CS 559: Machine Learning Fundamentals & Applications

Lecture 4: Linear Regression

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

4.1. Linear Basis Function Models

The simplest linear model for regression is called linear regression that is a linear combination of the input variables

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots w_D x_D. \tag{4-1}$$

- Input variable $\boldsymbol{x} = (x_1, \ldots, x_D)^T$
- Parameters $\boldsymbol{w} = (w_0, w_1, \ldots, w_D)$
- For a simplicity, Eq. (4-1) above can be expressed as

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=0}^{D} w_i x_i = \boldsymbol{w}\boldsymbol{x} \tag{4-2}$$

where $\boldsymbol{x} = (1, x_1, \ldots, x_D)^T$.

4.1. Linear Basis Function Models

If $y(\boldsymbol{x}, \boldsymbol{w})$ is **not linear**, the model will impose significant limitation. Instead, we can make the model be a linear combination of fixed nonlinear function of the input variables in the form

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} \boldsymbol{w_j \phi_j}(\boldsymbol{x}) = \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{4-3}$$

where $\phi_j(\boldsymbol{x})$ are known as ***basis function***.

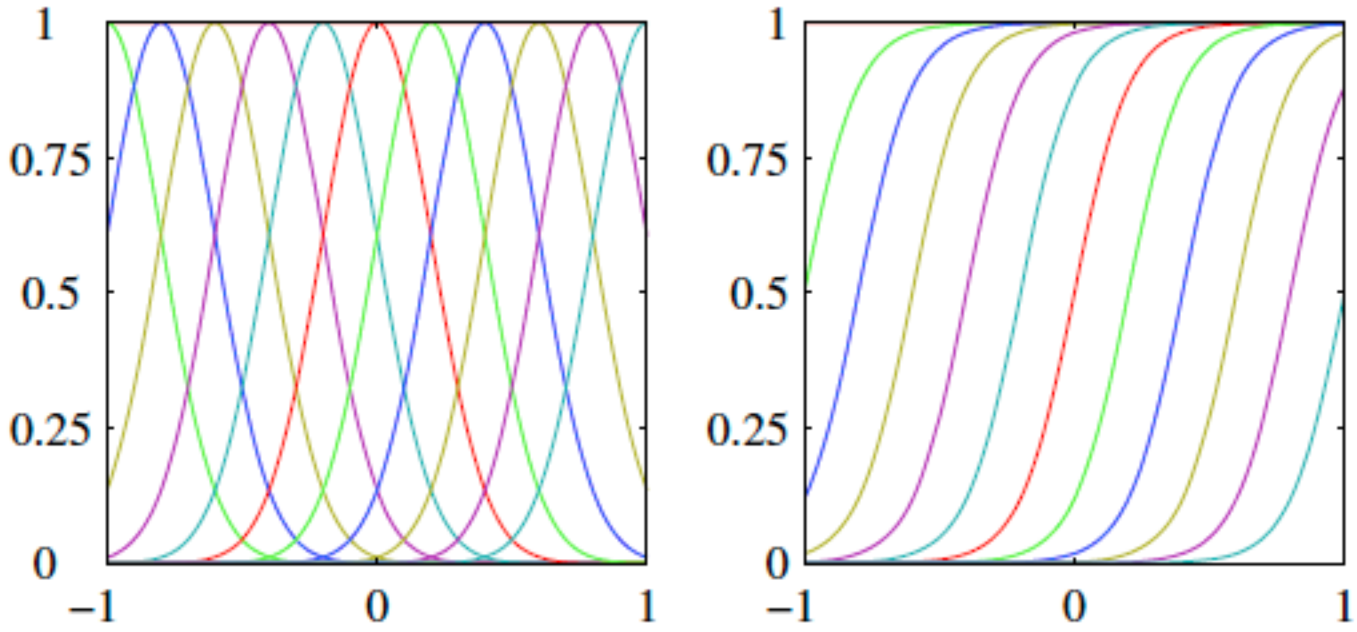
When nonlinear basis function is used, y is not linear in terms of $\boldsymbol{x}$ but is linear in $\boldsymbol{w}$.
This linearity in parameters simplifies the model greatly!

4.1. Linear Basis Function Models

Some possible choices for the basis functions are

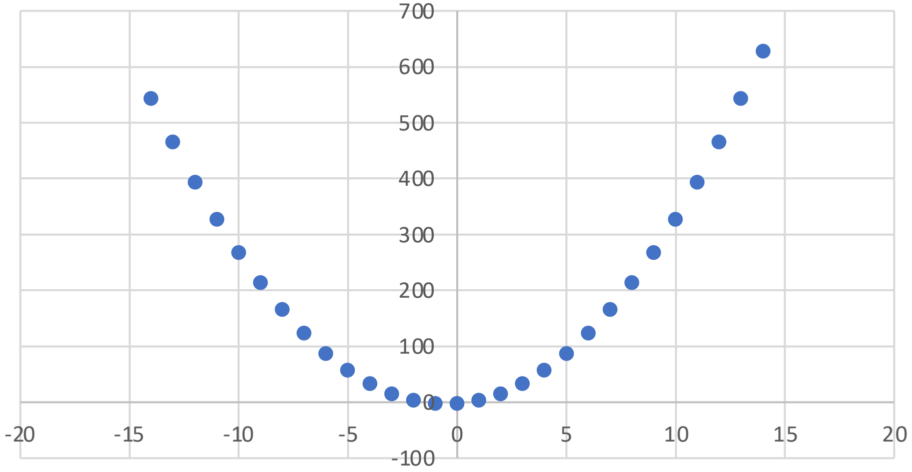
$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2\sigma^2}\right\}$$

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right) \text{ where } \sigma(a) = \frac{1}{1+\exp(-a)}$$



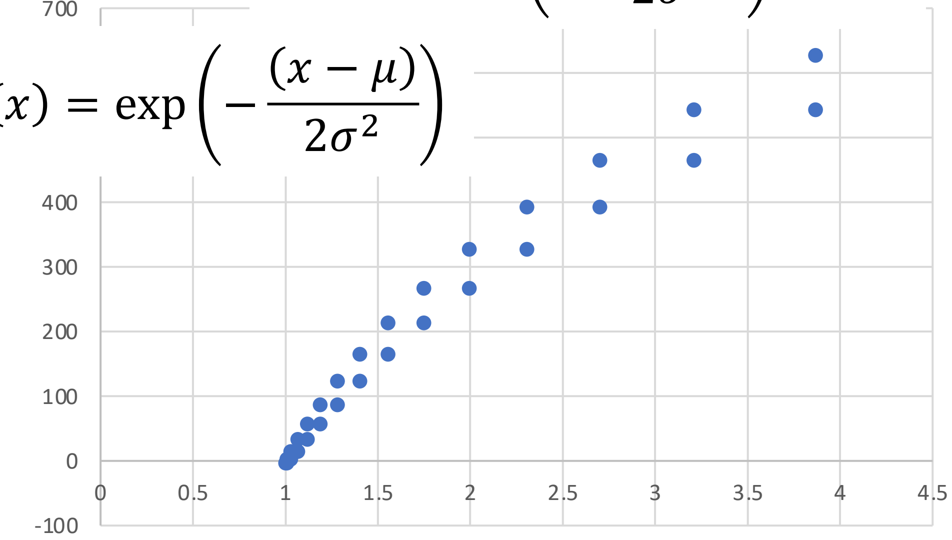
4.1. Linear Basis Function Models

$$y = ax^2 + bx + c$$



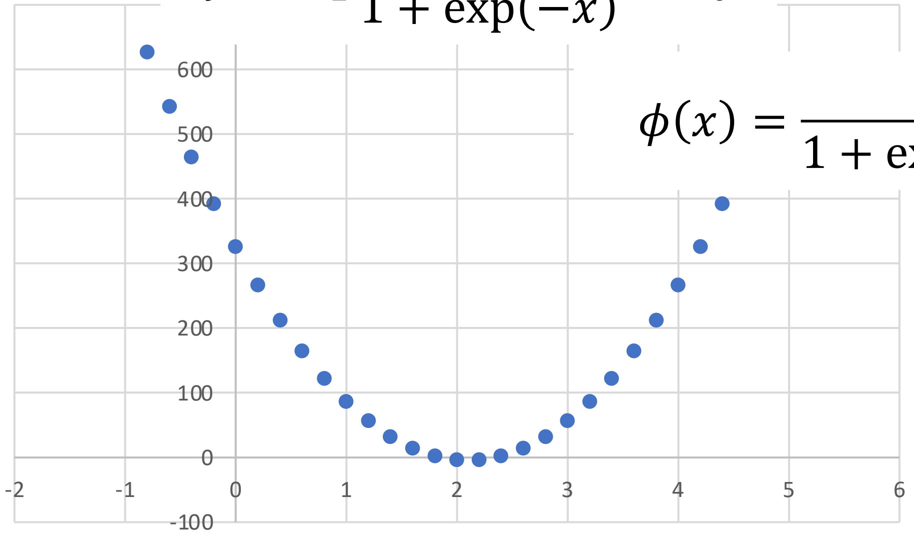
$$y = w_1 \exp\left(-\frac{(x-\mu)}{2\sigma^2}\right) + w_0$$

$$\phi(x) = \exp\left(-\frac{(x-\mu)}{2\sigma^2}\right)$$



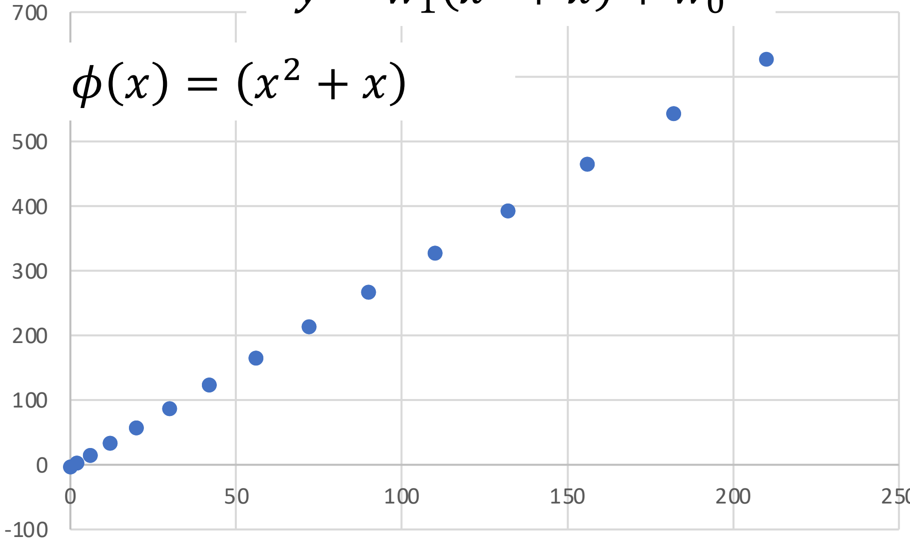
$$y = w_1 \frac{1}{1+\exp(-x)} + w_0$$

$$\phi(x) = \frac{1}{1+\exp(-x)}$$



$$y = w_1(x^2 + x) + w_0$$

$$\phi(x) = (x^2 + x)$$



4.1.1. Maximum Likelihood and Least Squares

Assume the target variable t is given by a deterministic function $y(\boldsymbol{x}, \boldsymbol{w})$ with additive Gaussian nose $\epsilon = \mathcal{N}(0, \beta^{-1})$ where β is the inverse of variance ($\beta = \sigma^{-2}$),

$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon. \tag{4-4}$$

Recall the conditional probability in Gaussian, the likelihood of Eq. (4-4) can be expressed as

$$p(t|x, w, \beta) = \mathcal{N}(t|y(x, w), \beta^{-1}). \tag{4-5}$$

Consider a data set of inputs $\boldsymbol{X} = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_N}\}$ with corresponding target values $\boldsymbol{t} = (t_1, \dots, t_N)$. Eq. (4-5) becomes

$$\begin{aligned} p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) &= \prod_{n=1}^{N} \mathcal{N}\left(t_n|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n}), \beta^{-1}\right) \\ &= \prod_{n=1}^{N} \frac{\beta^{N/2}}{\sqrt{(2\pi)^N}} \exp\left(-\frac{\beta}{2}(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n})^2)\right) \end{aligned} \tag{4-6}$$

4.1.1. Maximum Likelihood and Least Squares

Assume that data points are i.i.d., then we can write Eq. 4-6 as $p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\beta) = p(\boldsymbol{t}|\boldsymbol{w},\beta)$. Then log-likelihood then becomes

$$\ln p(\boldsymbol{t}|\boldsymbol{w},\beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}),\beta^{-1})$$

$$\boxed{\mathcal{N} = \frac{\beta^{N/2}}{\sqrt{(2\pi)^N}} \exp\left(-\frac{\beta}{2}(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n})^2)\right)}$$

$$= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \frac{\beta}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n})\}^2$$

$$= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\boldsymbol{w}) \qquad (4\text{-}7)$$

where the sum-square-error function is $E_D = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n})\}^2$.

4.1.1. Maximum Likelihood and Least Squares

To find $\boldsymbol{w}$, we need to maximize the log-likelihood function that is equivalent to *minimizing* $E_D(\boldsymbol{w})$:

$$\nabla \ln p(\boldsymbol{t}|\boldsymbol{w}, \beta) = 0 \to \nabla E_D(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}} E_D(\boldsymbol{w}) = \sum_{n=1}^{N} \{t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n})\} \boldsymbol{\phi}(\boldsymbol{x_n})^T = 0 \tag{4-8}$$

where β is the constant and therefore, it does not impact the result of $\boldsymbol{w}$.

Solving for $\boldsymbol{w}$,

$$\boldsymbol{w_{ML}} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \tag{4-9}$$

which is known as the *normal equations* for the least squares problem. The *design matrix* $\boldsymbol{\Phi}$ is an $N \times M$ matrix whose elements are $\Phi_{nj} = \phi_j(\boldsymbol{x_n})$:

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \cdots & \phi_{M-1}(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix}.$$

4.1.1. Maximum Likelihood and Least Squares

If w_0 is explicit, then

$$E_D = \frac{1}{2}\sum_{n=1}^{N}\left\{t_n - w_0 - \sum_{j=1}^{M-1} w_j\phi_j(x_n)\right\}^2 \qquad (4\text{-}10)$$

and

$$w_0 = \bar{t} - \sum_{j=1}^{M-1} w_j\overline{\phi_j} \qquad (4\text{-}11)$$

where

$$\bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n\,, \qquad \overline{\phi_j} = \frac{1}{N}\sum_{n=1}^{N}\phi_j(\boldsymbol{x_n})\,. \qquad (4\text{-}12)$$

4.1.1. Maximum Likelihood and Least Squares

We can also estimate the noise precision parameter β,

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}_{\boldsymbol{ML}}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{x}_{\boldsymbol{n}})\right\}^2 \tag{4-12}$$

where $\sigma_{ML}^2 = \beta_{ML}^{-1}$.

4.1.2. Sequential Learning

If the data set is sufficiently large, it is worth using the *sequential* algorithm, also known as the *online* algorithm. The solutions obtained via MLE (Eq. 4-8) will be underestimated.

The technique of ***stochastic gradient descent*** (aka *sequential gradient descent*) updates the parameter $\boldsymbol{w}$ through the iteration τ with an initial set of $\boldsymbol{w}^{(0)}$:

$$\boxed{\nabla E_n = \partial E_n / \partial \boldsymbol{w}}$$

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n$$

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \eta\left(t_n - \boldsymbol{w}^{(\tau)T} \boldsymbol{\phi}_n\right)\boldsymbol{\phi}_n \tag{4-13}$$

where η is a learning rate parameter and $\boldsymbol{\phi}_{\boldsymbol{n}} = \phi(\boldsymbol{x}_n)$ and it is $\eta \ll 1$.

4.1.2. Sequential Learning

1. **Gradient Descent (GD, bath gradient descent):**
   - It involves using the entire dataset or training set to compute the gradient to find the optimal solution.
   - Our movement towards the optimal solution, which could be the local or global optimal solution, is always direct.
   - However, this can become a major challenge when millions of samples are running.
2. **Stochastic Gradient Descent (SGD):**
   - The dataset is properly shuffled to avoid pre-existing orders then partitioned into *m* examples or an example at a time after reordering.
   - This random approximation of the data set removes the computational burden associated with gradient descent while achieving iteration faster and at a lower convergence rate.
   - Tt tends to result in more noise than GD.
3. **Mini-Batch Gradient Descent:** Bridge between GD and SGD.

4.1.2. Sequential Learning

Suppose there are m examples, $\boldsymbol{x}$ has n many features.

- GD: Repeat until the error function converges

Repeat {

for $j = 1, \dots, n$ {

$$w_j = w_j - \frac{\eta}{m}\sum_{i=1}^{m}\left(y_w\left(\boldsymbol{x}^{(i)}\right) - \boldsymbol{t}^{(i)}\right)\boldsymbol{x}_j^{(i)}$$

}

}

- SGD: Randomly shuffle and repeat until the error function converges

Repeat {

for $i = 1, \dots, m$ {

$$w_j = w_j - \eta\left(y_w\left(x^{(i)}\right) - t^{(i)}\right)x_j^{(i)} \quad \forall\, j = [0, \dots, n]$$

}

}

4.1.2. Sequential Learning

- Mini-Batch GD: Randomly shuffle and repeat the error function until converges
  - Suppose we make 10 batches with 1000 examples.

Repeat {

for $i = 1{,}11{,}21, \dots, 991$ {

$$w_j = w_j - \frac{\eta}{10} \sum_{k=i}^{i+9} \left(y_w\left(x^{(k)}\right) - t^{(k)} \right) x_j^{(k)} \quad \forall\, j = [0, \dots, n]$$

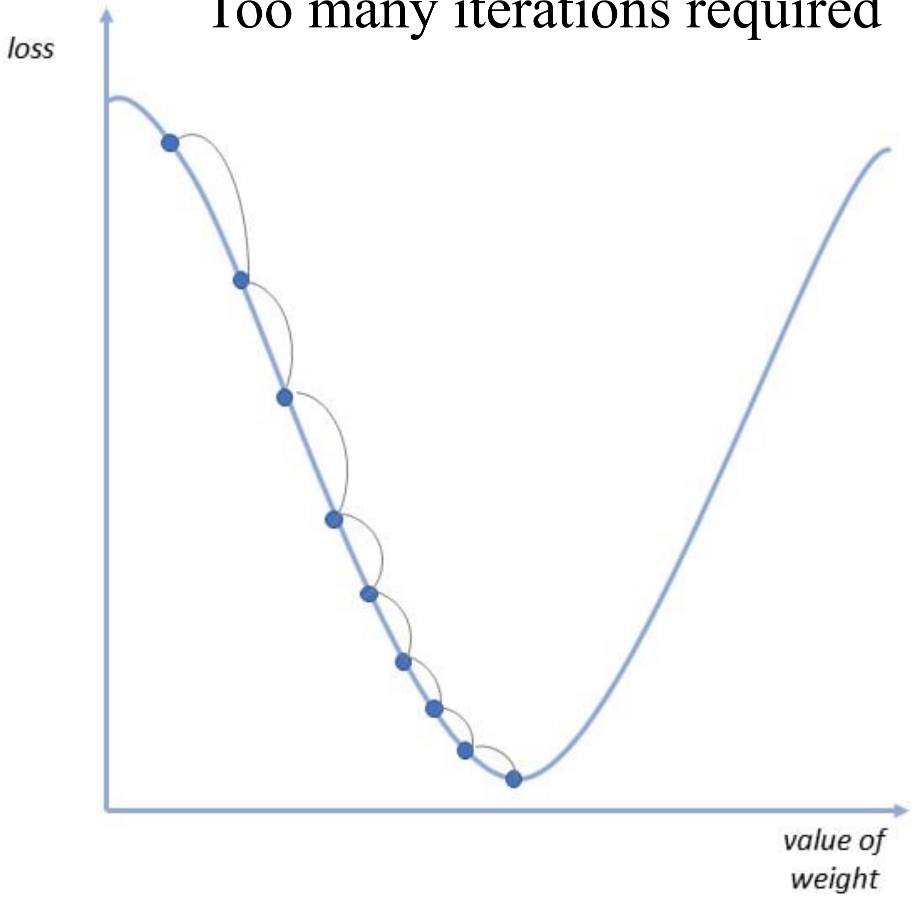
}

}

4.1.2. Sequential Learning

Small Learning Rate

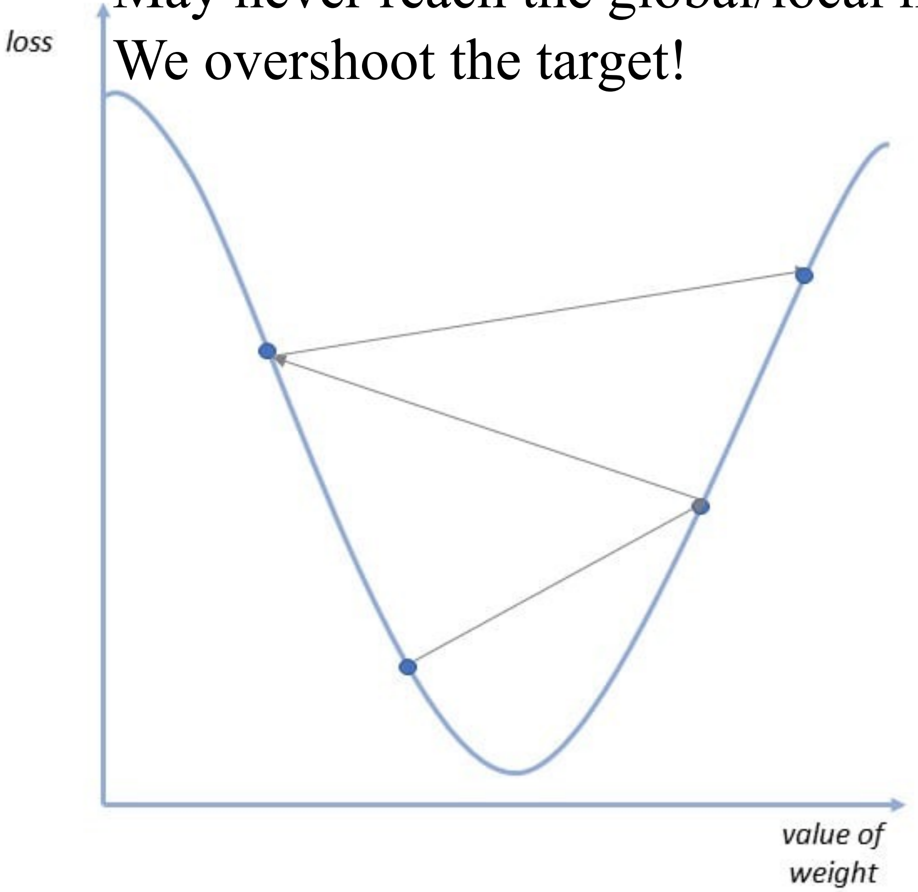
Too many iterations required



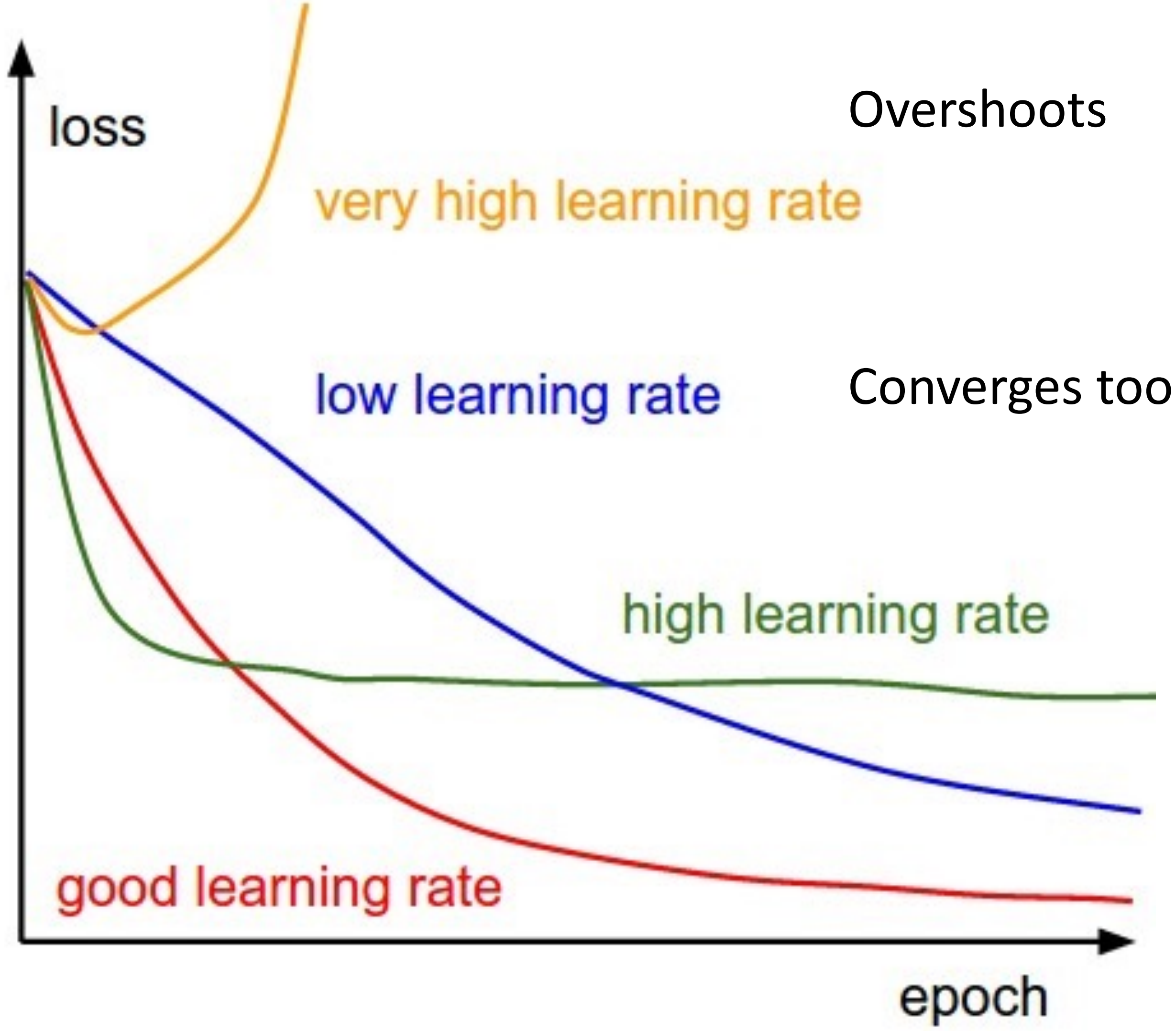
Large Learning Rate

May never reach the global/local minimum

We overshoot the target!



4.1.2. Sequential Learning



Overshoots

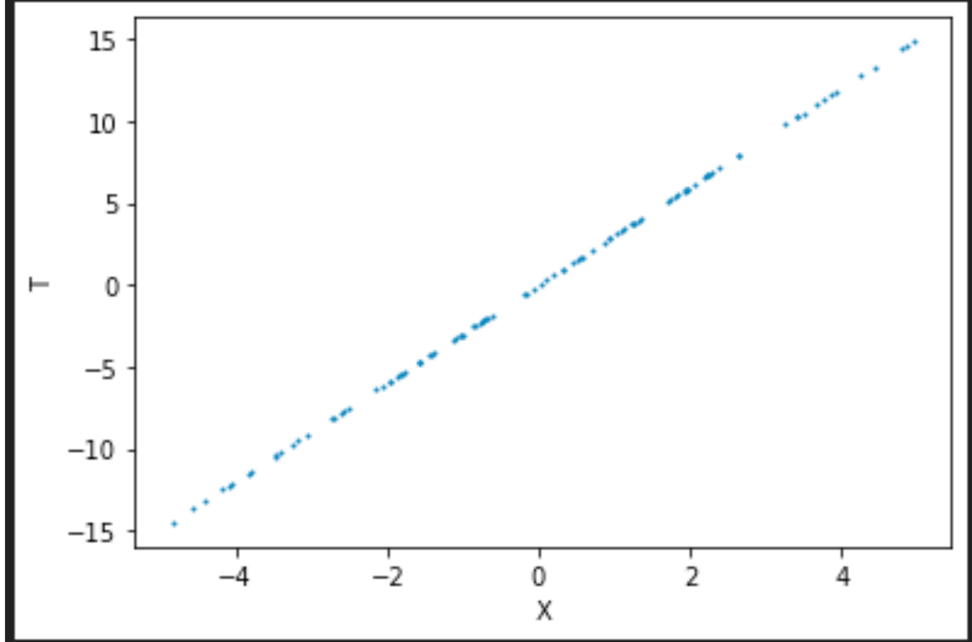
Converges too slow

Converges too early

4.1.2. Sequential Learning

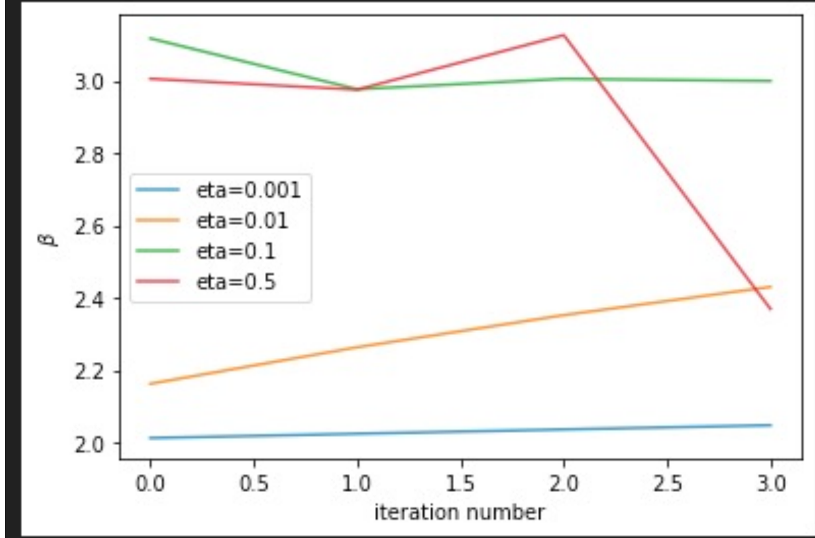
```
np.random.seed(123)
X = 10*np.random.sample(100)-5
T = 3*X
plt.scatter(X,T,s=1.2)
plt.ylabel('T')
plt.xlabel('X')
plt.show()
```

✓ 0.4s

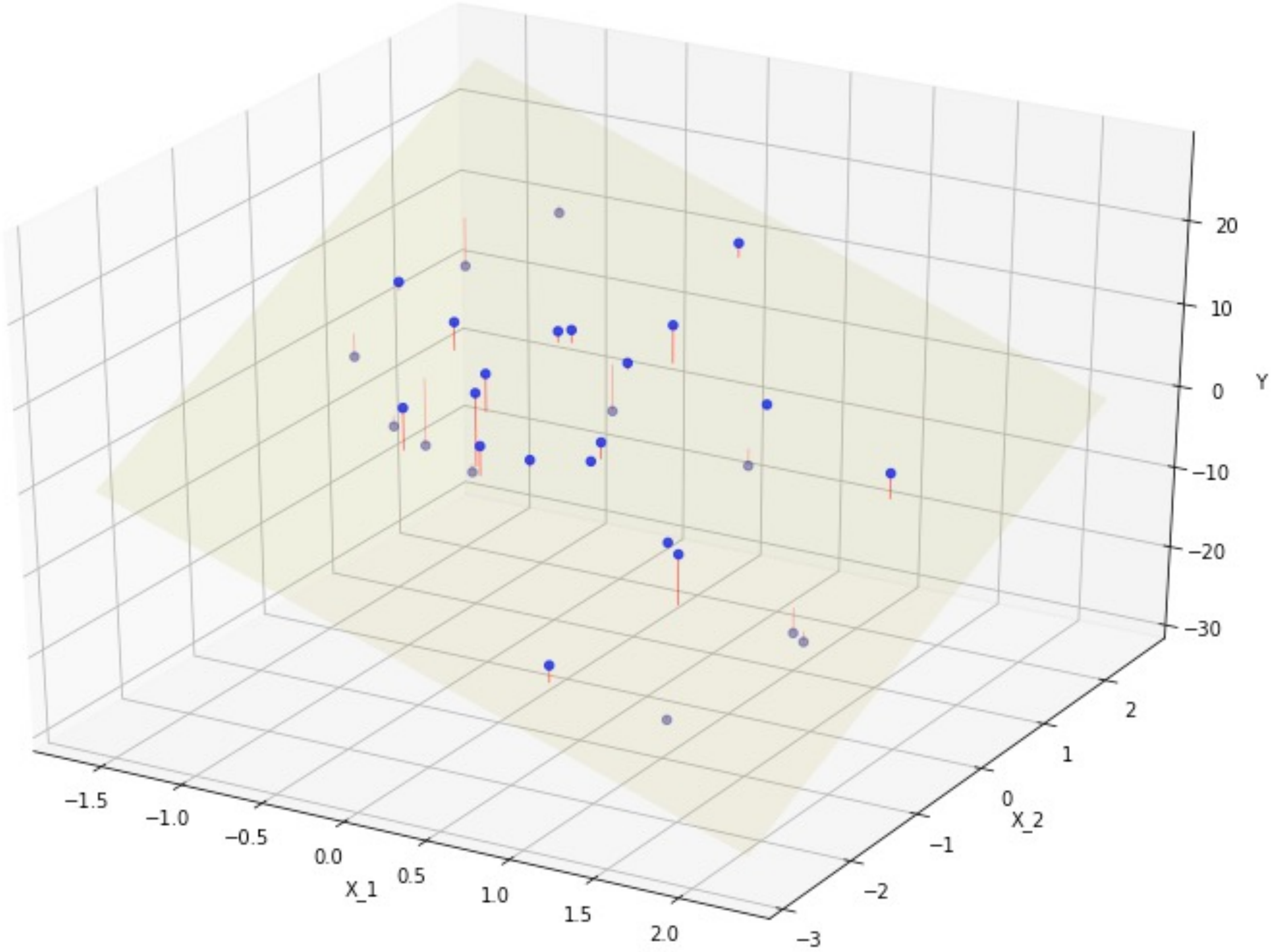



```
beta = 2
learning_rate = [1E-3, 1E-2, 0.1, 5E-1]
beta_list = []
for eta in learning_rate:
    beta_temp = []
    for i in range(4):
        Y = beta*X
        grad_loss = np.mean(2*(T-beta*X)*-X)
        beta = beta - eta*grad_loss
        beta_temp.append(beta)
    beta_list.append(beta_temp)
```

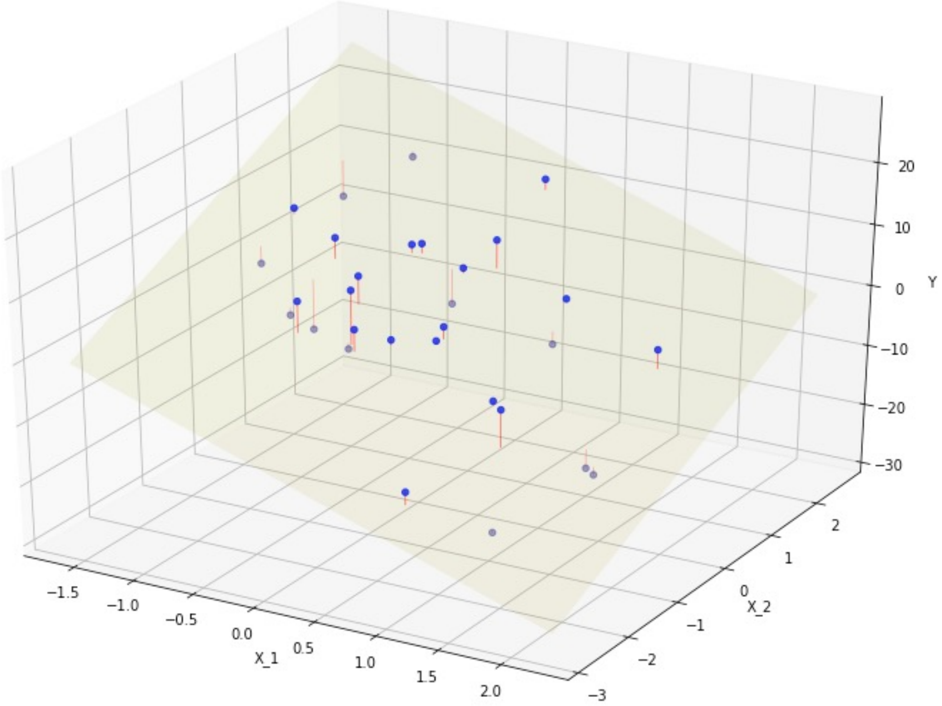
```
eta=0.001,beta=[2.0120427634766513, 2.023940498801148, 2.035694952513224, 2.0473078501194486]
eta=0.01,beta=[2.162038312390189, 2.2629520564539956, 2.3517129970047645, 2.429784667425353]
eta=0.1,beta=[3.116481505521015, 2.9762055834816294, 3.004860636501196, 2.999007086928208]
eta=0.5,beta=[3.0049857955664416, 2.9749644171916443, 3.1257132183222187, 2.3687459412506553]
```

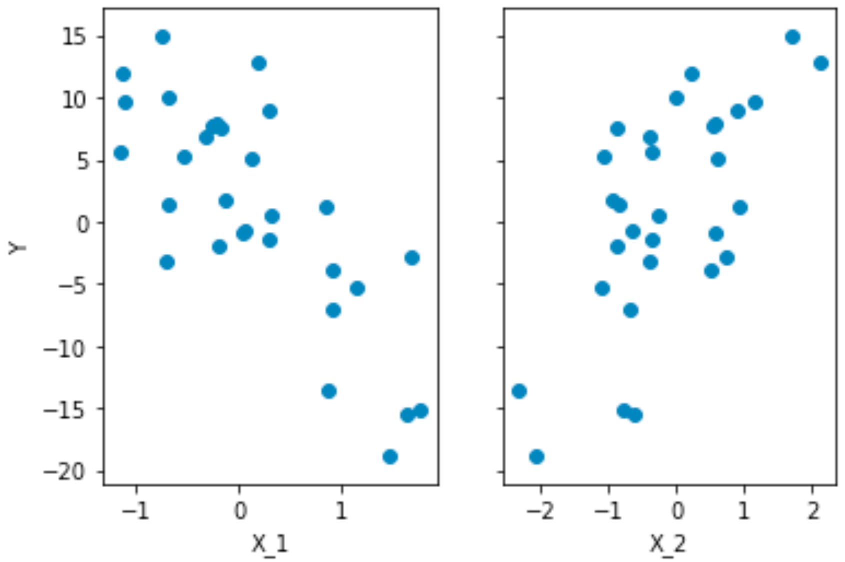


4.1.3. sklearn Linear Regression



4.1.3. sklearn Linear Regression






```
ols.fit(x_m, y_m)
print("beta_1, beta_2: " + str(np.round(ols.coef_, 3)))
print("beta_0: " + str(np.round(ols.intercept_, 3)))
print("RSS: %.2f" % np.sum((ols.predict(x_m) - y_m) ** 2))
print("R^2: %.5f" % ols.score(x_m, y_m))
```

```
beta_1, beta_2: [-6.619  4.436]
beta_0: 2.523
RSS: 356.34
R^2: 0.83938
```

Ordinary Least Square (OLS) is the result of MLE:

If $y_i = \beta_0 + \beta x_i + \epsilon_i$,

$$\beta = \frac{n\sum x_i y_i - \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2}, \beta_0 = \bar{y} - \beta\bar{x}.$$

If $y_i = \sum_{j=1}^{n} \beta_j X_{ij}, (i = 1, \dots, n)$,

$$\boldsymbol{\beta} = \frac{\boldsymbol{X}^T\boldsymbol{y}}{(\boldsymbol{X}^T\boldsymbol{X})^{-1}}$$

4.1.3. sklearn Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
```
✓ 0.3s

Parameters:

- **fit_intercept** (*bool, default=True*): If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
- **Normalize** (*bool, default=False*)**:** This parameter is ignored when **fit_intercept** is set to False. If True, the regressors X will be normalized before regression. Need to standardize the data with false when the standardized data is used.
- **copy_X** (*bool, default=True*): If True, X will be copied; else, it may be overwritten.
- **n_jobs** (*int, default=None*): The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly n_targets > 1 and secondly X is sparse or if positive is set to True. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.
- **positive:** *bool, default=False.* When set to True, forces the coefficients to be positive. This option is only supported for dense arrays.

4.1.3. sklearn Linear Regression

Recall the data from lecture 2,

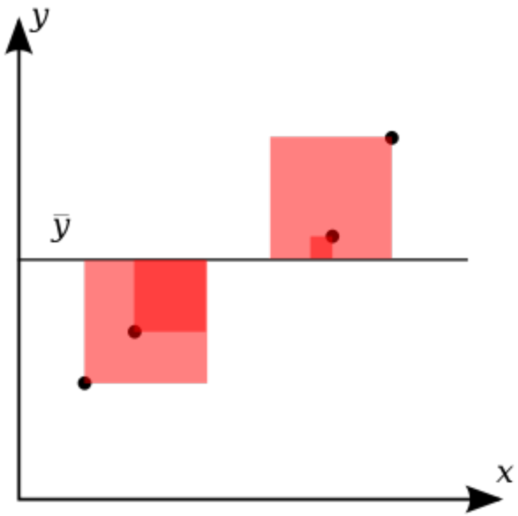
longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	rooms_per_household	bedrooms_per_room	population_per_household	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
-122.23	37.88	32.672294	8.208606	5.645033	7.675271	5.705034	2.728723	61.003713	3.113149	-2.957260	0.876915	0	0	0	1	0
-122.22	37.86	17.070476	11.413717	8.717972	11.487953	8.974699	2.724071	58.036799	2.848110	-2.822167	0.707407	0	0	0	1	0
-122.26	37.84	33.434943	9.799169	7.954136	10.099133	7.961036	0.797670	52.585885	2.062241	-1.811675	0.671329	0	0	0	1	0
-122.26	37.85	39.491212	8.560402	6.722807	9.055987	6.750997	0.822822	47.360433	2.038908	-1.864239	0.905250	0	0	0	1	0
-122.26	37.84	39.491212	9.597088	7.399371	9.717414	7.430798	0.746034	48.672270	2.506253	-2.271723	0.808116	0	0	0	1	0

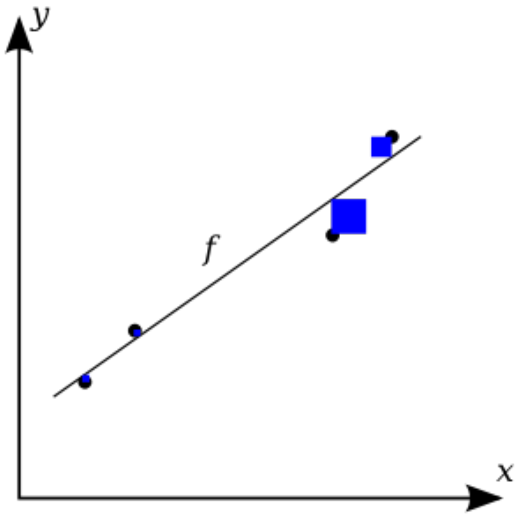
```
Y = df['median_house_value']
features = list(set(df.columns.tolist()[3:])-set(['median_house_value']))
print(features)
X = df[features]
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.3,random_state=42)
```

4.1.3. sklearn Linear Regression

```
def lr_model(xT,xt,yT,yt,features):
    """
    xT: Original Train X, yT: Original Train Y
    xt: Original Test X, yt: Original Test Y
    features: features in train dataset
    """
    XT,Xt = xT[features], xt[features]
    lr = LinearRegression()                     #Make a model
    lr.fit(XT,yT)                               #Fit a train data
    y_train_pred = lr.predict(XT)               #Predict the train data
    rmse = mean_squared_error(yT,y_train_pred)  #Calculate root-mean-squared-error
    mae = mean_absolute_error(yT,y_train_pred)  #Calculate mean-absolute-error
    rsq = r2_score(yT,y_train_pred)             #Calculate R^2
    print(f'train: rmse={rmse}, mae={mae}, R^2={rsq}')
    y_test_pred = lr.predict(Xt)                #Predict the test data
    rmse = mean_squared_error(yt,y_test_pred)
    mae = mean_absolute_error(yt,y_test_pred)
    rsq = r2_score(yt,y_test_pred)             #Calculate R^2
    print(f'train: rmse={rmse}, mae={mae}, R^2={rsq}')
    train_res = yT - y_train_pred               #Calcualte residuals
    test_res = yt - y_test_pred
    plt.scatter(yT,train_res,alpha=0.4,label='train') #Plot residuals
    plt.scatter(yt,test_res,alpha=0.4, label='test')
    plt.hlines(y=0,xmin=yT.min()-2,xmax=yT.max()+2,linestyles='--',color='r')
    plt.legend()
    plt.show()
    return lr
```

4.1.3. sklearn Linear Regression





- Total Sum of Squares (TSS):

$$\sum_{i}^{N}(y_i - \bar{y})^2$$

- Residual Sum of Squares (RSS):

$$\sum_{i}^{N}(t_i - y_i)^2$$

- R-square:

$$R^2 = 1 - \frac{RSS}{TSS}$$

- Root Mean Squared Error (RMSE):

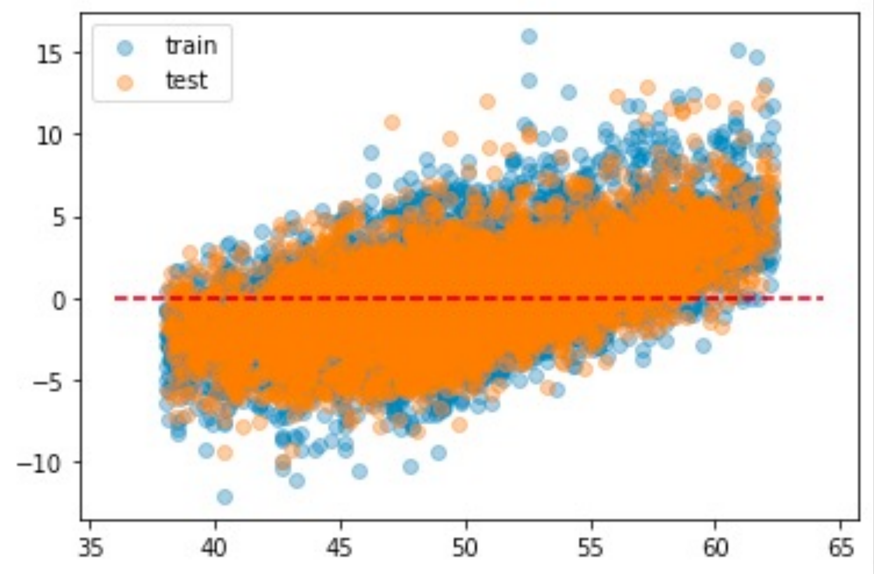
$$\sqrt{\frac{1}{N}\sum_{i}^{N}(t_i - y_i)^2} = \sqrt{\frac{RSS}{N}}$$

- Mean Squared Error (MSE):

$$\frac{1}{N}\sum_{i}^{N}|t_i - y_i|^2 = \frac{RSS}{N}$$

4.1.3. sklearn Linear Regression

```
1  lr_all = lr_model(X_train,X_test,y_train,y_test,features)
✓ 1.9s
train: rmse=8.206180522556119, mae=2.2169320667615215, R^2=0.7106131410345096
train: rmse=7.960813891167741, mae=2.1907446196748688, R^2=0.7251373701845317
```



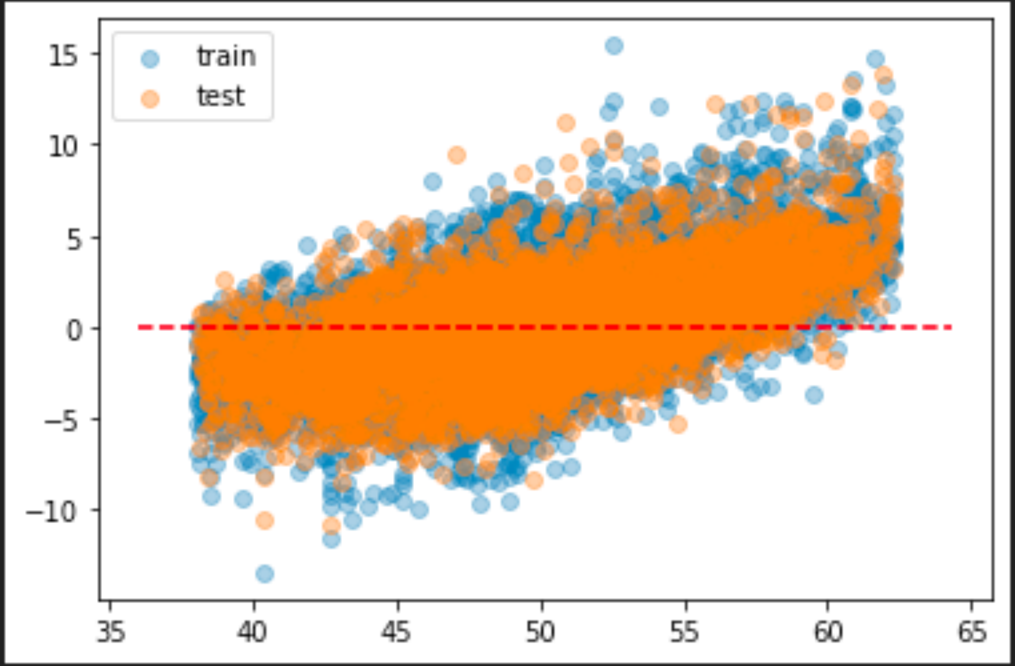
	features	weight
9	total_rooms	-26.731592
13	population_per_household	-19.814847
3	INLAND	-5.188291
5	NEAR BAY	-1.619131
1	<1H OCEAN	-0.800810
6	NEAR OCEAN	-0.226635
12	housing_median_age	0.054975
4	total_bedrooms	0.411882
0	bedrooms_per_room	1.481176
2	population	6.031911
11	median_income	6.550643
7	ISLAND	7.834866
8	households	19.421038
10	rooms_per_household	19.867423

4.1.3. – sklearn Linear Regression

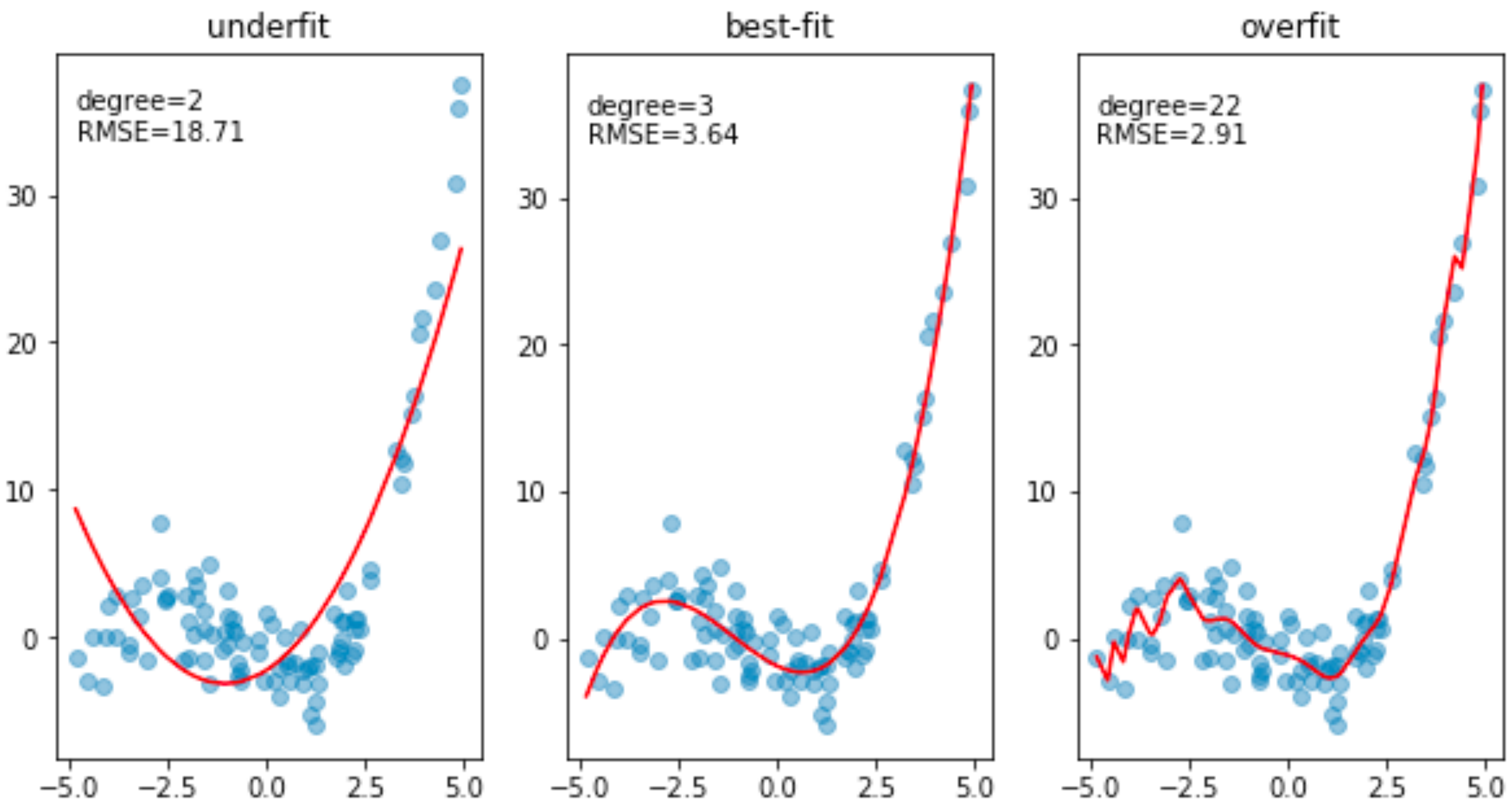
```
new_features = ['population_per_household','bedrooms_per_room','rooms_per_household','median_income','households',
'<1H OCEAN', 'population', 'INLAND', 'NEAR BAY', 'NEAR OCEAN', 'ISLAND']
lr_new = lr_model(X_train,X_test,y_train,y_test,new_features)
```

✓ 1.2s

```
train: rmse=8.547629583852357, mae=2.2610559882036503, R^2=0.6985721103657796
train: rmse=8.324502107296473, mae=2.2414503816061435, R^2=0.71258032502751
```



4.2.1. Overfit vs. Underfit



4.2.2. Regularized Least Squares

To control over-fitting, we can construct the total error function to be minimized in the form by adding the regularization term $E_w(\boldsymbol{w})$

$$E_D(\boldsymbol{w}) + \lambda E_w(\boldsymbol{w})$$

where λ is the regularization coefficient that controls the relative importance of the data-dependent error $E_D(\boldsymbol{w})$ and the regularization term $E_w(\boldsymbol{w})$.

4.2.2. Regularized Least Squares

Approaching from Bayesian theorem,

Likelihood function
See, slide 9.

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)},$$

Prior probability of ***w***

$p(D)$ is a normalizer so it can be ignored.

$$p(\boldsymbol{w}|D) = \prod_{n=1}^{N} p(D|\boldsymbol{w})p(\boldsymbol{w}) \rightarrow \ln p(\boldsymbol{w}|D) = \sum_{n=1}^{N} p(D|\boldsymbol{w}) + \sum_{n=1}^{N} p(\boldsymbol{w})$$

Assume the prior distribution of the coefficients are i.i.d. Gaussian, $w_k \sim \mathcal{N}(0, 1/\lambda)$, then

$$\sum p(\boldsymbol{w}) = \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}.$$

The total error function becomes

$$E = E_D + \lambda E_w = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}\}^2 + \frac{1}{2}\lambda\boldsymbol{w}^T\boldsymbol{w}.$$

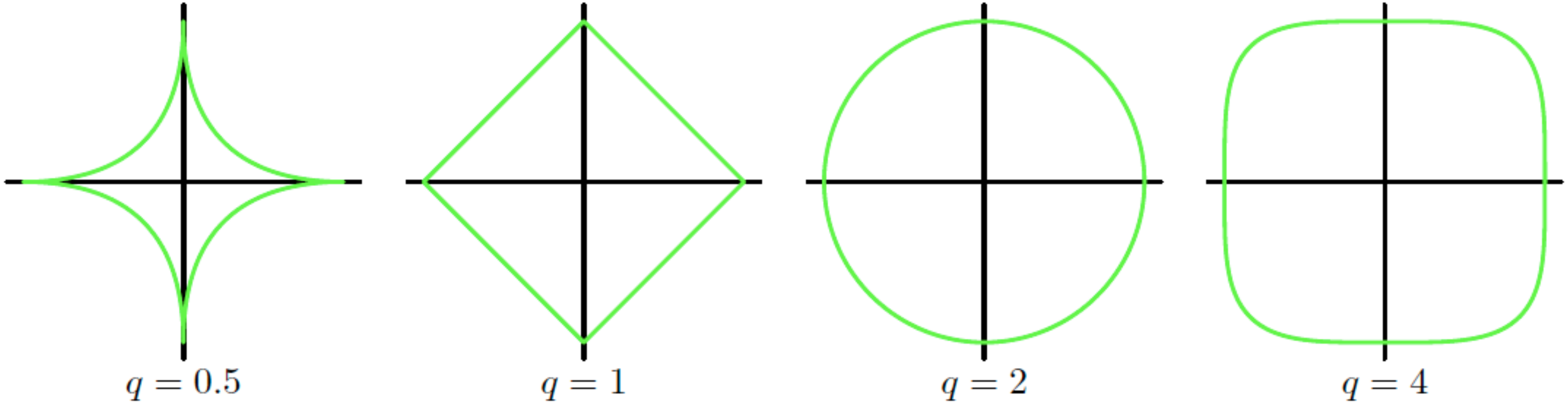
Solving for ***w***,

$$\boldsymbol{w} = \left(\lambda\boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}. \qquad (4\text{-}13)$$

4.2.2. Regularized Least Squares

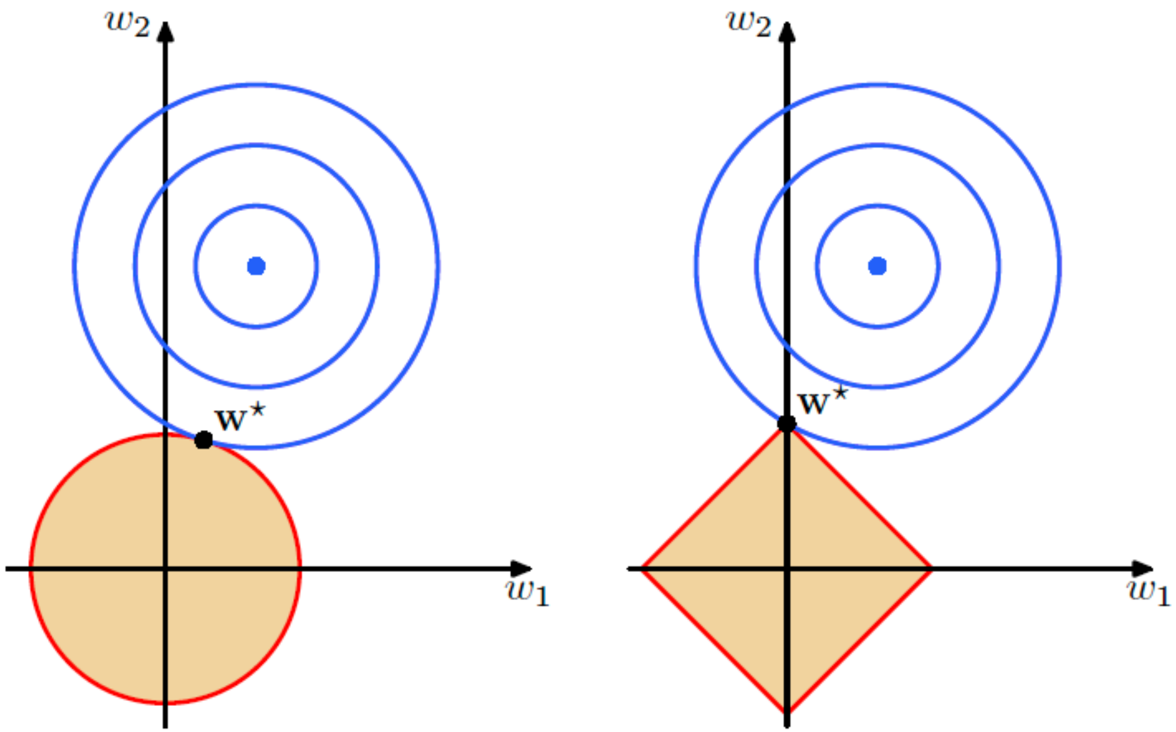
A more general regularized error form is

$$E = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n})\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q \,. \qquad (4\text{-}14)$$



Usually, $q = 1|2$.

4.2.2. Regularized Least Squares



If $q = 2$, it is called the *ridge* regularization.

- The function is strictly convex and differentiable everywhere.
- The dense solutions are possible, and all features are used.

If $q = 1$, is is called the *lasso* regularization.

- If λ is sufficiently large, some of the coefficients w_j are driven to zero.
- It leads to a *sparse* solution

4.2.2. Regularized Least Squares

Often, we can include them both regularizations. This is called *elastic net* regularization in the form

$$E = \frac{1}{2}\sum_{n=1}^{N}\{t_n - w^T\phi(x_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}\left|w_j\right|^2 + \left|w_j\right|.$$

- The function is not differentiable.
- We have a unique solution.

4.2.2. Regularized Least Squares

```python
from sklearn.linear_model import Lasso, Ridge, ElasticNet
```

✓ 0.1s

```python
np.random.seed(123)
X = 10*np.random.sample(100)-5
X1 = np.array(sorted(X))
X2 = X**2
X3 = X**3
X4 = X**3/2
X = np.column_stack((X1,X2,X3,X4))
```

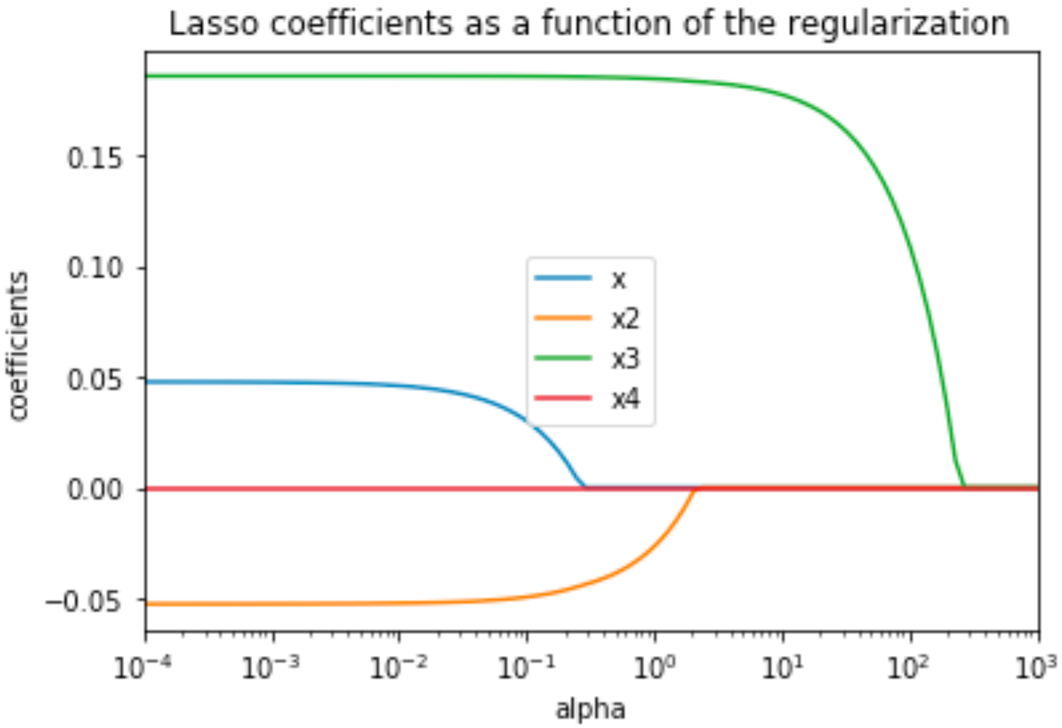
✓ 0.1s

4.2.2. Regularized Least Squares

```
alphas_lasso = np.logspace(-4, 3, 100)
coef_lasso = []
lasso = Lasso()
rmse, mae, rsq = [], [], []
for i in alphas_lasso:
    lasso.set_params(alpha = i).fit(X,Y)
    coef_lasso.append(lasso.coef_)
    y_train_pred = lasso.predict(X)

    rmse.append(mean_squared_error(Y,y_train_pred))  #Calculate root-mean-squared-error
    mae.append(mean_absolute_error(Y,y_train_pred))  #Calculate mean-absolute-error
    rsq.append(r2_score(Y,y_train_pred))             #Calculate R^2

features = ['x','x2','x3','x4']
df_coef = pd.DataFrame(coef_lasso, index=alphas_lasso, columns=features)
title = 'Lasso coefficients as a function of the regularization'
df_coef.plot(logx=True, title=title)
plt.xlabel('alpha')
plt.ylabel('coefficients')
plt.show()
```

✓ 2.2s

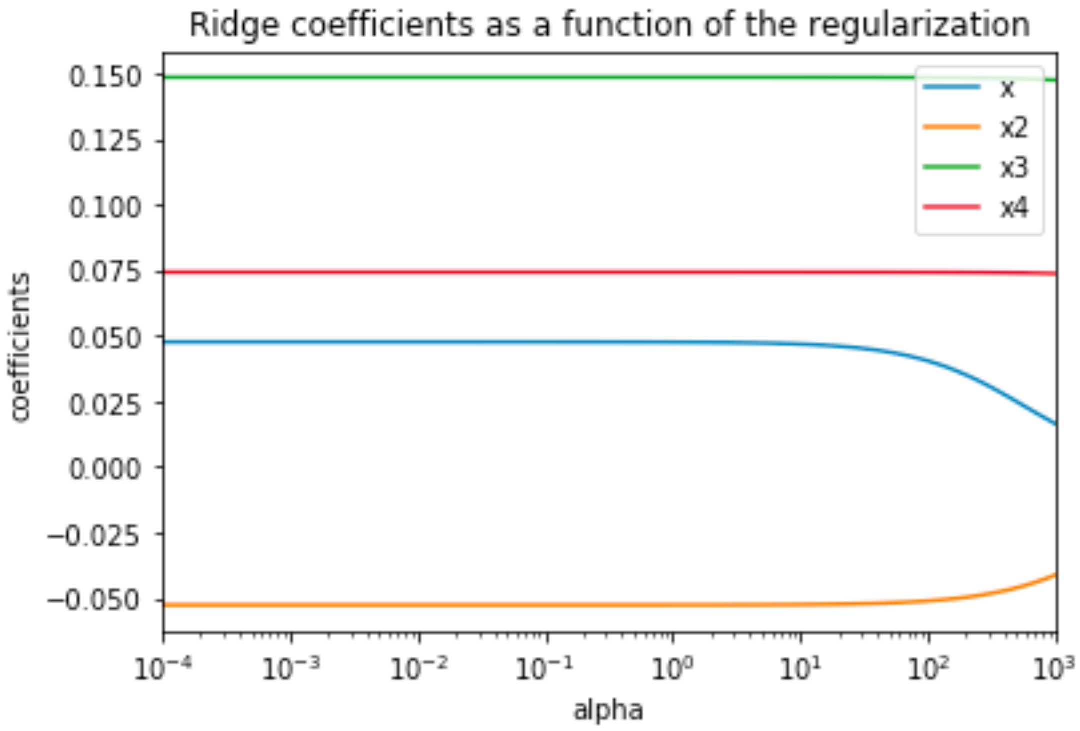
4.2.2. Regularized Least Squares



Lasso regularization therefore can be used for feature selection.

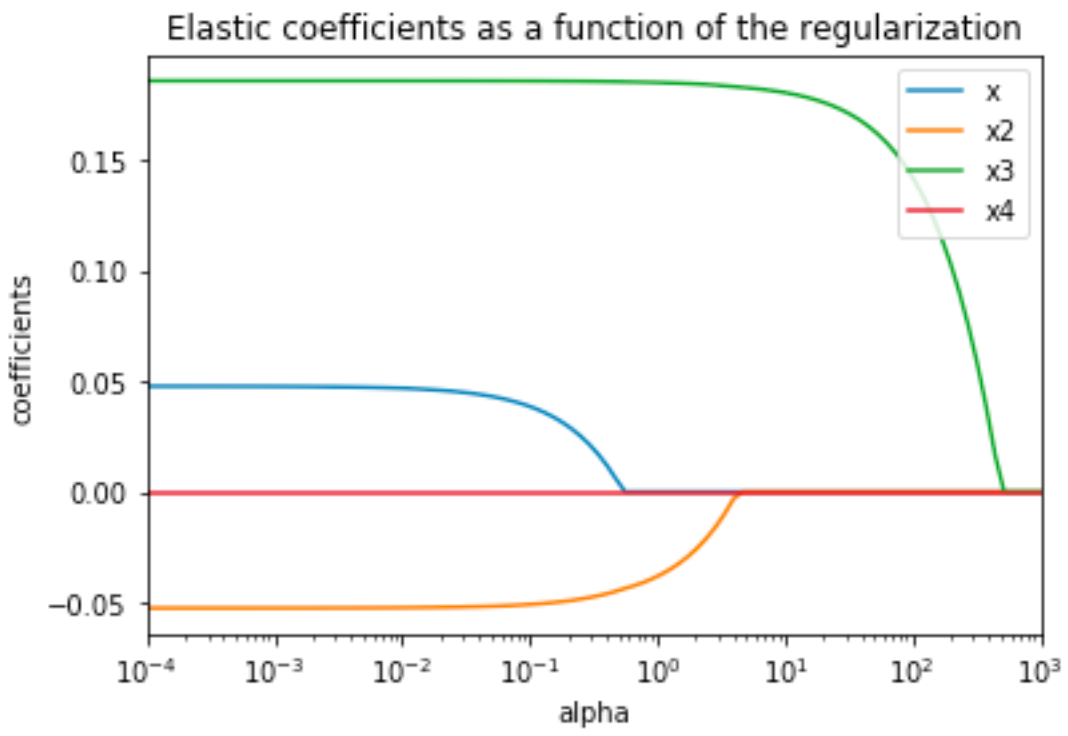
```
Lasso, alpha=0.01, intercept=0.24, coefficients=[ 0.04591006 -0.05221259  0.18560946  0.        ], rmse=13.006343902736498
Lasso, alpha=0.17783, intercept=0.21, coefficients=[ 0.01648733 -0.04703605  0.18533752  0.        ], rmse=13.012893764555317
Lasso, alpha=3.16228, intercept=-0.08, coefficients=[ 0.         -0.          0.18232391  0.        ], rmse=13.136403351426349
Lasso, alpha=56.23413, intercept=-0.05, coefficients=[ 0.         -0.          0.14271235  0.        ], rmse=15.48918780840939
Lasso, alpha=1000.0, intercept=0.04, coefficients=[-0.  0.  0.  0.], rmse=58.82740534381207
Linear Regression, intercept=0.24, coefficients=[ 0.04766321 -0.05252104  0.14850053  0.07425027], rmse=13.006323124645764

Linear Regression, intercept=0.24, coefficients=[ 0.04766321 -0.05252104  0.18562567], rmse=13.006323124645764
```

4.2.2. Regularized Least Squares




```
Ridge, alpha=0.01, intercept=0.17, coefficients=[ 0.0164893  -0.04107874  0.14741995  0.07370997], rmse=13.018155082313179
Ridge, alpha=0.17783, intercept=0.17, coefficients=[ 0.0164893  -0.04107874  0.14741995  0.07370997], rmse=13.018155082313179
Ridge, alpha=3.16228, intercept=0.17, coefficients=[ 0.0164893  -0.04107874  0.14741995  0.07370997], rmse=13.018155082313179
Ridge, alpha=56.23413, intercept=0.17, coefficients=[ 0.0164893  -0.04107874  0.14741995  0.07370997], rmse=13.018155082313179
Ridge, alpha=1000.0, intercept=0.17, coefficients=[ 0.0164893  -0.04107874  0.14741995  0.07370997], rmse=13.018155082313179
Linear Regression, intercept=0.24, coefficients=[ 0.04766321 -0.05252104  0.14850053  0.07425027], rmse=13.006323124645764
```

4.2.2. Regularized Least Squares




```
Elastic, alpha=0.01, intercept=0.24, coefficients=[ 0.04674513 -0.05235867  0.18561665  0.        ], rmse=13.006328837484837
Elastic, alpha=0.17783, intercept=0.22, coefficients=[ 0.03156567 -0.04964837  0.1854661   0.        ], rmse=13.008084572066696
Elastic, alpha=3.16228, intercept=-0.0, coefficients=[ 0.         -0.01232621  0.18348409  0.        ], rmse=13.084889385363315
Elastic, alpha=56.23413, intercept=-0.06, coefficients=[ 0.         -0.          0.16033351  0.        ], rmse=13.923383716994287
Elastic, alpha=1000.0, intercept=0.04, coefficients=[-0.  0.  0.  0.], rmse=58.82740534381207
Linear Regression, intercept=0.24, coefficients=[ 0.04766321 -0.05252104  0.14850053  0.07425027], rmse=13.006323124645764
```

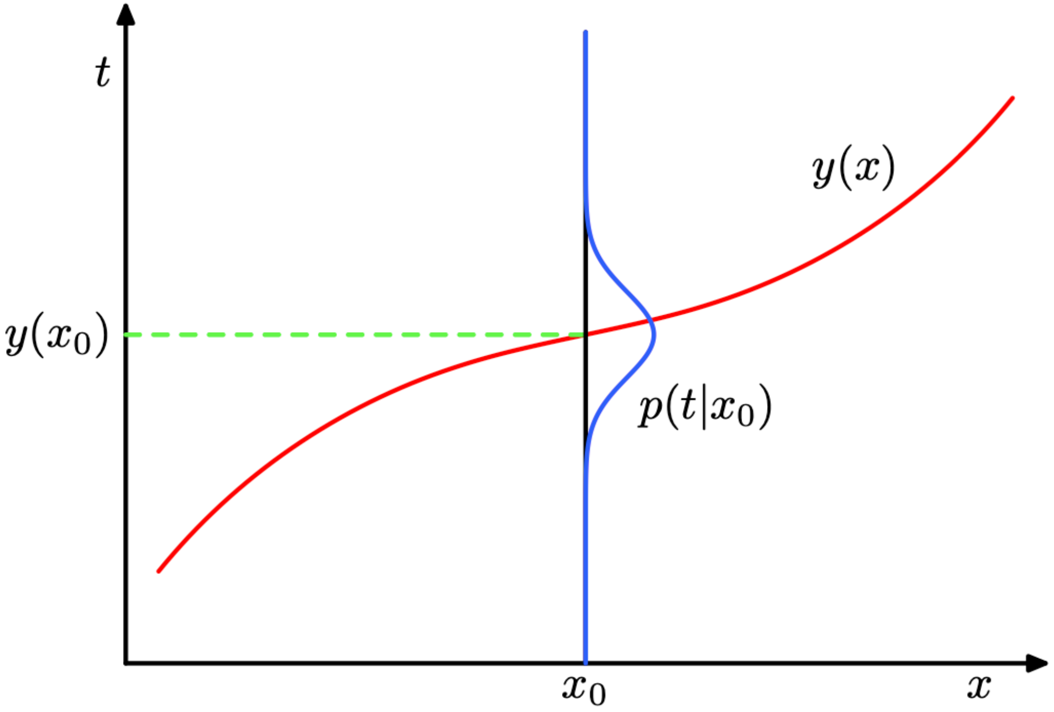
4.3. The Bias-Variance Decomposition

The average of expected loss also can be expressed as

$$\mathbb{E}(L) = \int\int (y - t)^2 p(x, t) \mathrm{d}x\mathrm{d}t. \tag{4-14}$$

If $\mathbb{E}(L)$ is minimized, then $\frac{\partial \mathbb{E}(L)}{\partial y} = 0$ solves $y = \mathbb{E}_t[t|x]$.

$\mathbb{E}_t(t|\boldsymbol{x})$ is the regression function that is the average over the ensemble of data sets and is the optimal prediction that is given by the conditional expectation.



4.3. The Bias-Variance Decomposition

We will use $h = \mathbb{E}_t[t|x]$ for the simplicity.

When the error function $(y-t)^2$ in Eq. 4-14 is expanded as

$$\begin{aligned}(y-t)^2 &= (y-h+h-t)^2 \\ &= (y-h)^2 + 2(y-h)(h-t) + (h-t)^2\end{aligned} \tag{4-15}$$

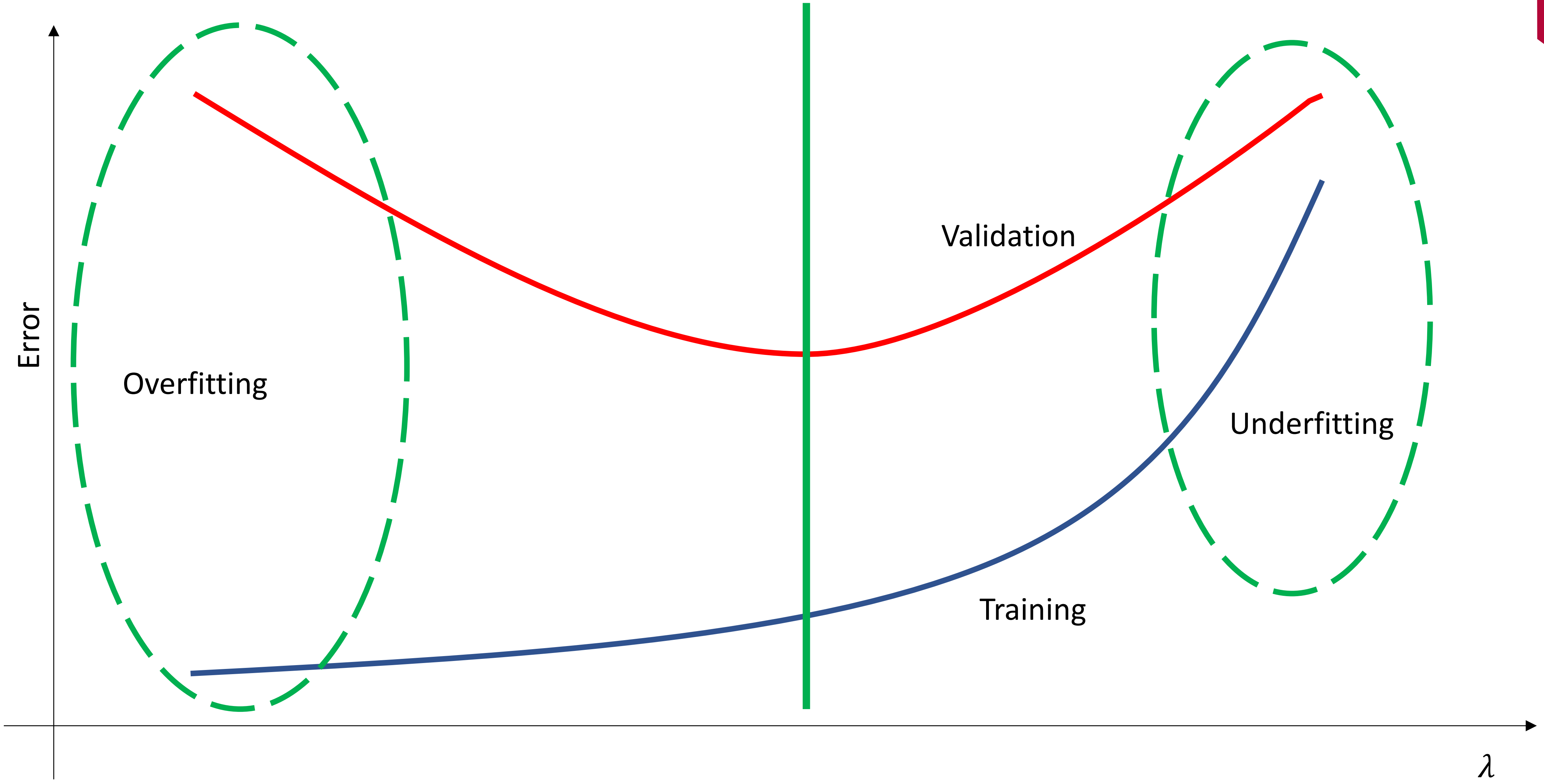
When Eq. (4-15) is substitute into Eq. (4-14), the second term vanishes and it becomes

$$\mathbb{E}(L) = \int\int (y-t)^2 p(x,t)\mathrm{d}x\mathrm{d}t = \int (y-h)^2\, p(x)\mathrm{d}x + \int\int (h-t)^2 p(x)\mathrm{d}x\mathrm{d}t \tag{4-16}$$

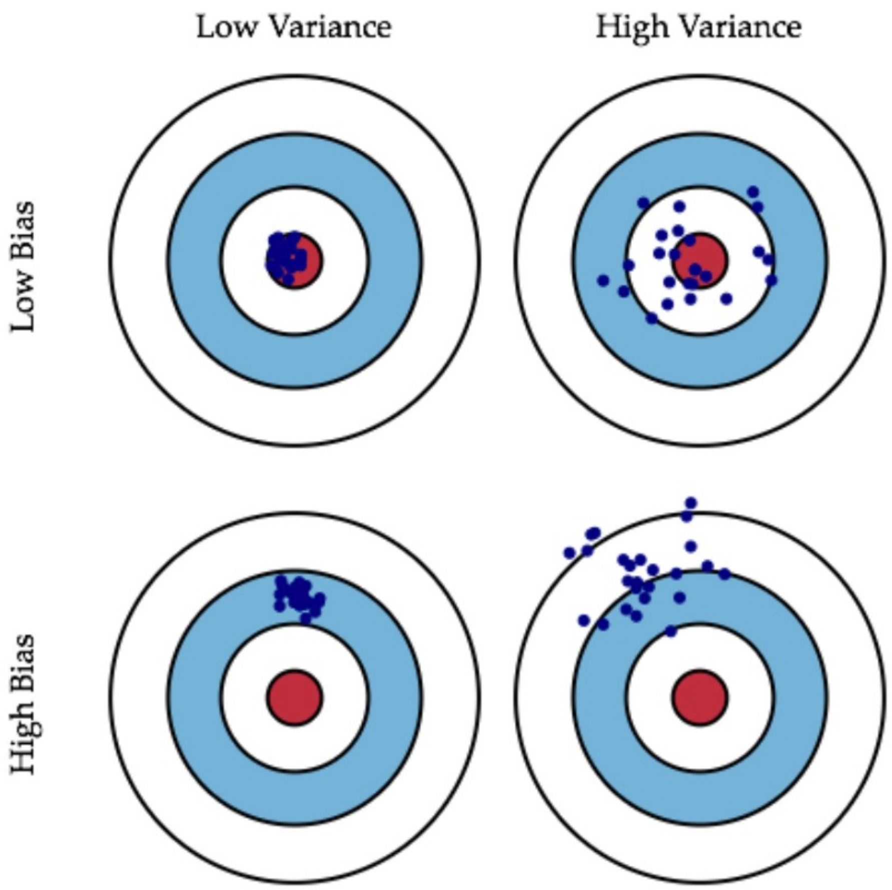
The first integration is the expectation of the data and it is the combination of $(\text{bias})^2$ and variance.

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

The Bias-Variance Decomposition



The Bias-Variance Decomposition



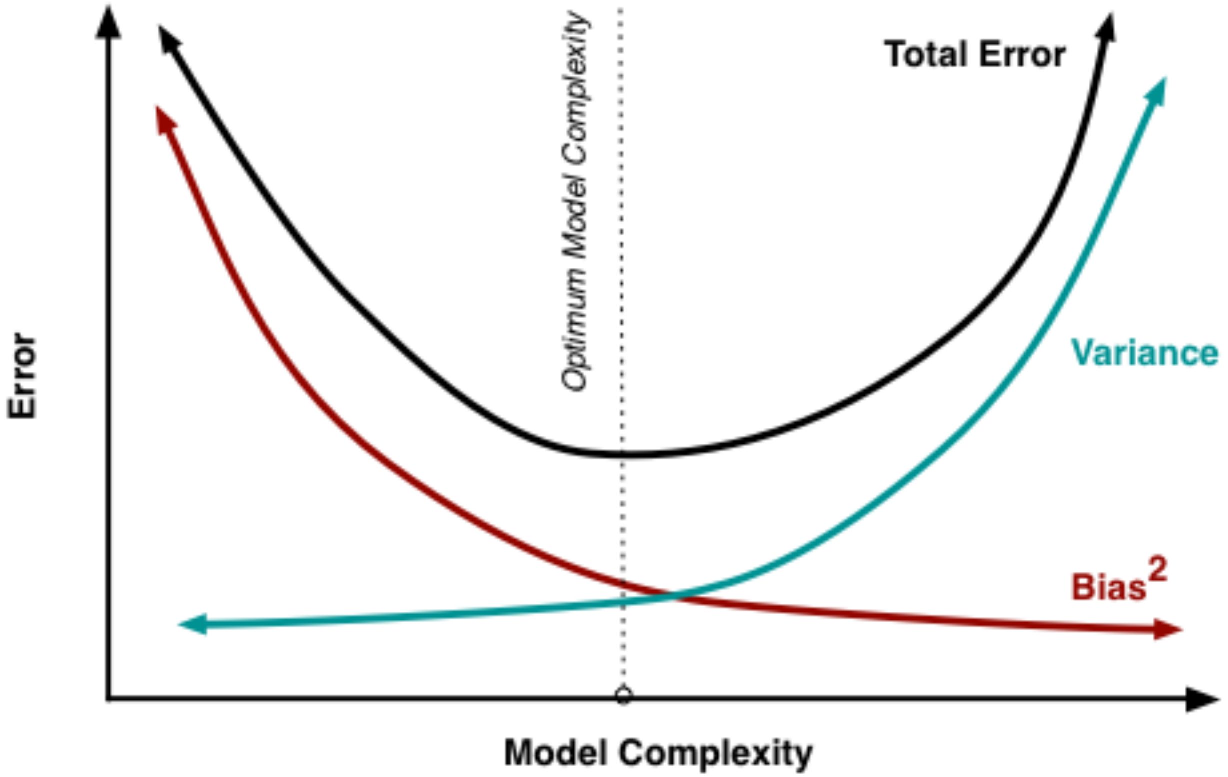


Figure: Graphical illustration of bias vs. variance

4.4. Conclusion

4.1. Linear Basis Function Models

- Assumptions:
  - Linearity: X and Y are in linear relationship.
  - Homoscedasticity: The variance of residual is the same for any features.
  - Independence: Examples are independent of each other.
  - Normality: All features are in Gaussian distribution.
- Optimization
  - GD, SGD, Mini-Batch GD

4.2. & 4.3. Regularization & Bias-Variance Decomposition

- The total errors are composite of Bias, Variance, and Noise.
  - Even though we want to optimize the model by reducing the errors,
    - Need to consider the trade-off between the bias and variance
    - Need to consider the trade-off between the overfitting and underfitting
    - Regularizations help to avoid the overfitting easily.
      - Lasso regularization also may be used for the feature selection or feature extraction.

4.4. Conclusion

Overall, The linear regression model implementation and interpretation are easy and straightforward. But,

- Simplicity: The model is too simple to capture the complexity of real data.
  - It is an easy start of modeling but hardly will be a final model.
  - It is essential to have different model.
- Assumptions are not realistic.
  - Need to do feature engineering, transformation, and more.
- Sensitivity: The model is sensitive to outliers as they are accounted in the estimation causing high variance and low bias.
  - Remove outliers.