V2A2_Regression

December 14, 2018

```
In [1]: #!/usr/bin/env python
        # V2A2_Regression.py
        # Programmgeruest zu Versuch 2, Aufgabe 2

        import numpy as np
        import scipy.spatial
        from random import randint

        # ------------------------------------------------------------------------------------------
        # base class for regressifiers
        # ------------------------------------------------------------------------------------------
        class Regressifier:
            """
            Abstract base class for regressifiers
            Inherit from this class to implement a concrete regression algorithm
            """

            def fit(self,X,T):        # train/compute regression with lists of feature vectors
                """
                Train regressifier by training data X, T, should be overwritten by any derived
                :param X: Data matrix of size NxD, contains in each row a data vector of size
                :param T: Target vector matrix of size NxK, contains in each row a target vect
                :returns: -
                """
                pass

            def predict(self,x):      # predict a target vector given the data vector x
                """
                Implementation of the regression algorithm; should be overwritten by any deriv
                :param x: test data vector of size D
                :returns: predicted target vector
                """
                return None

            def crossvalidate(self,S,X,T,dist=lambda t: np.linalg.norm(t)):  # do a S-fold cro
                """
                Do a S-fold cross validation
```

```
:param S: Number of parts the data set is divided into
:param X: Data matrix (one data vector per row)
:param T: Matrix of target vectors; T[n] is target vector of X[n]
:param dist: a fuction dist(t) returning the length of vector t (default=Eukli
:returns (E_dist,sd_dist,E_min,E_max) : mean, standard deviation, minimum, and
:returns (Erel_dist,sdrel_dist,Erel_min,Erel_max) : mean, standard deviation, 
"""
X,T=np.array(X),np.array(T)                         # ensure array type
N=len(X)                                            # N=number of data vectors
perm = np.random.permutation(N)                     # do a random permutation 
X1,T1=[X[i] for i in perm], [T[i] for i in perm]    # ... to get random partit
idxS = [range(i*N//S,(i+1)*N//S) for i in range(S)] # divide data set into S p
E_dist,E_dist2,E_max,E_min=0,0,-1,-1                # initialize first two mom
Erel_dist,Erel_dist2,Erel_max,Erel_min=0,0,-1,-1    # initialize first two mom
for idxTest in idxS:                                # loop over all possible t
    # (i) generate training and testing data sets and train classifier
    idxLearn = [i for i in range(N) if i not in idxTest]                      #
    if(S<=1): idxLearn=idxTest                                                 #
    X_learn, T_learn = np.array([X1[i] for i in idxLearn]), np.array([T1[i] fo
    X_test , T_test  = np.array([X1[i] for i in idxTest ]), np.array([T1[i] fo
    self.fit(X_learn,T_learn)                       # train regressifier
    # (ii) test regressifier
    for i in range(len(X_test)):  # loop over all data vectors to be tested
        # (ii.a) regress for i-th test vector
        xn_test = X_test[i].T                               # data vector for test
        t_test = self.predict(xn_test)                      # predict target value
        # (ii.b) check for regression errors
        t_true = T_test[i].T                                # true target value
        d=dist(t_test-t_true)                               # (Euklidean) distance
        dttrue=dist(t_true)                                 # length of t_true
        E_dist  = E_dist+d                                  # sum up distances (fo
        E_dist2 = E_dist2+d*d                               # sum up squared dista
        if(E_max<0)or(d>E_max): E_max=d                     # collect maximal erro
        if(E_min<0)or(d<E_min): E_min=d                     # collect minimal erro
        drel=d/dttrue
        Erel_dist   = Erel_dist+drel                        # sum up relative dist
        Erel_dist2 = Erel_dist2+(drel*drel)                 # sum up squared relat
        if(Erel_max<0)or(drel>Erel_max): Erel_max=drel      # collect maximal rela
        if(Erel_min<0)or(drel<Erel_min): Erel_min=drel      # collect minimal rela
E_dist      = E_dist/float(N)                               # estimate of first mo
E_dist2     = E_dist2/float(N)                              # estimate of second m
Var_dist    = E_dist2-E_dist*E_dist                         # variance of error
sd_dist     = np.sqrt(Var_dist)                             # standard deviation o
Erel_dist   = Erel_dist/float(N)                            # estimate of first mo
Erel_dist2  = Erel_dist2/float(N)                           # estimate of second m
Varrel_dist = Erel_dist2-Erel_dist*Erel_dist                # variance of error
sdrel_dist  = np.sqrt(Varrel_dist)                          # standard deviation o
return (E_dist,sd_dist,E_min,E_max), (Erel_dist,sdrel_dist,Erel_min,Erel_max) #
```

```
                                                                  # and maximum error (f
In [2]: # ------------------------------------------------------------------------------------
        # DataScaler: scale data to standardize data distribution (for mean=0, standard deviat
        # ------------------------------------------------------------------------------------
        class DataScaler:
            """
            Class for standardizing data vectors
            Some regression methods require standardizing of data before training to avoid num
            """

            def __init__(self,X):                # X is data matrix, where rows are data vector
                """
                Constructor: Set parameters (mean, std,...) to standardize data matrix X
                :param X: Data matrix of size NxD the standardization parameters (mean, std, .
                :returns: object of class DataScaler
                """
                self.meanX = np.mean(X,0)        # mean values for each feature column
                self.stdX  = np.std(X,0)         # standard deviation for each feature column
                if isinstance(self.stdX,(list,tuple,np.ndarray)):
                    self.stdX[self.stdX==0]=1.0 # do not scale data with zero std (that is, co
                else:
                    if(self.stdX==0): self.stdX=1.0   # in case stdX is a scalar
                self.stdXinv = 1.0/self.stdX     # inverse standard deviation


            def scale(self,x):                   # scales data vector x to mean=0 and std=1
                """
                scale data vector (or data matrix) x to mean=0 and s.d.=1
                :param x: data vector or data matrix
                :returns: scaled (standardized) data vector or data matrix
                """
                return np.multiply(x-self.meanX,self.stdXinv)

            def unscale(self,x):                 # unscale data vector x to original distributi
                """
                unscale data vector (or data matrix) x to original data ranges
                :param x: standardized data vector or data matrix
                :returns: unscaled data vector or data matrix
                """
                return np.multiply(x,self.stdX)+self.meanX

            def printState(self):
                """
                print standardization parameters (mean value, standard deviation (std), and in
                """
                print("mean=",self.meanX, " std=",self.stdX, " std_inv=",self.stdXinv)

In [3]: from itertools import *
```

```
        from functools import reduce
        # ------------------------------------------------------------------------------------
        # function to compute polynomial basis functions
        # ------------------------------------------------------------------------------------
        def phi_polynomial(x,deg=1):           # x should be list or np.array or 1xD matrix; r
            """
            polynomial basis function vector; may be used to transform a data vector x into a
            :param x: data vector to be transformed into a feature vector
            :param deg: degree of polynomial
            :returns phi: feature vector
            Example: phi_polynomial(x,3) returns for one-dimensional x the vector [1, x, x*x,
            """
            x=np.array(np.mat(x))[0]           # ensure that x is a 1D array (first row of x)
            D=len(x)
            #assert (D==1) or ((D>1) and (deg<=3)), "phi_polynomial(x,deg) not implemented for
            if(D==1):
                phi = np.array([x[0]**i for i in range(deg+1)])
            else:
                phi = np.array(phi_helper(x.tolist(),deg))
                #phi = np.array([])
                #if(deg>=0):
                #    phi = np.concatenate((phi,[1]))      # include degree 0 terms
                #    if(deg>=1):
                #        phi = np.concatenate((phi,x))    # includes degree 1 terms
                #        if(deg>=2):
                #            for i in range(D):
                #                phi = np.concatenate(( phi, [x[i]*x[j] for j in range(i+1)] ))
                #            if(deg>=3):
                #                for i in range(D):
                #                    for j in range(i+1):
                #                        phi = np.concatenate(( phi, [x[i]*x[j]*x[k] for k in
                                # EXTEND CODE HERE FOR deg>3!!!!
            return phi.T  # return basis function vector (=feature vector corresponding to dat

        def phi_helper(x, deg):
            if deg <= 0:
                return [1]
            else:
                return phi_helper(x, deg-1) + [reduce(lambda a,b:a*b,combi) for combi in combin

In [4]: # ------------------------------------------------------------------------------------
        # Least Squares (ML) linear regression with sum of squares Regularization,
        # ------------------------------------------------------------------------------------
        class LSRRegressifier(Regressifier):
            """
            Class for Least Squares (or Maximum Likelihood) Linear Regressifier with sum of sq
            """
```

```
def __init__(self,lmbda=0,phi=lambda x: phi_polynomial(x,1),flagSTD=0,eps=0.000001)
    """
    Constructor of class LSRegressifier
    :param lmbda: Regularization coefficient lambda
    :param phi: Basis-functions used by the linear model (default linear polynomia
    :param flagSTD: If >0 then standardize data X and target values T (to mean 0 a
    :param eps: maximal residual value to tolerate (instead of zero) for numerical
    :returns: -
    """
    self.lmbda=lmbda       # set regression parameter (default 0)
    self.phi=phi           # set basis functions used for linear regression (defau
    self.flagSTD=flagSTD;  # if flag >0 then data will be standardized, i.e., scal
    self.eps=eps;          # maximal residual value to tolerate (instead of zero)


def fit(self,X,T,lmbda=None,phi=None,flagSTD=None): # train/compute LS regression
    """
    Train regressifier (see lecture manuscript, theorem 3.11, p33)
    :param X: Data matrix of size NxD, contains in each row a data vector of size
    :param T: Target vector matrix of size NxK, contains in each row a target vect
    :param lmbda: Regularization coefficient lambda
    :param phi: Basis-functions used by the linear model (default linear polynomia
    :param flagSTD: If >0 then standardize data X and target values T (to mean 0 a
    :returns: flagOK: if >0 then all is ok, otherwise matrix inversion was bad cond
    """
    # (i) set parameters
    if lmbda==None: lmbda=self.lmbda       # reset regularization coefficient?
    if phi==None: phi=self.phi             # reset basis functions?
    if flagSTD==None: flagSTD=self.flagSTD # standardize data vectors?
    # (ii) scale data for mean=0 and s.d.=0 ?
    if flagSTD>0:                          # if yes, then...
        self.datascalerX=DataScaler(X)     # create datascaler for data matrix X
        self.datascalerT=DataScaler(T)     # create datascaler for target matrix T
        X=self.datascalerX.scale(X)        # scale all features (=columns) of data
        T=self.datascalerT.scale(T)        # ditto for target matrix T
    # (iii) compute weight matrix and check numerical condition
    flagOK,maxZ=1,0;                       # if <1 then matrix inversion is numeri
    try:
        self.N,self.D = X.shape            # data matrix X has size N x D (N is num
        self.M = self.phi(self.D*[0]).size # get number of basis functions
        #self.K = T.shape[1]                # DELTE dummy code (just required for
        PHI = np.array([np.transpose(phi(x)) for x in X])                        #
        PHIT_PHI_lmbdaI = np.add(np.dot(PHI.T,PHI),np.dot(self.lmbda,np.identity(se
        #print(PHIT_PHI_lmbdaI)
        # REPLACE dummy code: compute PHI_T*PHI+lambda*I
        PHIT_PHI_lmbdaI_inv = np.linalg.inv(PHIT_PHI_lmbdaI)         # REPLACE dum
        self.W_LSR = np.dot(np.dot(PHIT_PHI_lmbdaI_inv,PHI.T),T)# REPLACE dummy co
        # (iv) check numerical condition
```

```
            Z=np.dot(PHIT_PHI_lmbdaI,PHIT_PHI_lmbdaI_inv)-np.identity(self.M)  # REPLAC
            maxZ = np.max(Z)    # REPLACE dummy code: Compute maximum (absolute) compo
            assert maxZ<=self.eps              # maxZ should be <eps for good conditio
        except:
            flagOK=0;
            print("EXCEPTION DUE TO BAD CONDITION:flagOK=", flagOK, " maxZ=", maxZ, " e
            raise
        return flagOK

    def predict(self,x,flagSTD=None):      # predict a target value given data vector x
        """
        predicts the target value y(x) for a test vector x
        :param x: test data vector of size D
        :param flagSTD: If >0 then standardize data X and target values T (to mean 0 a
        :returns: predicted target vector y of size K
        """
        if flagSTD==None: flagSTD=self.flagSTD      # standardazion?
        if flagSTD>0: x=self.datascalerX.scale(x)   # if yes, then scale x before comp
        phi_of_x = self.phi(x)                      # compute feature vector phi_of_x f
        y=np.dot(self.W_LSR.T, phi_of_x)                    # REPLACE dummy code:  com
        if flagSTD>0: y=self.datascalerT.unscale(y) # scale prediction back to origina
        return y                  # return prediction y for data vector x
```

```
In [5]: # ----------------------------------------------------------------------------
        # KNN regression
        # ----------------------------------------------------------------------------
        class KNNRegressifier(Regressifier):
            """
            Class for fast K-Nearest-Neighbor-Regression using KD-trees
            """

            def __init__(self,K,flagKLinReg=0):
                """
                Constructor of class KNNRegressifier
                :param K: number of nearest neighbors that are used to compute prediction
                :flagKLinReg: if >0 then the do a linear (least squares) regression on the the
                              otherwise just take the mean of the K nearest neighbors target v
                :returns: -
                """
                self.K = K                              # K is number of nearest-neighbors
                self.X, self.T = [],[]                  # initially no data is stored
                self.flagKLinReg=flagKLinReg            # if flag is set then do a linear r

            def fit(self,X,T): # train/compute regression with lists of data vectors X and tar
                """
                Train regressifier by stroing X and T and by creating a KD-Tree based on X
                :param X: Data matrix of size NxD, contains in each row a data vector of size
                :param T: Target vector matrix of size NxK, contains in each row a target vect
                :returns: -
                """
                self.X, self.T = np.array(X),np.array(T)   # just store feature vectors X and 
                self.N, self.D = self.X.shape              # store data number N and dimension
                self.kdtree = scipy.spatial.KDTree(self.X) # do an indexing of the feature vec

            def predict(self,x,K=None,flagKLinReg=None):   # predict a target value given data
                """
                predicts the target value y(x) for a test vector x
                :param x: test data vector of size D
                :param K: number of nearest neighbors that are used to compute prediction
                :flagKLinReg: if >0 then the do a linear (least squares) regression on the the
                              otherwise just take the mean of the K nearest neighbors target v
                :returns: predicted target vector of size K
                """
                if(K==None): K=self.K                      # do a K-NN search...
                if(flagKLinReg==None): flagKLinReg=self.flagKLinReg # if flag >0 then do a reg
                nn = self.kdtree.query(x,K)                # get indexes of K nearest neighbor
                if K==1: idxNN=[nn[1]]                     # cast nearest neighbor indexes nn 
                else: idxNN=nn[1]
                t_out=0
                if(self.flagKLinReg==0):
                    # just take mean value of KNNs
                    t_out=np.mean([self.T[i] for i in idxNN])
                else:
                    # do a linear regression of the KNNs
                    lsr=LSRRegressifier(lmbda=0.0001,phi=lambda x:phi_polynomial(x,1),flagSTD=
                    lsr.fit(self.X[idxNN],self.T[idxNN])
                    t_out=lsr.predict(x)
                return t_out
```

In [17]:
```
# *******************************************************
# __main___
# Module test
# *******************************************************

if __name__ == '__main__':
    print("\n-----------------------------------------")
    print("Example: 1D-linear regression problem")
    print("-----------------------------------------")
    # (i) generate data
    N=100
    w0,w1=4,2                 # parameters of line
    X=np.zeros((N,1))         # x data: allocate Nx1 matrix as numpy ndarray
    X[:,0]=np.arange(0,50.0,50.0/N)  # equidistant sampling of the interval [0,50)
    T=np.zeros((N,1))         # target values: allocate Nx1 matrix as numpy ndarray
    sd_noise = 1.0            # noise power (=standard deviation)
    T=T+w1*X+w0 + np.random.normal(0,sd_noise,T.shape)  # generate noisy target value
```

```
            par_lambda = 0            # regularization parameter
            print("X=",X)
            print("T=",T)

            # (ii) define basis functions (phi should return list of basis functions; x shoul
            deg=2;                                  # degree of polynomial
            phi=lambda x: phi_polynomial(x,1)       # define phi by polynomial basis-functions u
            print("phi(4)=", phi([4]))               # print basis function vector [1, x, x*x ..
            print("phi([1,2])=", phi([1,2]))         # print basis function vector for two-dim.

            # (iii) compute LSR regression
            print("\n-----------------------------------------")
            print("Do a Least-Squares-Regression")
            print("-----------------------------------------")
            lmbda=0;
            lsr = LSRRegressifier(lmbda,phi)
            lsr.fit(X,T)
            print("lsr.W_LSR=",lsr.W_LSR)           # weight vector (should be approximately
            x=np.array([3.1415]).T
            print("prediction of x=",x,"is y=",lsr.predict(x))

            # do S-fold crossvalidation
            S=3
            err_abs,err_rel = lsr.crossvalidate(S,X,T)
            print("LSRRegression cross-validation: absolute errors (E,sd,min,max)=", err_abs,

            # (iv) compute KNN-regression
            print("\n-----------------------------------------")
            print("Do a KNN-Regression")
            print("-----------------------------------------")
            K=2;
            knnr = KNNRegressifier(K)
            knnr.fit(X,T)
            print("prediction of x=",x,"is y=",knnr.predict(x))

            # do S-fold crossvalidation
            err_abs,err_rel = knnr.crossvalidate(S,X,T)
            print("KNNRegression cross-validation: absolute errors (E,sd,min,max)=", err_abs,
```

```
-----------------------------------------
Example: 1D-linear regression problem
-----------------------------------------
X= [[ 0. ]
 [ 0.5]
 [ 1. ]
 [ 1.5]
 [ 2. ]
[ 2.5]
[ 3. ]
[ 3.5]
[ 4. ]
[ 4.5]
[ 5. ]
[ 5.5]
[ 6. ]
[ 6.5]
[ 7. ]
[ 7.5]
[ 8. ]
[ 8.5]
[ 9. ]
[ 9.5]
[10. ]
[10.5]
[11. ]
[11.5]
[12. ]
[12.5]
[13. ]
[13.5]
[14. ]
[14.5]
[15. ]
[15.5]
[16. ]
[16.5]
[17. ]
[17.5]
[18. ]
[18.5]
[19. ]
[19.5]
[20. ]
[20.5]
[21. ]
[21.5]
[22. ]
[22.5]
[23. ]
[23.5]
[24. ]
[24.5]
[25. ]
[25.5]
[26. ]
```

```
 [26.5]
 [27. ]
 [27.5]
 [28. ]
 [28.5]
 [29. ]
 [29.5]
 [30. ]
 [30.5]
 [31. ]
 [31.5]
 [32. ]
 [32.5]
 [33. ]
 [33.5]
 [34. ]
 [34.5]
 [35. ]
 [35.5]
 [36. ]
 [36.5]
 [37. ]
 [37.5]
 [38. ]
 [38.5]
 [39. ]
 [39.5]
 [40. ]
 [40.5]
 [41. ]
 [41.5]
 [42. ]
 [42.5]
 [43. ]
 [43.5]
 [44. ]
 [44.5]
 [45. ]
 [45.5]
 [46. ]
 [46.5]
 [47. ]
 [47.5]
 [48. ]
 [48.5]
 [49. ]
 [49.5]]
T= [[  3.96539552]
```

```
[  5.40900355]
[  6.80479406]
[  8.03458322]
[  8.3053615 ]
[  8.73249338]
[  8.77151182]
[  9.63643954]
[ 12.01447684]
[ 14.10503997]
[ 14.53639261]
[ 16.56342681]
[ 15.55803052]
[ 17.54062696]
[ 18.10718482]
[ 20.19849118]
[ 19.69947678]
[ 21.30766924]
[ 21.40690018]
[ 22.14653271]
[ 22.94023538]
[ 25.36725429]
[ 26.22817455]
[ 27.78178612]
[ 28.41925754]
[ 27.71492035]
[ 29.23188689]
[ 29.9793288 ]
[ 31.95354147]
[ 32.47758882]
[ 32.27269297]
[ 36.15945205]
[ 35.22589939]
[ 38.2674975 ]
[ 39.33027519]
[ 38.71212506]
[ 40.19112542]
[ 40.37822356]
[ 42.08476023]
[ 41.09623498]
[ 43.07393777]
[ 44.58435281]
[ 46.23278059]
[ 46.80294075]
[ 48.494492  ]
[ 47.73994133]
[ 50.94662245]
[ 50.73035166]
[ 50.82361583]
[ 54.41661601]
[ 54.32981823]
[ 55.41985461]
[ 54.82862886]
[ 58.04696979]
[ 57.30488021]
[ 59.91427266]
[ 60.44129026]
[ 62.53071307]
[ 60.97598519]
[ 62.26103967]
[ 64.32530732]
[ 64.36329645]
[ 68.68025287]
[ 66.79291027]
[ 68.05828692]
[ 69.51147636]
[ 69.78195668]
[ 72.01666756]
[ 72.378812  ]
[ 73.80618286]
[ 73.32671575]
[ 74.58556612]
[ 75.35734468]
[ 77.2098026 ]
[ 79.00187307]
[ 79.67579371]
[ 79.86904299]
[ 80.98003463]
[ 82.03965717]
[ 82.55033359]
[ 81.91388516]
[ 86.1249457 ]
[ 84.73057236]
[ 86.42908599]
[ 86.79082706]
[ 88.72028873]
[ 90.40730705]
[ 91.78703418]
[ 91.15956868]
[ 93.33433493]
[ 93.12277541]
[ 96.2559933 ]
[ 95.55246418]
[ 95.90050568]
[ 97.07683317]
[ 98.35082574]
[ 99.51604217]
```

```
 [100.21538238]
 [103.11867279]
 [102.75984867]]
phi(4)= [1 4]
phi([1,2])= [1 1 2]

-----------------------------------------
Do a Least-Squares-Regression
-----------------------------------------
lsr.W_LSR= [[4.09052788]
 [1.99478017]]
prediction of x= [3.1415] is y= [10.35712978]
LSRRegression cross-validation: absolute errors (E,sd,min,max)= (0.7210491003592423, 0.5070165

-----------------------------------------
Do a KNN-Regression
-----------------------------------------
prediction of x= [3.1415] is y= 9.203975679168057
KNNRegression cross-validation: absolute errors (E,sd,min,max)= (1.1743092886650586, 0.9170722
```

```
In [16]: temp = []
         for i,e in enumerate(KNNErrors):
             temp.append((e[2][3],i))
         temp.sort()
         #print(temp)

         smallest_mean= KNNErrors[1][2]
         smallest_sd= KNNErrors[1][2]
         smallest_min= KNNErrors[13][2]
         smallest_max= KNNErrors[1][2]
         print(smallest_mean)
         print(smallest_sd)
         print(smallest_min)
         print(smallest_max)
         print(KNNErrors[1])
```

```
(0.04137596473341779, 0.07148122842087293, 6.810069605730603e-05, 0.44728370446074633)
(0.04137596473341779, 0.07148122842087293, 6.810069605730603e-05, 0.44728370446074633)
(0.1297162287702424, 0.39638164915995594, 5.395317915671042e-05, 3.1265953272944778)
(0.04137596473341779, 0.07148122842087293, 6.810069605730603e-05, 0.44728370446074633)
(2, (1.0953802187892574, 0.8689819772601937, 0.003291802811396849, 3.884868724720441), (0.0413
```

0.1 a) Versuchen Sie zunächst den Aufbau des Moduls V2A2_Regression.py zu verstehen:

Betrachten Sie den Aufbau des Moduls durch Eingabe von pydoc V2A2_Regressifier.
Welche Klassen gehören zu dem Modul und welchen Zweck haben sie jeweils?

- DataScaler: Standatisiert Daten
- Regressifier: Abstrakte Klasse für Regressierer
- KNNRegressifier: ermöglicht Regression mithilfe von Fast-KNN
- LSRRegressifier: ermöglicht Regression mithilfe von Fehlerquadratsummen

Betrachten Sie nun die Basis-Klasse Regressifier im Quelltext: Wozu dienen jeweils die Methoden fit(self,X,T), predict(self,x) und crossvalidate(self,S,X,T) ?

- fit(self,X,T): Verlangt nach der implimentierung einer Methode die den Regressierer mit X und T trainiert.
- predict(self,x): Verlangt nach der implimentierung einer Methode die auf x eine Regression anwendet und eine Vorhersage zurückliefert
- crossvalidate(self,S,X,T): Macht S-fache Kreuzvaliedierung mit den Daten,Labeln (X,T) und liefert als Ergebniss Informationen über die relativen und absoluten Fehlerwerte

Worin unterscheidet sich crossvalidate(.) von der entsprechenden Methode für Klassifikation (siehe vorigen Versuch)?

- hier kann man noch angeben mit welcher Längenfunktion gearbeitet werden soll, und es werden andere Fehlerstatistiken zurückgegeben

b) Betrachten Sie nun die Funktion phi_polynomial(x,deg):
Was berechnet die Funktion? Welches Ergebnis liefert phi_polynomial([3],5)? Welches Ergebnis liefert phi_polynomial([3,5],2)?

```
In [74]: phi_polynomial([3],5)

Out[74]: array([  1,   3,   9,  27,  81, 243])

In [75]: phi_polynomial([3,5],2)

Out[75]: array([ 1,  3,  5,  9, 15, 25])
```

- phi_polynomial(x,deg): berechnet die Basisfunktion für die Werte x mit dem grad deg

Geben Sie eine allgemeine Formel an für das Ergebnis von phi_polynomial([x1,x2],2)?

- [1, x1, x2, (x1)š, x1*x2, (x2)š]

Wozu braucht man diese Funktion im Zusammenhang mit Regression?

- Das Kreuzprodukt aus dieser Funktion und dem Zielwertevektor geben die Prognose bei der Regression

Bis zu welchem Polynomgrad kann die Funktion Basisfunktionen berechnen? Erweitern Sie die Funktion mindestens bis Grad 5.

- momentan bis grad 3, siehe Verbesserung im code

```
In [83]: phi_polynomial([3,5],5)
```

```
Out[83]: array([   1,    3,    5,    9,   15,   25,   27,   45,   75,  125,   81,
                 135,  225,  375,  625,  243,  405,  675, 1125, 1875, 3125])
```

c) Betrachten Sie die Klasse LSRRegressifier:
Welche Art von Regressions-Modell berechnet diese Klasse?

- Fehlerquadratsummenregression mit Regularisierung

Wozu dienen jeweils die Parameter lmbda, phi, flagSTD und eps ?

- lmbda: Regularisierungsparameter
- phi: Basisfunktion
- flagSTD: gibt an ob die Daten Standatisiert werden sollen
- eps: höchster erlaubter Fehlerrestwert

Welche Rolle spielt hier die Klasse DataScaler? In welchen Methoden und zu welchem Zweck werden die Daten ggf. umskaliert? Welches Problem kann auftreten wenn man dies nicht tut? Wozu braucht man die Variablen Z und maxZ in der Methode fit(.)?

- DataScaler wird in fit() und predict() verwendet um Daten zu Standartisieren
- nicht scalierte Daten werde können kleinste abweichungen in Gewichten zu groSSen und/oder unterschiedlichen Änderungen führen.
- Z ist die Matrix mit den Fehlerwerten die durch Invertierung entstehen.
- maxZ ist der gröSSte Wert aus Z und wird verwendet um zu bestimmen ob der Fehlerwert annehmbar ist

Vervollständigen Sie die Methoden fit(self,X,T,...) und predict(self,x,...) (vgl. vorige Aufgabe).

- siehe code

d) Betrachten Sie die Klasse KNNRegressifier:
Welche Art von Regressions-Modell berechnet diese Klasse?

- fast K-Nearest-Neighbor-Regression

Wozu dienen jeweils die Parameter K und flagKLinReg?

- K: Anzahl der NN die verwendet werden um eine Vorhersage zu treffen
- flagKLinReg: gibt an ob eine Regression verwendet werden soll oder nur der Mittelwert aus den NN genommen wird

Beschreiben Sie kurz in eigenen Worten (2-3 Sätze) auf welche Weise die Prädiktion y(x) berechnet wird.

- Es werden die KNNs für x ermittelt. Damit wird ein LSRRegressifier trainiert, der dann nach einer prediction gefragt wird.

e) Betrachten Sie abschlieSSend den Modultest:
Beschreiben Sie kurz was im Modultest passiert.

- es wird ein Linearer Datensatz erstellt

- die Funktion phi wird neu definiert und auf Grad 2 festgesetzt
- LSR Regression wird durchgeführt und Kreuzvalidiert
- KNN Regression wird durchgeführt und Kreuzvalidiert

Welche Gewichte W werden gelernt? Wie lautet also die gelernte Prädiktionsfunktion? Welche Funktion sollte sich idealerweise (für N) ergeben?

w0 = [3.78705442]
w1 = [2.01036841]

y = 3.787 + 2.010*x

y = 4 + 2*x für N

Welche Ergebnisse liefert die Kreuzvalidierung? Was bedeuten die Werte?

- (E,sd,min,max)= (0.8335277318740052, 0.6423712068147653, 0.030450286307427632, 3.059745154084382)
- E ist der mittlere Fehlerwert
- sd ist die Standartabweichung der Fehler
- min ist der kleinste Fehlerwert
- max ist der gröSSte Fehlerwert

Vergleichen und Bewerten Sie die Ergebnisse von Least Squares Regression gegenüber der KNN-Regression (nach Optimierung der Hyper-Parameter , K, ...).

LSR: absolute errors (E,sd,min,max)= (0.7210491003592423, 0.5070165487566235, 0.0066007843635347285, 2.780939336314603)
KNN: absolute errors (E,sd,min,max)= (1.1743092886650586, 0.9170722237516987, 0.0038023455924474092, 4.2882465799825695)
LSR: relative errors (E,sd,min,max)= (0.02240702968734219, 0.028216719179408916, 0.0001525998169486947, 0.14635559861181296)
KNN: relative errors (E,sd,min,max)= (0.05354319692710508, 0.13202983166413382, 5.0979643788569236e-05, 1.0603171407681475)

- LSR scheint in fast allen Bereichen wesentlich bessere Ergebnisse als KNNR zu liefern.