STA314 Summer 2023

Week 1: Introduction & Assessing Model Performance

Ziang Zhang

Department of Statistics, University of Toronto

Overview

What's in this course?

We will cover the fundamental tools/concepts used in ML methods.
This course focuses more on understanding the mechanics behind ML methods, with less emphasis on the coding part (different from CSC311).
More specifically, we will cover the following topics:

- Supervised Methods: Linear Regression, Logistic Regression, KNN, Decision Trees...
- Unsupervised Methods: PCA, K-means
- Statistical Tools: Bias/Variance Tradeoff, Shrinkage, Bayesian Inference, Cross-validation, Bootstrapping ...

What are the learning outcomes?

By the end of this course...

1. Understand how different ML models/methods work.
2. Being able to diagnose the problem when a particular model/method fails
3. Code up simple ML algorithms for practical problems

We will *not* cover deep-learning/neural networks in this course. But, this course will prepare you for more advanced ML/AI courses, such as STA414, CSC412 etc.

Components of the course

Lecture: Introducing new concepts & Illustrating their applications.
Tutorial: Focus on problem-solving skills & More emphasis on coding/computation. This is also a good chance for you to ask computation-related questions

Note that you have to attend your enrolled tutorial session on Acorn, as the quizzes will be held during the tutorial.
Also, neither the lecture nor the tutorial will be recorded, so please come in-person!

Computation

The official computation environment in this course is R.
You won't be asked to write codes in midterm or final, *but you are expected to read codes or output from R.*
For the coding question in problem sets, the starter code will be given in R, but you may code in Python if you prefer.

Grading Scheme

Grades: 30% Quiz (lowest grade will be dropped), 30% Midterm, 40% Final

Quiz 1	July 14th
Quiz 2	July 21st
Midterm	July 28th
Last day to drop	July 31st
Quiz 3	Aug 4th
Quiz 4	Aug 11th
Last day for CR/NCR, or LWD	August 15th
Final Exam	TBA

Problem set & Quizzes

Problem set: The assigned problem set will be posted approximately weekly. Most problem sets will include a computational component. *You should bring printouts of your work to the quiz, which should contain both your code and its outputs.*
The non-computational parts of the assignments will not be collected or graded.
Quiz: The quizzes will be held during the tutorial time, with questions based on the assigned problem set each week. You will be asked to answer questions based on your printouts. *One of the quiz questions may require you to hand in your printout.* The lowest quiz grade will be dropped.

Collaboration & Discussion

Collaboration: For the non-computer part of each assigned problem set, students are allowed to collaborate with each other. For the computer part of each assigned problem set, students are required to work on the question alone.

Discussion: Piazza will be used for the course forum.

As a motivation for students to participate in Piazza, the instructor team will regularly endorse good questions and responses. By the end of the term, students who have received $\geq$ *5 good questions or endorsed responses will be awarded a 1 percent bonus at their final grades.*

What is ML?

Machine Learning: Predicting outcome, learning pattern, automating decision making.
Statistics: Making inferences, determining important factors, distinguishing different effects.

Statistics is more concentrated on mathematical rigorousness and result interpretability; ML is more on predictive accuracy and autonomy;

Statistical learning depends on both worlds.

- **Outcome** (y): Also called response, dependent variable, label, output.
- **Feature** (x): Also called covariate, independent variable, explanatory variable, predictor, regressor, input.
- **Statistical learning**: Estimate the data-generating process/distribution and hence make prediction.
  - **Supervised**: Given outcomes and features, aim to predict new outcome ($p(y|x)$, $p(y,x)$). Examples such as regression (y continuous) and classification (y categorical).
  - **Unsupervised**: Given features, aim to understand their structure ($p(x)$). Examples such as clustering and PCA.
  - **Semi-supervised**: Given some labelled data, but most of the data are unlabelled (not cover in this course).

Parametric vs Nonparametric

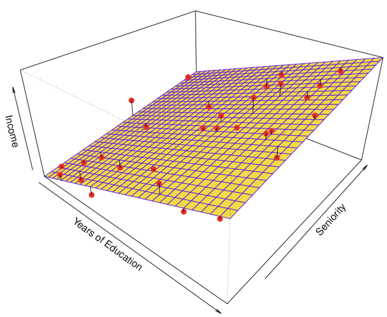
Consider the regression problem: $Y = f(X) + \epsilon$:

- $X = (X_1, ..., X_p)$ denotes p features.
- f is some unknown function.
- ϵ is a independent random noise with $\mathbb{E}(\epsilon) = 0$.

Target is to obtain $\hat{f}$, hence predict Y_{new} as $\hat{f}(X_{new})$

- **Parametric:** Assume a functional form on f, for example, linear regression $f(X) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$.
- **Nonparametric:** Does not assume the form of f, but only some measure of smoothness, for example, smoothing spline.

Parametric vs Nonparametric



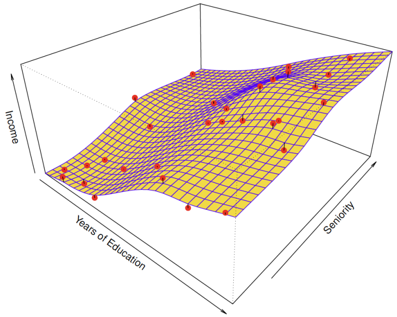


Figure: Fitted parametric model and nonparametric model. Source: James et al., 2021.

Parametric method: easier to implement and interpret, work bad when the functional form is mis-specified.
Nonparametric method: flexible to accommodate different types of f, require larger data to give precise result.

Review of the prerequisite

To understand the ML concepts, the following prerequisite knowledge is important:

1. Linear algebra (MAT223)
2. Calculus (MAT235 or MAT237)
3. Probability theory and statistics (STA237/238 or STA257/261, STA302)

In case some of the core knowledge was not covered in your pre-requisite courses, we will review some intermediate-level linear algebra as well as multivariate statistics.

But please keep in mind that you should review the more basic stuff on your own, if you feel unfamiliar with topics such as matrix/vector algebra, convexity/concavity, optimization, univariate statistics and probability theory...

Review of linear algebra: Trace

The **trace** of a $n \times n$ (square) matrix A is defined as:

$$tr(A) = \sum_{i=1}^{n} A_{ii}.$$

This linear operator satisfies the following property:

- $tr(A + B) = tr(A) + tr(B)$ for any $n \times n$ matrices A, B (linearity).
- $tr(cA) = ctr(A)$ for any square matrix A and scalar c (linearity).
- $tr(A) = tr(A^T)$ for any square matrix A.
- $tr(AB) = tr(BA)$ for any $n \times n$ matrices A, B.
- $tr(ABC) = tr(BCA) = tr(CAB)$ for any $n \times n$ matrices A, B, C (cyclic).

Note that $tr(ABC) = tr(BAC)$ is *not* guaranteed to hold.

Review of linear algebra: Determinant

- The determinant of $A \in \mathbb{R}^{n \times n}$ is denoted as $|A|$ or $\det(A)$.
- For $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $|A| = a_{11}a_{22} - a_{12}a_{21}$.
- For larger square matrices, we use **cofactor expansion**.
- For each element a_{ij} in a chosen row or column, calculate its **cofactor** C_{ij}:

$$C_{ij} = (-1)^{i+j} \cdot \det(M_{ij}),$$

  where M_{ij} is the matrix that remains after deleting the i-th row and j-th column from the original matrix.
- The determinant of the matrix is then: $|A| = \sum_{j=1}^{n} a_{ij}C_{ij}$ (row-expansion) or $|A| = \sum_{i=1}^{n} a_{ij}C_{ij}$(column-expansion).
- This process is recursive, and continues until the determinants of 2×2 matrices are calculated.

Review of linear algebra: Determinant

This determinant operator satisfies the following property:

- $|AB| = |A||B|$ for any $n \times n$ matrices A, B.
- $|cA| = c^n|A|$ for any $n \times n$ matrix A and scalar c.
- $|A| = |A^T|$ for any square matrix A.
- The matrix A is invertible (non-singular) iff $|A| \neq 0$.

Note that $|A + B| = |A| + |B|$ is *not* guaranteed to hold.

Review of linear algebra: PSD and PD

A $n \times n$ square matrix A is **symmetric** if $A = A^T$.
A $n \times n$ symmetric matrix A is **positive semi-definite (PSD)** if

$$x^T A x \geq 0 \ \forall x \in \mathbb{R}^n.$$

If in addition, $x^T A x = 0$ implies $x = 0$, the PSD matrix A is called positive definite.

Negative semi-definite matrix and negative definite matrix are defined by replacing $\geq$ above with $\leq$.

Review of linear algebra: SVD

Given *any* $n \times p$ matrix A, its singular value decomposition (SVD) can always be written as

$$A = UDV^T,$$

where U is an $n \times n$ orthonormal matrix, V is an $p \times p$ orthonormal matrix, and D is an $n \times p$ diagonal matrix with *non-negative* entries.

- An $n \times p$ diagonal matrix has $\min(n, p)$ elements on the diagonal, and all other entries are zero.
- An orthonormal matrix U is a square matrix that satisfies $UU^T = U^T U = I$.
- The diagonal values of D is called the *singular values* of A, sorted from the largest to the smallest.

Review of linear algebra: Eigenvectors and eigenvalues

- An **eigenvector** of $\mathbf{A}$ is a *non-zero vector* $\mathbf{v}$ such that

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

  where λ is a scalar called the eigenvalue corresponding to this eigenvector.
- An **eigenvalue** λ of $\mathbf{A}$ is a scalar for which there exists a non-zero vector $\mathbf{v}$ (an eigenvector) such that the above holds.
- To find the eigenvalues of a matrix, we solve the characteristic equation:

$$|\mathbf{A} - \lambda\mathbf{I}| = 0$$

  where $\mathbf{I}$ is the identity matrix of the same size as $\mathbf{A}$.
- The eigenvalues can be substituted back into the equation $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ to find the corresponding eigenvectors.

Review of linear algebra: Eigen-decomposition

Given *any* $n \times n$ real symmetric matrix A, its Eigen (Spectral) decomposition can always be written as

$$A = UDU^T,$$

where U is an $n \times n$ orthonormal matrix and D is an $n \times n$ diagonal matrix with diagonal values $\{d_i\}_{i=1}^n$.

- The ith column of U is the ith *eigenvector* of A.
- The diagonal values $\{d_i\}_{i=1}^n$ are the *eigenvalues* of A, sorted from the largest to the smallest.
- $|A| = \prod_{i=1}^n d_i$ and $tr(A) = \sum_{i=1}^n d_i$ (proof as exercise).

Eigen-decomposition continued

If $A \in \mathbb{R}^{n\times n}$ is PSD, its SVD and Eigen-decomposition are the *same*: $A = UDU^T$.

- It is easy to show the eigenvalues $\{d_i\}_{i=1}^n$ of the PSD matrix must be non-negative.
- If the PSD matrix is further PD, the eigenvalues $\{d_i\}_{i=1}^n$ are strictly positive.
- For a PSD matrix $A = UDU^T$, we define $A^q = UD^qU^T$, where D^q has diagonal value $\{d_i^q\}_{i=1}^n$.

For a general $A \in \mathbb{R}^{n\times p}$, the singular values of A are the square root of the eigenvalues of A^TA.

Review of Random Vectors and Matrices

A *random matrix* is just a matrix of random variables. Their joint probability distribution is the distribution of the random matrix. Random matrices with just one column (say, $p \times 1$) may be called *random vectors*.

Review of Expected Value

The expected value of a matrix (vector) is defined as the matrix (vector) of expected values. Denoting the $p \times c$ random matrix $\mathbf{X}$ by $[X_{i,j}]$,

$$\mathbb{E}(\mathbf{X}) = [E(X_{i,j})].$$

Immediately we have natural properties like:

- For a random matrix $\mathbf{Z}$, $\mathbb{E}(\mathbf{X}+\mathbf{Z}) = \mathbb{E}(\mathbf{X}) + \mathbb{E}(\mathbf{Z})$
- For a constant matrix $\mathbf{A}$, $\mathbb{E}(\mathbf{AX}) = \mathbf{A}\mathbb{E}(\mathbf{X})$
- For two constant matrices $\mathbf{A}, \mathbf{B}$, $\mathbb{E}(\mathbf{AXB}) = \mathbf{A}\mathbb{E}(\mathbf{X})\mathbf{B}$
- For a random matrix $\mathbf{Z}$, $tr(\mathbb{E}(\mathbf{Z})) = \mathbb{E}(tr(\mathbf{Z}))$

Review of variance and covariance

Let $\mathbf{X}$ be a $p \times 1$ random vector with $\mathbb{E}(\mathbf{X}) = \boldsymbol{\mu}$. The *variance-covariance matrix* of $\mathbf{X}$ (sometimes just called the *covariance matrix*), denoted by $cov(\mathbf{X})$, is defined as

$$cov(\mathbf{X}) = \mathbb{E}\left\{(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top\right\}.$$

Exercise: *Prove from the definition that the covariance matrix is always PSD.*

$cov(\mathbf{X}) = \mathbb{E}\left\{(\mathbf{X}-\boldsymbol{\mu})(\mathbf{X}-\boldsymbol{\mu})^\top\right\}$

$$
\begin{aligned}
cov(\mathbf{X}) &= \mathbb{E}\left\{\begin{pmatrix} X_1-\mu_1 \\ X_2-\mu_2 \\ X_3-\mu_3 \end{pmatrix}\begin{pmatrix} X_1-\mu_1 & X_2-\mu_2 & X_3-\mu_3 \end{pmatrix}\right\} \\
&= \mathbb{E}\left\{\begin{pmatrix} (X_1-\mu_1)^2 & (X_1-\mu_1)(X_2-\mu_2) & (X_1-\mu_1)(X_3-\mu_3) \\ (X_2-\mu_2)(X_1-\mu_1) & (X_2-\mu_2)^2 & (X_2-\mu_2)(X_3-\mu_3) \\ (X_3-\mu_3)(X_1-\mu_1) & (X_3-\mu_3)(X_2-\mu_2) & (X_3-\mu_3)^2 \end{pmatrix}\right. \\
&= \left(\begin{matrix} \mathbb{E}\{(X_1-\mu_1)^2\} & \mathbb{E}\{(X_1-\mu_1)(X_2-\mu_2)\} & \mathbb{E}\{(X_1-\mu_1)(X_3- \\ \mathbb{E}\{(X_2-\mu_2)(X_1-\mu_1)\} & \mathbb{E}\{(X_2-\mu_2)^2\} & \mathbb{E}\{(X_2-\mu_2)(X_3- \\ \mathbb{E}\{(X_3-\mu_3)(X_1-\mu_1)\} & \mathbb{E}\{(X_3-\mu_3)(X_2-\mu_2)\} & \mathbb{E}\{(X_3-\mu_3)^2\} \end{matrix}\right. \\
&= \begin{pmatrix} Var(X_1) & Cov(X_1,X_2) & Cov(X_1,X_3) \\ Cov(X_1,X_2) & Var(X_2) & Cov(X_2,X_3) \\ Cov(X_1,X_3) & Cov(X_2,X_3) & Var(X_3) \end{pmatrix}.
\end{aligned}
$$

So, the covariance matrix $cov(\mathbf{X})$ is a $p \times p$ symmetric matrix with variances on the main diagonal and covariances on the off-diagonals.

Matrix of covariances between two random vectors

Let $\mathbf{X}$ be a $p \times 1$ random vector with $\mathbb{E}(\mathbf{X}) = \boldsymbol{\mu}_x$ and let $\mathbf{Y}$ be a $q \times 1$ random vector with $\mathbb{E}(\mathbf{Y}) = \boldsymbol{\mu}_y$. The $p \times q$ matrix of covariances between the elements of $\mathbf{X}$ and the elements of $\mathbf{Y}$ is

$$cov(\mathbf{X}, \mathbf{Y}) = \mathbb{E}\left\{(\mathbf{X} - \boldsymbol{\mu}_x)(\mathbf{Y} - \boldsymbol{\mu}_y)^\top\right\}.$$

Quick exercise: *Is the matrix of covariances between two random vectors always PSD as well? why or why not.*

Adding a constant has no effect

On variances and covariances

- $cov(\mathbf{X} + \mathbf{a}) = cov(\mathbf{X})$
- $cov(\mathbf{X} + \mathbf{a}, \mathbf{Y} + \mathbf{b}) = cov(\mathbf{X}, \mathbf{Y})$

These results are clear from the definitions:

- $cov(\mathbf{X}) = \mathbb{E}\left\{(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top\right\}$
- $cov(\mathbf{X}, \mathbf{Y}) = \mathbb{E}\left\{(\mathbf{X} - \boldsymbol{\mu}_x)(\mathbf{Y} - \boldsymbol{\mu}_y)^\top\right\}$

Sometimes it is useful to let $\mathbf{a} = -\boldsymbol{\mu}_x$ and $\mathbf{b} = -\boldsymbol{\mu}_y$ (*Centering*).

Analogous to $Var(a\,X) = a^2\,Var(X)$

Let $\mathbf{X}$ be a $p \times 1$ random vector with $\mathbb{E}(\mathbf{X}) = \boldsymbol{\mu}$ and $cov(\mathbf{X}) = \boldsymbol{\Sigma}$, while $\mathbf{A} = [a_{i,j}]$ is an $r \times p$ matrix of constants. Then

$$\begin{aligned}
cov(\mathbf{AX}) &= \mathbb{E}\left\{(\mathbf{AX} - \mathbf{A}\boldsymbol{\mu})(\mathbf{AX} - \mathbf{A}\boldsymbol{\mu})^\top\right\} \\
&= \mathbb{E}\left\{\mathbf{A}(\mathbf{X} - \boldsymbol{\mu})\left(\mathbf{A}(\mathbf{X} - \boldsymbol{\mu})\right)^\top\right\} \\
&= \mathbb{E}\left\{\mathbf{A}(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top\mathbf{A}^\top\right\} \\
&= \mathbf{A}\mathbb{E}\{(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top\}\mathbf{A}^\top \\
&= \mathbf{A}cov(\mathbf{X})\mathbf{A}^\top \\
&= \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top
\end{aligned}$$

Review of multivariate normal

The $p \times 1$ random vector $\mathbf{X}$ is said to have a *multivariate normal distribution*, and we write $\mathbf{X} \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, if $\mathbf{X}$ has (joint) density

$$f(\mathbf{x}) = \frac{1}{|\boldsymbol{\Sigma}|^{\frac{1}{2}}(2\pi)^{\frac{p}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where $\boldsymbol{\mu}$ is $p \times 1$ and $\boldsymbol{\Sigma}$ is $p \times p$ symmetric and positive definite.

- $|\boldsymbol{\Sigma}|$ is the determinant of Σ.
- The matrix $\boldsymbol{Q} = \Sigma^{-1}$ is called the *precision matrix* of $\mathbf{x}$, with $|\boldsymbol{Q}| = 1/|\boldsymbol{\Sigma}|$.

Analogies

(Multivariate normal reduces to the univariate normal when $p = 1$)

- Univariate Normal
  - $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}$
  - $E(X) = \mu, Var(X) = \sigma^2$
  - $\frac{(X-\mu)^2}{\sigma^2} \sim \chi^2(1)$
- Multivariate Normal
  - $f(\mathbf{x}) = \frac{1}{|\mathbf{\Sigma}|^{\frac{1}{2}}(2\pi)^{\frac{p}{2}}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$
  - $E(\mathbf{X}) = \boldsymbol{\mu}$, $cov(\mathbf{X}) = \mathbf{\Sigma}$
  - $(\mathbf{X}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{X}-\boldsymbol{\mu}) \sim \chi^2(p)$

More properties of the multivariate normal

- If $\mathbf{c}$ is a vector of constants, $\mathbf{X} + \mathbf{c} \sim N(\mathbf{c} + \boldsymbol{\mu}, \boldsymbol{\Sigma})$
- If $\mathbf{A}$ is a matrix of constants, $\mathbf{A}\mathbf{X} \sim N(\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top)$
- Linear combinations of multivariate normals are multivariate normal.
- All the marginals (dimension less than p) of $\mathbf{X}$ are (multivariate) normal, but it is possible in theory to have a collection of univariate normals whose joint distribution is not multivariate normal.
- For the multivariate normal, zero covariance implies independence. The multivariate normal is the only continuous distribution with this property.

Linear Regression

Assume there are n i.i.d data-pairs $(y_i, x_{i1}, ..., x_{ip})$.

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

- $\mathbf{X} \in \mathbb{R}^{n\times p}$. Each column denotes the observation of a particular feature.
- $\boldsymbol{y} = (y_1, ...y_n) \in \mathbb{R}^n$. $\boldsymbol{\beta} = (\beta_1, ...\beta_p) \in \mathbb{R}^p$.
- $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_n) \in \mathbb{R}^n$, $\mathbb{E}(\boldsymbol{\epsilon}) = \mathbf{0}$ and $\mathrm{Var}(\boldsymbol{\epsilon}) = \sigma^2\boldsymbol{I}$.

- Ordinary Least Square (OLS):

$$\hat{\boldsymbol{\beta}} = \arg\ \min_{\boldsymbol{\beta}}(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}).$$

- Maximum likelihood estimation (MLE):

$$\hat{\boldsymbol{\beta}} = \arg\ \max_{\boldsymbol{\beta}}\ p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{\beta}) = \arg\ \max_{\boldsymbol{\beta}}\ \log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{\beta}).$$

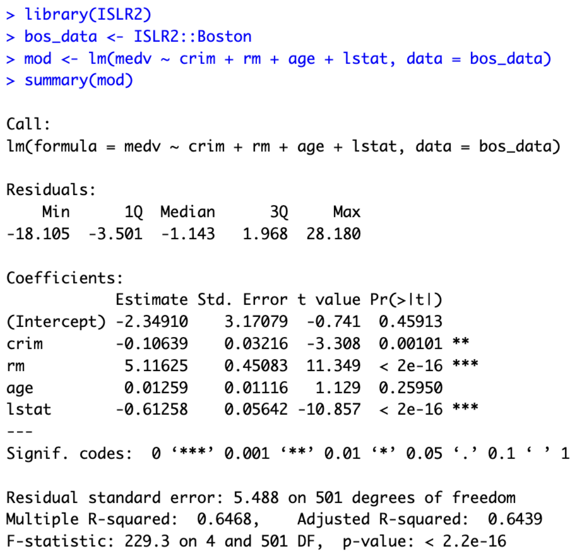
Both will lead to the same result:

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

Boston Housing Value

1. The Boston data set records medv (median house value) for 506 census tracts in Boston.
2. We will seek to predict medv using features: rm (average number of rooms per house), age (average age of houses), and lstat (percent of households with low socioeconomic status) and crim (per capital crime rate).
3. To start with, we will consider a linear regression model.

Model fitted through MLE

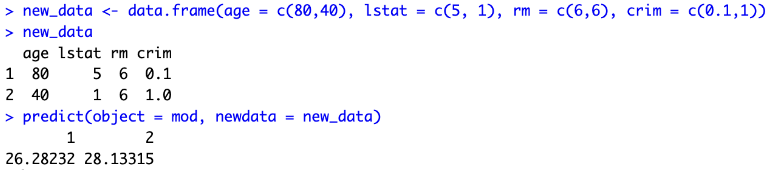


```
> library(ISLR2)
> bos_data <- ISLR2::Boston
> mod <- lm(medv ~ crim + rm + age + lstat, data = bos_data)
> summary(mod)

Call:
lm(formula = medv ~ crim + rm + age + lstat, data = bos_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-18.105  -3.501  -1.143   1.968  28.180 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.34910    3.17079  -0.741  0.45913    
crim        -0.10639    0.03216  -3.308  0.00101 ** 
rm           5.11625    0.45083  11.349  < 2e-16 ***
age          0.01259    0.01116   1.129  0.25950    
lstat       -0.61258    0.05642 -10.857  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.488 on 501 degrees of freedom
Multiple R-squared:  0.6468,	Adjusted R-squared:  0.6439 
F-statistic: 229.3 on 4 and 501 DF,  p-value: < 2.2e-16
```

Prediction with lm

Suppose we are interested in predicting the house value:

1. and in an older but safer region with rm = 6, age = 80, crim = 0.1, lstat = 5;
2. in a newer but more risky region with rm = 6, age = 40, crim = 1, lstat = 1;
3. $\hat{y}_{new} = \mathbf{X}_{new}\hat{\boldsymbol{\beta}}$.



```
> new_data <- data.frame(age = c(80,40), lstat = c(5, 1), rm = c(6,6), crim = c(0.1,1))
> new_data
  age lstat rm crim
1  80     5  6  0.1
2  40     1  6  1.0
> predict(object = mod, newdata = new_data)
       1        2 
26.28232 28.13315 
```

But this is not the only possible predictive model, there are many other ways...

1. Reduce the model complexity by removing unnecessary features (i.e. removing age from the model)
2. Expand the model flexibility by adding interaction, polynomial (i.e. adding age $\times$ rm, adding age^2).
3. Fitting model on the transformed scales (regressing on $\sqrt{medv}$ then transforming back to the original scale).

Which model should we use for prediction?

```
> # smaller model:
> mod1 <- lm(medv ~ crim + rm  + lstat, data = bos_data)
> predict(object = mod1, newdata = new_data)
       1        2 
25.83676 28.05805 
>
> # bigger model:
> mod2 <- lm(medv ~ crim + rm + age + I(age^2) + age:rm + lstat, data = bos_data)
> predict(object = mod2, newdata = new_data)
       1        2 
26.56024 26.83932 
>
> # transformed model:
> mod3 <- lm(I(sqrt(medv)) ~ crim + rm + age + lstat, data = bos_data)
> (predict(object = mod3, newdata = new_data))^2
       1        2 
26.37293 28.72395 
```

One natural criterion is the Mean Square Error (MSE):

Definition (Mean Square Error)

Let y denotes the outcome, and $\hat{y}(x)$ denotes the prediction constructed using the feature x, then:

$$\text{MSE}(\hat{y}(x)) = \mathbb{E}\left[(y - \hat{y}(x))^2\right].$$

1. The expectation can be taken over $p(y|x)$ or $p(y, x)$, depending on the application (i.e. whether the value of the feature is fixed).
2. If the expectation is taken over $p(y|x)$, the MSE will be a function of x.
3. MSE is not the only criterion; for certain application, MAE $\mathbb{E}\left[|y - \hat{y}(x)|\right]$ is popular.

Claim: At a given value of x, the best deterministic prediction in terms of MSE is $\hat{y}(x) = \mathbb{E}(y|x)$.

Proof: Let $\mathbb{E}$ be taken over $p(y|x)$, then

$$\begin{aligned}
\mathbb{E}\left[(y-\hat{y}(x))^2|x\right] =& \mathbb{E}\left[(y - \mathbb{E}(y|x) + \mathbb{E}(y|x) - \hat{y}(x))^2|x\right] \\
=& \mathbb{E}\Big[(y-\mathbb{E}(y|x))^2 + (\mathbb{E}(y|x)-\hat{y}(x))^2 \\
& + 2(y-\mathbb{E}(y|x))(\mathbb{E}(y|x)-\hat{y}(x))|x\Big] \\
=& \mathbb{E}\left[(y-\mathbb{E}(y|x))^2|x\right] + (\mathbb{E}(y|x)-\hat{y}(x))^2 \\
& + 2\mathbb{E}\left[(y-\mathbb{E}(y|x))|x\right](\mathbb{E}(y|x)-\hat{y}(x)) \\
=& \underbrace{\mathrm{Var}(y|x)}_{\text{not depend on } \hat{y}(x)} + \underbrace{(\mathbb{E}(y|x)-\hat{y}(x))^2}_{\text{non-negative}}.
\end{aligned}$$

Therefore, $\hat{y}(x) = \mathbb{E}(y|x)$ minimizes the MSE.

What does this imply?

1. When (y, x) is joint-normal, justifies the use of linear regression (why?)
2. If $p(y|x)$ is known, then $\mathbb{E}(y|x)$ can be computed and used as the prediction to minimize MSE (but not for other criterion such as MAE).

But in reality..

1. $p(y|x)$ is never known... The best we can do is to estimate $p(y|x)$ using an observed dataset.
2. The constructed $\hat{y}(x)$ is no longer deterministic... Its randomness comes from the randomness of each dataset.
3. In other words, $\hat{y}(x)$ is a *random variable*, with variation across different observed datasets.

Revisit the MSE decomposition

Again, let's consider the MSE of $\hat{y}$ at a *particular* value of the feature x. But now, we assume $\hat{y}$ is constructed from a training dataset, *independent* of the target outcome y:

$$\begin{aligned}
\mathbb{E}\big[(y-\hat{y})^2|x\big] &= \mathbb{E}\big[(y-\mathbb{E}(\hat{y}|x)+\mathbb{E}(\hat{y}|x)-\hat{y})^2|x\big] \\
&= \mathbb{E}\big[(\hat{y}-\mathbb{E}(\hat{y}))^2|x\big]+\mathbb{E}\big[(y-\mathbb{E}(\hat{y}|x))^2|x\big] \\
&- 2\mathbb{E}\big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(\hat{y}|x))|x\big] \\
&= \mathrm{Var}(\hat{y}|x)+\mathbb{E}\big[(y-\mathbb{E}(y|x)+\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x))^2|x\big] \\
&- 2\mathbb{E}\big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(\hat{y}|x))|x\big] \\
&= \mathrm{Var}(\hat{y}|x)+\mathrm{Var}(y|x)+(\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x))^2 \\
&- 2\mathbb{E}\big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(\hat{y}|x))|x\big]
\end{aligned}$$

MSE decomposition continued

To take a look at the last term:

$$\begin{aligned}
&\mathbb{E}\Big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(\hat{y}|x))|x\Big] \\
&= \mathbb{E}\Big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(y|x)+\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x))|x\Big] \\
&= \underbrace{\mathbb{E}\Big[(\hat{y}-\mathbb{E}(\hat{y}|x))(y-\mathbb{E}(y|x))|x\Big]}_{\text{Cov}(\hat{y},y|x)=0}+\underbrace{\mathbb{E}\Big[(\hat{y}-\mathbb{E}(\hat{y}|x)|x\Big]}_{0}(\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x)).
\end{aligned}$$

Therefore,

$$\mathbb{E}\Big[(y-\hat{y})^2|x\Big] = \underbrace{\text{Var}(\hat{y}|x)}_{\textit{variance}} + \underbrace{\text{Var}(y|x)}_{\textit{Bayes error}} + \underbrace{(\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x))^2}_{\textit{squared Bias}}$$

Bias Variance Tradeoff

$$\mathbb{E}\left[(y-\hat{y})^2|x\right] = \underbrace{\mathrm{Var}(\hat{y}|x)}_{variance} + \underbrace{\mathrm{Var}(y|x)}_{Bayes\ error} + \underbrace{(\mathbb{E}(y|x)-\mathbb{E}(\hat{y}|x))^2}_{squared\ bias}$$

- Adding flexibility into the model could decreases the bias, but may increases the variance.
- Simplifying the model structure could decreases the variance, but may increase the bias.
- No model can behave better than just the Bayes error. It is irreducible.

Bias Variance Tradeoff : Typical U-curve

We typically need to find a nice middle point to optimize the overall predictive performance.

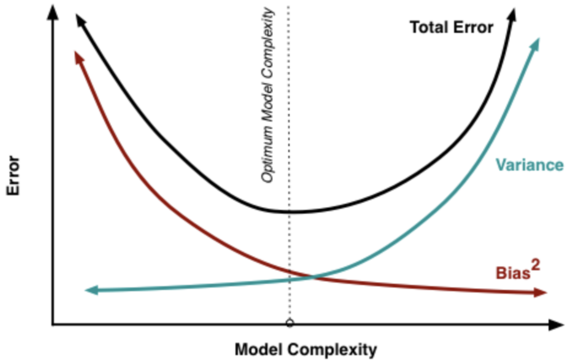


Figure: The total error (MSE) as a function of model-complexity.
source: http://scott.fortmann-roe.com/docs/BiasVariance.html

In practice, computation of expectation is too hard if not impossible. Therefore, most of the expectation is approximated by the sample mean.

Assume $\{(y_i, x_i)\}_{i=1}^{n}$ is a set of n i.i.d observations, the (unconditional) MSE can be approximated by:

$$\text{MSE}(\hat{y}) \approx \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}(x_i))^2.$$

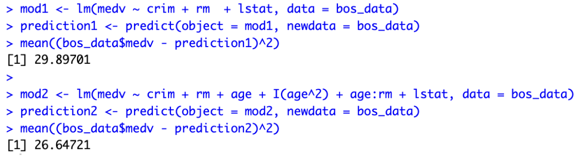
If there are m i.i.d observations that satisfy $x_i = x_0$ for some value x_0, the conditional MSE at $x = x_0$ can be approximated by:

$$\text{MSE}(\hat{y}(x_0)) \approx \frac{1}{m}\sum_{i:x_i=x_0}(y_i - \hat{y}(x_0))^2.$$

These are called empirical MSE.

When n is large, the empirical MSE should be very close to the true MSE, by LLN.

For Boston housing price data, $n = 506$. Large enough for LLN to work!



```
> mod1 <- lm(medv ~ crim + rm  + lstat, data = bos_data)
> prediction1 <- predict(object = mod1, newdata = bos_data)
> mean((bos_data$medv - prediction1)^2)
[1] 29.89701
>
> mod2 <- lm(medv ~ crim + rm + age + I(age^2) + age:rm + lstat, data = bos_data)
> prediction2 <- predict(object = mod2, newdata = bos_data)
> mean((bos_data$medv - prediction2)^2)
[1] 26.64721
```

So based on the empirical MSE: the larger model is much better than the smaller model!
Have we reached a reasonable conclusion here?...

- Unfortunately, the answer is no. We compute the empirical MSE using the same dataset to train the models...
- We cannot evaluate student's performance by giving them the same set of questions from their homework exercises, as their final exam!
- Because they can just memorize the answers (*overfitting*), without the ability to generalize.
- Recall in our previous proof, we used the fact that $\hat{y}$ and y_{new} are independent...

Data separation:

We need to separate the dataset into (at least) two:

1. **Training Data**: The part of data that is used to train/construct each model (homework exercises for your model).
2. **Testing Data**: The part of data that is used to evaluate your final model performance, which must be kept secret until your model training and selection is done (final exam for your model).

How much data should we give to each of these categories?
It depends on the complexity of your model.

- Simpler parametric model such as a simple linear regression, requires less data to train.
- More complex nonparametric model typically requires large amount of data to train.

Data separation: Boston Housing Example

```
> # Data separation:
> set.seed(123)
> sample_indx <- sample(x = 1:nrow(bos_data), size = 300, replace = FALSE)
> training <- bos_data[sample_indx, ]
> testing <- bos_data[-sample_indx, ]
>
> mod1 <- lm(medv ~ crim + rm  + lstat, data = training)
> prediction1 <- predict(object = mod1, newdata = testing)
> mean((testing$medv - prediction1)^2)
[1] 33.71328
>
> mod2 <- lm(medv ~ crim + rm + age + I(age^2) + age:rm + lstat, data = training)
> prediction2 <- predict(object = mod2, newdata = testing)
> mean((testing$medv - prediction2)^2)
[1] 30.71367
>
> mod3 <- lm(I(sqrt(medv)) ~ crim + rm + age + lstat, data = training)
> prediction3 <- predict(object = mod3, newdata = testing)
> mean((testing$medv - (prediction3^2))^2)
[1] 28.63631
```

- Set seed before taking the random indices for reproducibility.
- Important to set replace = FALSE to avoid the sample overlap.

- Usually the testing MSE should be larger than the training MSE, because most model is trained by minimizing the training MSE or some similar notion.
- The testing MSE also takes into account the variance part of the bias-variance trade-off; whereas training MSE only measures the bias part.
- Once you see the testing performance, you cannot go back and change your model-choice.

Error curve for training and testing data

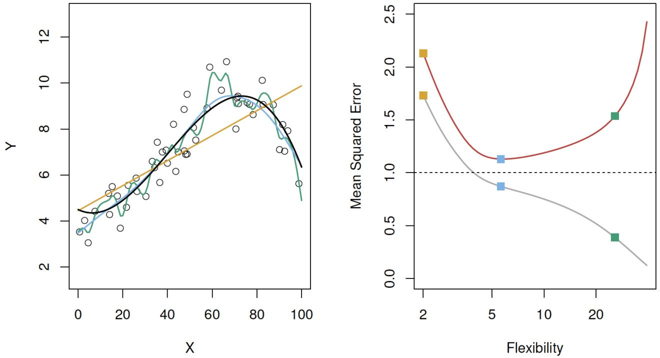


Figure: Source: James et al., 2021.

- Yellow point: underfitting. Error dominated by bias.
- Green point: overfitting. Error dominated by variance.
- Blue point: optimal point. Error is mostly Bayes error.

Limitation of linear regression

Linear regression model is a simple and interpretable tool in ML/statistics, however:

1. The model is linear in the regression parameters, and may not accommodate well more complex relationship.
2. Adding more features (i.e. $\text{age}^2, \text{age}^3, \text{age}^4$) is not always effective. MLE cannot be computed when $p > n$.
3. For such problem, a more flexible non-parametric approach could be helpful.

K Nearest Neighbourhood (KNN)

KNN regression is a non-parametric substitute for linear regression. Given an integer K and target feature vector $\mathbf{x}_0$, the algorithm proceeds as follow:

1. For each training data, compute the distance between its features $\mathbf{x}_i$ and $\mathbf{x}_0$:

$$d_i = ||\mathbf{x}_i - \mathbf{x}_0||_2 = \sqrt{\sum_{d=1}^{p} |x_{id} - x_{0d}|^2}.$$

2. Rank the distance $\{d_i\}_{i=1}^{n}$ from smallest to largest. Define $\mathcal{N}_0$ to be the indices of the nearest K points.
3. Compute

$$\hat{y}(\mathbf{x}_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} y_i.$$

Property of KNN

For KNN, the bias-variance tradeoff is controlled by the choice of K:

1. When $K = n$, $\hat{y}(\mathbf{x}_0)$ is just the sample mean $\bar{y}$.
2. When $K = 1$, $\hat{y}(\mathbf{x}_0)$ is just the y_i whose feature is closest to x_0.
3. As K increases, the bias becomes larger, but the variance becomes smaller.

KNN when $p = 1$

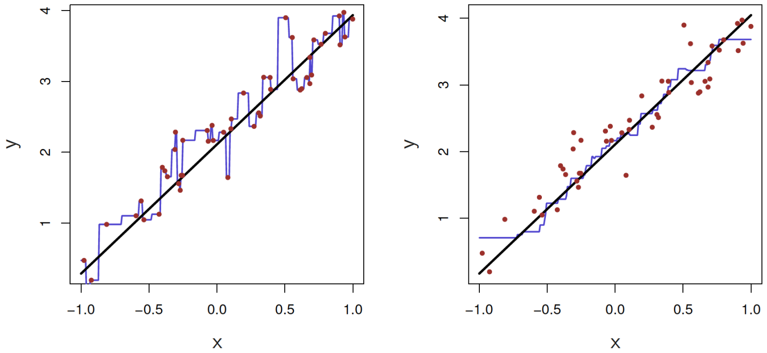


Figure: Left: K = 1. Right: K = 9; Truth is in black. Source: James et al., 2021.

KNN when $p = 2$

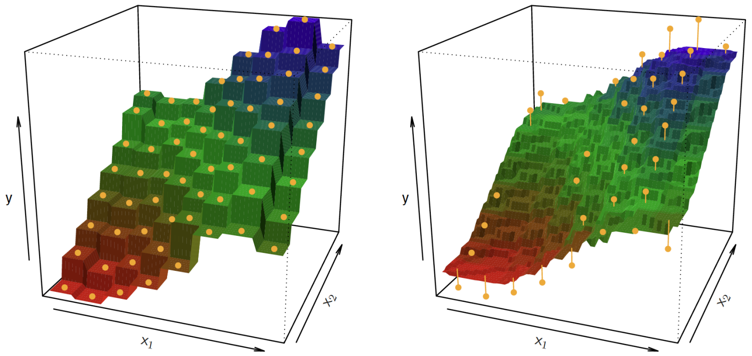
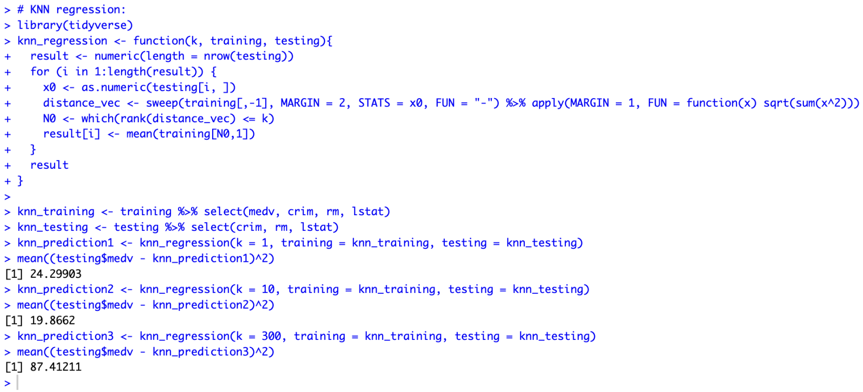


Figure: Left: K = 1. Right: K = 9; Source: James et al., 2021.



```
> # KNN regression:
> library(tidyverse)
> knn_regression <- function(k, training, testing){
+   result <- numeric(length = nrow(testing))
+   for (i in 1:length(result)) {
+     x0 <- as.numeric(testing[i, ])
+     distance_vec <- sweep(training[,-1], MARGIN = 2, STATS = x0, FUN = "-") %>% apply(MARGIN = 1, FUN = function(x) sqrt(sum(x^2)))
+     N0 <- which(rank(distance_vec) <= k)
+     result[i] <- mean(training[N0,1])
+   }
+   result
+ }
>
> knn_training <- training %>% select(medv, crim, rm, lstat)
> knn_testing <- testing %>% select(crim, rm, lstat)
> knn_prediction1 <- knn_regression(k = 1, training = knn_training, testing = knn_testing)
> mean((testing$medv - knn_prediction1)^2)
[1] 24.29903
> knn_prediction2 <- knn_regression(k = 10, training = knn_training, testing = knn_testing)
> mean((testing$medv - knn_prediction2)^2)
[1] 19.8662
> knn_prediction3 <- knn_regression(k = 300, training = knn_training, testing = knn_testing)
> mean((testing$medv - knn_prediction3)^2)
[1] 87.41211
>
```

For next week:

- We will cover k-fold cross-validation, model regularization, and principal component analysis.
- The problem set 1 will be posted today. Remember to bring the printout of the computation question (code + output) to the tutorial.
- You need to take the quiz in your enrolled tutorial section.