Overview of Supervised Learning

김지수 (Jisu KIM)

통계적 기계학습(Statistical Machine Learning), 2026 1학기 (spring)

The lecture note is a minor modification of the lecture notes from Prof. Yongdai Kim's "Statistical Machine Learning".Also, see Section 2.1-2.6, 2.9 from [1].

1 Statistical Machine Learning

There are several fields in machine learning.

- Supervised Learning (지도학습)
  - Predict outcome variable (출력변수) using input variable (입력변수)
  - 회귀(regression), 분류(classification), etc
- Unsupervised Learning (비지도학습)
  - Clarify relations within input variables.
  - 군집분석(clustering), 주성분분석(principal component analysis), 요인분석(factor analysis), etc
- Reinforcement Learning (강화학습)
  - Learn decision making where environment changes according to actions
  - multi-armed bandit problem, markov decision process, etc

This class mostly covers supervised learning.

Data(자료) always comes with random noise(잡음). To see the effect of random noise and whether the algorithm works correctly, we need statistical analysis. In machine learning, statistical theory provides the following:

- understanding overfitting (과적합)
- understanding curse of dimensionality (차원의 저주)
- statistical theory for algorithm: consistency(일치성), optimality(최적합), uncertaintly quantification(불확실성의 정량화)

2 Basic Model for Supervised Learning

- Input(입력) / Covariate(설명 변수) : $x \in \mathbb{R}^d$, so $x = (x_1, \ldots, x_d)$.
- Output(출력) / Response(반응 변수): $y \in \mathcal{Y}$. If y is categorical, then supervised learning is "classification", and if y is continuous, then supervised learning is "regression".
- Model(모형) :

$$y \approx f(x), \qquad f \in \mathcal{M}.$$

If we include the error ϵ to the model, then it can be also written as

$$y = \phi(f(x), \epsilon).$$

For many cases, we assume additive noise, so

$$y = f(x) + \epsilon.$$

- Assumption(가정): f belongs to a family of functions $\mathcal{M}$. This is the assumption of a model: a model can be still used when the corresponding assumption is not satisfied in your data.
- Loss function(손실 함수): $\ell(y, a)$. A loss function measures the difference between estimated and true values for an instance of data. The most common ones are:
  - square: $\ell(y, a) = (y - a)^2$.
  - $0 - 1$: $\ell(y, a) = I(y \neq a)$.
- Training data(학습 자료): $\mathcal{T} = \{(y_i, x_i), i = 1, \ldots, n\}$, where (y_i, x_i) is a sample from a probability distribution P_i. For many cases we assume i.i.d., or x_i's are fixed and y_i's are i.i.d..
- Goal(목적): we want to find f that minimizes the expected prediction error,

$$f^0 = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(Y,X)\sim P} \left[\ell(Y, f(X))\right].$$

  Here, $\mathcal{F}$ can be different from $\mathcal{M}$; $\mathcal{F}$ can be smaller than $\mathcal{M}$.
- Prediction model(예측 모형): f^0 is unknown, so we estimate f^0 by $\hat{f}$ using data. For many cases we minimizes on the empirical prediction error, that is taking the expectation on the empirical distribution $P_n = \frac{1}{n}\sum_{i=1}^n \delta_{(Y_i,X_i)}$.

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{P_n} \left[\ell(Y, f(X))\right] = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f(X_i)).$$

- Prediction(예측): if $\hat{f}$ is a predicted function, and x is a new input, then we predict unknown y by $\hat{f}(x)$.

3 Linear Regression Models and Nearest Neighbors

From the additive noise model

$$y = f(x) + \epsilon,\ f \in \mathcal{M},$$

Linear Regression Model (선형회귀모형) is that

$$\mathcal{M} = \mathcal{F} = \left\{ \beta_0 + \sum_{j=1}^d \beta_j x_j : \beta_j \in \mathbb{R} \right\}.$$

For estimating β, we use least squares method (최소제곱법): suppose the training data is $\{(y_i, x_{ij}) : 1 \leq i \leq n, 1 \leq j \leq d\}$. We use square loss

$$\ell(y, a) = (y - a)^2,$$

then the empirical loss (경험적 손실) becomes the residual sum of square (RSS, 잔차제곱합) as

$$\begin{aligned} RSS(\beta) &= \sum_{i=1}^n (y_i - f(x_i))^2 \\ &= \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^d x_{ij}\beta_j \right)^2. \end{aligned}$$

Let $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d)$ be the minimizer of RSS, then the predicted function is

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{j=1}^d \hat{\beta}_j x_j.$$

k-Nearest Neighbor(k-NN, k-최근접 이웃) is to use those observations in the training set $\mathcal{T}$ closest in input space to x to form $\hat{f}(x)$. For an input $x \in \mathbb{R}^d$, let $N_k(x)$ be the neighborhood of x defined by the k closest points among the training sample $x_1, \ldots, x_n$. Then the predicted function for k-nearest neighbor is

$$\hat{f}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i.$$

4 Statistical Decision Theory

Let the training data $\mathcal{T} = \{(y_i, x_i), i = 1, \ldots, n\}$ be i.i.d. from P, or $y|X = x$ is i.i.d. (weaker). For the regression, the most common loss is the square loss

$$\ell(y, a) = (y - a)^2.$$

Then the Risk (위험) or Expected Prediction Error (EPE, 평균예측오차) is

$$\begin{aligned} R(f) &= \mathbb{E}_{(Y,X)\sim P}\left[\ell(Y, f(X))\right] = \mathbb{E}\left[(Y - f(X))^2\right] \\ &= \mathbb{E}_X\left[\mathbb{E}_{Y|X}\left[(Y - f(X))^2|X\right]\right]. \end{aligned}$$

Then

$$\mathbb{E}_{Y|X}\left[(Y - f(X))^2|X\right] = \mathbb{E}_{Y|X}\left[\left(Y - \mathbb{E}\left[Y|X\right]\right)^2|X\right] + \left(\mathbb{E}\left[Y|X\right] - f(X)\right)^2,$$

so $R(f)$ is minimized when $f^0(x) = \mathbb{E}\left[Y|X = x\right]$. This conditional expectation is called the regression function (회귀함수). Still, $\mathbb{E}\left[Y|X = x\right]$ is usually unknown.

The nearest neighbor (최근접 이웃) directly implement this.

$$\hat{f}(x) = Ave\left(y_i|x_i \in N_k(x)\right),$$

where "Ave" denotes average, and $N_k(x)$ is the neighbor containing the k points in $\mathcal{T}$ closest to x. Two approximations are happening here:

- expectation is approximated by averaging over sample data;
- conditioning at a point is relaxed to conditioning on neighbor of x.

In fact, under mild regularity conditions on the joint probability distribution $P(X, Y)$, one can show that as $n, k \to \infty$ such that $k/n \to 0$, then

$$f(x) \to \mathbb{E}\left[Y|X = x\right].$$

The condition $k/n \to 0$ means that the model complexity k should grow slower than the sample size n.

For linear regression (선형 회귀), we assume that the regression function is approximately linear. For notational convenience, write $x = (1, x_1, \ldots, x_d)^\top$ so that the constant is also included in x. Then we assume

$$\mathbb{E}\left[Y|X = x\right] \approx x^\top\beta = \sum_{j=0}^{d}\beta_j x_j.$$

This is a model-based approach - we specify a model for the regression function. Let

$$\beta_* \coloneqq \left(\mathbb{E}\left[XX^\top\right]\right)^{-1}\mathbb{E}\left[XY\right],$$

then the risk becomes

$$\mathbb{E}\left[(Y - X^\top\beta)^2\right] = \mathbb{E}\left[\left(Y - X^\top\beta_*\right)^2\right] + \mathbb{E}\left[\left(X^\top\beta_* - X^\top\beta\right)^2\right],$$

So β_* is minimizing the risk. And the least square $\hat{\beta} = (X^\top X)^{-1}X^\top Y$ can be thought as the average over empirical distribution (경험적 분포) of training data.

For the classification so that $y \in \mathcal{Y} = \{0, \ldots, K-1\}$, the risk is

$$\begin{aligned} R(f) &= \mathbb{E}_{(Y,X)\sim P}\left[\ell(Y, f(X))\right] = \mathbb{E}\left[I(Y \neq f(X))\right] \\ &= \mathbb{E}_X\left[\sum_{j=0}^{K-1}\ell(j, f(X))P(Y = j|X)\right]. \end{aligned}$$

Hence the risk $R(f)$ is minimized when

$$f^0(x) = \arg\min_{k=0,\ldots,K-1}\sum_{j=0}^{K-1}\ell(j, k)P(Y = j|X = x).$$

For the classification, the most common loss is the 0-1 loss

$$\ell(y, a) = I(y \neq a).$$

And the optimal prediction function is

$$f^0(x) = \arg \max_{j=0,\ldots,K-1} P(Y = j|X = x).$$

This optimal classifier is called Bayes rule / Bayes classifier (베이즈분류 / 베이즈모형), and the error rate is called the Bayes risk (베이즈위험). Still, $P(Y = j|X = x)$ is usually unknown.

Again, k-NN (k-최근접 이웃) can directly approximate Bayes rule. The conditional probability $P(Y = j|X = x)$ is approximated as

$$\frac{1}{k} \sum_{x_i \in N_k(x)} I(y_i = j),$$

so k-NN classifier becomes

$$\hat{f}(x) = \arg \max_{j=0,\ldots,K-1} \left\{ \sum_{x_i \in N_k(x)} I(y_i = j) \right\}.$$

5 Curse of Dimensionality

Suppose $X = (X_1, \ldots, X_d) \sim Uniform([0, 1]^d)$. Suppose we want to set a hypercubical neighbor of a target point $x \in [0, 1]^d$ to capture a fraction r of the observations. For this, the side length of the hypercube should be

$$e_d(r) = r^{1/d}.$$

$e_{10}(0.01) = 0.63$ and $e_{10}(0.1) = 0.80$, so to capture 1% or 10% of the data to form a local average, we must cover 63% or 80% of the range of each input variable. Such neighbor is no longer "local".

Consider $X = (X_1, \ldots, X_d) \sim Uniform(\mathcal{B}_{\mathbb{R}^d}(0, 1))$, where $\mathcal{B}_{\mathbb{R}^d}(0, 1) = \{x \in \mathbb{R}^d : \|x\|_2 \leq 1\}$ is the unit ball in $\mathbb{R}^d$. For i-th data $X_i = (X_{i1}, \ldots, X_{id})$, $R_i \coloneqq \sqrt{\sum_{j=1}^d X_{ij}^2}$ is the distance of X_i to the origin, and $R_{(1)} = \min_{1 \leq i \leq n} R_i$ is the distance from the origin to the nearest neighbor. Then $R_{(1)}$ is also a random variable, and the median of $R_{(1)}$ is

$$Med(R_{(1)}) = \left(1 - (1/2)^{1/n}\right)^{1/d}.$$

If $n = 5000$ and $d = 10$, then $Med(R_{(1)}) \approx 0.52$, more than halfway to the boundary. Hence most data points are closer to the boundary of the sample space than to any other data point. This presents a problem since prediction is much more difficult near the edges of the training sample.

Beyond statistical difficulty for inference, high dimension is very counter-intuitive: reference https://marckhoury.github.io/blog properties-of-high-dimensional-space/

Consider a square with side length 1. At each corner of the square place a circle of radius 1/2, so that the circles cover the edges of the square. Then consider the circle centered at the center of the square that is just large enough to touch the circles at the corners of the square. In two dimensions it's clear that the inner circle is entirely contained in the square.

We can do the same thing in three dimensions. At each corner of the unit cube place a sphere of radius 1/2, again covering the edges of the cube. The sphere centered at the center of the cube and tangent to spheres at the corners of the cube is shown in red in Figure 3. Again we see that, in three dimensions, the inner sphere is entirely contained in the cube.

To understand what happens in higher dimensions we need to compute the radius of the inner sphere in terms of the dimension.

(the radius of the inner sphere) = (the length of the half diagonal of the cube)−(the radius of the spheres at the corners).

The radius of the inner sphere is equal to the length of the diagonal of the cube minus the radius of the spheres at the corners. See Figure . The half diagonal length is $\sqrt{d}/2$, and the latter value is always 1/2, and hence the radius of the inner sphere is

$$\frac{\sqrt{d} - 1}{2}.$$

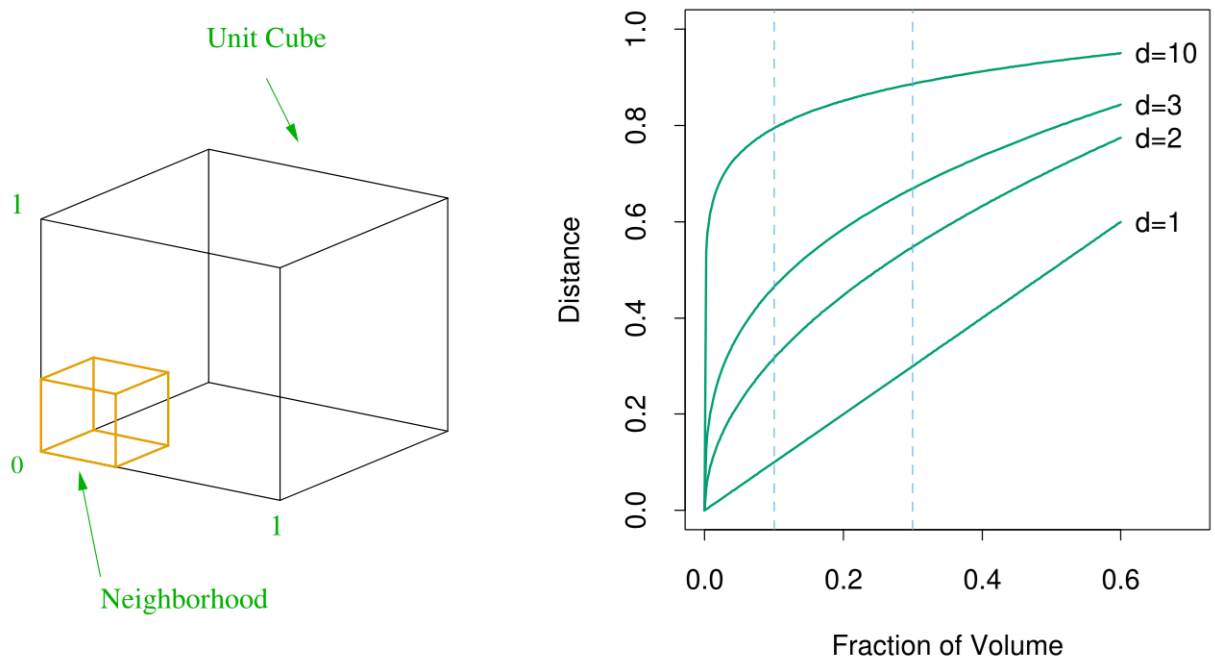


Figure 1: The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data. Figure 2.6 from [1].

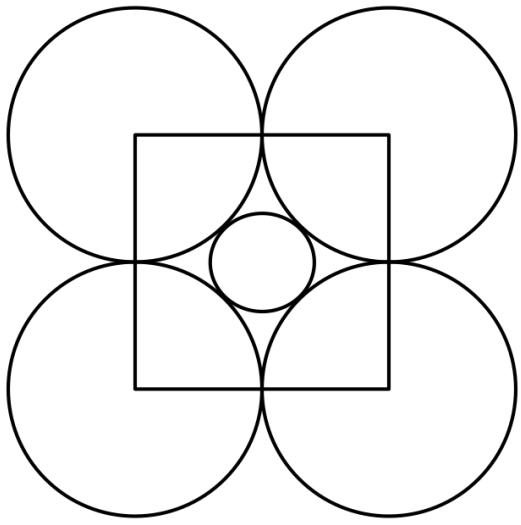


Figure 2: At each corner of the square we place a circle of radius 1/2. The inner circle is just large enough to touch the circles at the corners. Figure 1 from [2].

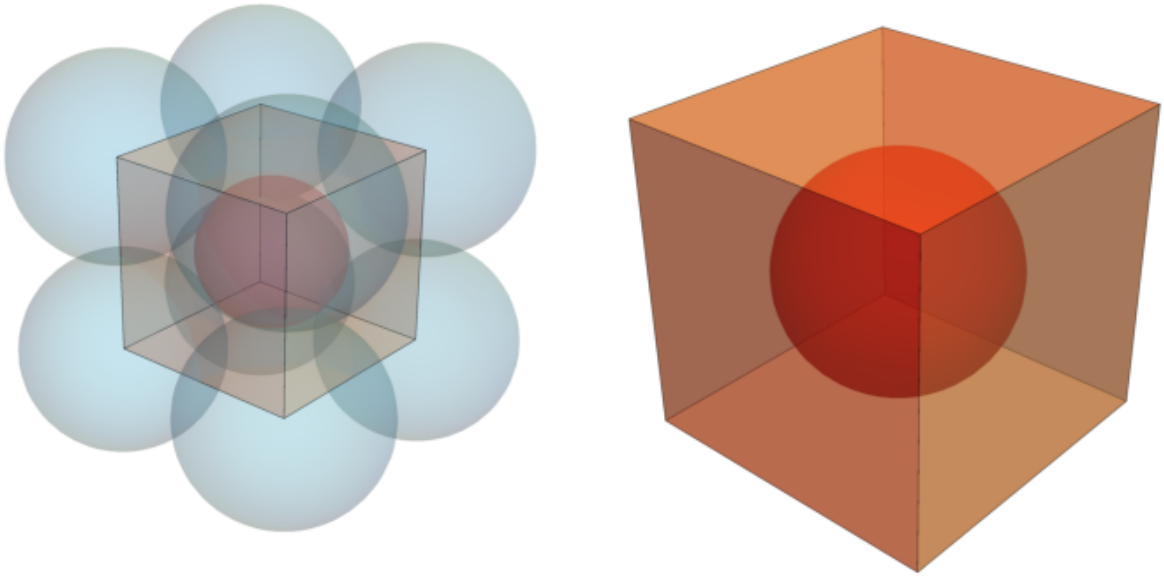


Figure 3: In three dimensions we place a sphere at the each of the eight corners of a cube. Figure 2 from [2].

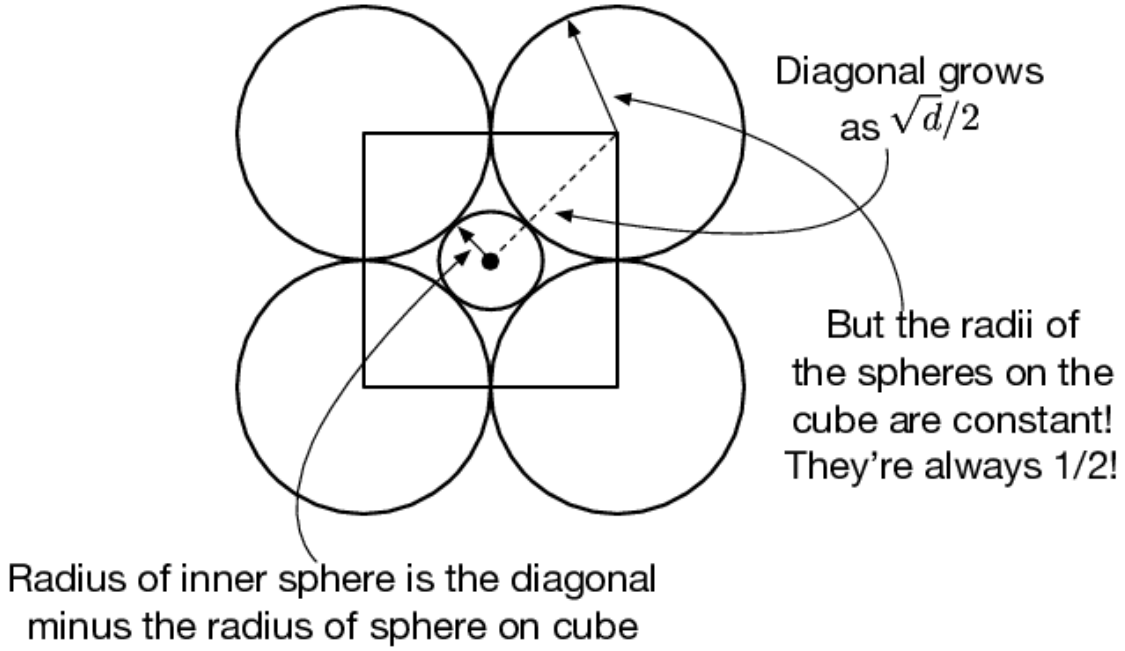


Figure 4: The size of the radius of the inner sphere is growing as the dimension increases because the distance to the corner increases while the radius of the corner sphere remains constant. Figure 3 from [2].

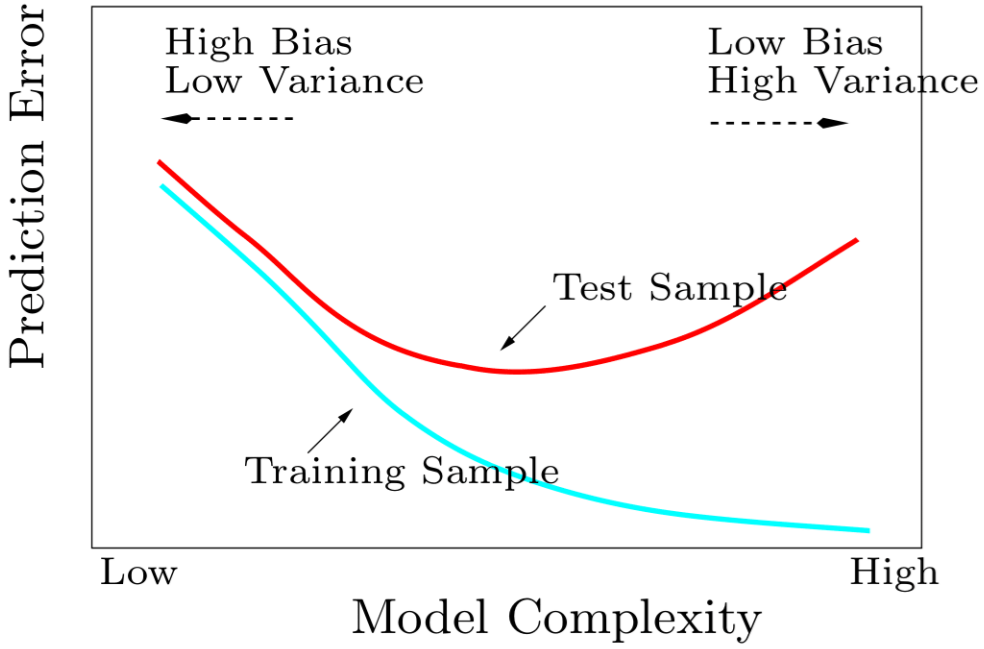


Figure 5: Test and training error as a function of model complexity. Figure 2.11 from [1].

When $d \leq 3$, $\frac{\sqrt{d}-1}{2} < \frac{1}{2}$ and the sphere is strictly inside the cube. However in four dimensions, the radius of the inner sphere is $\frac{\sqrt{d}-1}{2} = \frac{1}{2}$, so the inner sphere touches the sides of the cube. In five dimensions, the radius of the inner sphere is 0.618034, and the sphere starts poking outside of the cube! By ten dimensions, the radius is 1.08114 and the sphere is poking very far outside of the cube!

6 Overfitting and the Bias-Variance Tradeoff

In k-Nearest Neighbor, the neighbor size k determines the complexity of the prediction model. To choose k, if we know the probability distribution $(Y, X) \sim P$, then we can minimize the risk (위험)

$$\mathbb{E}_{(Y,X)\sim P}\left[\ell(Y, f(X))\right].$$

However, P is usually unknown. One way to estimate risk is by Training Error (TE, 학습오차)

$$\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f(X_i)),$$

where the training data $\mathcal{T} = \{(Y_i, X_i) : i = 1, \ldots, n\}$.

However, training error always underestimates prediction error (예측오차), since the same dataset $\mathcal{T}$ is used for estimating the prediction model and estimating prediction error. Better estimation for the prediction error is the test error, where the prediction error is computed on the separate test data. The typical behavior of the prediction error on the test set is as in Figure 5. As the model becomes more complex, the prediction error first decreases but then increases, while test error monotonically decreases. Hence, when the model is chosen according to the training error, then the prediction error can be large: this is called overfitting (과적합). This phenomenon is explained by the bias-variance tradeoff.

Suppose the regression model

$$Y = f(X) + \epsilon,$$

with $\mathbb{E}[\epsilon] = 0$ and $Var[\epsilon] = \sigma^2$, and we use the square loss $\ell(y, a) = (y - a)^2$. Let $f(X, \mathcal{T})$ be the predicted function

trained on the training set $\mathcal{T}$. Then the risk for $f(X, \mathcal{T})$ is

$$\begin{aligned} R(f) &= \mathbb{E}\left[(Y - f(X,\mathcal{T}))^2\right] \\ &= \mathbb{E}\left[(Y - f(X))^2\right] + \mathbb{E}\left[(f(X) - f(X,\mathcal{T}))^2\right] \\ &= \sigma^2 + \mathbb{E}_X\left[(f(X) - \mathbb{E}_{\mathcal{T}}\left[f(X,\mathcal{T})|X\right])^2\right] + \mathbb{E}_X\left[\mathbb{E}_{\mathcal{T}}\left[(f(X,\mathcal{T}) - \mathbb{E}_{\mathcal{T}}\left[f(X,\mathcal{T})|X\right])^2|X\right]\right] \\ &= \sigma^2 + \mathbb{E}_X\left[Bias_{\mathcal{T}}(X)^2 + Variance_{\mathcal{T}}(X)\right]. \end{aligned}$$

Then

$$\mathbb{E}_{Y|X}\left[(Y - f(X))^2|X\right] = \mathbb{E}_{Y|X}\left[(Y - \mathbb{E}\left[Y|X\right])^2|X\right] + \left(\mathbb{E}\left[Y|X\right] - f(X)\right)^2,$$

In general, as the model becomes more complex, then the bias(편의) decreases and the variance(분산) increases.

For k-Nearest Neighbor,

$$\hat{f}(x) = f(x,\mathcal{T}) = \frac{1}{k}\sum_{l=1}^{k}(f(x_{(l)}) + \epsilon_{(l)}),$$

where (l) is the index for l-th nearest neighbor. Then the bias and variance becomes

$$Bias_{\mathcal{T}}(x) = f(x) - \frac{1}{k}\sum_{l=1}^{k} f(x_{(l)}),$$

and

$$Variance_{\mathcal{T}}(x) = \frac{\sigma^2}{k}.$$

When $k = 1$, then the bias is minimized and the variance is maximized. On the other hand, when $k = n$, then the bias is maximized and the variance is minimized.

7 Comparison of Linear Regression with Nearest Neighbors

Below is the code for comparing linear regression with nearest neighbors. We measure the performance by Mean square Error (MSE, 평균제곱오차)

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - f(X_i))^2,$$

either on training data or test data.

```
require(caret)

## Loading required package: caret
## Loading required package: ggplot2
## Loading required package: lattice

mse <- function(y_pred, y_true) {

  return (mean((y_pred - y_true)^2))
}


linear_knn_compare <- function(data_train, data_test) {

  par(mfrow = c(2, 2))

  model_lr <- lm(y ~ x, data = data_train)
  plot(data_train[["x"]], data_train[["y"]], main = "Linear Regression",
       xlab = "x", ylab = "y", pch = 16)
  abline(model_lr)
  cat(c("MSE of Linear Regression on Training:",
        mse(data_train[["y"]], predict(model_lr, newdata = data_train)), "\n"))
  cat(c("MSE of Linear Regression on Test:",
        mse(data_test[["y"]], predict(model_lr, newdata = data_test)), "\n"))

  for (k in c(1, 5, 15)) {
```

```
    model_knn <- caret::knnreg(y ~ x, data = data_train, k = k)
    plot(data_train[["x"]], data_train[["y"]],
         main = paste("Nearest Neighbor with k =", k), xlab = "x", ylab = "y",
         pch = 16)
    lines(data_train[["x"]], predict(model_knn, newdata = data_train))
    cat(c("MSE of Nearest Neighbor with k =", k, "on Training:",
          mse(data_train[["y"]], predict(model_knn, newdata = data_train)),
          "\n"))
    cat(c("MSE of Nearest Neighbor with k =", k, "on Test:",
          mse(data_test[["y"]], predict(model_knn, newdata = data_test)), "\n"))
  }
}
```

Suppose we have 100 training data and 900 test data, and $\epsilon \sim N(0, 1)$.

```
set.seed(0)
x <- rnorm(1000)
ep <- rnorm(1000)
train <- sort(x[1:100], index.return = TRUE)[["ix"]]
test <- 101:1000
```

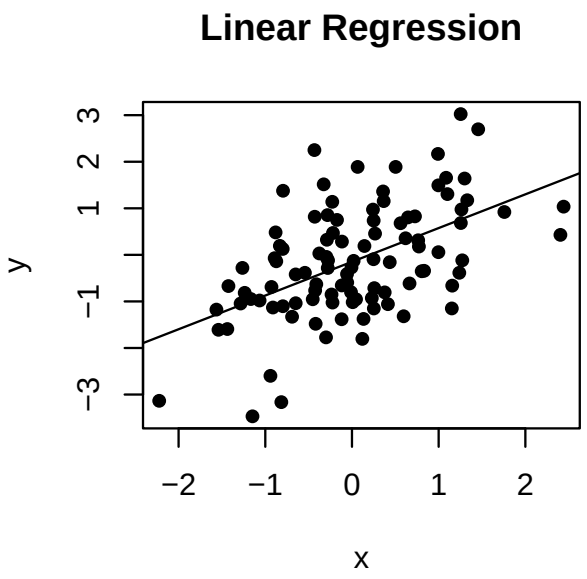
First, we consider the simulation

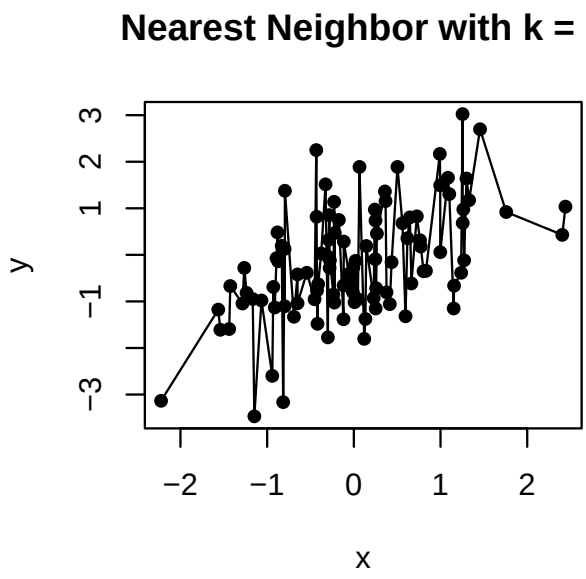
$$y = x + \epsilon.$$

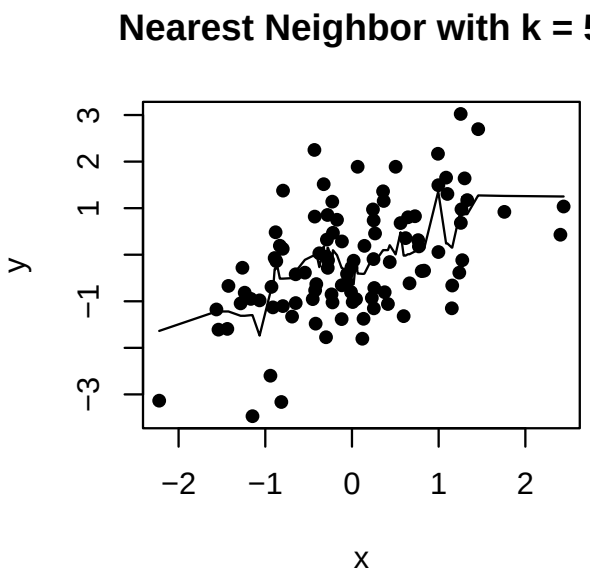
The result is as follows:

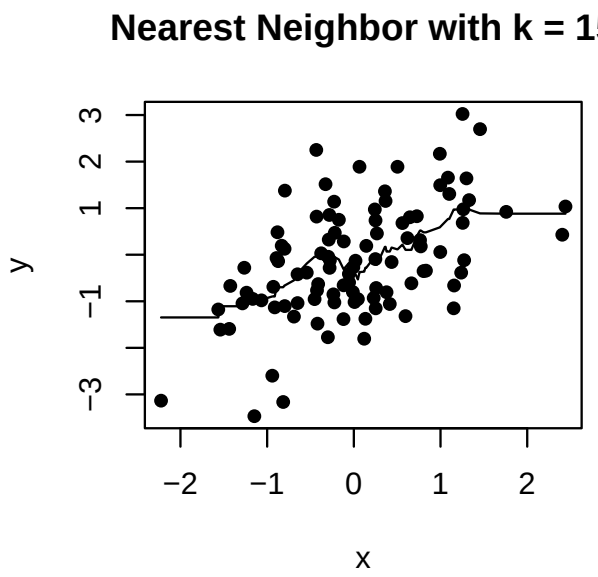
Method	MSE on Training	MSE on Test
Linear	1.0447	1.1621
1-NN	0	2.0972
5-NN	0.9381	1.2779
15-NN	1.0053	1.2757

```
# y = x + ep
y_l <- x + ep
data_l_train <- list(x = x[train], y = y_l[train])
data_l_test <- list(x = x[test], y = y_l[test])

linear_knn_compare(data_l_train, data_l_test)

## MSE of Linear Regression on Training: 1.04471978605025
## MSE of Linear Regression on Test: 1.16212381385859
## MSE of Nearest Neighbor with k = 1 on Training: 0
## MSE of Nearest Neighbor with k = 1 on Test: 2.09716012610056
## MSE of Nearest Neighbor with k = 5 on Training: 0.938067727758741
## MSE of Nearest Neighbor with k = 5 on Test: 1.27788806945154
## MSE of Nearest Neighbor with k = 15 on Training: 1.00528738504805
## MSE of Nearest Neighbor with k = 15 on Test: 1.27565679650095
```









Second, we consider the simulation

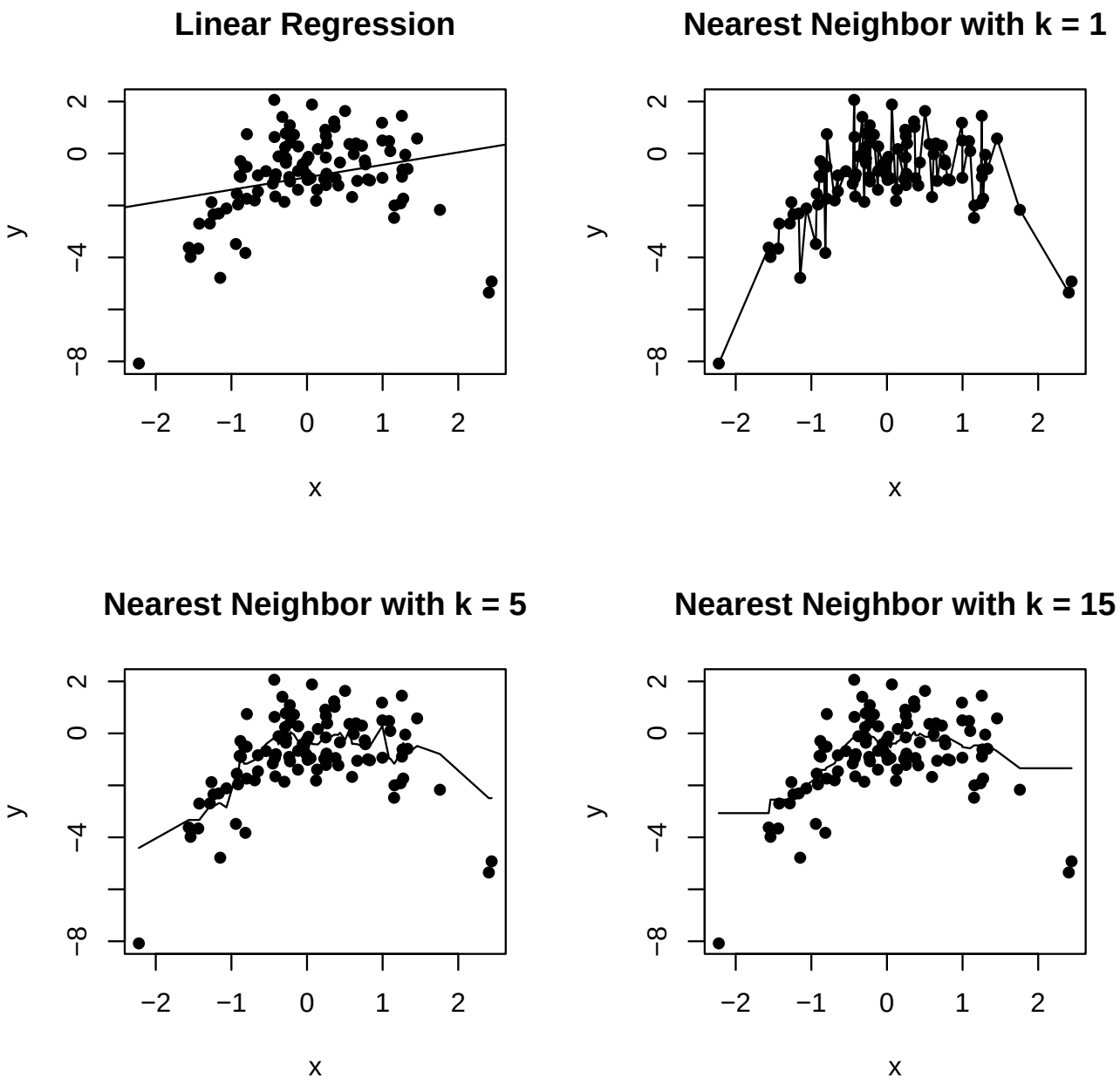
$$y = x(1-x) + \epsilon.$$

The result is as follows:

Method	MSE on Training	MSE on Test
Linear	2.4477	3.2330
1-NN	0	2.2437
5-NN	1.1955	1.7266
15-NN	1.5284	2.0868

```
# y = x(1-x) + ep
y_q <- x * (1 - x) + ep
data_q_train <- list(x = x[train], y = y_q[train])
data_q_test <- list(x = x[test], y = y_q[test])
```

```
linear_knn_compare(data_q_train, data_q_test)

## MSE of Linear Regression on Training: 2.44774751766011
## MSE of Linear Regression on Test: 3.23304666428842
## MSE of Nearest Neighbor with k = 1 on Training: 0
## MSE of Nearest Neighbor with k = 1 on Test: 2.24369404278609
## MSE of Nearest Neighbor with k = 5 on Training: 1.19552889833504
## MSE of Nearest Neighbor with k = 5 on Test: 1.72664570974827
## MSE of Nearest Neighbor with k = 15 on Training: 1.52842407126519
## MSE of Nearest Neighbor with k = 15 on Test: 2.08680791517323
```



From the simulations, we can compare Linear Regression vs Nearest Neighbor as follows:

Method	Linear Regression	Nearest Neighbor
Assumption	Linear	X
Required Data Size (Relative)	Small	Large
Interpretation	Easy	Impossible
Prediction	Good if Linear is satisfied, Bad otherwise	works regardless of assumptions
Tuning Parameter	X	neighborhood size k

Also, we can see that:

- training error is not a good estimate of prediction error.
- As model complexity grows, we can see bias-variance tradeoff
- Neighborhood size k determines the model complexity of Nearest Neighbor

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. Data mining, inference, and prediction.

[2] Marc Khoury. Counterintuitive properties of high dimensional space, 2018.