KUBIG
Data Science and Machine Learning

Week 3.
Preprocessing and Optimization
Model Evaluation

Content

1. Loss Function Review

Loss Function Review

- Regression

$$Y_i \overset{ind}{\sim} N(\mu_i(\mathbf{X}_i),\ \sigma) \qquad \text{where} \quad E[Y_i] = \mu_i(\mathbf{X}_i)$$

$$\mu_i(\mathbf{X}_i) = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}$$

$$= \boldsymbol{\beta}^T \mathbf{X}_i$$

Loss Function Review

- Logistic Regression

$$Y_i \overset{ind}{\sim} \text{Bernoulli}(\pi_i(\mathbf{X}_i)) \quad \text{where} \quad E[Y_i] = \pi_i(\mathbf{X}_i)$$

$$logit(\pi_i(\mathbf{X}_i)) = \log\left(\frac{\pi_i(\mathbf{X}_i)}{1-\pi_i(\mathbf{X}_i)}\right) = \beta_0 + \beta_1 X_{1\mathrm{i}} + \beta_2 X_{2\mathrm{i}} + \cdots + \beta_p X_{p\mathrm{i}}$$

$$= \boldsymbol{\beta}^T \mathbf{X}_i$$

Loss Function Review

- Estimation

$$\underset{\boldsymbol{\beta}}{argmin}\, L[\tau(\theta), T(X)] \Leftrightarrow \underset{\boldsymbol{\beta}}{argmax}\, L(\boldsymbol{\beta}, \sigma)$$

- Regression → SSE
- Logistic Regression → Cross Entropy

Loss Function Review

- Cross Entropy

$$CE_i = -\sum_{k=1}^{C} y_{ik} \log \pi_i(k)$$

y_{ik} : the k^{th} value in y_i

$\pi_i(k)$: the probability for the i^{th} observation to belonging to Class k

Loss Function Review

- For C = 3 (number of Class)

Class 1: y_i =(1,0,0) Class 2: y_i =(0,1,0) Class 3: y_i =(0,0,1)

⇒ One-Hot encoding

$$\sum_{i=1}^{n} CE_i = -\sum_{k=1}^{C} [y_{i1} \log \pi_i(1) + y_{i2} \log \pi_i(2) + y_{i3} \log \pi_i(3)]$$

Loss Function Review

- IF Class 2,

Class 1: y_i =(1,0,0)   Class 2: y_i =(0,1,0)   Class 3: y_i =(0,0,1)

$$CE_i = -[0 * \log \pi_i(1) + 1 * \log \pi_i(2) + 0 * \log \pi_i(3)]$$

$$= -\log \pi_i(2)$$

$$\Rightarrow \text{IF } \pi_i(2) = 1, \text{ then } CE_i = -\log 1 = 0 \text{ (minimum Loss)}$$

Loss Function Review

- Categorical Cross Entropy

$$CE_i = -\sum_{k=1}^{C} y_{ik} \log \pi_i(k)$$

- Likelihood of ?

$$Y_i \overset{ind}{\sim} Multi(\pi_1, \cdots, \pi_k)$$

Loss Function Review

- For C = 2 (number of Class)

$$\mathrm{P}(Y_i = 1|\mathbf{X}_i) = \pi_i(\mathbf{X}_i) = \frac{e^{\boldsymbol{\beta}^T\mathbf{X}_i}}{1 + e^{\boldsymbol{\beta}^T\mathbf{X}_i}} = \frac{1}{1 + e^{-\boldsymbol{\beta}^T\mathbf{X}_i}}$$ (sigmoid function)

Loss Function Review

- For C = 2 (number of Class)

$$P(Y = 1|\mathbf{X}_i) = \pi(\mathbf{X}_i) = \frac{e^{\boldsymbol{\beta}^T\mathbf{X}_i}}{1 + e^{\boldsymbol{\beta}^T\mathbf{X}_i}} = \frac{1}{1 + e^{-\boldsymbol{\beta}^T\mathbf{X}_i}}$$ (sigmoid function)

- For C = K > 2 (number of Class)

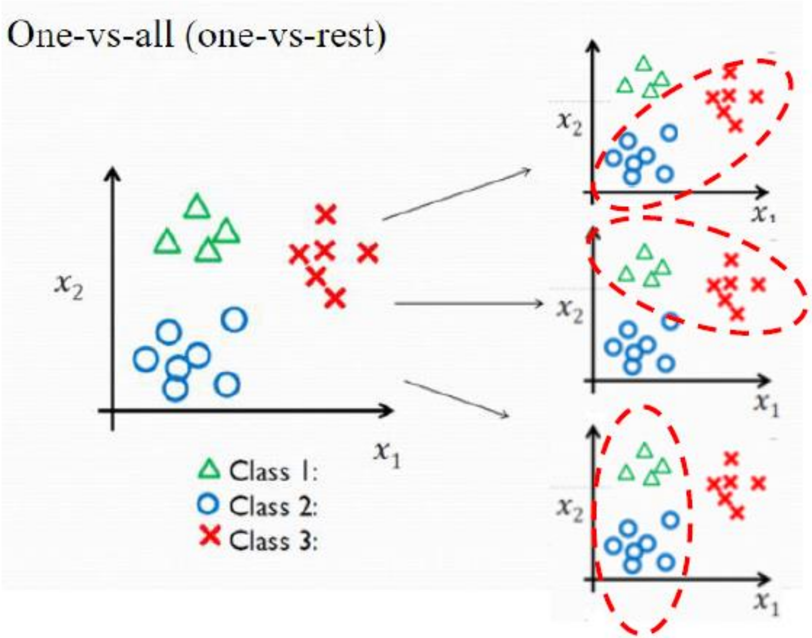
$$P(Y = l|\mathbf{X}_i) = \pi_l(\mathbf{X}_i) = \frac{e^{\boldsymbol{\beta}_l^T\mathbf{X}_i}}{\sum_{c=1}^{K} e^{\boldsymbol{\beta}_c^T\mathbf{X}_i}}$$ (softmax function)

Loss Function Review

Binary Classification	Binomial Distribution	Binary Cross Entropy	Sigmoid Function
Multi-Class Classification	Multinomial Distribution	Categorical Cross Entropy	Softmax Function

Loss Function Review

- One-Vs-Rest



Loss Function Review

- One-Vs-Rest

```
LogisticRegression(solver='sag', multi_class='multinomial')
```

```
'ovr'
```

2. Optimization of Loss Function

Optimization

- Loss function

$$L[\tau(\theta), T(X)] = L[Y,\ \hat{Y}]$$

$$Regression \quad \Rightarrow \quad L[Y,\ \hat{Y}] = \sum_{i}^{n} (Y_i - \hat{Y}_i)^2$$

$$Classification \quad \Rightarrow \quad L\left[Y,\ \hat{Y}\right] = -\sum_{i}^{n}\sum_{k}^{C} Y_i \ log\hat{\pi}_i(k)$$

Optimization

- Machine “Learning”

$$\underset{\boldsymbol{\theta}}{argmin}\ L\left[Y,\ \hat{Y}\right] = \widehat{\boldsymbol{\theta}}$$

$$\Rightarrow\ Optimization$$

Optimization

- Logistic Regression

$$L[Y,\ \hat{Y}] = -\sum_{i=1}^{n}[\,y_i \log \pi_i + (1 - y_i)\log(1 - \pi_i)]$$

$$= -\sum_{i=1}^{n}[\,y_i(\boldsymbol{\beta}^T \mathbf{X}_i) - \log(1 + \exp(\boldsymbol{\beta}^T \mathbf{X}_i)]$$

Optimization

- Logistic Regression

$$L[Y,\ \hat{Y}] = -\sum_{i=1}^{n}[\,y_i(\boldsymbol{\beta}^T\mathbf{X}_i) - \log(1+\exp(\boldsymbol{\beta}^T\mathbf{X}_i)]$$

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{argmin}\ L[Y,\ \hat{Y}]$$

$\Rightarrow$ Can you solve it?

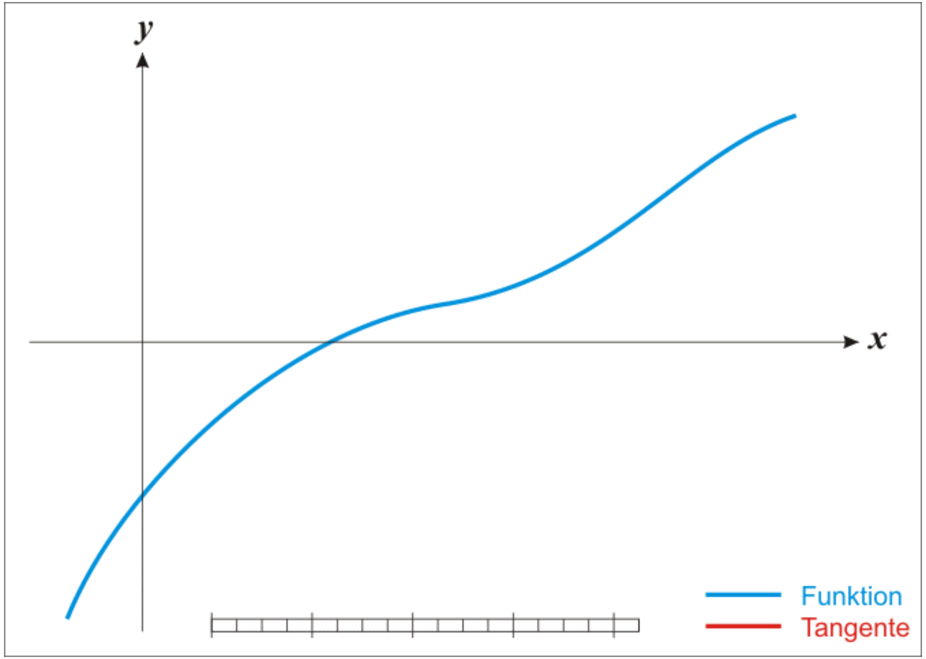
Optimization

- Optimization often can be rewritten as solving equations.

  ex) Normal equation

- Some problems do not have an explicit solution and a numerical approach should be exploited.

Newton-Raphson Method



Newton-Raphson Method

- Linear approximation (1st order Taylor Expansion)

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) = 0$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$\Rightarrow \quad \theta^{(\mathrm{t}+1)} = \theta^{(\mathrm{t})} - \frac{f(\theta^{(\mathrm{t})})}{f'(\theta^{(\mathrm{t})})}$$

Newton-Raphson Method

1. Initialize $\theta^{(0)} = \theta_0$ which can be arbitrary on the domain of the function

2. Update for $\mathrm{t} = 0, 1, 2, 3, \cdots$

$$\theta^{(\mathrm{t}+1)} = \theta^{(\mathrm{t})} - \frac{f\left(\theta^{(\mathrm{t})}\right)}{f'(\theta^{(\mathrm{t})})}$$

until

$$\left|\theta^{(\mathrm{t}+1)} - \theta^{(\mathrm{t})}\right| < \epsilon$$

for small $\epsilon > 0$

Newton-Raphson Method

- Quadratic approximation (2nd order Taylor Expansion)

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

$$\frac{\partial}{\partial x} f(x) \approx f'(x_0) + f''(x_0)(x - x_0) = 0$$

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)} \quad \Rightarrow \quad \theta^{(\mathrm{t}+1)} = \theta^{(\mathrm{t})} - \frac{f'(\theta^{(\mathrm{t})})}{f''(\theta^{(\mathrm{t})})}$$

Newton-Raphson Method

- Quadratic approximation (2nd order Taylor Expansion)

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_0) + L'(\boldsymbol{\theta}_0)^T(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T L''(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Newton-Raphson Method

- Quadratic approximation (2nd order Taylor Expansion)

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_0) + \nabla L(\boldsymbol{\theta}_0)^T(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

where

$$\nabla L(\boldsymbol{\theta}_0) = \frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

$$\mathbf{H}(\boldsymbol{\theta}_0) = \frac{\partial^2}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} L(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

Newton-Raphson Method

- Updating equation is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \mathbf{H}^{-1}(\boldsymbol{\theta}^{(t)})\nabla L(\boldsymbol{\theta}^{(t)})$$

$$= \boldsymbol{\theta}^{(t)} - \mathbf{H}^{-1}(\boldsymbol{\theta}^{(t)})\frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} L(\boldsymbol{\theta}^{(t)})$$

$$cf. \quad \theta^{(t+1)} = \theta^{(t)} - \frac{f'(\theta^{(t)})}{f''(\theta^{(t)})}$$

Newton-Raphson Method

$$Loss[\boldsymbol{\beta}] = -\sum_{i=1}^{n}[\,y_i(\boldsymbol{\beta}^T\mathbf{X}_i) - \log(1+\exp(\boldsymbol{\beta}^T\mathbf{X}_i))]$$

$$\nabla L(\boldsymbol{\beta}) = \frac{\partial}{\partial\boldsymbol{\beta}}L(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\left[y_i\mathbf{X}_i - \frac{\exp(\boldsymbol{\beta}^T\mathbf{X}_i)}{1+\exp(\boldsymbol{\beta}^T\mathbf{X}_i)}\mathbf{X}_i\right]$$

$$\mathbf{H}(\boldsymbol{\beta}) = \frac{\partial^2}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T}L(\boldsymbol{\beta}) = \sum_{i=1}^{n}\left[\left(\frac{\exp(\boldsymbol{\beta}^T\mathbf{X}_i)}{1+\exp(\boldsymbol{\beta}^T\mathbf{X}_i)}\right)\left(\frac{1}{1+\exp(\boldsymbol{\beta}^T\mathbf{X}_i)}\right)\mathbf{X}_i\mathbf{X}_i^T\right]$$

Newton-Raphson Method

$$Loss[\boldsymbol{\beta}] = -\sum_{i=1}^{n}[y_i(\boldsymbol{\beta}^T\mathbf{X}_i) - \log(1 + \exp(\boldsymbol{\beta}^T\mathbf{X}_i)]$$

Update

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \mathbf{H}^{-1}\left(\boldsymbol{\beta}^{(t)}\right)\nabla L\left(\boldsymbol{\beta}^{(t)}\right)$$

until

$$||\boldsymbol{\beta}^{t+1} - \boldsymbol{\beta}^{t}|| < \epsilon \qquad \text{for small } \epsilon > 0$$

Newton-Raphson Method

					Solvers
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

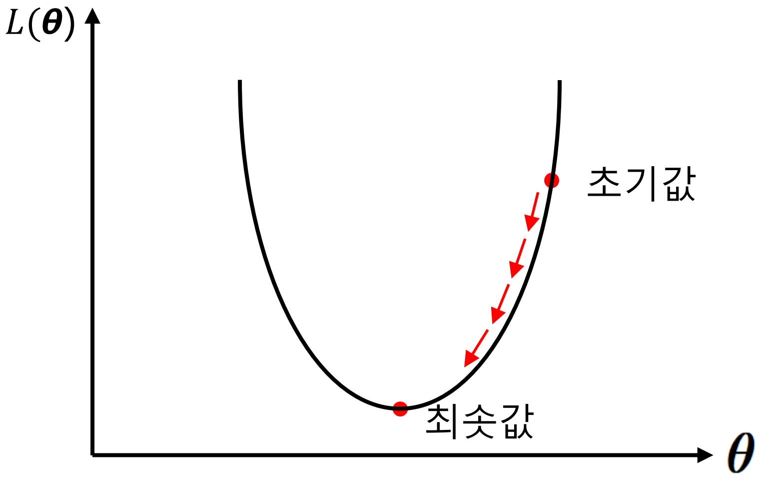
Gradient Descent

- Newton-Raphson is expensive to compute due to the computation of the inverse of Hessian.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \mathbf{H}^{-1}(\boldsymbol{\theta}^{(t)})\nabla L(\boldsymbol{\theta}^{(t)})$$

$$\Rightarrow \quad \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)}\,\nabla L(\boldsymbol{\theta}^{(t)})$$

Gradient Descent



Gradient Descent

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)}\, \nabla L(\boldsymbol{\theta}^{(t)})$$

or

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta\, \nabla L(\boldsymbol{\theta}^{(t)})$$

```
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)
```

```
eta = learning_schedule(epoch * m + i)
theta = theta - eta * gradients
```

Batch Gradient Descent

- Regression → SSE

$$\nabla L(\boldsymbol{\beta}) = \frac{\partial}{\partial \boldsymbol{\beta}} L(\boldsymbol{\beta}) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

- Logistic Regression → Cross Entropy

$$\nabla L(\boldsymbol{\beta}) = \frac{\partial}{\partial \boldsymbol{\beta}} L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left[y_i \mathbf{X}_i - \frac{\exp(\boldsymbol{\beta}^T \mathbf{X}_i)}{1 + \exp(\boldsymbol{\beta}^T \mathbf{X}_i)} \mathbf{X}_i \right]$$

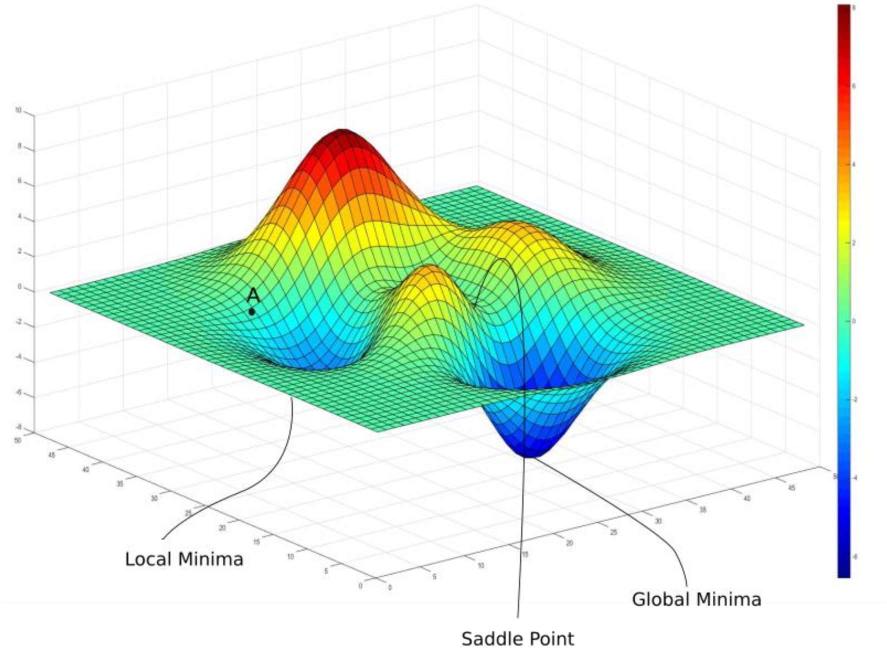
Steepest Descent

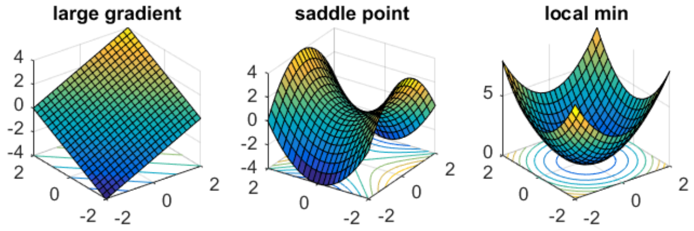
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} L(\boldsymbol{\theta}^{(t)}) \quad \Leftrightarrow \quad (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) \propto -\frac{\partial L(\boldsymbol{\theta}^{(t)})}{\partial \boldsymbol{\theta}^{(t)}}$$

$$\frac{L(\boldsymbol{\theta}^{(t+1)})}{\partial \eta} = [\frac{\partial L(\boldsymbol{\theta}^{(t+1)})}{\partial \boldsymbol{\theta}^{(t+1)}}]^T \frac{\partial \boldsymbol{\theta}^{(t+1)}}{\partial \eta} = -[\frac{\partial L(\boldsymbol{\theta}^{(t+1)})}{\partial \boldsymbol{\theta}^{(t+1)}}]^T \frac{\partial L(\boldsymbol{\theta}^{(t)})}{\partial \boldsymbol{\theta}^{(t)}} \overset{\text{set}}{=} 0$$

$\Rightarrow \quad \boldsymbol{\theta}^{(t+2)}$ and $\boldsymbol{\theta}^{(t+1)}$ are orthogonal

Stochastic Gradient Descent





Stochastic Gradient Descent

$$\underbrace{\boxed{\theta^{(1)}}\boxed{\theta^{(2)}}\boxed{\theta^{(3)}}\boxed{\quad\cdots\cdots\quad}\boxed{\theta^{(n-1)}}\boxed{\theta^{(n)}}}_{\text{1 epoch}}$$

- Stochastic (Randomness) → Shuffle the data

Stochastic Gradient Descent

```python
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(loss='squared_loss',
                       max_iter=50, tol=-np.infty, penalty=None, eta0=0.1, random_state=42)
sgd_reg.fit(X, y.ravel())

sgd_reg.intercept_, sgd_reg.coef_
```

```
(array([4.16782089]), array([2.72603052]))
```

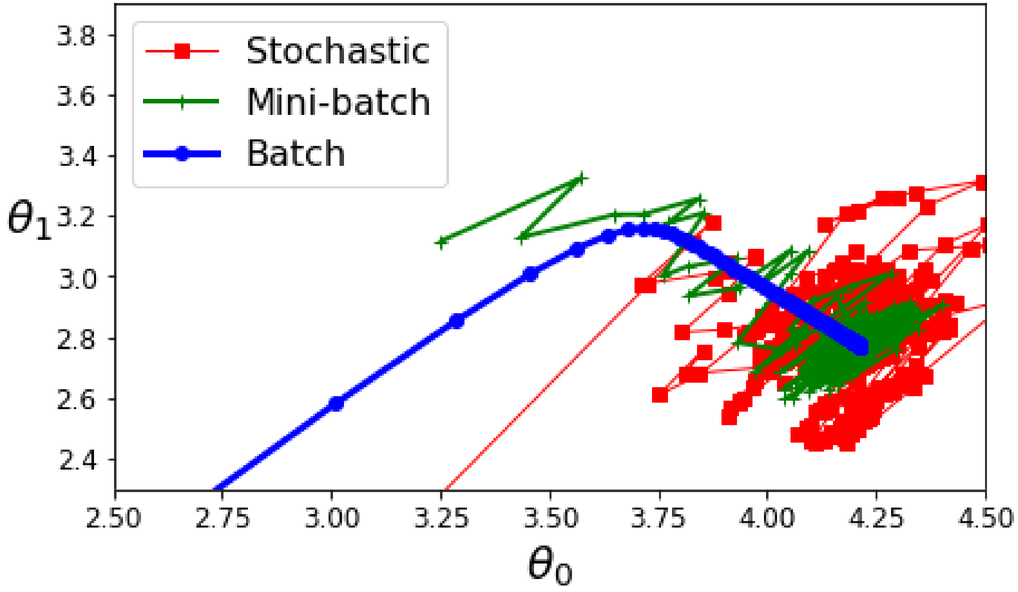
```python
from sklearn.linear_model import SGDClassifier

sdg_cls = SGDClassifier(loss='log',
                       max_iter=50, tol=-np.infty, penalty=None, eta0=0.1, random_state=42))
```

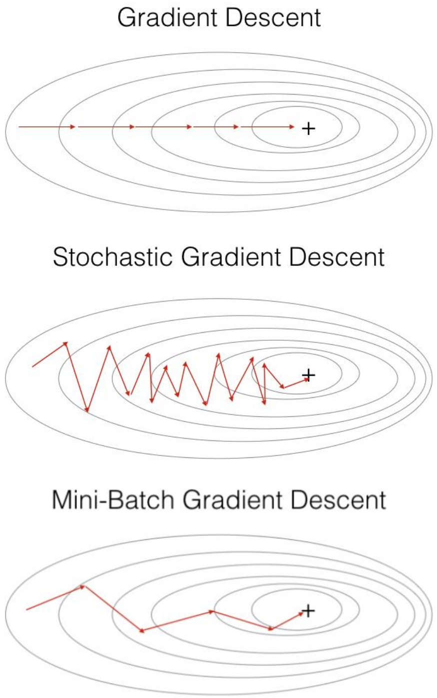
Mini-Batch Gradient Descent

$$\underbrace{\boxed{\theta^{(1)}}\boxed{\theta^{(2)}}\boxed{\cdots\cdots}\boxed{\theta^{(k)}}}_{\text{1 epoch}}$$

- k batches have p data → n = k x p

Mini-Batch Gradient Descent



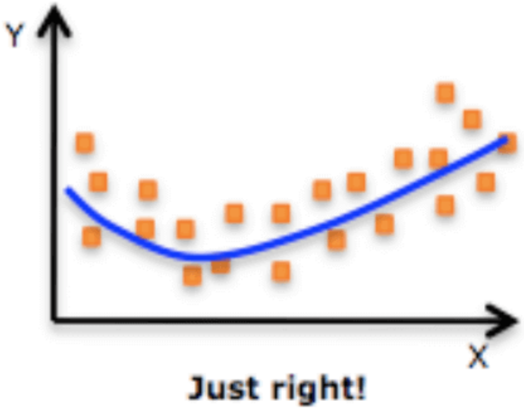


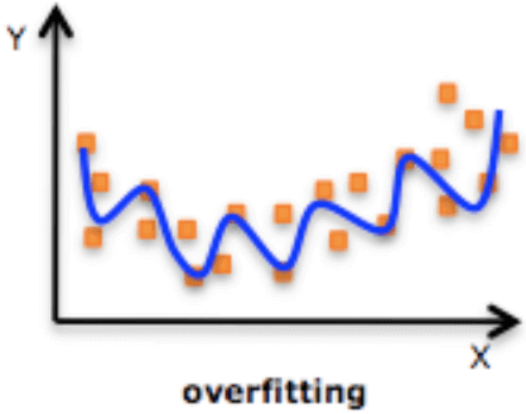
Gradient Descent

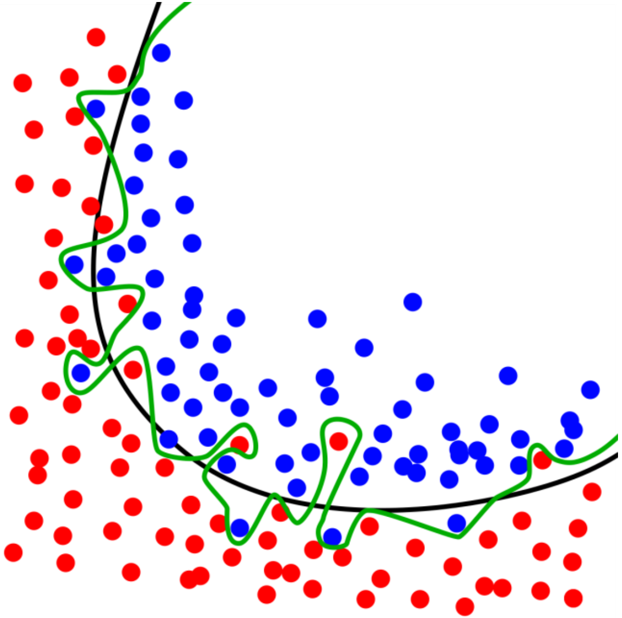
- What about $\eta^{(t)}$? $\rightarrow$ in Deep Learning

Overfitting and the Need for Model Evaluation

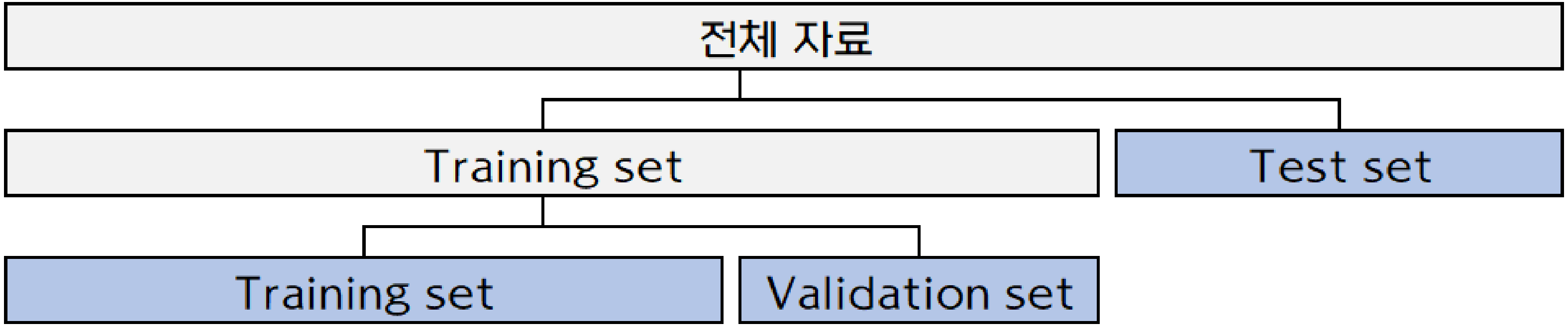
Overfitting





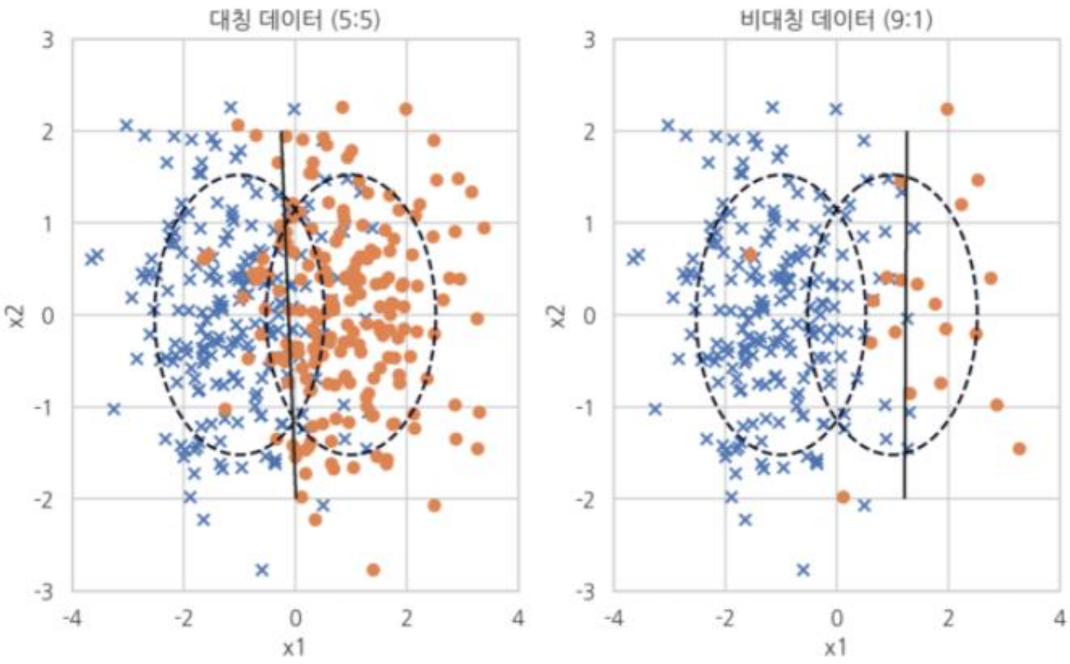


Cross Validation

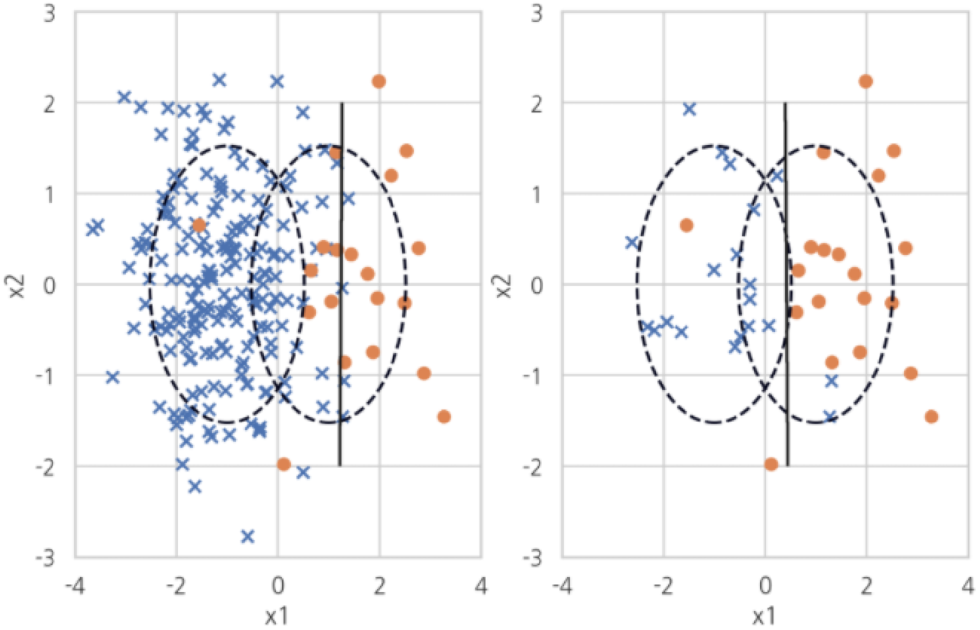


Preprocessing for Model Evaluation

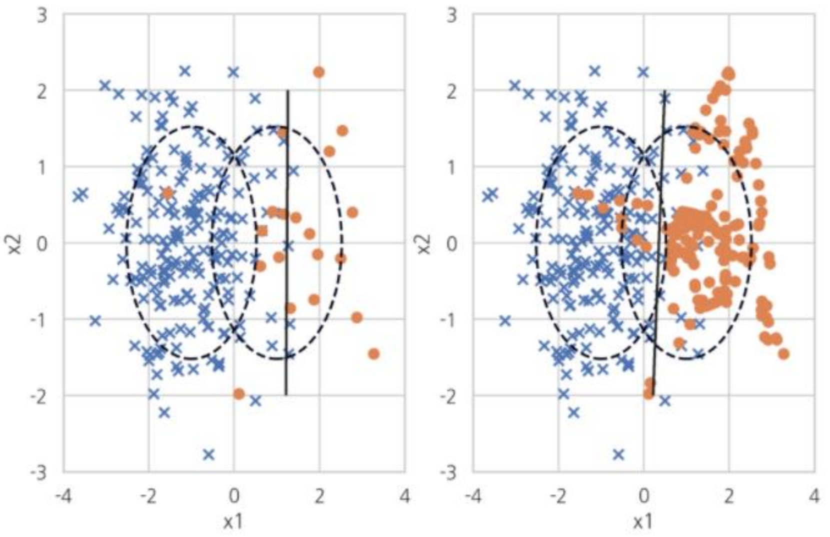
Handling Imbalanced Data

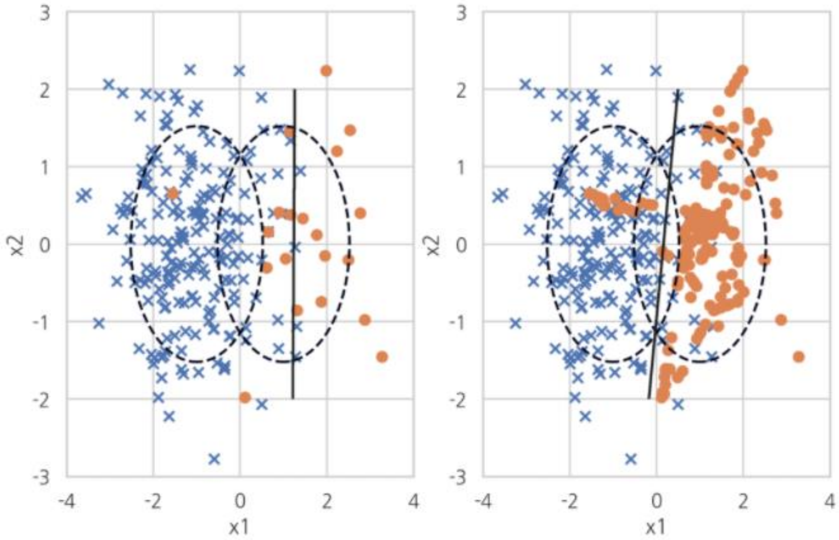


Undersampling



Oversampling: SMOTE and ADASYN

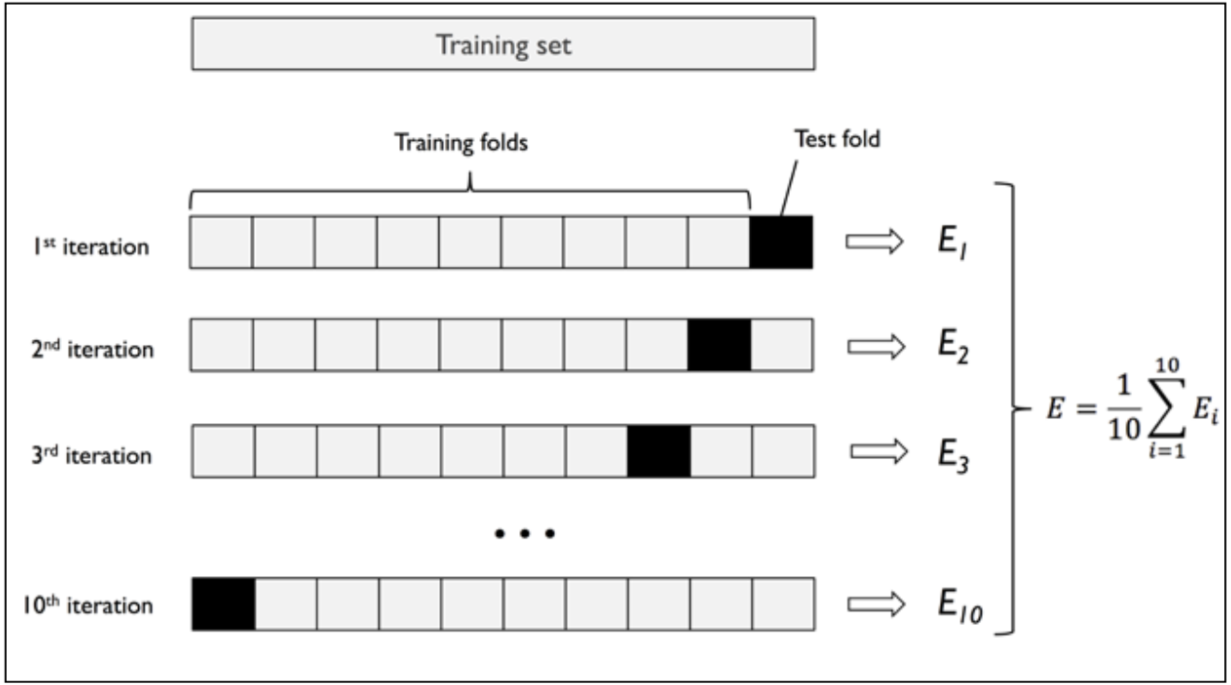




GridSearch

K-fold Cross Validation

- K = 10



K-fold Cross Validation

```
### Pipeline Streaming: 표준화 → PCA → Logistic Regression ###
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=4),
                        LogisticRegression(random_state=1, solver='lbfgs'))  # 적용 순서대로 나열
pipe_lr.fit(X_train, y_train)           # 표준화(fit → transform) → PCA(fit → transform) → Logistic Reg fit의 순서로 처리
```

K-fold Cross Validation

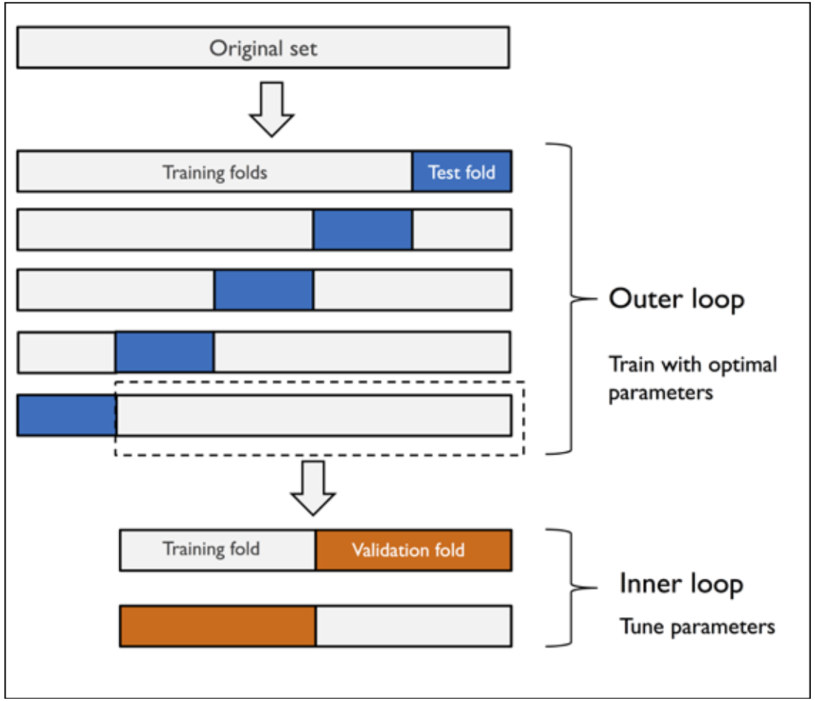
```
[ ]  ### K-fold cross-validation using pipeline ###
     from sklearn.model_selection import cross_val_score
     scores = cross_val_score(estimator=pipe_lr, X=X_train, y=y_train, cv=10)  # Accuracy scores
     print('CV accuracy scores: %s' % scores)

     import numpy as np
     print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
CV accuracy scores: [0.97826087 0.95652174 0.95652174 0.95652174 0.91304348 0.95555556
 0.97777778 0.97777778 1.         0.97777778]
CV accuracy: 0.965 +/- 0.022
```

Nested Cross Validation

- $K_1 = 5$

  $K_2 = 2$



Grid Search CV

```
[ ]  # Decision tree
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import KFold
     inner_cv=KFold(n_splits=3, shuffle=True, random_state=0)
     outer_cv=KFold(n_splits=5, shuffle=True, random_state=0)
     gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                       param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None]}],
                       scoring='accuracy', cv=inner_cv)
     scores = cross_val_score(gs, X, y, scoring='accuracy', cv=outer_cv)
     print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
CV accuracy: 0.942 +/- 0.012
```

Grid Search CV

cv.glmnet {glmnet} R Documentation

Cross-validation for glmnet

Description

Does k-fold cross-validation for glmnet, produces a plot, and returns a value for `lambda` (and `gamma` if `relax=TRUE`)

Usage

```
cv.glmnet(x, y, weights = NULL, offset = NULL, lambda = NULL,
  type.measure = c("default", "mse", "deviance", "class", "auc", "mae",
  "C"), nfolds = 10, foldid = NULL, alignment = c("lambda",
  "fraction"), grouped = TRUE, keep = FALSE, parallel = FALSE,
  gamma = c(0, 0.25, 0.5, 0.75, 1), relax = FALSE, trace.it = 0, ...)
```

reference

자료

19-2 STAT424 통계적 머신러닝 - 박유성 교수님

교재

파이썬을 이용한 통계적 머신러닝 (2020) – 박유성

ISLR (2013) - G. James, D. Witten, T. Hastie, R. Tibshirani

The elements of Statistical Learning (2001) - J. Friedman, T. Hastie, R. Tibshirani

Statistical Learning with Sparsity: The Lasso and Generalizations(2015)
 - R. Tibshirani, , T. Hastie, M. Wainwright