

# Linear Models

Injung Kim  
Handong Global University

# Agenda

---



- Linear Regression
- Bias–Variance Trade–off
- Regularized Regression
- Regression using scikit–learn
- Logistic Regression

# Linear Regression

- Linear regression estimates the target variable by a linear combination of input variables

$$\mathbf{x} = (x_1, x_2, \dots, x_p)^T \Rightarrow y$$

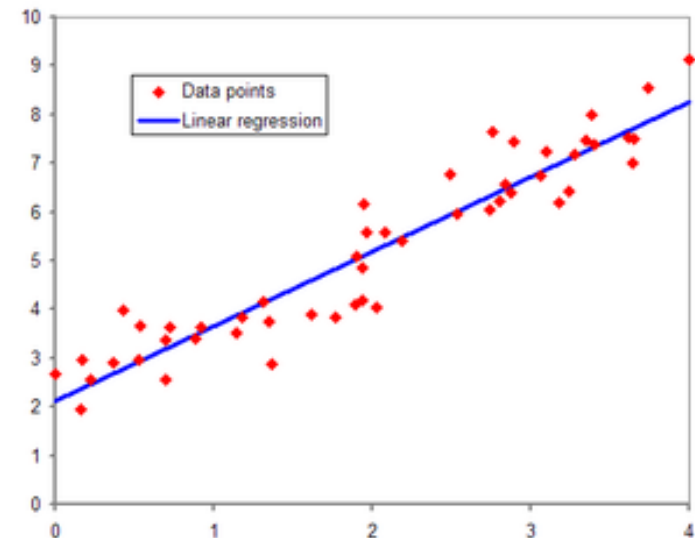
- Element-wise notation

$$\hat{y} = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + \dots + b$$

- Vector notation

$$y = \mathbf{w}\mathbf{x} + b$$

- $\mathbf{x} = (x_1, x_2, \dots, x_p)^T$
- $\mathbf{w} = (w_1, w_2, \dots, w_p)^T$



# Linear Regression

- Parameter estimation

- Training data

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$y = \mathbf{w}\mathbf{x} + b$$

- Find  $\mathbf{w}^*$  that minimizes loss  $L_{linear} = \frac{1}{2n} \sum_j (\hat{y}_j - y_j)^2$

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left[ \frac{1}{2n} \sum_j (\hat{y}_j - y_j)^2 \right]$$

- Solution

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

# Linear Regression

## ■ Notations

- $X$ : input vector,  $\theta$ : weight vector,  $y$ : output (scalar or vector)

## ■ Loss function

$$J(\theta) = (X\theta - y)^T(X\theta - y)$$

$$J(\theta) = ((X\theta)^T - y^T)(X\theta - y)$$

$$J(\theta) = (X\theta)^T(X\theta) - (X\theta)^T y - y^T(X\theta) + y^T y$$

- Since  $((X\theta)^T y)^T = y^T(X\theta)$

$$J(\theta) = (X\theta)^T(X\theta) - 2y^T(X\theta) + y^T y$$

$$J(\theta) = \theta^T X^T X \theta - 2y^T X \theta + y^T y$$

# Linear Regression

- Loss function (cont'd)

$$J(\theta) = \theta^T X^T X \theta - 2y^T X \theta + y^T y$$

- Derivative

$$\frac{\partial J(\theta)}{\partial \theta} = X^T X \theta + X^T X \theta - 2X^T y = 2X^T X \theta - 2X^T y$$

- Set derivative to zero

$$2X^T X \theta - 2X^T y = 0$$

$$X^T X \theta = X^T y$$

$$\theta = (X^T X)^{-1} X^T y$$

$$\frac{\partial (AX)}{\partial X} = A^T$$

$$\frac{\partial (X^T A)}{\partial X} = A$$

$$\frac{\partial (X^T X)}{\partial X} = 2X$$

$$\frac{\partial (X^T A X)}{\partial X} = AX + A^T X$$

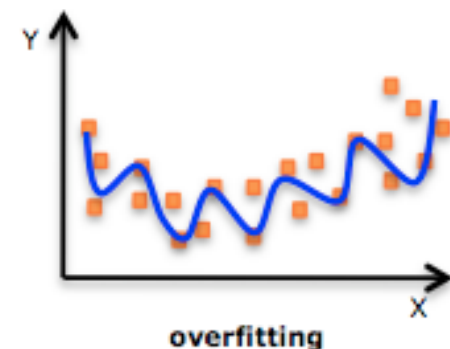
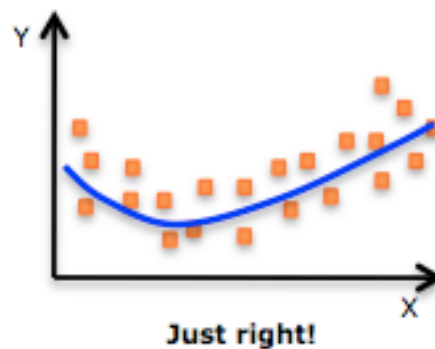
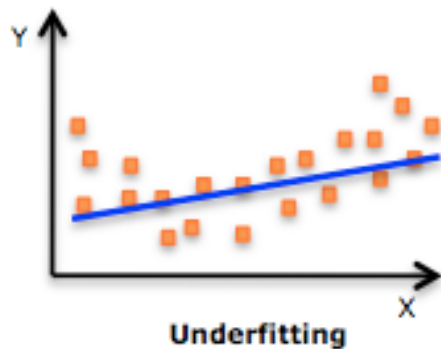
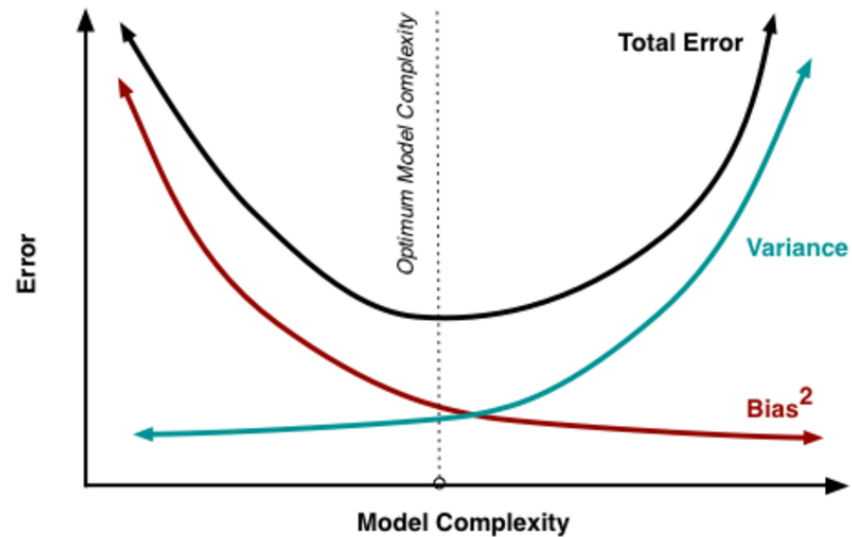
# Agenda

---



- Linear Regression
- Bias–Variance Trade-off
- Regularized Regression
- Regression using scikit-learn
- Logistic Regression

# Bias-Variance Trade-off





# Bias–Variance Trade–off

- Approximating  $Y$  by  $\hat{Y}$

- $Y = f(X) + \epsilon, \hat{Y} = \hat{f}(X)$

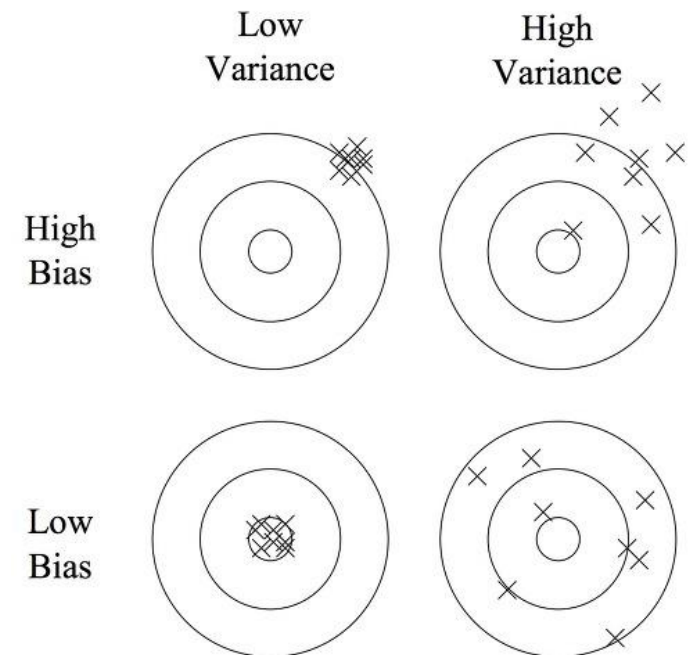
- $E(\hat{Y} - Y)^2 = E(\hat{f}(x) - Y)^2$   
 $= E[\hat{f}(X) - f(X)]^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \text{var}(\epsilon)$

- Bias  $\hat{f}(X) - f(X)$

- Error caused by inappropriate model (too simple model)
  - Related to under-fitting

- Variance  $(\hat{f}(x) - E[\hat{f}(x)])^2$

- Error caused by incorrect parameters (too complex model)
  - Related to over-fitting



# Bias–Variance Trade–off

- Approximating  $Y$  by  $\hat{Y}$ 
  - $Y = f(X) + \epsilon$ ,  $\hat{Y} = \hat{f}(X)$
- For a random variable  $X$ ,

$$\text{Var}[X] = \text{E}[X^2] - (\text{E}[X])^2 \qquad \text{E}[X^2] = \text{Var}[X] + (\text{E}[X])^2$$

$$\begin{aligned} \text{E}[(y - \hat{f})^2] &= \text{E}[(f + \epsilon - \hat{f})^2] \\ &= \text{E}[(f + \epsilon - \hat{f} + \text{E}[\hat{f}] - \text{E}[\hat{f}])^2] \\ &= \text{E}[(f - \text{E}[\hat{f}])^2] + \text{E}[\epsilon^2] + \text{E}[(\text{E}[\hat{f}] - \hat{f})^2] + 2\text{E}[(f - \text{E}[\hat{f}])\epsilon] + 2\text{E}[\epsilon(\text{E}[\hat{f}] - \hat{f})] + 2\text{E}[(\text{E}[\hat{f}] - \hat{f})(f - \text{E}[\hat{f}])] \\ &= (f - \text{E}[\hat{f}])^2 + \text{E}[\epsilon^2] + \text{E}[(\text{E}[\hat{f}] - \hat{f})^2] + 2(f - \text{E}[\hat{f}])\text{E}[\epsilon] + 2\text{E}[\epsilon]\text{E}[\text{E}[\hat{f}] - \hat{f}] + 2\text{E}[\text{E}[\hat{f}] - \hat{f}](f - \text{E}[\hat{f}]) \\ &= (f - \text{E}[\hat{f}])^2 + \text{E}[\epsilon^2] + \text{E}[(\text{E}[\hat{f}] - \hat{f})^2] \\ &= (f - \text{E}[\hat{f}])^2 + \text{Var}[y] + \text{Var}[\hat{f}] \\ &= \text{Bias}[\hat{f}]^2 + \text{Var}[y] + \text{Var}[\hat{f}] \\ &= \text{Bias}[\hat{f}]^2 + \sigma^2 + \text{Var}[\hat{f}] \end{aligned}$$

# Regularized Regression

- Ridge regression = linear regression +  $L_2$  regularization

$$L_{ridge} = \frac{1}{2n} \sum_j (\hat{y}_j - y_j)^2 + \alpha \|\mathbf{w}\|_2^2$$

- $L_2$  norm  $\|\mathbf{w}\|_2 = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$

- Lasso regression = linear regression +  $L_1$  regularization

$$L_{lasso} = \frac{1}{2n} \sum_j (\hat{y}_j - y_j)^2 + \alpha \|\mathbf{w}\|_1$$

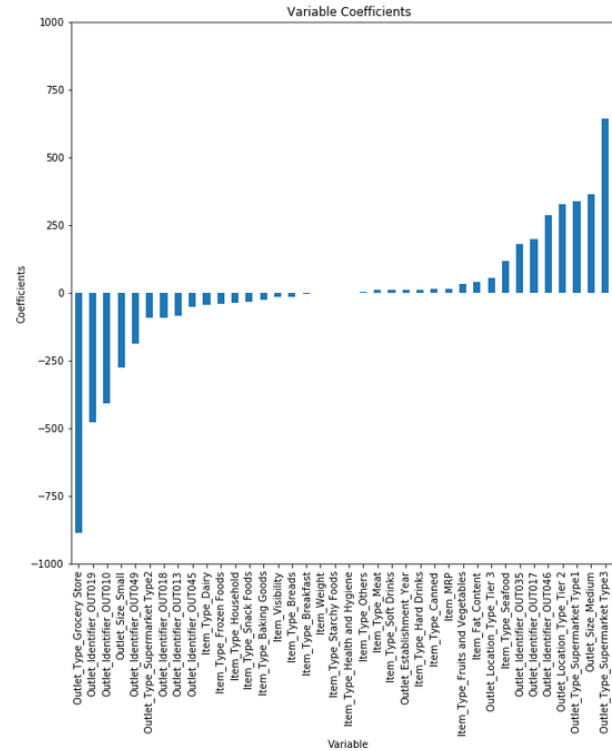
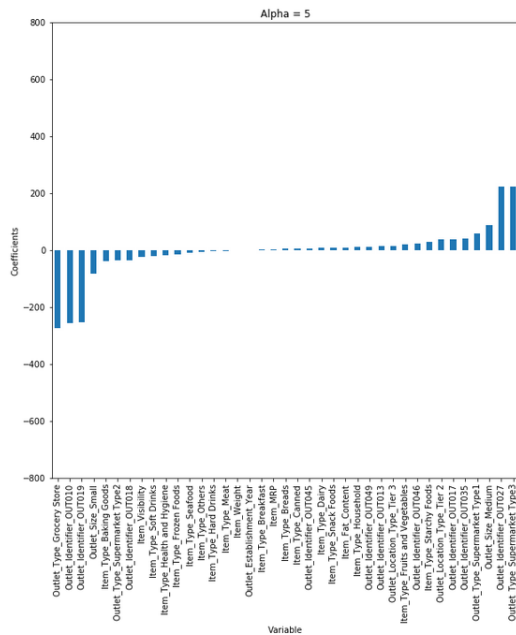
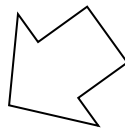
- $L_1$  norm  $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \dots + |w_n|$

- Elastic-Net = linear regression +  $L_1$  reg. +  $L_2$  reg.

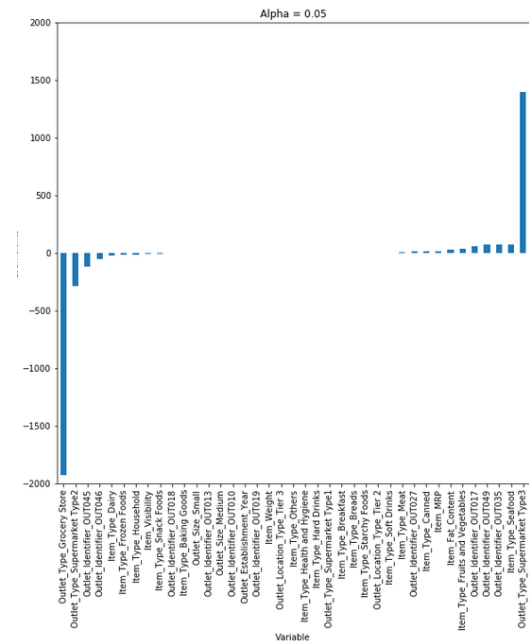
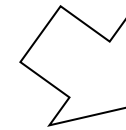
$$L_{elastic\_net} = \frac{1}{2n} \sum_j (\hat{y}_j - y_j)^2 + \alpha_1 \|\mathbf{w}\|_1 + \alpha_2 \|\mathbf{w}\|_2^2$$

# Ridge vs. Lasso

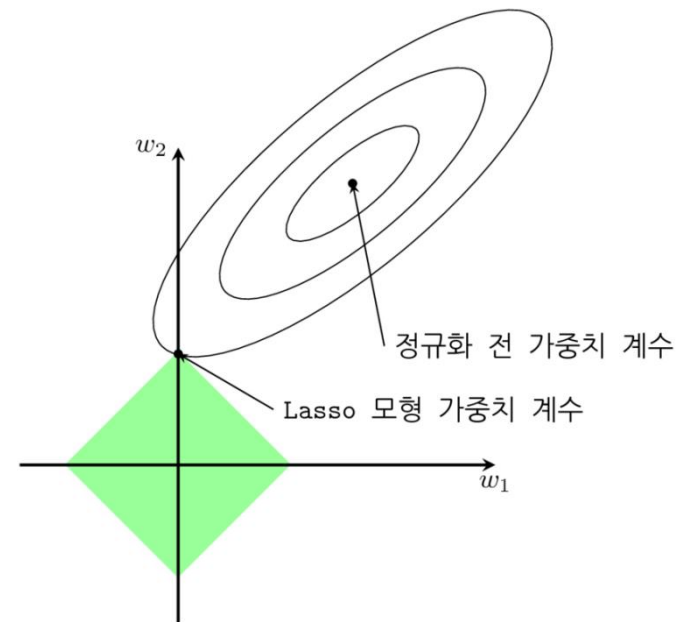
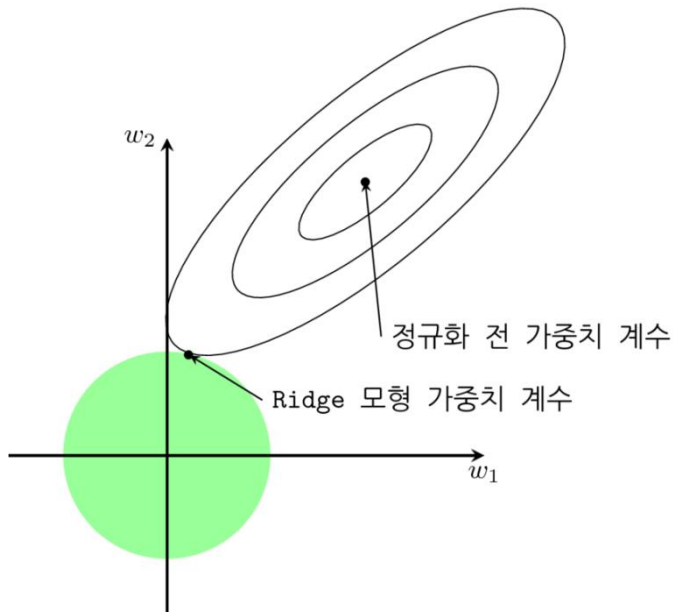
Ridge



Lasso



# Ridge vs. Lasso



# Regularized Regression in scikit-learn



- Linear regression
  - `from sklearn.linear_model import LinearRegression`
  - `lr = LinearRegression().fit(X_train, y_train)`
  - `y_hat = lr.predict(X_test)`
  
- Ridge regression
  - `from sklearn.linear_model import Ridge`
  - `ridge = Ridge(alpha=0.1).fit(X_train, y_train)` # by default,  $\alpha = 1$
  - `y_hat = ridge.predict(X_test)`
  
- Lasso regression
  - `from sklearn.linear_model import Lasso`
  - `lasso = Lasso(alpha=0.01).fit(X_train, y_train)` # by default,  $\alpha = 1$
  - `y_hat = lasso.predict(X_test)`
  
- ElasticNet
  - `from sklearn.linear_model import ElasticNet`
  - `elastic_net = ElasticNet(alpha=0.001, l1_ratio=0.5).fit(X_train, y_train)` # by default,  $\alpha = 1$ ,  $l1\_ratio = 0.5$
  - `y_hat = elastic_net.predict(X_test)`

# Regularized Regression in scikit-learn

## ■ Ridge

- $L_{ridge} = \sum ||\hat{y} - y||_2^2 + \alpha ||w||_2^2$
- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Ridge.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

## ■ Lasso

- $L_{lasso} = \frac{1}{2 * N} \sum ||\hat{y} - y||_2^2 + \alpha ||w||_1$
- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Lasso.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html)

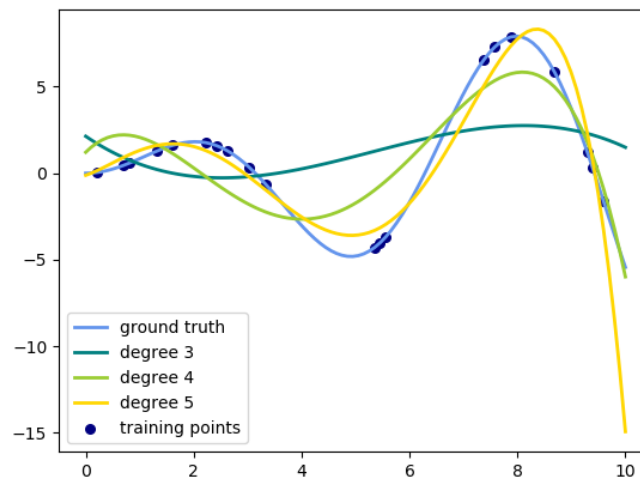
## ■ Elastic-Net

- $L_{elastic\_net} = \frac{1}{2 * N} \sum ||\hat{y} - y||_2^2 + \alpha \cdot l1\_ratio ||w||_1 + \frac{1}{2} \alpha \cdot (1 - l1\_ratio) ||w||_2^2$
- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.ElasticNet.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html)

# Polynomial Regression

- Polynomial regression is a form of regression analysis in which the dependent variable  $y$  is modelled as an  $n^{\text{th}}$  degree polynomial in  $x$ .

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon$$



Ref. [https://scikit-learn.org/stable/auto\\_examples/linear\\_model/plot\\_polynomial\\_interpolation.html](https://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html)



# Regression in scikit-learn



- Linear regression
  - `from sklearn.linear_model import LinearRegression`
  - `lr = LinearRegression().fit(X_train, y_train)`
  - `y_hat = lr.predict(X_test)`
  
- Ridge regression
  - `from sklearn.linear_model import Ridge`
  - `ridge = Ridge(alpha=0.1).fit(X_train, y_train)` # by default,  $\alpha = 1$
  - `y_hat = ridge.predict(X_test)`
  
- Lasso regression
  - `from sklearn.linear_model import Lasso`
  - `lasso = Lasso(alpha=0.01).fit(X_train, y_train)` # by default,  $\alpha = 1$
  - `y_hat = lasso.predict(X_test)`
  
- ElasticNet
  - `from sklearn.linear_model import ElasticNet`
  - `elastic_net = ElasticNet(alpha=0.001, l1_ratio=0.5).fit(X_train, y_train)` # by default,  $\alpha = 1$ ,  $l1\_ratio = 0.5$
  - `y_hat = elastic_net.predict(X_test)`

# Regression in scikit-learn

## ■ Ridge

- $L_{ridge} = \sum \|\hat{y} - y\|_2^2 + \alpha \|\mathbf{W}\|_2^2$

- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Ridge.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

## ■ Lasso

- $L_{lasso} = \frac{1}{2n} \sum \|\hat{y} - y\|_2^2 + \alpha \|\mathbf{W}\|_1$

- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Lasso.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html)

## ■ Elastic-Net

- $L_{elastic\_net} = \frac{1}{2n} \sum \|\hat{y} - y\|_2^2 + \alpha \cdot l1\_ratio \|\mathbf{W}\|_1 + \frac{1}{2} \alpha \cdot (1 - l1\_ratio) \|\mathbf{W}\|_2^2$

- [http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.ElasticNet.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html)

# Regression in scikit-learn



- Import packages
  - `import numpy as np`
  - `import sklearn as sk`
  - `from sklearn.model_selection import train_test_split`
  - `import matplotlib.pyplot as plt`
- Prepare datasets
  - `X, y = mglearn.datasets.load_extended_boston()`
  - `X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)`

# Regression in scikit-learn

## ■ Create and train Regressor

- `from sklearn.linear_model import Ridge`
- `ridge = Ridge().fit(X_train, y_train)` # by default,  $\alpha = 1$

## ■ Check coefficients

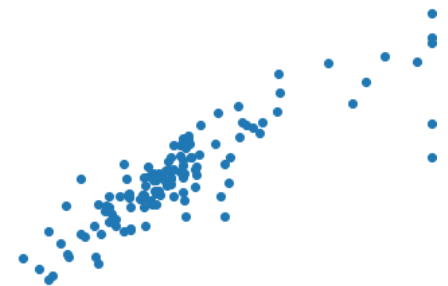
- `print("ridge.coef_: {}".format(ridge.coef_))` #  $w$
- `print("ridge.intercept_: {}".format(ridge.intercept_))` #  $b$
- `plt.plot(ridge.coef_, 'v', label="Ridge  $\alpha=1$ ")`

## ■ Predict

- `y_hat = ridge.predict(X_test)`
- `plt.plot(y_test, y_hat, 'o')`

## ■ Evaluation

- `print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))`
- `print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))`



# Agenda

---



- Linear Regression
- Bias–Variance Trade–off
- Regularized Regression
- Regression using scikit–learn
- Logistic Regression

# Logistic Regression

- Logistic regression = Linear regression + logisistic function

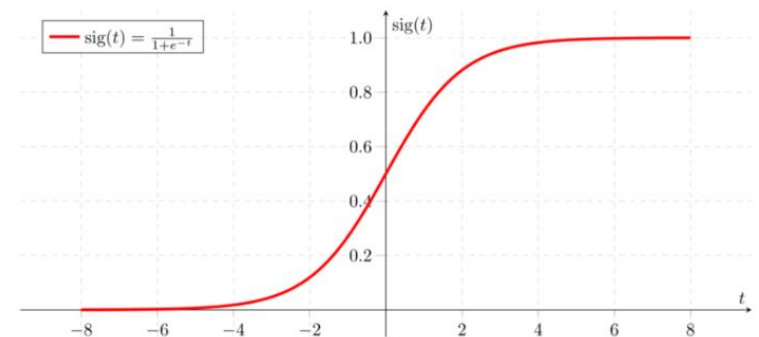
- $Logistic(x) = Sigmoid(x) = \frac{1}{1+\exp(-x)}$

- Discrimination function

- $f_j(x) = logistic(\sum_i w_{ji}x_i + b_j)$ 
    - $f_j(x)$  has range  $[0,1]$

- Parameter estimation

- $W^* = argmin_W L(\hat{Y}, Y; W)$
  - $L(\cdot)$ : a loss function such as cross entropy or MSE



# Loss Functions

- Mean squared error

$$E_{MSE} = \frac{1}{2} \frac{\sum_N (\hat{y}_t - y_t)^2}{N}$$

- Cross entropy (with softmax activation)

- Softmax activation:  $\hat{y}_t = \frac{\exp(net_t)}{\sum_t \exp(net_t)}$

$$E_{CE} = - \sum_N y_t \log(\hat{y}_t)$$

- $y_i \in \{0,1\}$ : label (only  $y_{true} = 1$ )

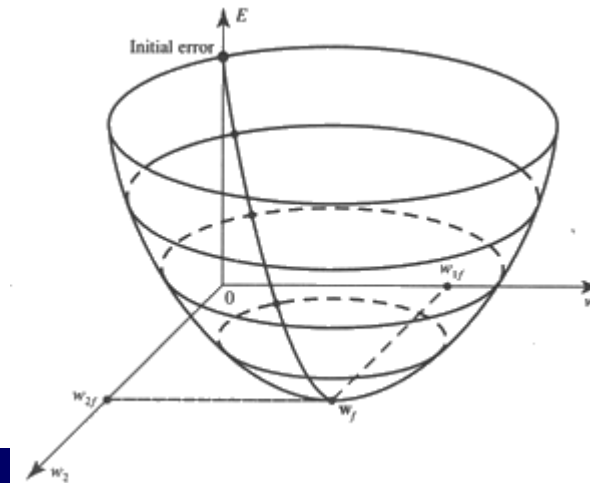
# Gradient-based Learning

- Given current weights  $W$ , the gradient gives a direction in which increases the error most rapidly
  - Gradient of  $E(W)$  with respect to weight

$$\frac{\nabla E(W)}{\nabla W} = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_i}, \dots, \frac{\partial E}{\partial w_M} \right)$$

- Update rule

$$W^{t+1} = W^t - \eta \cdot \frac{\nabla E(W)}{\nabla W^t}$$





# LogisticRegression in scikit-learn



- Import LogisticRegression
  - `from sklearn.linear_model import LogisticRegression`
- Instance creation and training
  - `logi_reg = LogisticRegression().fit(X_train,y_train)`
- Checking coefficients and intercept
  - `print("logi_reg.coef_: ", logi_reg.coef_)` # W
  - `print("logi_reg.intercept_:", logi_reg.intercept_)` # b
- Applying to new data
  - `y_pred = logi_reg.predict(X_test)`



Thank you  
for your attention!

