213081e-02  2.84851888e-01
 -8.48806231e-02 -2.00727798e-01 -6.93365225e-02 -5.88853578e-02
  7.29728355e-02 -1.36528243e-01 -2.26202702e-01 -1.28035094e-01
 -2.24863325e-01 -1.13338956e-01  4.19411521e-01 -1.68848150e-02
  9.78093460e-02 -9.27714500e-02 -1.60258656e-02 -3.22486764e-01
  1.01134491e-01  7.97108235e-03 -1.81313744e-01 -3.69616798e-01
 -2.47146762e-01  2.02432625e-01 -4.38061153e-01  3.64617492e-02
 -7.59308984e-01  1.95668256e-01  6.70564131e-01 -1.47283279e-01
 -2.77532541e-01 -3.08022282e-01  1.19842567e-02 -7.66976903e-02
```

```
 2.18511621e-02  1.72857687e-01 -6.25981195e-01  1.36043453e-01
 6.68313146e-02 -1.65130385e-02 -5.46617990e-02  5.48724195e-02
-5.19208304e-02 -7.63188408e-02  3.78906795e-02  1.94056398e-01
-2.78668584e-01 -7.05574660e-01  8.01753250e-05 -1.97576222e-01
 5.53327040e-02 -2.03772994e-02 -1.47505902e-01  5.91246025e-02
-1.79737621e-02 -7.12236378e-02 -2.34113997e-01 -4.01567558e-02
 2.90612583e-02 -7.81051692e-02  2.06254670e-05  5.56183281e-02
 4.80864131e-03  2.42168243e-02 -1.15039749e-01  1.75335472e-02
 2.94441955e-02 -2.98387467e-01  2.24608964e-03 -5.67996318e-02
 3.60431186e-03  3.50473096e-02 -9.11562735e-02 -2.58145605e-02
 3.76595743e-02  1.50183200e-01 -3.75077570e-02  1.27138401e-02
 1.16133919e-01 -2.18208523e-01  1.41690489e-01  1.29547055e-01
 1.02847461e-01  7.46744531e-03 -5.50688491e-02 -1.09489486e-01
 4.33124764e-02  3.52862834e-02  1.02885915e-01  1.58937930e-01
 3.22368270e-02  1.33278437e-01  1.54200823e-02  1.03812366e-02
 2.68478094e-02 -4.69455614e-03 -1.03924197e-01  3.63370895e-01
 1.95941884e-01  8.55311399e-03 -6.75202423e-01  6.17886875e-02
-5.57216536e-03 -6.85068836e-01  9.73106644e-04  8.93303864e-04
 3.32687011e-01  1.03800671e-01  2.21151780e-01 -1.24168329e-01
 5.08255859e-02 -6.41651342e-02 -3.57687689e-02  1.84616236e-01
-5.38713351e-02  2.16353496e-01 -4.49026953e-02  5.07394826e-02
-1.58277599e-03 -3.52975947e-02 -2.32615544e-02  1.11961868e-01
-3.17638241e-01  3.66642352e-03  1.01689005e-01 -9.14031375e-02
 1.07835193e-01 -5.52562467e-02 -1.69661381e-01  1.65517660e-02
-1.62822814e-01 -3.68756103e-01  8.04196574e-02 -8.18975854e-02
-1.12357085e-01  3.69551139e-02 -5.07037615e-02 -1.69293973e-01
 1.78704521e-01  2.25341988e-01 -8.85750241e-02  5.14764795e-02
 1.97391087e-01  5.44295165e-02  2.08205031e-01  8.63666478e-03
 2.50397522e-01  2.12918833e-01  3.01972410e-02 -4.05213862e-03
-1.89041169e-02 -1.59662947e-01  2.21658174e-01  2.82466967e-02
 8.76627247e-02  4.79010683e-02  9.35769244e-02  9.70424870e-02
-2.11928518e-02 -2.45208682e-02 -2.29960413e-02 -1.97186002e-02
 1.40300710e-01  2.80593592e-01  1.01183417e-01  7.86300171e-03
-1.36568507e-01  1.93395520e-01  1.49290958e-02  9.97525775e-02
-5.93850011e-02 -3.88941947e-02 -2.24188695e-01  1.14703293e-01
-8.61448667e-02 -3.94278300e-02 -3.86775078e-02 -1.38123752e-01
 3.28920405e-02 -1.46146334e-01 -8.05652042e-02 -9.70461716e-02
 2.15697923e-01  3.68525314e-02  2.31497755e-01 -2.69752137e-02
 1.55073486e-02  1.84839607e-01  8.53004635e-03  5.65295263e-02
-2.07214477e-01 -2.92021485e-02  4.13387816e-03  3.20040301e-02
-6.48052853e-02 -9.35343805e-02 -1.63259658e-02 -1.15971880e-01
-5.33499384e-02 -1.74711442e-01 -7.62523429e-02 -3.15365640e-03
-3.13599999e-02  1.47577802e-02  8.98079738e-02  5.76990605e-02
 3.07022562e-02  6.10550828e-02  4.30451202e-02  2.50332155e-02
-1.23965738e-01 -1.06664213e-01 -3.90944224e-02  3.18752489e-02
-5.83388127e-02  6.63287979e-03  8.53339681e-02  2.11106178e-02]
III.1) Some predictions on the training data:
Prediction for X[ 642 ]= [5.00000e+03 9.90000e+00 1.52400e-01 7.13000e+01 1.93001e-02]  is y= 1
```

```
Prediction for X[ 6 ]= [4.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]  is y= 123
Prediction for X[ 1359 ]= [4.00000e+02 6.70000e+00 1.01600e-01 3.96000e+01 5.78076e-03]  is y=
Prediction for X[ 990 ]= [1.00000e+04 0.00000e+00 2.54000e-02 3.96000e+01 4.28464e-04]  is y= 1
Prediction for X[ 953 ]= [8.0000e+02 1.9700e+01 5.0800e-02 3.9600e+01 3.6484e-02]  is y= 123.99
III.2) Some predicitions for new test vectors:
Prediction for x_test_1 is y= 130.45406757371268
Prediction for x_test_2 is y= 133.1060885540002
III.3) S= 3 fold Cross Validation:
absolute errors (E,sd,min,max)= (2.1022709165494167, 2.63711268032694, 0.000984334197639214, 40
relative errors (E,sd,min,max)= (0.016946838227214888, 0.021770475968140576, 8.455680284846053e
```

```
In [ ]: lmbdaRange = [1,2,5,10,30,60,100]
        degRange = list(range(1,8))
        LSRErrors = []

        for lmb in lmbdaRange:
            for d in degRange:
                # (III) Do least-squares regression with regularization
                print("\n#### Least Squares Regression with regularization lambda=", lmb, " deg
                lsr = LSRRegressifier(lmbda=lmb,phi=lambda x: phi_polynomial(x,d),flagSTD=flagS
                lsr.fit(X,T)
                print("lsr.W_LSR=",lsr.W_LSR)    # REPLACE dummy code: print weight vector for
                #print("III.1) Some predictions on the training data:")
                #for i in range(N_pred):
                    #n=idx_perm[i]
                    #print("Prediction for X[",n,"]=",X[n]," is y=",lsr.predict(X[n]),", where
                #print("III.2) Some predicitions for new test vectors:")
                #print("Prediction for x_test_1 is y=", lsr.predict(x_test_1))    # REPLACE dum
                #print("Prediction for x_test_2 is y=", lsr.predict(x_test_2))    # REPLACE dum
                print("III.3) S=",S,"fold Cross Validation:")
                err_abs,err_rel = lsr.crossvalidate(S,X,T)                  # REPLACE dummy co
                LSRErrors.append((lmb, d, err_abs, err_rel))
                print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min
```

```
In [7]: # (IV) Do KNN regression
        print("\n#### KNN regression with flagKLinReg=", flagKLinReg, " ####")
        knnr = KNNRegressifier(K,flagKLinReg)                                  # REPLACE dummy
        knnr.fit(X,T)
        print("IV.1) Some predictions on the training data:")
        for i in range(N_pred):
            n=idx_perm[i]
            print("Prediction for X[",n,"]=",X[n]," is y=",knnr.predict(X[n]),", whereas true v
        print("IV.2) Some predicitions for new test vectors:")
        print("Prediction for x_test_1 is y=", knnr.predict(x_test_1))    # REPLACE dummy code
        print("Prediction for x_test_2 is y=", knnr.predict(x_test_2))    # REPLACE dummy code
        print("IV.3) S=",S,"fold Cross Validation:")
        err_abs,err_rel = knnr.crossvalidate(S,X,T)                  # REPLACE dummy code: do
        print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min,max)=",
```

```
#### KNN regression with flagKLinReg= 1  ####
IV.1) Some predictions on the training data:
Prediction for X[ 642 ]= [5.00000e+03 9.90000e+00 1.52400e-01 7.13000e+01 1.93001e-02]  is y= 
Prediction for X[ 6 ]= [4.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]  is y= 12
Prediction for X[ 1359 ]= [4.00000e+02 6.70000e+00 1.01600e-01 3.96000e+01 5.78076e-03]  is y=
Prediction for X[ 990 ]= [1.00000e+04 0.00000e+00 2.54000e-02 3.96000e+01 4.28464e-04]  is y= 
Prediction for X[ 953 ]= [8.0000e+02 1.9700e+01 5.0800e-02 3.9600e+01 3.6484e-02]  is y= 127.19
IV.2) Some predicitions for new test vectors:
Prediction for x_test_1 is y= 126.56192024990972
Prediction for x_test_2 is y= 132.35085577388358
IV.3) S= 3 fold Cross Validation:
absolute errors (E,sd,min,max)= (3.1014230510756793, 3.1881851528245053, 0.0020327675492239905
relative errors (E,sd,min,max)= (0.02490752287142648, 0.02601855286718329, 1.5901618890310798e-
```

```
In [ ]: flagRange = [0,1]
        kRange = list(range(1,15))
        KNNErrors = []

        for f in flagRange:
            for k in kRange:
                # (IV) Do KNN regression
                print("\n#### KNN regression with flagKLinReg=", f," K=",k, " ####")
                knnr = KNNRegressifier(k,f)                                    # REPLACE dummy 
                knnr.fit(X,T)
                #print("IV.1) Some predictions on the training data:")
                #for i in range(N_pred):
                 #   n=idx_perm[i]
                 #   print("Prediction for X[",n,"]=",X[n]," is y=",knnr.predict(X[n]),", wher
                #print("IV.2) Some predicitions for new test vectors:")
                #print("Prediction for x_test_1 is y=", knnr.predict(x_test_1))    # REPLACE d
                #print("Prediction for x_test_2 is y=", knnr.predict(x_test_2))    # REPLACE d
                print("IV.3) S=",S,"fold Cross Validation:")
                err_abs,err_rel = knnr.crossvalidate(S,X,T)                    # REPLACE dummy 
                KNNErrors.append((f, k, err_abs, err_rel))
                print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min
```

a) Vervollständigen Sie das Programmgerüst V2A3_regression_airfoilnoise.py um eine Least-Squares-Regression auf den Daten zu berechnen. Optimieren Sie die HyperParameter um bei einer S = 3-fachen Kreuzvalidierung möglichst kleine Fehlerwerte zu erhalten.

Welche Bedeutung haben jeweils die Hyper-Parameter lmbda, deg, flagSTD?

- lmbda: Regularisierungs parameter
- deg: Ist der Grad er polynomiellen Basisfunktion
- flagSTD: gibt an ob die Daten standartisiert werden sollen

Was passiert ohne Skalierung der Daten (flagSTD=0) bei höheren Polynomgraden (achten Sie auf die Werte von maxZ)?

- EXCEPTION DUE TO BAD CONDITION:flagOK= 0 maxZ= 195590361182.93994 eps= 1e-06

- der Fehlerwert beim invertieren explodiert ohne die Skalierung

Geben Sie Ihre optimalen Hyper-Parameter sowie die resultierenden Fehler-Werte an.

- lambda=1, deg=5, flagSTD=1
- absolute errors (E,sd,min,max)= (2.2241976966993438, 3.6504292455503355, 0.0023069096170331704, 83.63604473632992)
- relative errors (E,sd,min,max)= (0.018030635646137383, 0.03132168189811296, 1.746150761488692e-05, 0.762838110293237)

Welche Prognosen ergibt Ihr Modell für die neuen Datenvektoren x_test_1=[1250,11,0.2,69.2,0.0051] bzw. x_test_2=[1305,8,0.1,57.7,0.0048] ?

- Prediction for x_test_1 is y= 130.45406757371268
- Prediction for x_test_2 is y= 133.1060885540002

Welchen Polynomgrad und wieviele Basisfunktionen verwendet Ihr Modell?

- Polynomgrad ist 5
- mit 6 Basisfunktionen

b) Vervollständigen Sie das Programmgerüst V2A3_regression_airfoilnoise.py um eine KNN-Regression auf den Daten zu berechnen. Optimieren Sie die Hyper-Parameter um bei einer S = 3-fachen Kreuzvalidierung möglichst kleine Fehlerwerte zu erhalten.
Welche Bedeutung haben jeweils die Hyper-Parameter K und flagKLinReg?

- K: anzahl der NN die zum Ergebniss beitragen
- flagKLinReg: gibt an ob die KNN noch in einen LSRegressifier gegeben werden oder ob nur der Durchschnitt der KNN berechnet wird.

Geben Sie Ihre optimalen Hyper-Parameter sowie die resultierenden Fehler-Werte an.

- flagKLinReg=1, K=13
- absolute errors (E,sd,min,max)= (3.0858901947930324, 3.0296772825440974, 0.006016325705218151, 32.13059666400561)
- relative errors (E,sd,min,max)= (0.02482629194750422, 0.024838560765987942, 4.864074982591945e-05, 0.2851617187841634)

Welche Prognosen ergibt Ihr Modell für die neuen Datenvektoren x_test_1=[1250,11,0.2,69.2,0.0051] bzw. x_test_2=[1305,8,0.1,57.7,0.0048] ?

- Prediction for x_test_1 is y= 126.56192024990972
- Prediction for x_test_2 is y= 132.35085577388358

c) Vergleichen Sie die beiden Modelle. Welches liefert die besseren Ergebnisse?
LSR: absolute errors (E,sd,min,max)= (2.2241976966993438, 3.6504292455503355, 0.0023069096170331704, 83.63604473632992)
KNN: absolute errors (E,sd,min,max)= (3.0858901947930324, 3.0296772825440974,

0.006016325705218151, 32.13059666400561)
absolute differences: (0.861692498, 0.620751963, 0.003709416, 51.505448072)
LSR: relative errors (E,sd,min,max)= (0.018030635646137383, 0.03132168189811296, 1.746150761488692e-05, 0.762838110293237)
KNN: relative errors (E,sd,min,max)= (0.02482629194750422, 0.024838560765987942, 4.864074982591945e-05, 0.2851617187841634) relative differences: (0.006795656, 0.006483121, 0.000031179, 0.477676392

- KNN scheint die bessereren Ergebnisse zu liefern, verwendet intern aber auch LSR