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

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

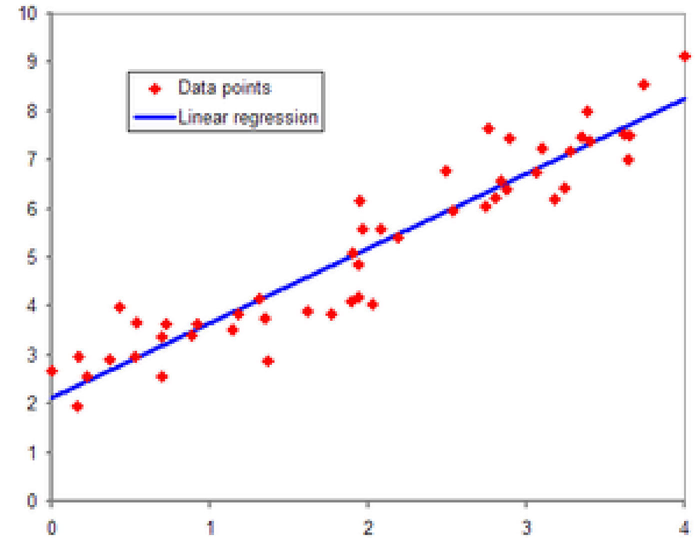
  - Element-wise notation

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

  - Vector notation

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

    - $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)^T$
    - $\boldsymbol{w} = (w_1, w_2, \ldots, x_p)^T$



Linear Regression

- Parameter estimation
  - Training data

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

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

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

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

  - Solution

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

Linear Regression

- Notations
  - X: input vector, θ: 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

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

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

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

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

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

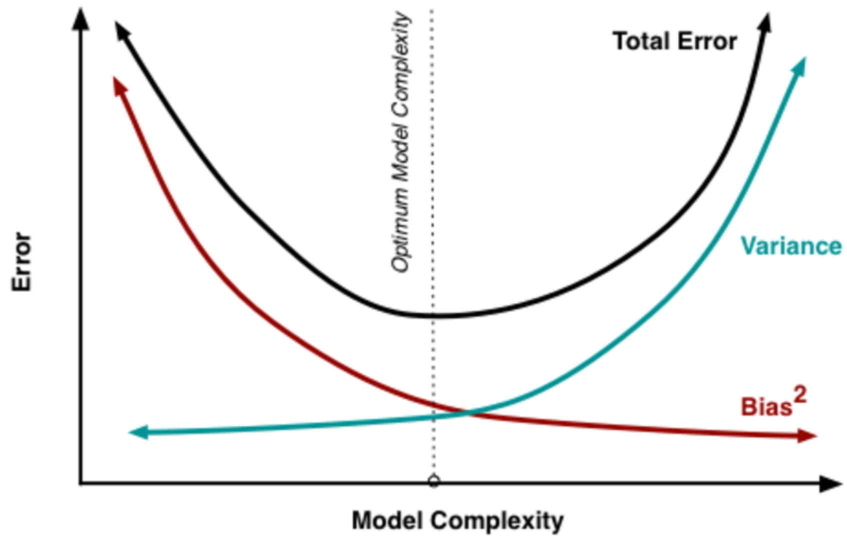
$$\frac{\partial (X^T X)}{\partial X} = 2X$$

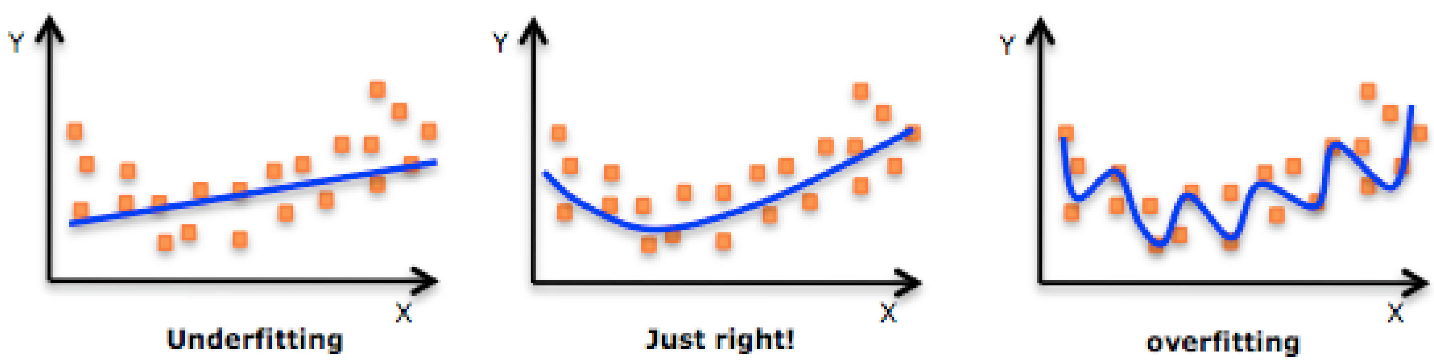
$$\frac{\partial (X^T AX)}{\partial X} = AX + A^T X$$

Agenda

- Linear Regression
- <u>Bias-Variance Trade-off</u>
- Regularized Regression
- Regression using scikit-learn
- Logistic Regression

Bias-Variance Trade-off



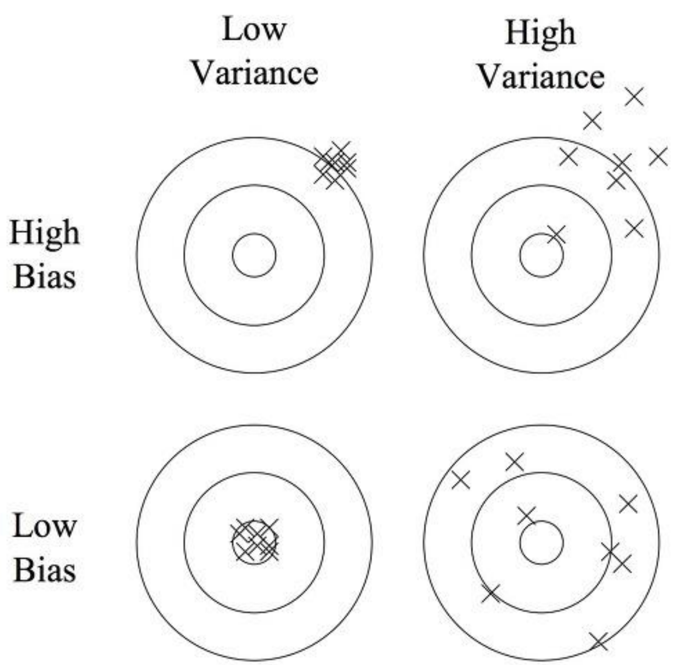


Bias-Variance Trade-off

- Approximating Y by $\hat{Y}$
  - $Y = f(X) + \epsilon,\ \hat{Y} = \hat{f}(X)$
  - $E(\hat{Y} - Y)^2 = E(\hat{f}(x) - Y)^2$
    $$= E[\hat{f}(X) - f(X)]^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + var(\epsilon)$$

- Bias $\hat{f}(X) - f(X)$
  - Error caused by inappropriate model (too simple model)
  - Related to under-fitting

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

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

$$\begin{aligned}
\mathrm{E}\left[(y-\hat{f})^2\right] &= \mathrm{E}\left[(f+\varepsilon-\hat{f})^2\right] \\
&= \mathrm{E}\left[(f+\varepsilon-\hat{f}+\mathrm{E}[\hat{f}]-\mathrm{E}[\hat{f}])^2\right] \\
&= \mathrm{E}\left[(f-\mathrm{E}[\hat{f}])^2\right] + \mathrm{E}[\varepsilon^2] + \mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right] + 2\,\mathrm{E}\left[(f-\mathrm{E}[\hat{f}])\varepsilon\right] + 2\,\mathrm{E}\left[\varepsilon(\mathrm{E}[\hat{f}]-\hat{f})\right] + 2\,\mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})(f-\mathrm{E}[\hat{f}])\right] \\
&= (f-\mathrm{E}[\hat{f}])^2 + \mathrm{E}[\varepsilon^2] + \mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right] + 2(f-\mathrm{E}[\hat{f}])\,\mathrm{E}[\varepsilon] + 2\,\mathrm{E}[\varepsilon]\,\mathrm{E}\left[\mathrm{E}[\hat{f}]-\hat{f}\right] + 2\,\mathrm{E}\left[\mathrm{E}[\hat{f}]-\hat{f}\right](f-\mathrm{E}[\hat{f}]) \\
&\boxed{= (f-\mathrm{E}[\hat{f}])^2 + \mathrm{E}[\varepsilon^2] + \mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right]} \\
&= (f-\mathrm{E}[\hat{f}])^2 + \mathrm{Var}[y] + \mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2 + \mathrm{Var}[y] + \mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2 + \sigma^2 + \mathrm{Var}\left[\hat{f}\right]
\end{aligned}$$

Regularized Regression

- Ridge regression = linear regression + L_2 regularization

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

  - L_2norm $\|\boldsymbol{w}\|_2 = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$

- Lasso regression = linear regression + L_1 regularization

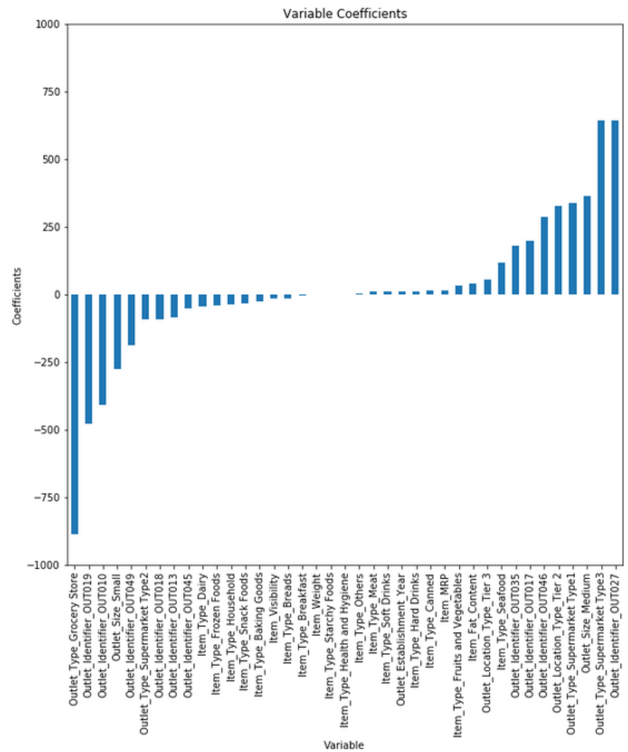
$$L_{lasso} = \frac{1}{2n}\sum_j (\hat{y}_j - y_j)^2 + \boldsymbol{\alpha}\|\boldsymbol{w}\|_1$$

  - L_1norm $\|\boldsymbol{w}\|_1 = |w_1| + |w_2| + \cdots + |w_n|$

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

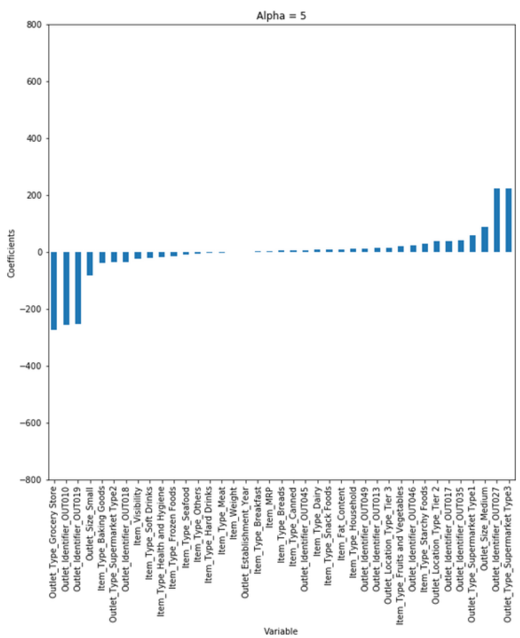
$$L_{elastic\_net} = \frac{1}{2n}\sum_j (\hat{y}_j - y_j)^2 + \boldsymbol{\alpha_1}\|\boldsymbol{w}\|_1 + \boldsymbol{\alpha_2}\|\boldsymbol{w}\|_2^2$$

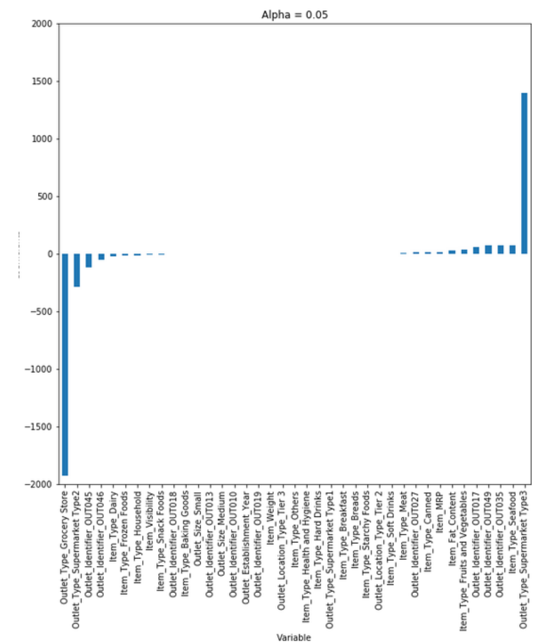
Ridge vs. Lasso



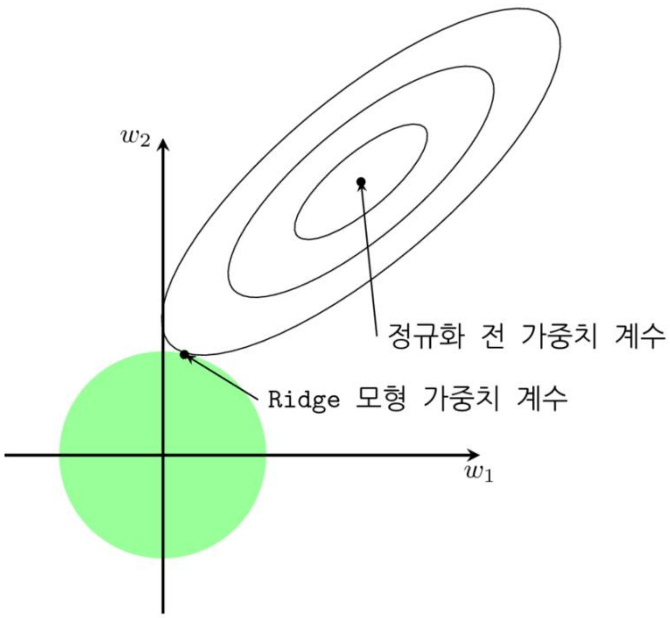
Ridge

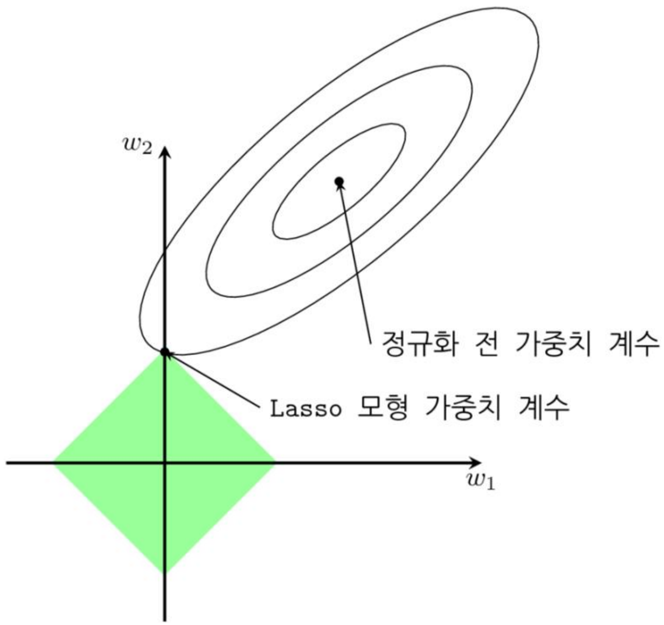
Lasso





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)

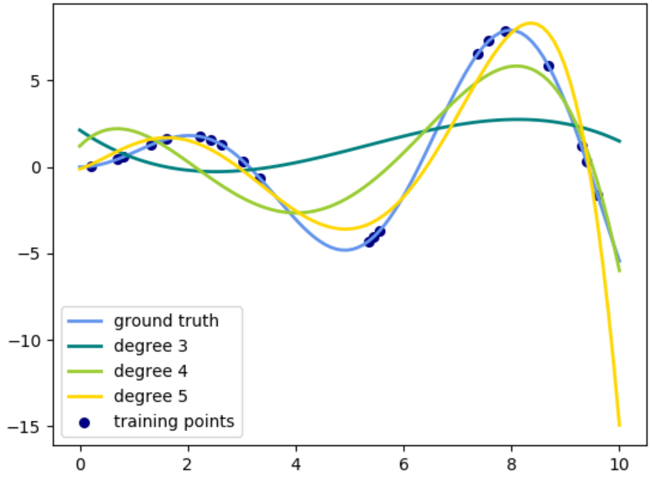
Regularized Regression in scikit-learn

- 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
- 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
- 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

Polynomial Regression

- **Polynomial regression** is a form of regression analysis in which the dependent variable y is modelled as an n^{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

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
  - $L_{ridge} = \sum \|\hat{y} - y\|_2^2 + \alpha \|\boldsymbol{W}\|_2^2$
  - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
- Lasso
  - $L_{lasso} = \frac{1}{2n} \sum \|\hat{y} - y\|_2^2 + \alpha \|\boldsymbol{W}\|_1$
  - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
- Elastic-Net
  - $L_{elastic\_net} = \frac{1}{2n} \sum \|\hat{y} - y\|_2^2 + \alpha \cdot l1\_ratio \|\boldsymbol{W}\|_1 + \frac{1}{2} \alpha \cdot (1 - l1\_ratio) \|\boldsymbol{W}\|_2^2$
  - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html

Regression in scikit-learn

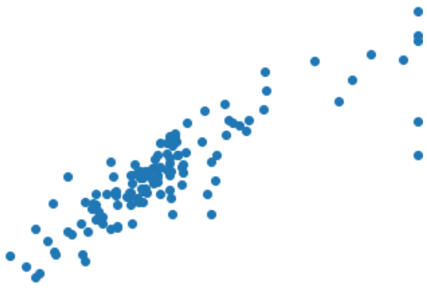
- Import packages
  - import numpy as np
  - import sklearn as sk
  - from sklearn.model_selection import train_test_split

  - 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)))

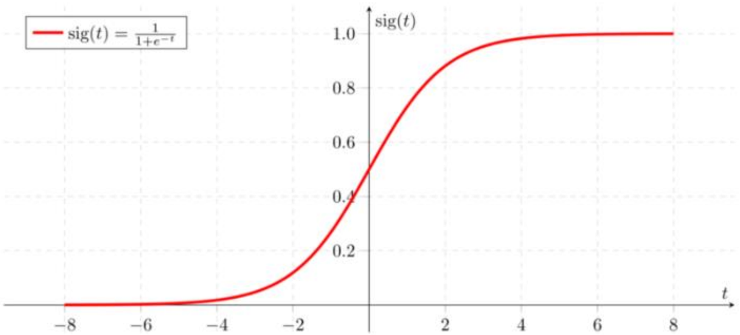
Agenda

- Linear Regression
- Bias-Variance Trade-off
- Regularized Regression
- Regression using scikit-learn
- <u>Logistic Regression</u>

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 $y_{true} = 1$)

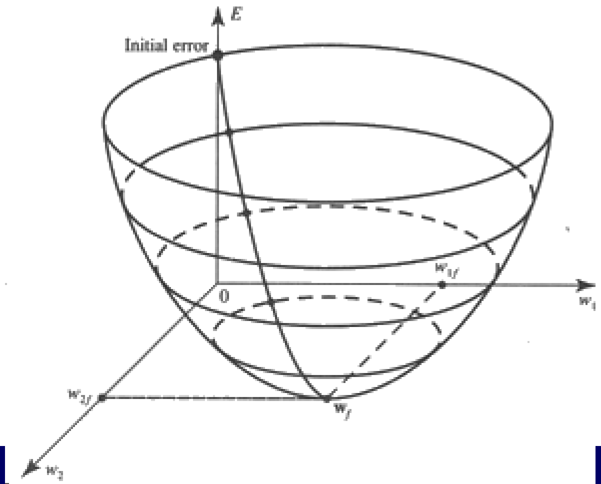
Gradient-based Learning

- 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)

Q&A

Thank you
for your attention!