

Introduction to Statistical Learning and Machine Learning

Mid-term Review



Chap1

Introduction

- Introduction to Machine Learning



What is Machine Learning?

- **Definition of ML (Mitchell, 1997): WELL-POSED LEARNING PROBLEMS.**
 - A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .
- **Example: A computer program that learns to play checkers**
 - Task: playing checkers games;
 - Experience: obtained by playing games against itself;
 - Performance Measure: percent of games won against opponents



Notations, formally

Task:

\mathcal{X} input variables (from input set), a.k.a., features, predictors, independent variables.

\mathcal{Y} output variables (from output set), a.k.a., response or dependent variable.

$f : \mathcal{X} \rightarrow \mathcal{Y}$ Prediction function,

Performance:

$l : \mathcal{X} \rightarrow \mathcal{Y}$ Loss function,

$l(y, y')$ is the cost of predicting y' if y is correct.

Experience: task-dependent, many different scenarios

- Supervised Learning, Unsupervised Learning, Reinforcement Learning,
- Semi-supervised Learning, Multiple Instance Learning, Active Learning.



Supervised Learning

- A labeled training set examples with outputs provided by an expert,

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$$

- Regression Vs. Classification problems,

- **Regression:** Y is quantitative (e.g price, blood pressure);
- **Classification:** Y takes values in a finite, unordered set (survived/died, digit 0-9, cancer class of tissue sample), qualitative.

Definition,

- A supervised learning system (or learner), L is a (computable) function from the set of (finite) training sets to the set of prediction functions:

$$L : \mathbb{P}^{<\infty}(\mathcal{X} \times \mathcal{Y}) \rightarrow \mathcal{Y}^{\mathcal{X}}$$
$$L : \mathcal{D} \mapsto f$$

So if presented with a training set \mathcal{D} , it provides a decision rule/function

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

Let L be a learning system.

- Process of computing is $f = L(\mathcal{D})$ called training (phase).
- Applying f to new data is called prediction, or testing. (phase).



Toy example: *How grade will I get in this course?*

General workflow of SL.

- **Data:** entry survey and marks from previous years
- **Process the data:**
 - Split into **training set; test set;**
 - Representation of **input features;** output
- Choose form of model: **linear regression**
- Decide how to evaluate the system's performance: **objective function**
- Set model parameters to optimize performance
- Evaluate on test set: **generalization**

CSC411/CSC2515: Entry Survey

Which course are you taking?

- CSC411
 CSC2515

Name

Student Number

Major

Years Until Graduation

- 1 2 3 4 5

Status

Email

Familiarity with Bayes Rule

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Maximum A Posteriori

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Logistic Regression

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Gradient Descent

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Chain Rule

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Matlab

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Python

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with Belief Networks

- Proficient
 Confortable
 Rusty
 Hunh?

Familiarity with EigenVectors

- Proficient
 Confortable
 Rusty
 Hunh?

What related courses have you taken?

e.g., CSC321, CSC384

credit: Urtasun & Zemel (University of Toronto)



Fudan-SDS Confidential - Do Not Distribute



大数据学院
School of Data Science

Toy example: *How grade will I get in this course?*

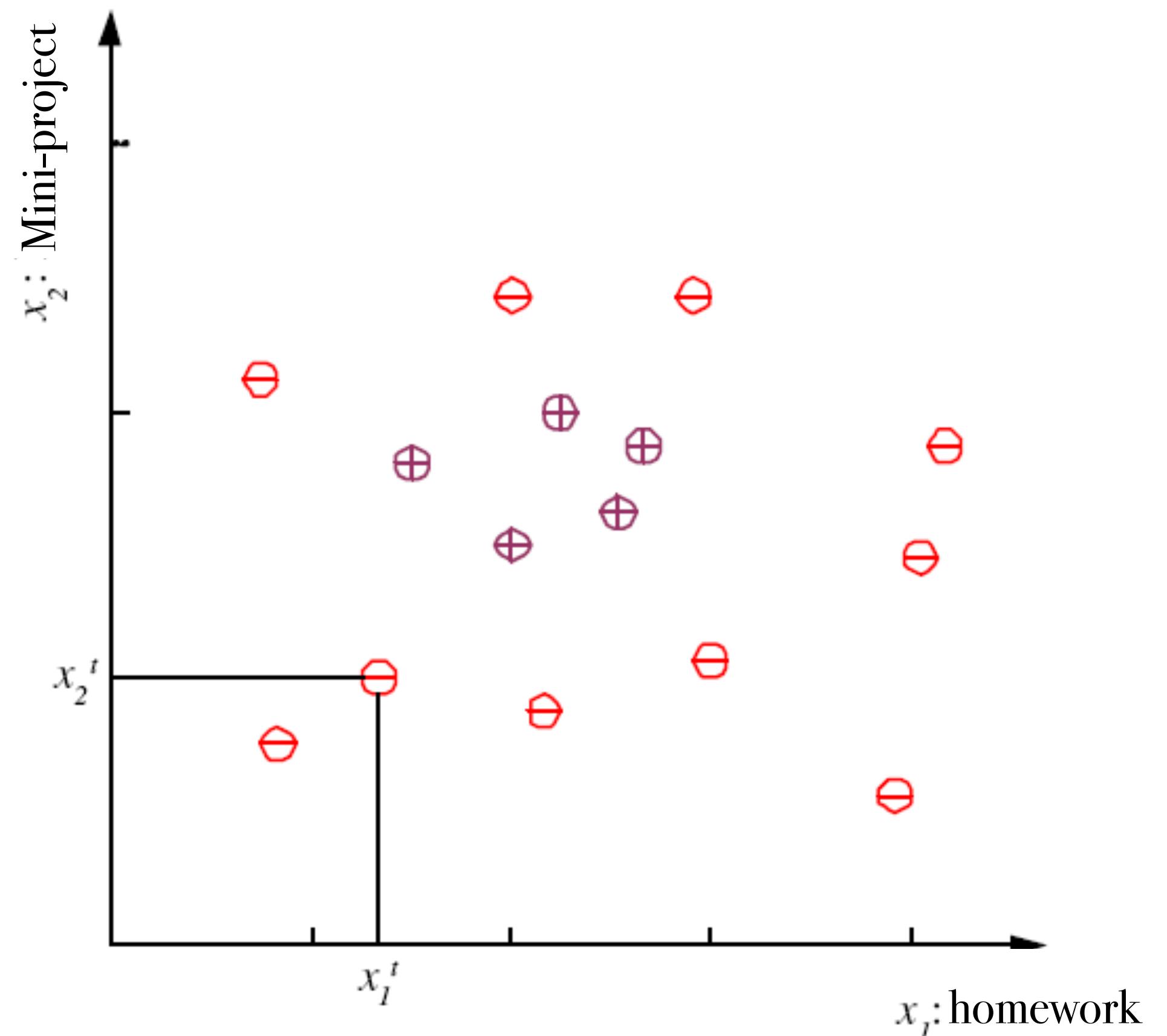
Settings:

- Class C of a “good score”
 - Knowledge extraction: What do people expect from a good score?
- Output:
 - Positive (+) and negative (−) examples
- Input representation:
 - x_1 : homework, x_2 : Mini-projects



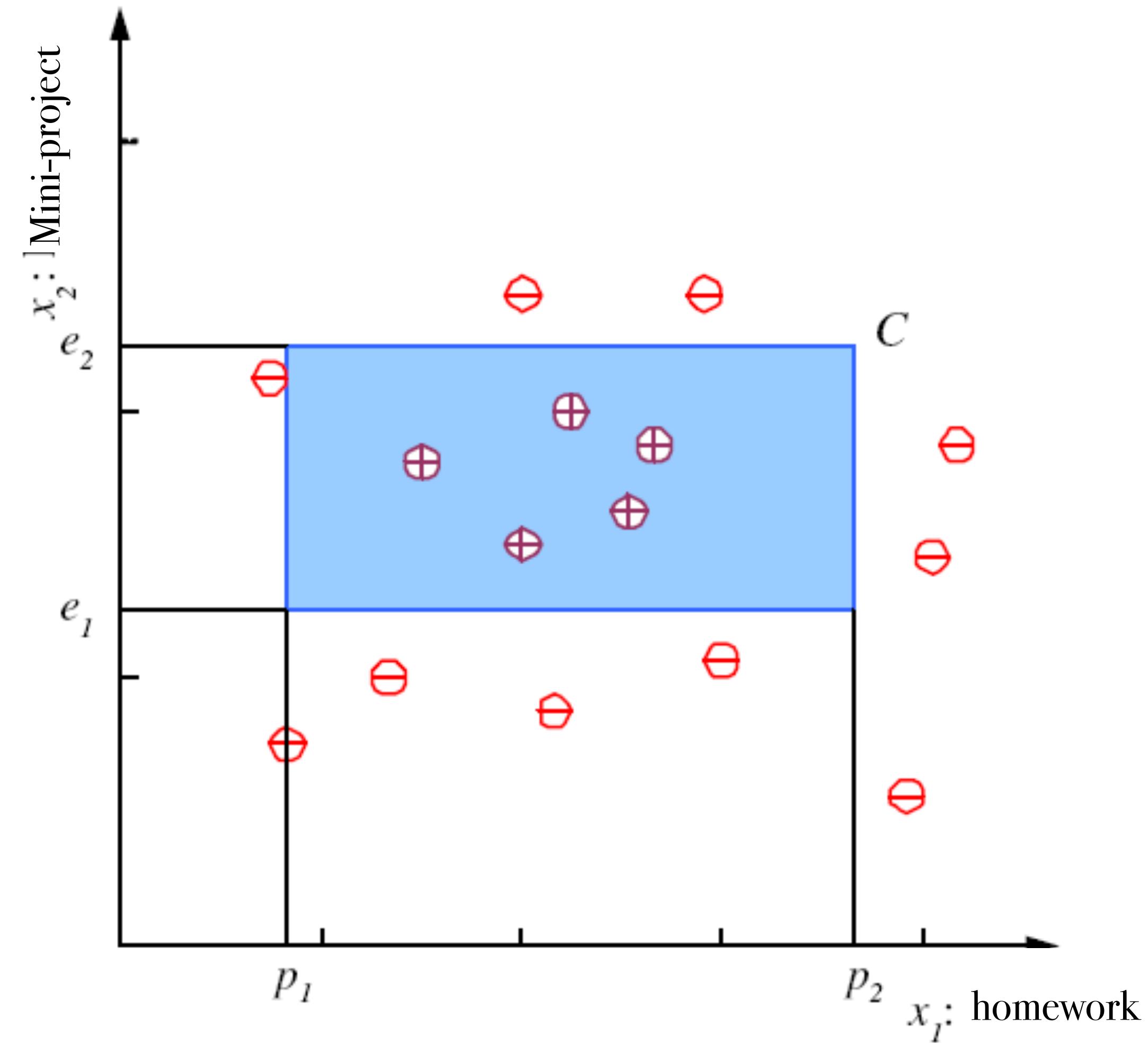
Training Set

$$\mathcal{D} = \{(x^1, y^1), (x^2, y^2), \dots, (x^N, y^N)\} \subset \mathcal{X} \times \mathcal{Y}$$



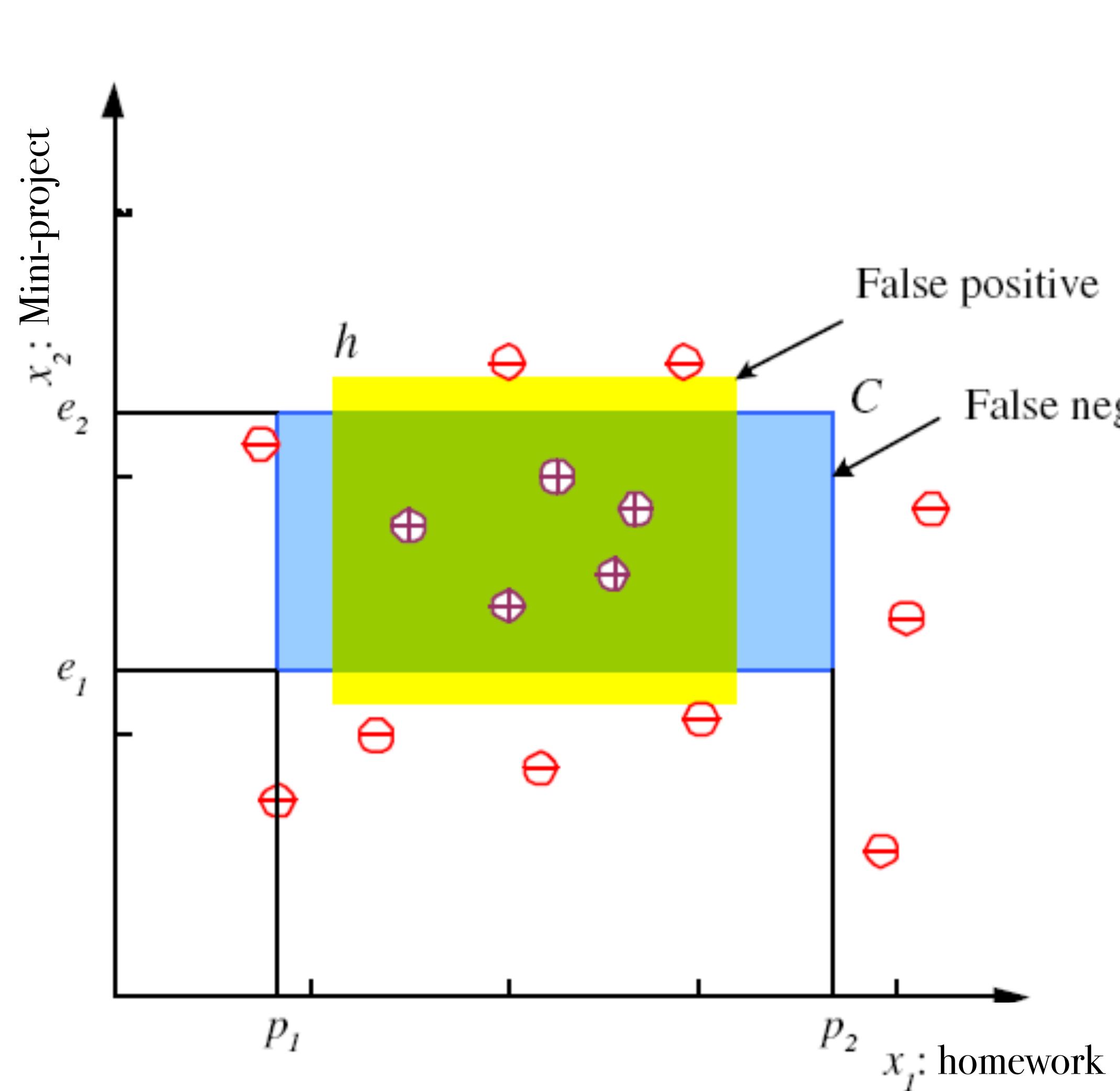
$$y = \begin{cases} 1 & \text{if } x \text{ is positive} \\ 0 & \text{if } x \text{ is negative} \end{cases}$$
$$x^t = \begin{bmatrix} x_1^t \\ x_2^t \end{bmatrix}$$

Class C



Tom M. Mitchell is an American computer scientist and E. Fredkin University Professor at the Carnegie Mellon University.

Hypothesis class \mathcal{H}

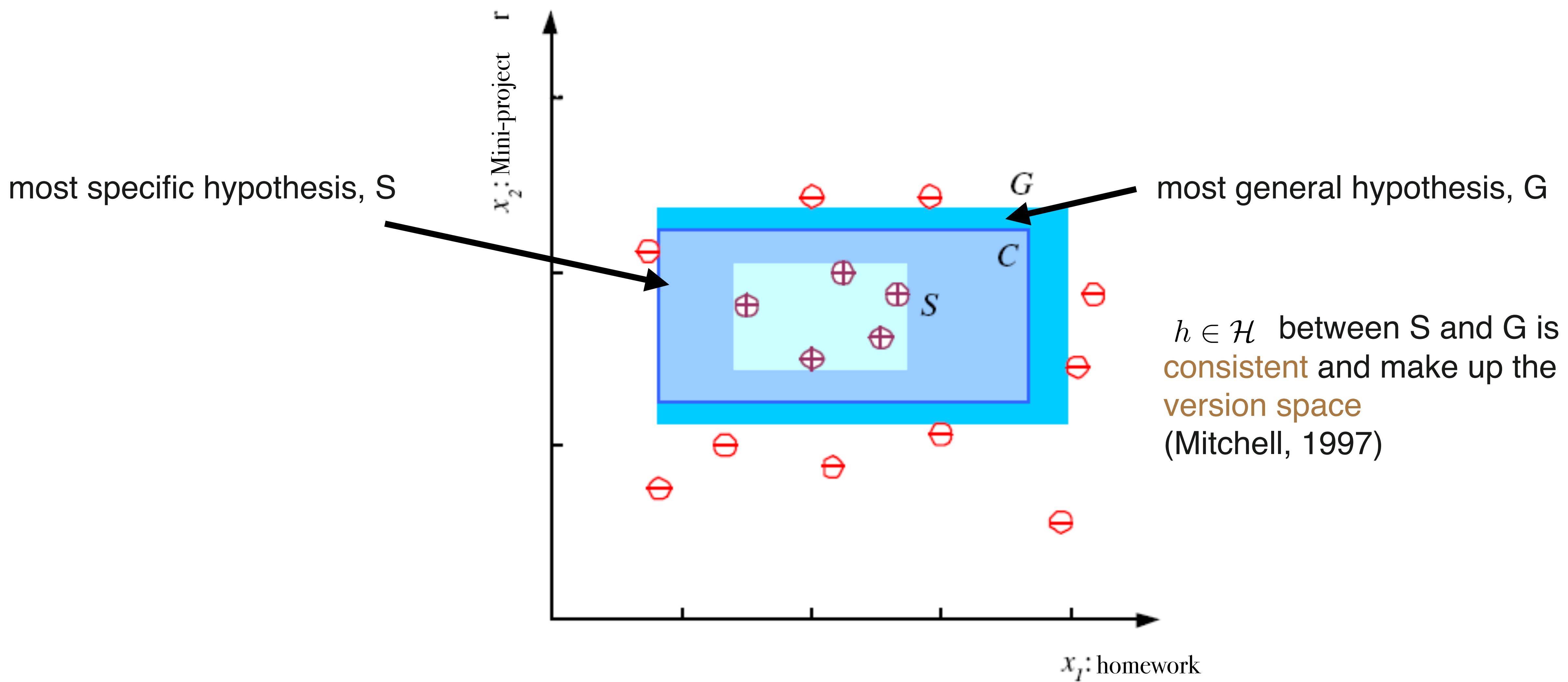


$$h(x) = \begin{cases} 1 & \text{if } x \text{ is positive} \\ 0 & \text{if } x \text{ is negative} \end{cases}$$

Error of h on \mathcal{H}

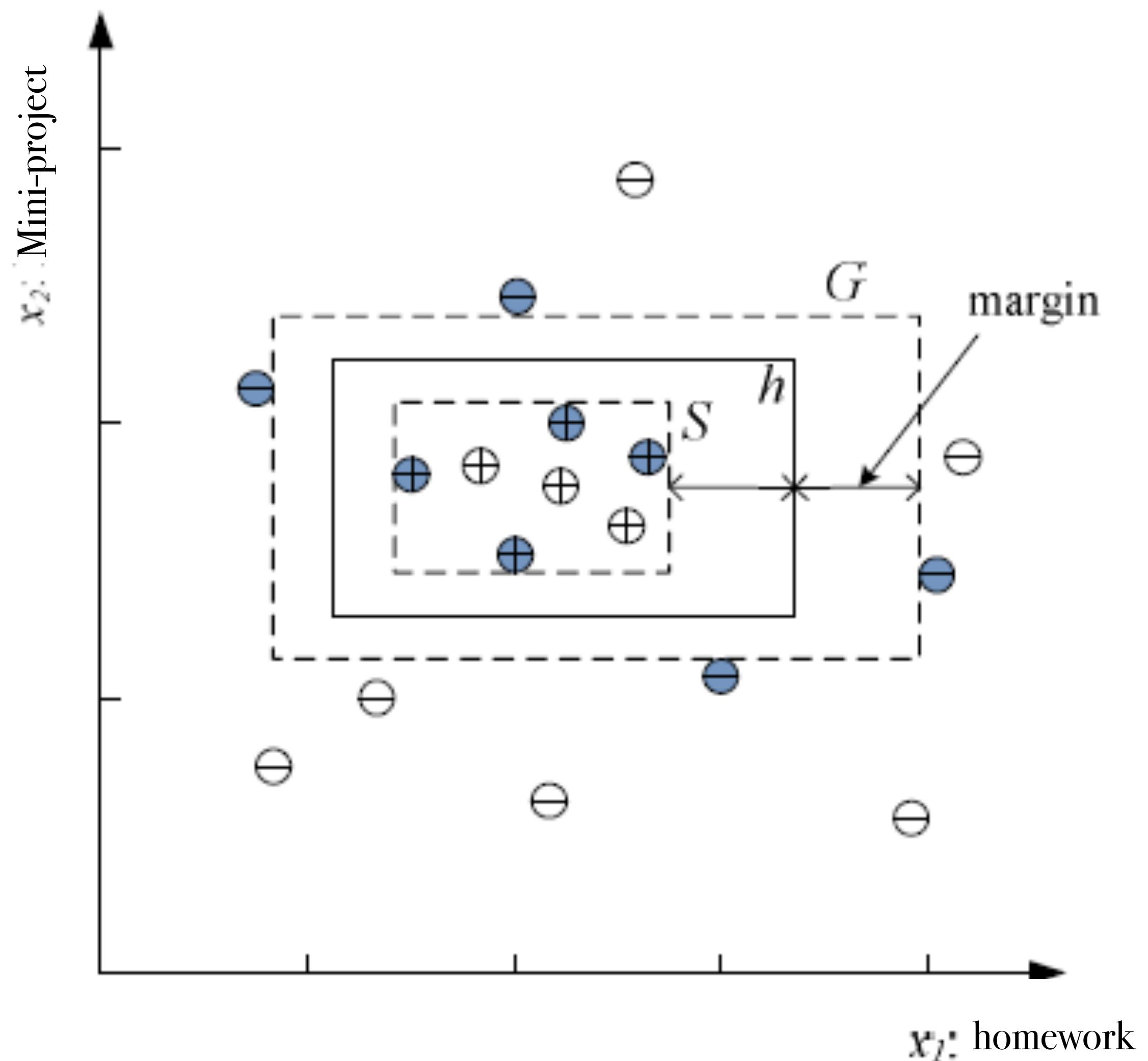
$$E(h|\mathcal{X}) = \sum_{t=1}^N \mathbf{1}(h(x^t) \neq y^t)$$

Version Space (Mitchell, 1997)



Margin

One solution:
Choose h with
largest margin



Chap 2、3

Linear Regression

- Linear Regression by least squares;
- The bias-variance tradeoff;



Simple Linear Regression

Parametric method

- **Simple Linear Regression:** Y is quantitative (e.g price, blood pressure); on the basis of a single predictor variable X .

$$Y \approx \beta_0 + \beta_1 X.$$

Symbols explanations:

- You might read “ \approx ” as “*is approximately modeled as*”;
- β_0 and β_1 are two unknown constants that represent the **intercept** and **slope** terms;
- saying that we are regressing Y on X (or Y onto X).
- hat symbol, $\hat{}$, to denote the estimated value for an unknown parameter or coefficient, or to denote the predicted value of the response.

So how to statistically estimate and analyse the Coefficients?



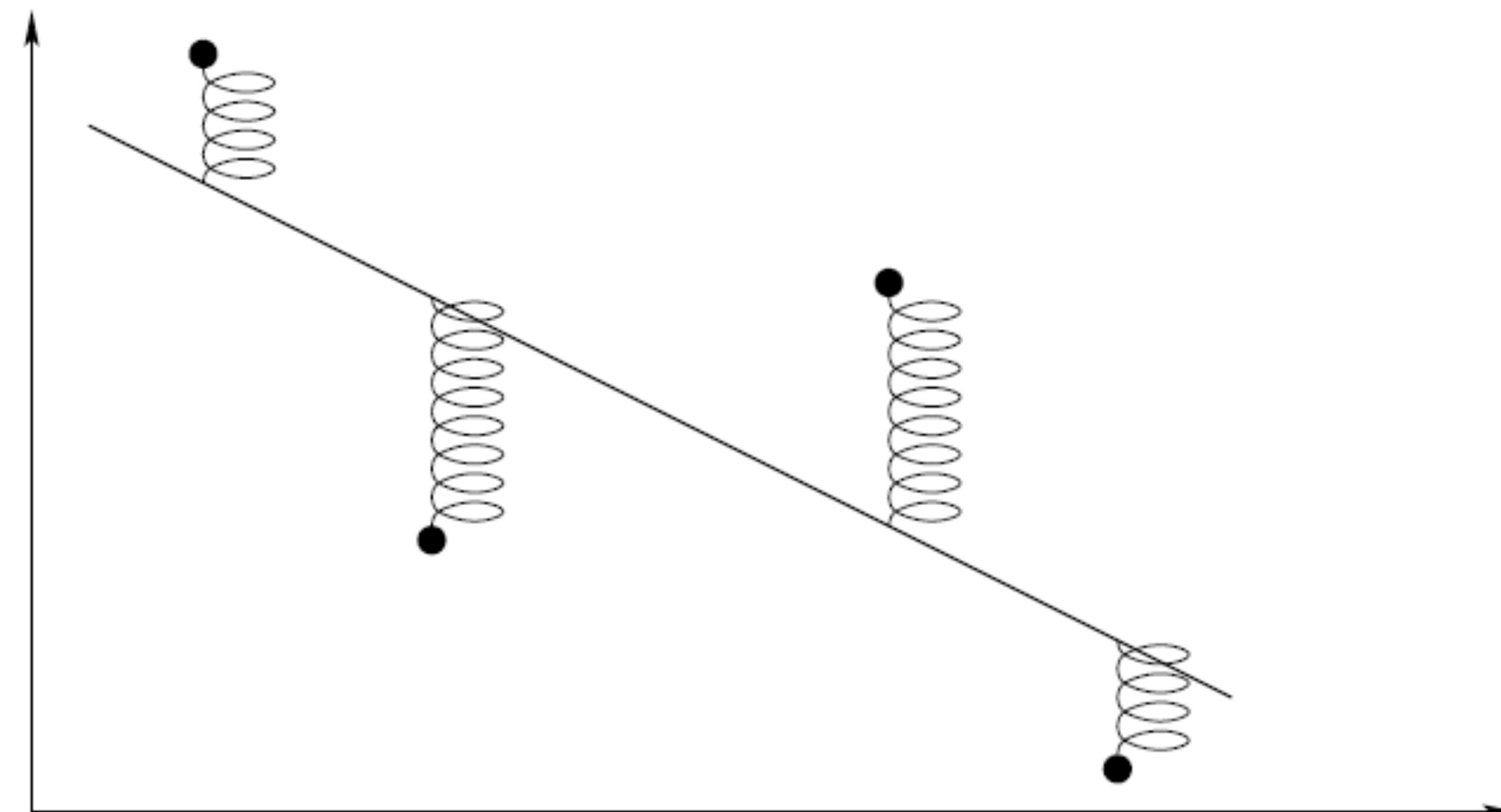
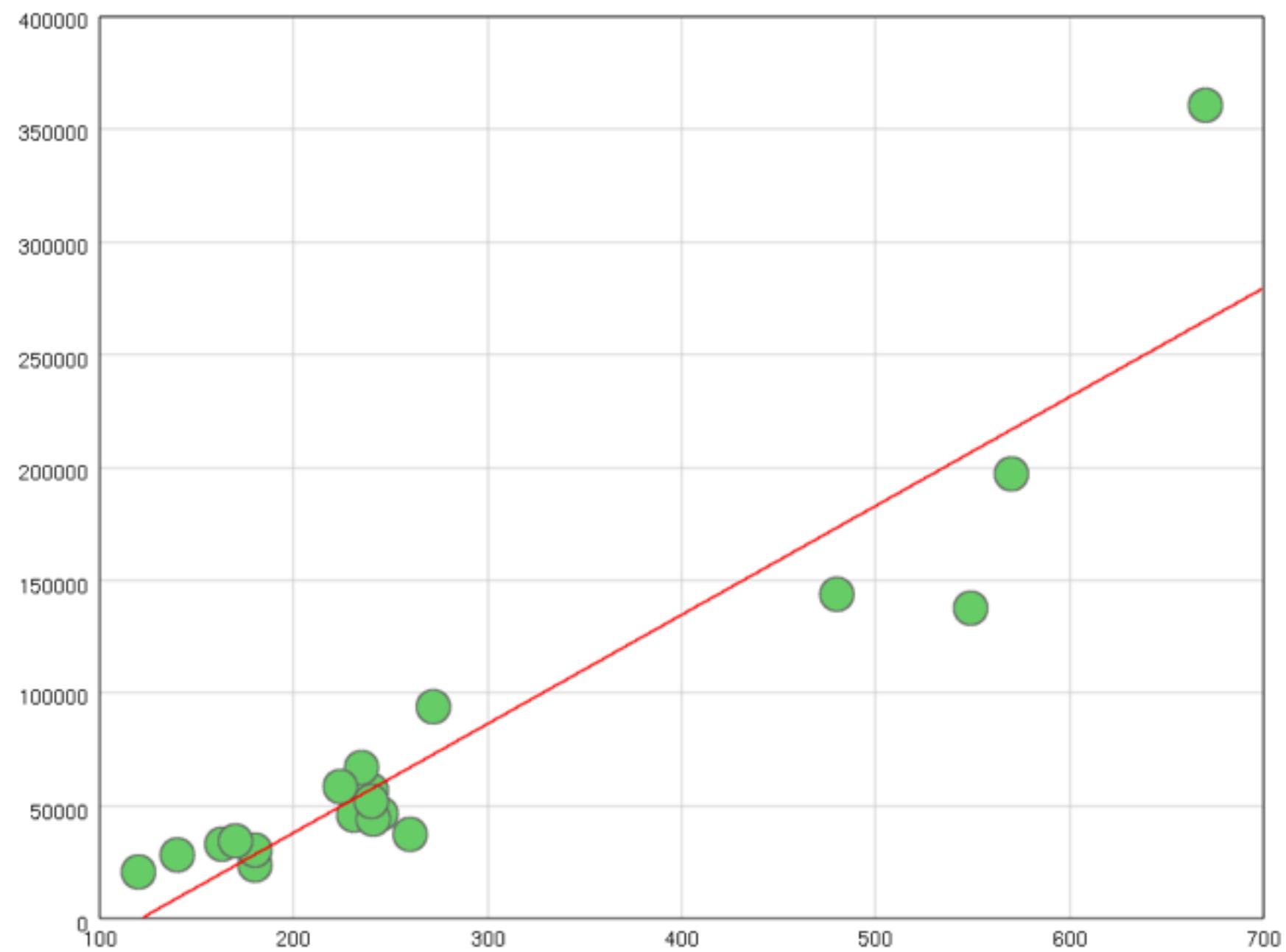
Optimizing the Objective

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

Standard loss/cost/objective function measures the squared error between Y and \hat{Y}

$$l(y, \hat{y}) = \sum_{i=1}^N [y_i - (\beta_0 + \beta_1 x_i)]^2$$

How do we obtain the parameters in general?



How to estimate and analyse the Coefficients?

Optional subtitle

- Least Square and Maximum Likelihood Estimate;
- Assessing the Accuracy of the Coefficient Estimates;
- Standard Error and Confidence Intervals;
- Hypothesis Testing;
- P-value



Insight of Linear Model

Polynomial Regression $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \dots + \beta_d x_i^d + \epsilon_i$

Bias-Variance Decomposition

assume: $y_i = f(x_i) + \epsilon_i$ for some function f and assume we have a “leaner” that make a training set \mathcal{D}

$$\epsilon_i \sim N(0, \theta^2)$$

Then for a new example (x_i, y_i) the error averaged over training sets is

$$E[(y_i - \hat{f}(x_i))^2] = \text{Bias}[\hat{f}(x_i)]^2 + \text{Var}[\hat{f}(x_i)] + \theta^2$$

Expected error due to
having wrong model.

where $\text{Bias}[\hat{f}(x_i)] = E[\hat{f}(x_i)] - f(x_i)$,

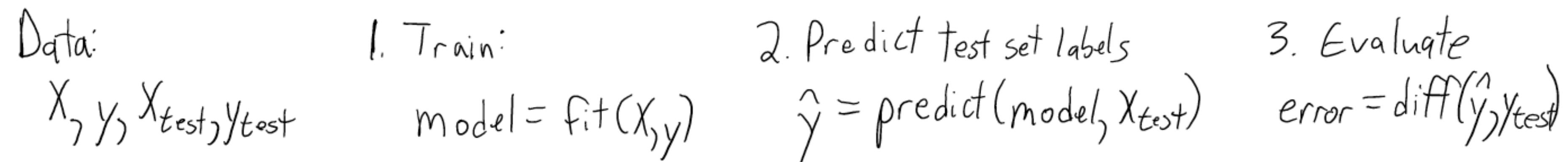
How sensitive is the model
to the particular training set? $\text{Var}[\hat{f}(x_i)] = E[(\hat{f}(x_i) - E[\hat{f}(x_i)])^2]$

“Irreducible
error”:
best we can
hope for
given the
noise level.



Supervised Learning Pipeline (Prepare for the Projects)

1, Given a training set X and y , **with i.i.d assumption** (training and test data drawn from same distribution), if we have an explicit test set to approximate test error:



2, What if we don't have an explicit test set?

Possible training procedures if you only have a training set:

- (1). Randomly split training set into “**train**” and “**validate**” set.
- (2). Train model based on **train set**.
- (3). Report validate set accuracy with this model.

$$X = \begin{bmatrix} train \\ \cdots \\ validate \end{bmatrix} \quad \bar{y} = \begin{bmatrix} train \\ \cdots \\ validate \end{bmatrix}$$



Tom Simonite
June 4, 2015

Why and How Baidu Cheated an Artificial Intelligence Test

Machine learning gets its first cheating scandal.

The sport of training software to act intelligently just got its first cheating scandal. Last month Chinese search company Baidu announced that its image recognition software had [inched ahead of Google's on a standardized](#)

**Golden rule: this test set cannot influence training in any way.
If you violate golden rule, you can overfit to the test data.**



Further issues of Supervised Learning

Optional subtitle

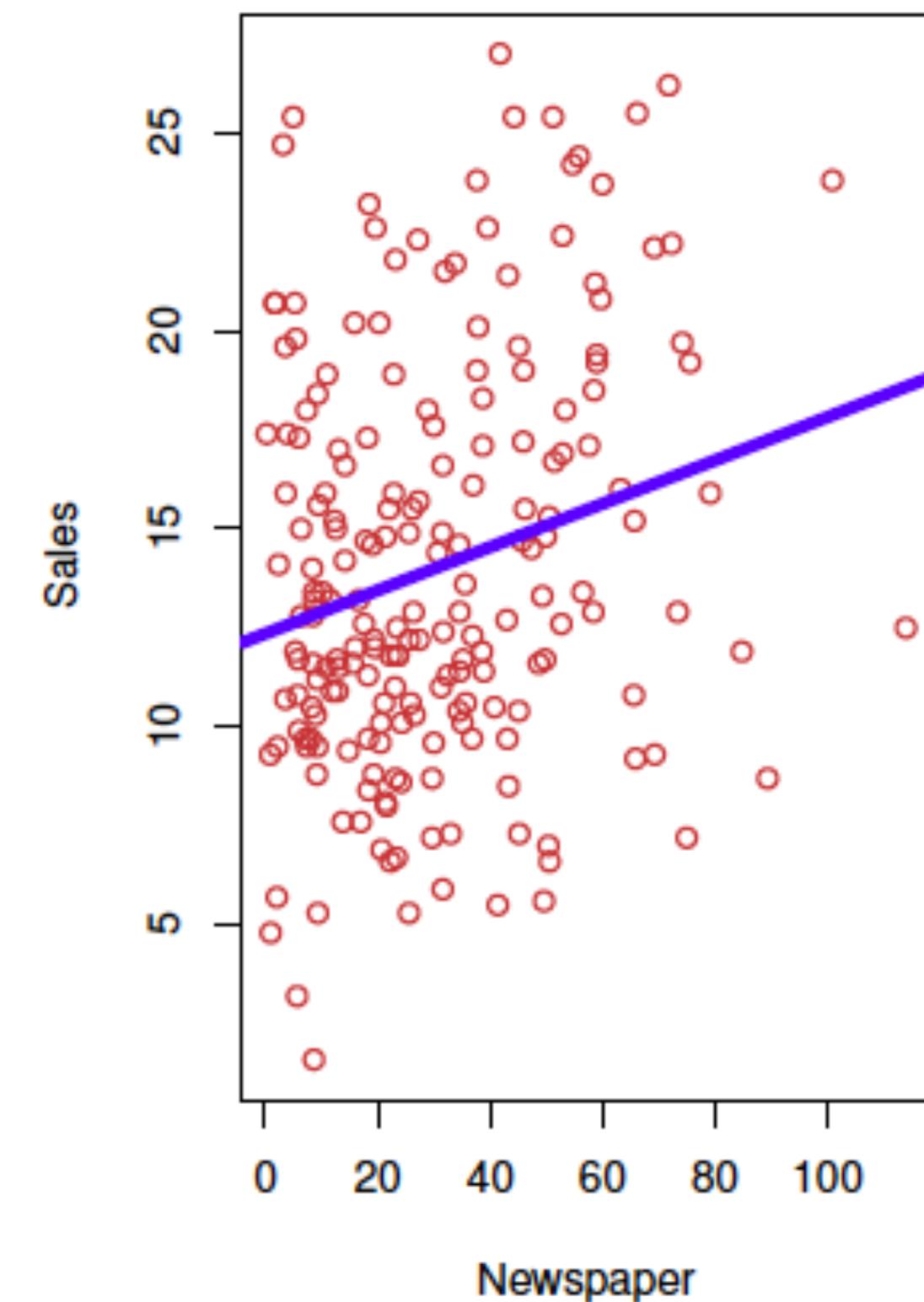
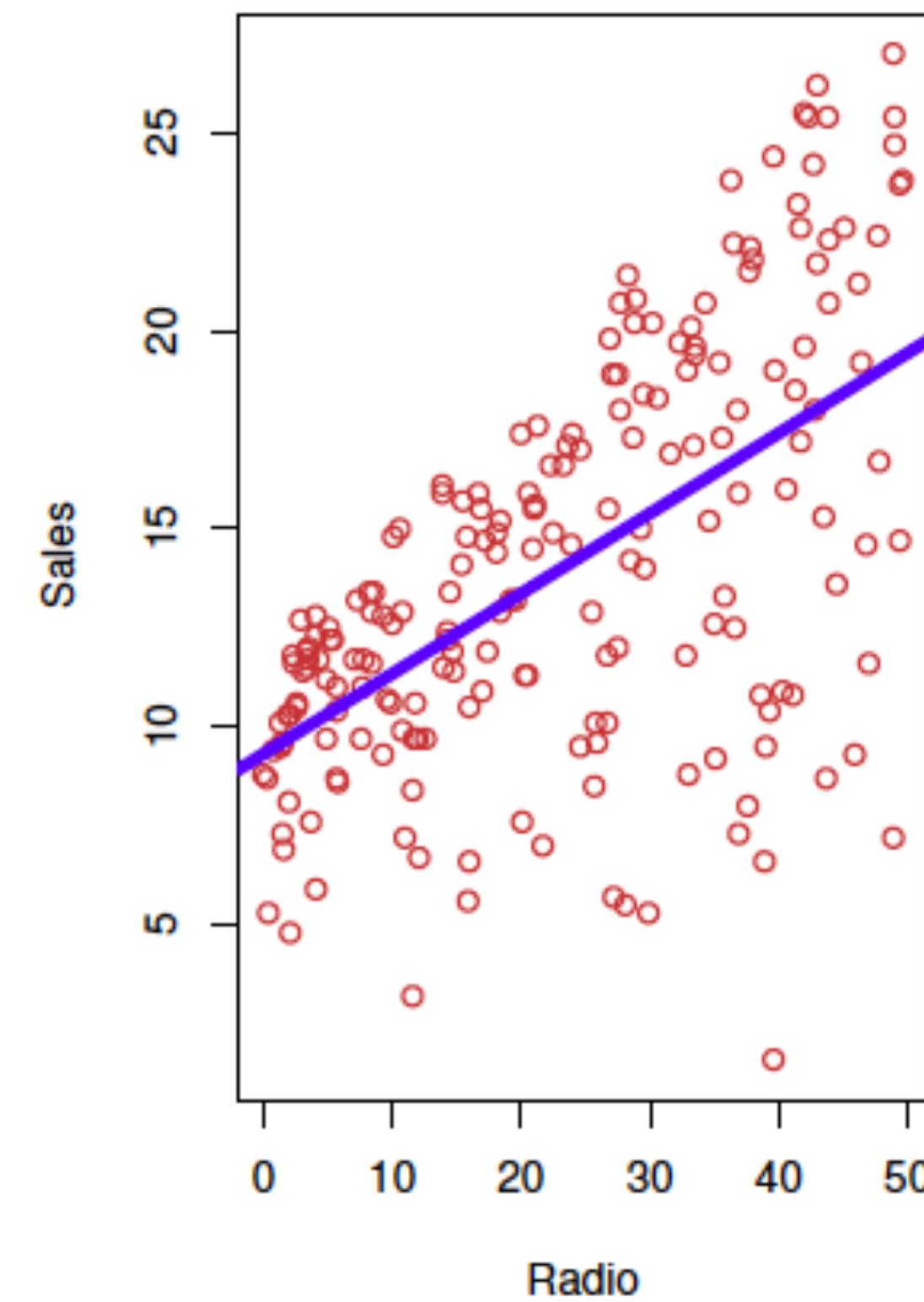
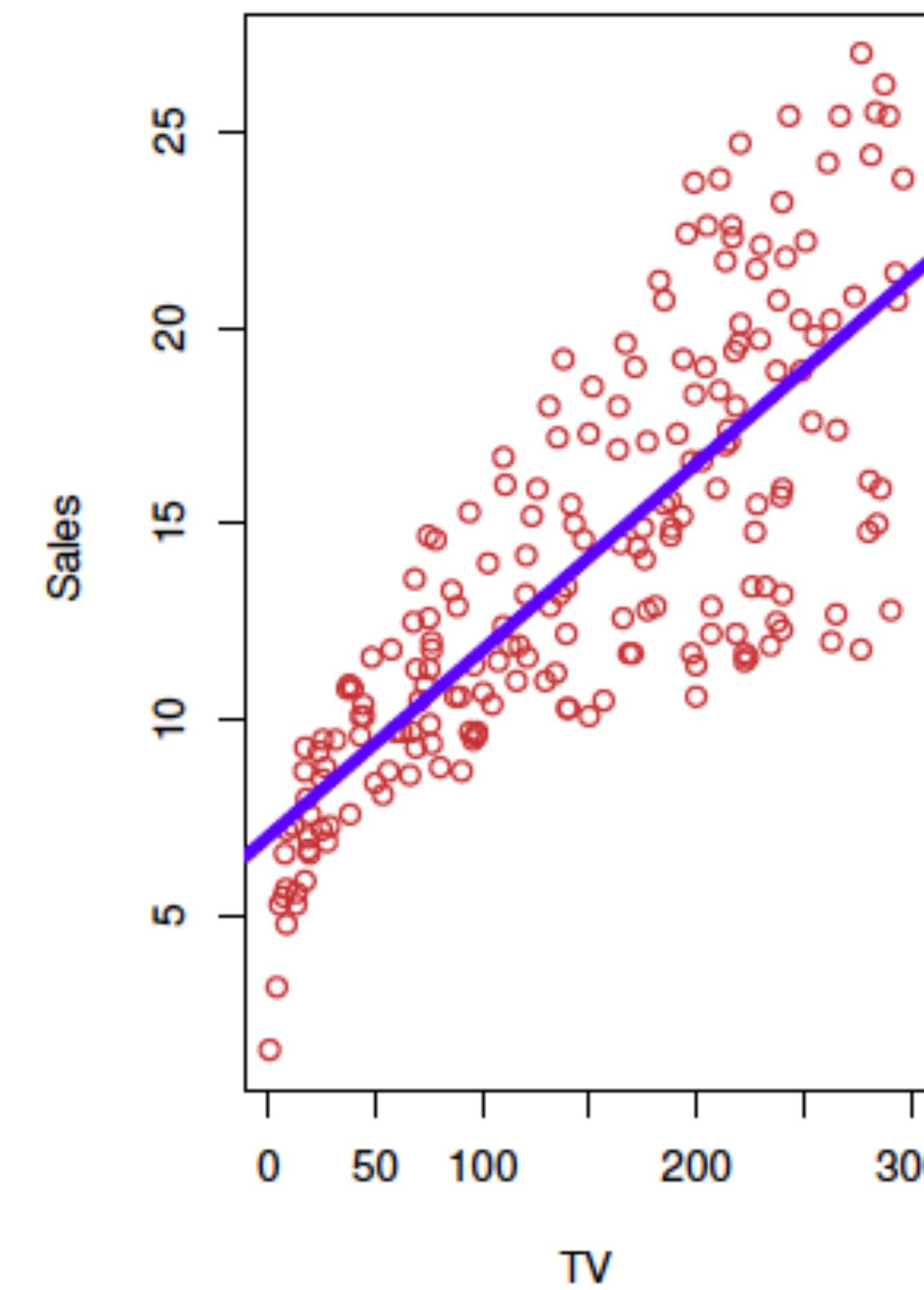
- Never Ever touch testing data (Not Overfitting);
- Loss Function
- K-fold Cross Validation for tuning hyper-parameter; (Model Selection)
- Accuracy, Confusion Matrix, ROC curve; (Measurement)



Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon,$$

$$\text{sales} = \beta_0 + \beta_1 \times \text{TV} + \beta_2 \times \text{radio} + \beta_3 \times \text{newspaper} + \epsilon.$$



Multiple Linear Regression

Sec 3.2 of “The Elements of Statistical Learning”

$$X^T = (X_1, X_2, \dots, X_p), \quad f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j.$$

RSS denotes the **empirical risk** over the training set. It doesn't assure the predictive performance over all inputs of interest.

Least squares to minimize the residual sum of squares:

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta).$$

$$\frac{\partial \text{RSS}}{\partial \beta} = -2\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial^2 \text{RSS}}{\partial \beta \partial \beta^T} = 2\mathbf{X}^T \mathbf{X}.$$

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0$$

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

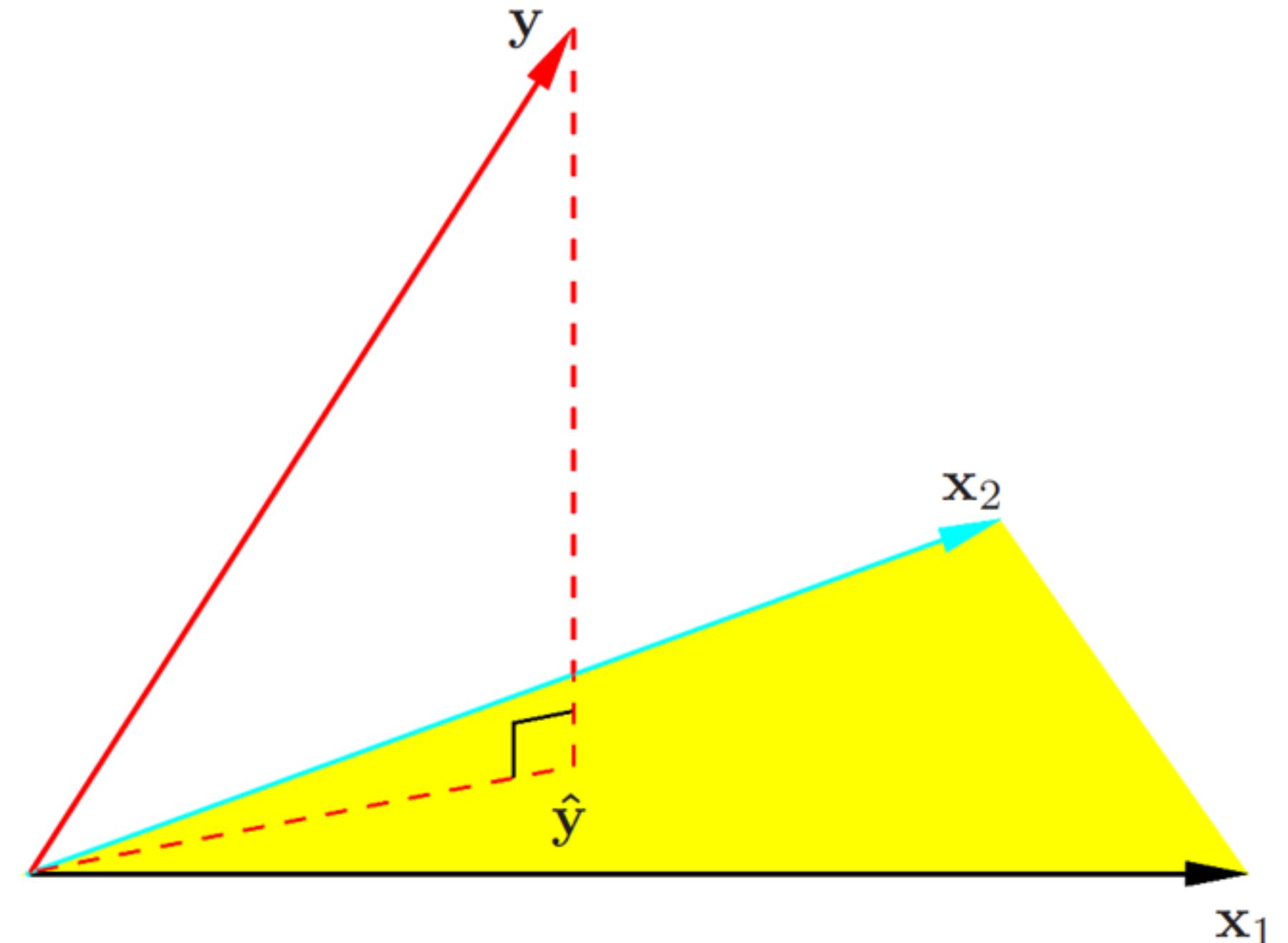
$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \boxed{\mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}},$$

Projection (Hat) matrix: $\mathbf{H} = \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$

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

Note that: For a unique solution, the matrix $\mathbf{X}^T \mathbf{X}$ must be full rank.

Orthogonal Projection of \mathbf{Y} on the space spanned by the columns of \mathbf{X} .



Geometric interpretation.

Alternatives instead of Least Squares

Optional subtitle

- **Prediction Accuracy:** especially when $p > n$, to control the variance.
- **Model interpretability:** By removing irrelevant features – that is, by setting the corresponding coefficient estimates to zero— we can obtain a model that is more easily interpreted.

Three methods to perform feature selection:

- **Subset Selection.** We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.
- **Shrinkage.** We fit a model involving all p predictors, but the estimated coefficients are shrunk towards zero relative to the least squares estimates. This shrinkage (also known as **regularization**) has the effect of reducing variance and can also perform variable selection.
- **Dimension Reduction.** We project the p predictors into a M -dimensional subspace, where $M < p$. This is achieved by computing M different **linear combinations**, or **projections**, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.



Shrinkage Methods(1)

- Ridge Regression

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^p \beta_j^2,$$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately.

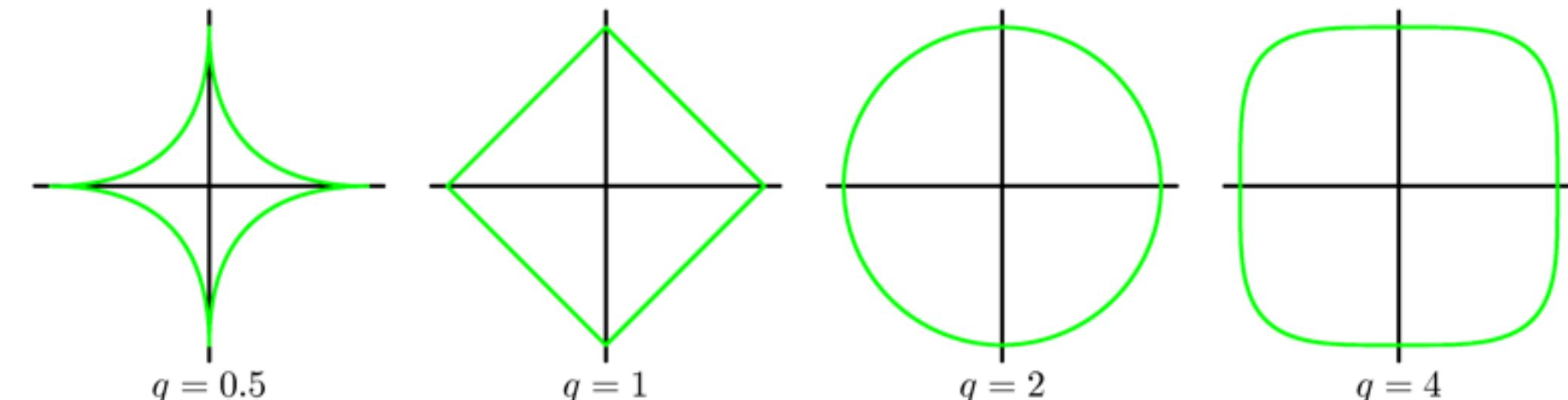
- Lasso

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \lambda \sum_{j=1}^p |\beta_j|.$$



Shrinkage Methods in Matrix Form

$$\operatorname{argmin}_{\beta} \| Y - X\beta \|_2^2 + \lambda \| \beta \|_q$$



$q=0$, L_0 -norm; \rightarrow finding the minimiser is NP-hard computational problem. (the Eq. is nonconvex).

- L_0 -norm has closed form solution [1].
- it is defined in Eq(6.10) of textbook. i.e., $\| \beta \|_0 = \#\sigma(\beta)$, $\#$ stands for cardinality; $\sigma(\beta)$ is the support of β

$q < 1$, **hard-thresholding**

$q=1$, L_1 -norm \rightarrow Lasso (convex), a.k.a., **soft-thresholding**.

$q \leq 1$ used for outlier detection [2,3].

$q=2$, L_2 -norm \rightarrow Ridge Regression (convex) $\|\beta\|_2 = \sqrt{\sum_{j=1}^p \beta_j^2}$.

Note: (1) tuning the parameter λ is very important.

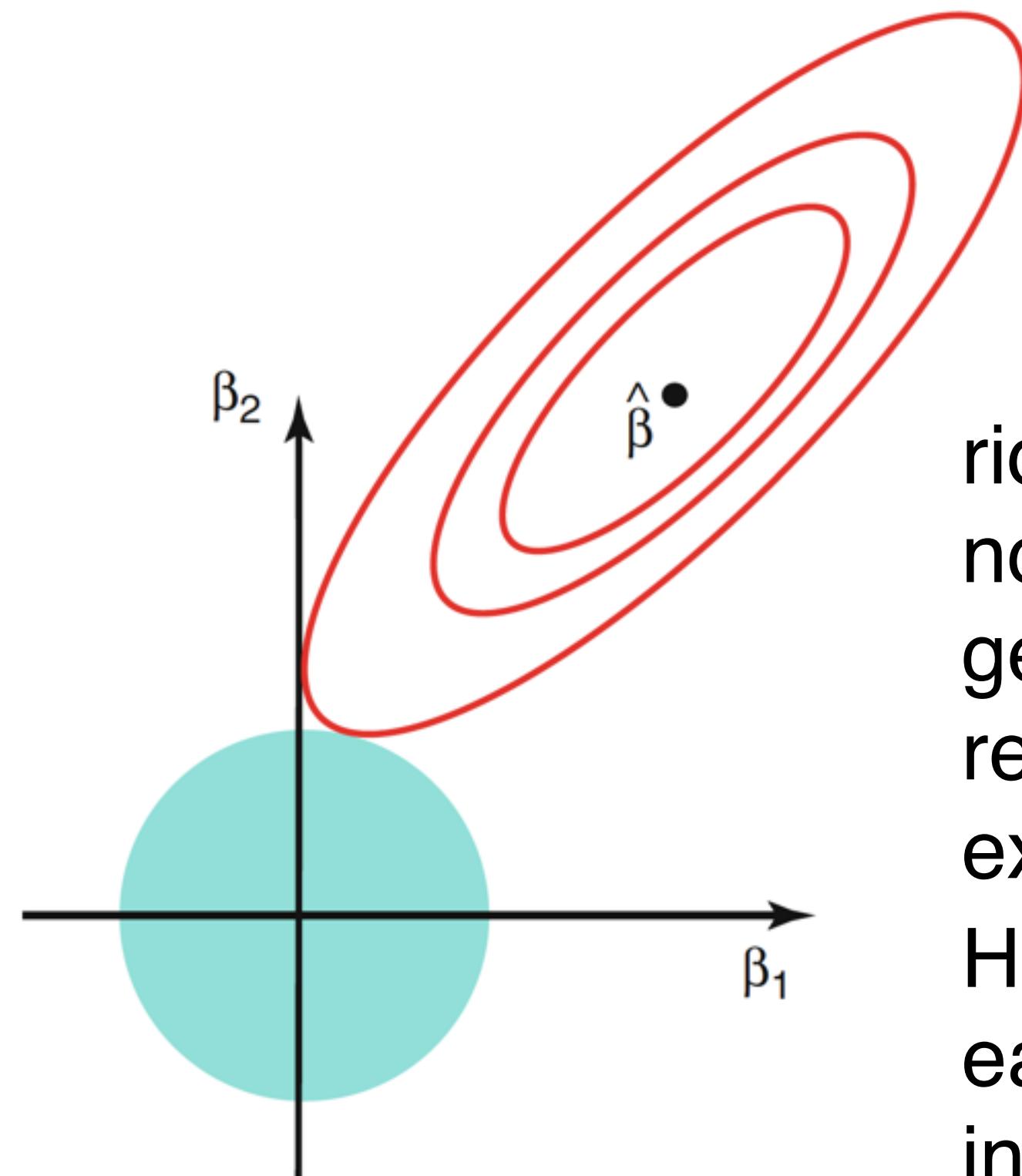
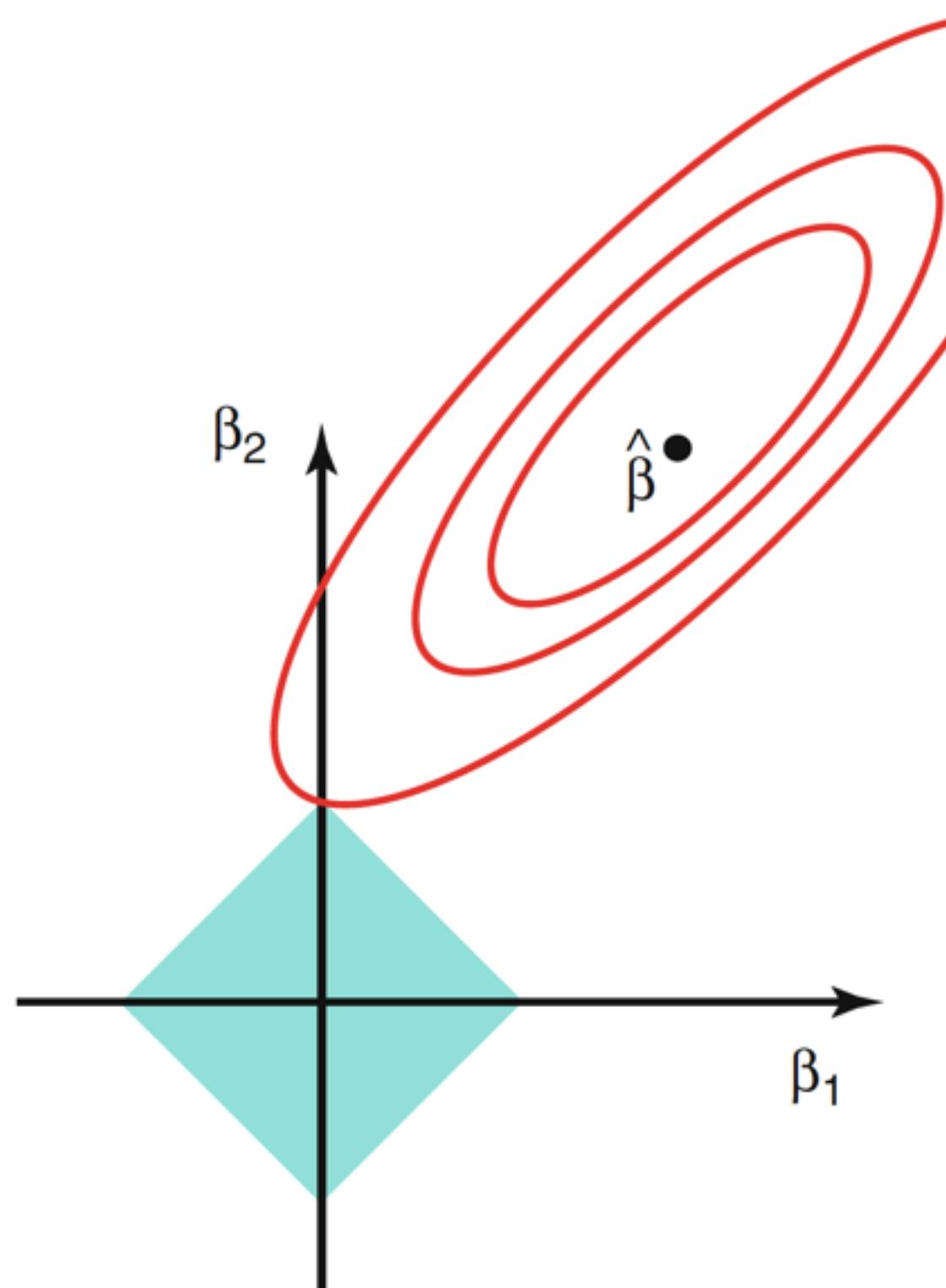
$$\| \beta \|_q = \left(\sum_{i=1}^p |\beta_i|^q \right)^{\frac{1}{q}}$$

[1] Mila Nikolova, Description of the minimizers of least squares regularized with ℓ_0 -norm. Uniqueness of the global minimizer, SIAM J. IMAGING SCIENCE 2013.

[2] Yiyuan She, and Art B. Owen, Outlier Detection Using Nonconvex Penalized Regression, 2011. Journal of the American Statistical Association

[3] Yanwei Fu et al. Robust Subjective Visual Property Prediction from Crowdsourced Pairwise Labels. IEEE Transaction on Pattern Analysis and Machine Intelligence, 2016

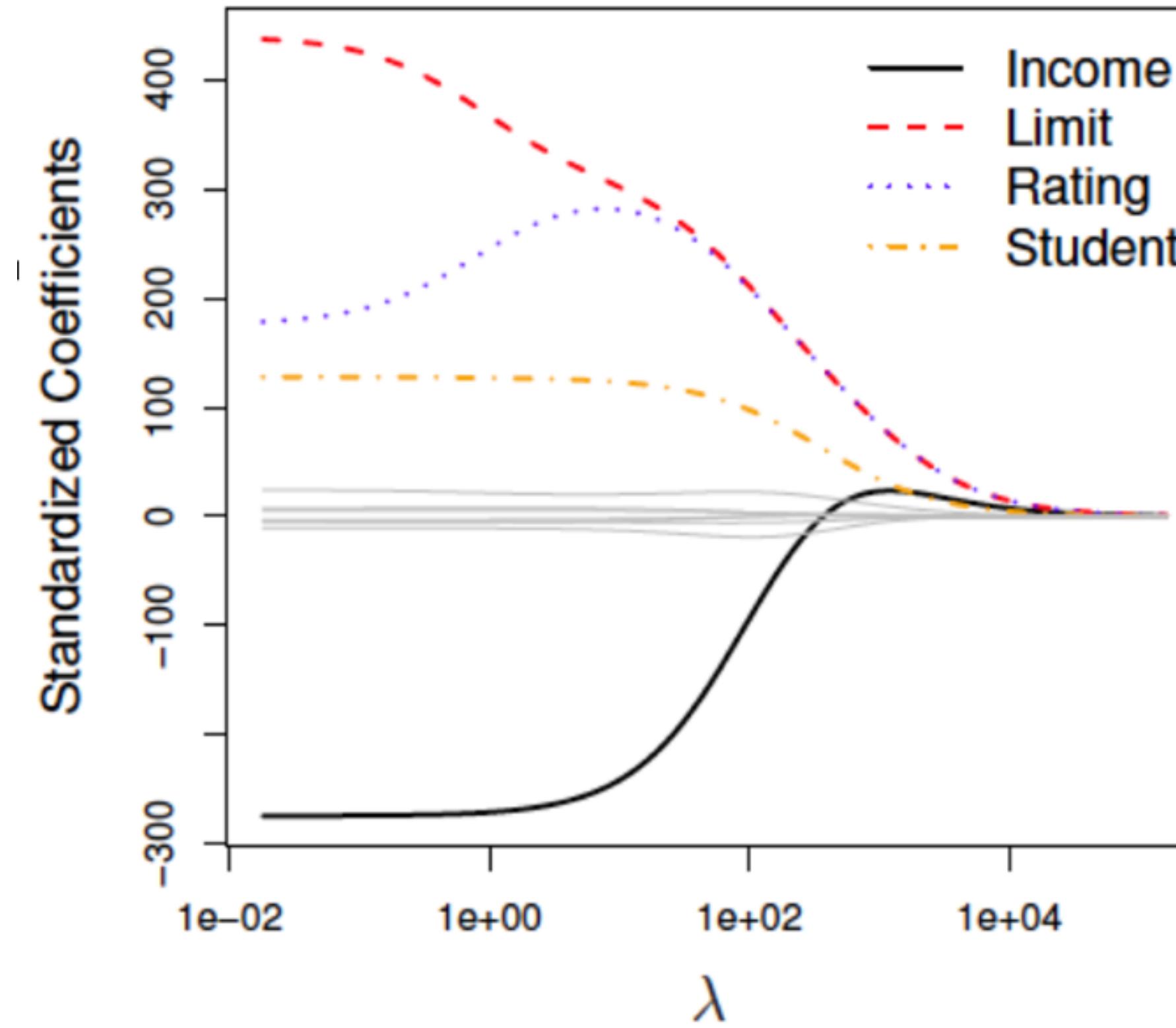
Regularized Least Squares



ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero.

However, the lasso constraint has corners at each of the axes, and so the ellipse will **OFFEN** intersect the constraint region at an axis.

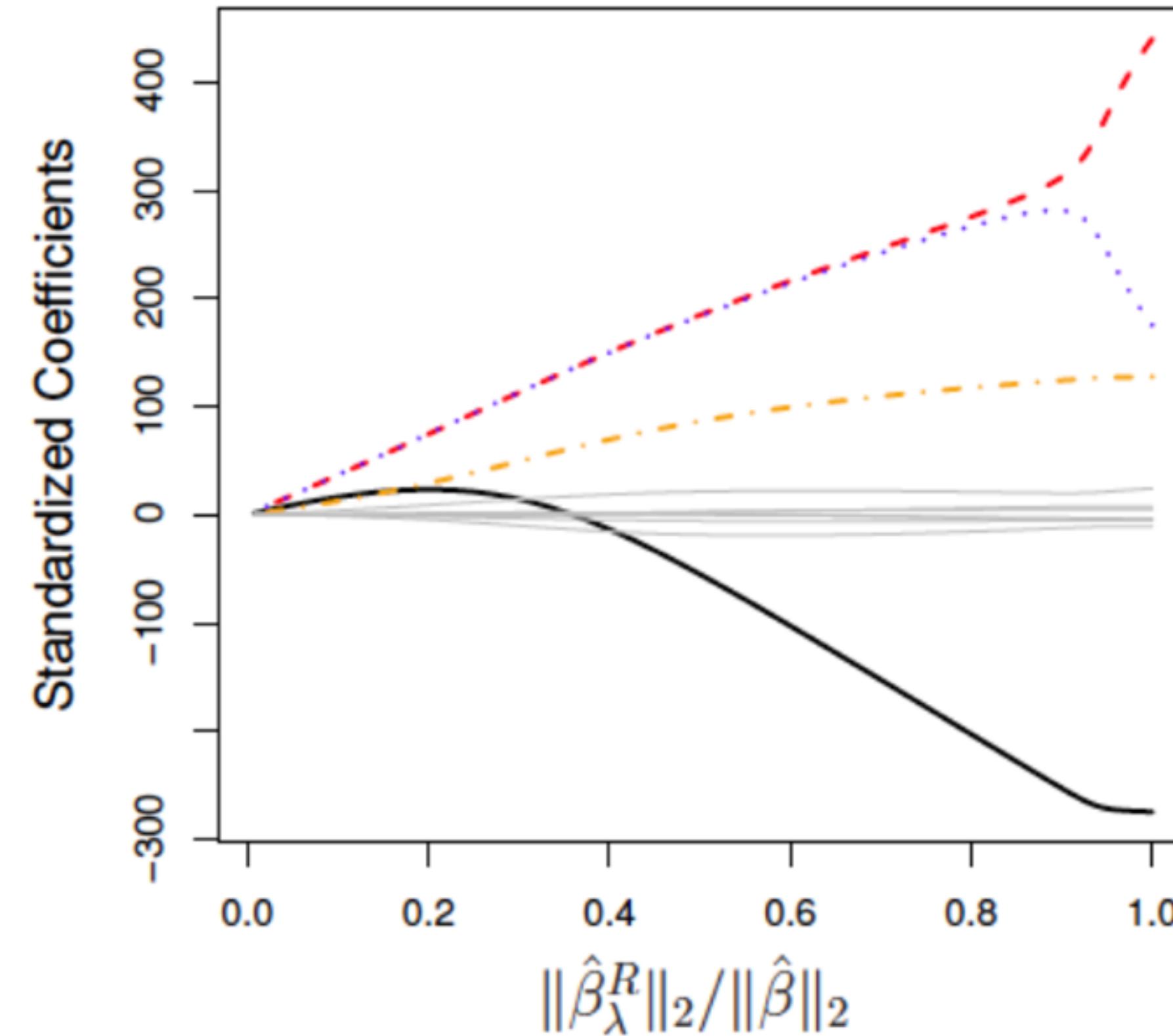
Credit Data Example of Ridge regression



$$\beta(\lambda) = \underset{\beta}{\operatorname{argmin}} \| Y - X\beta \|_2^2 + \lambda \| \beta \|_2$$

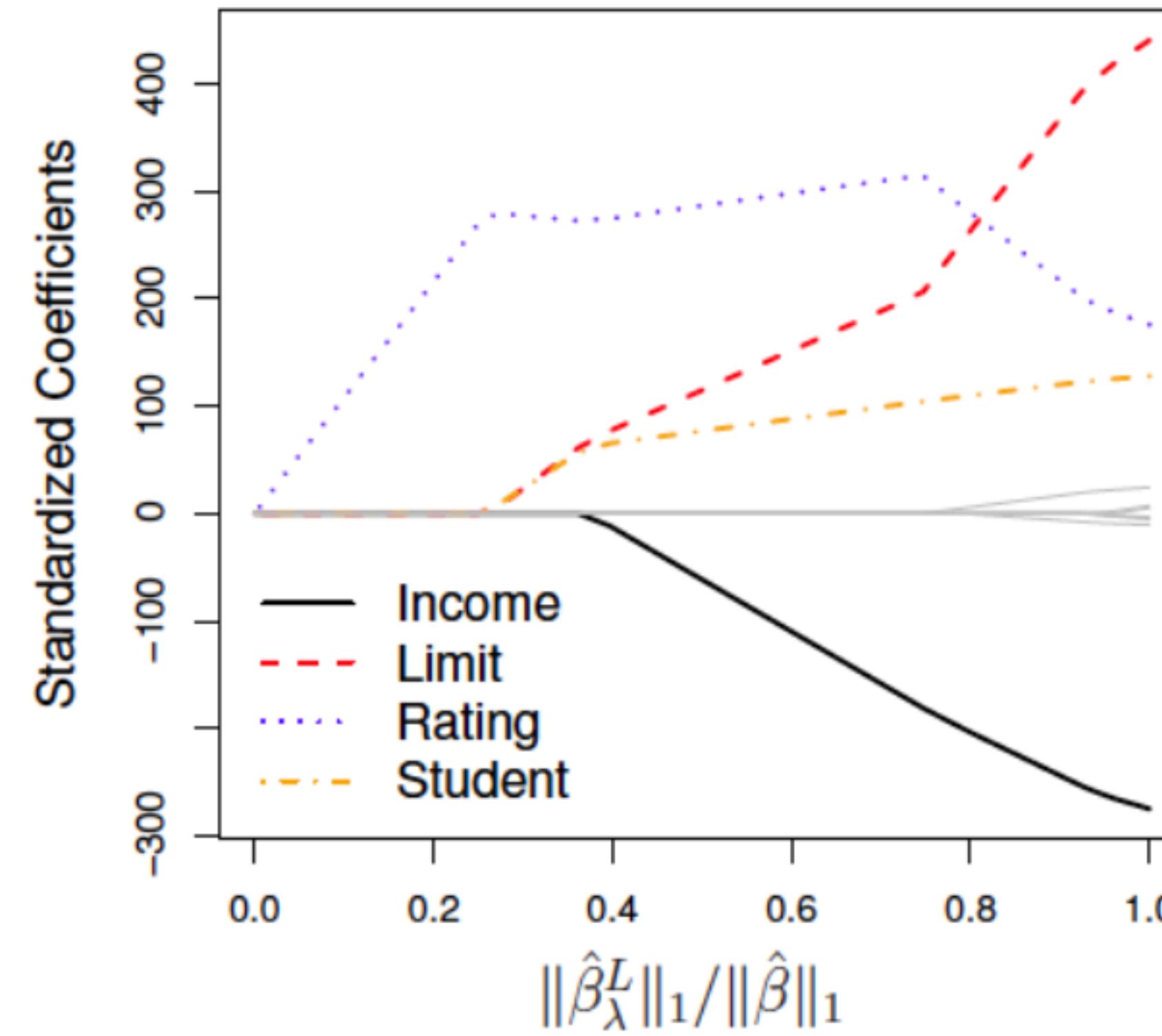
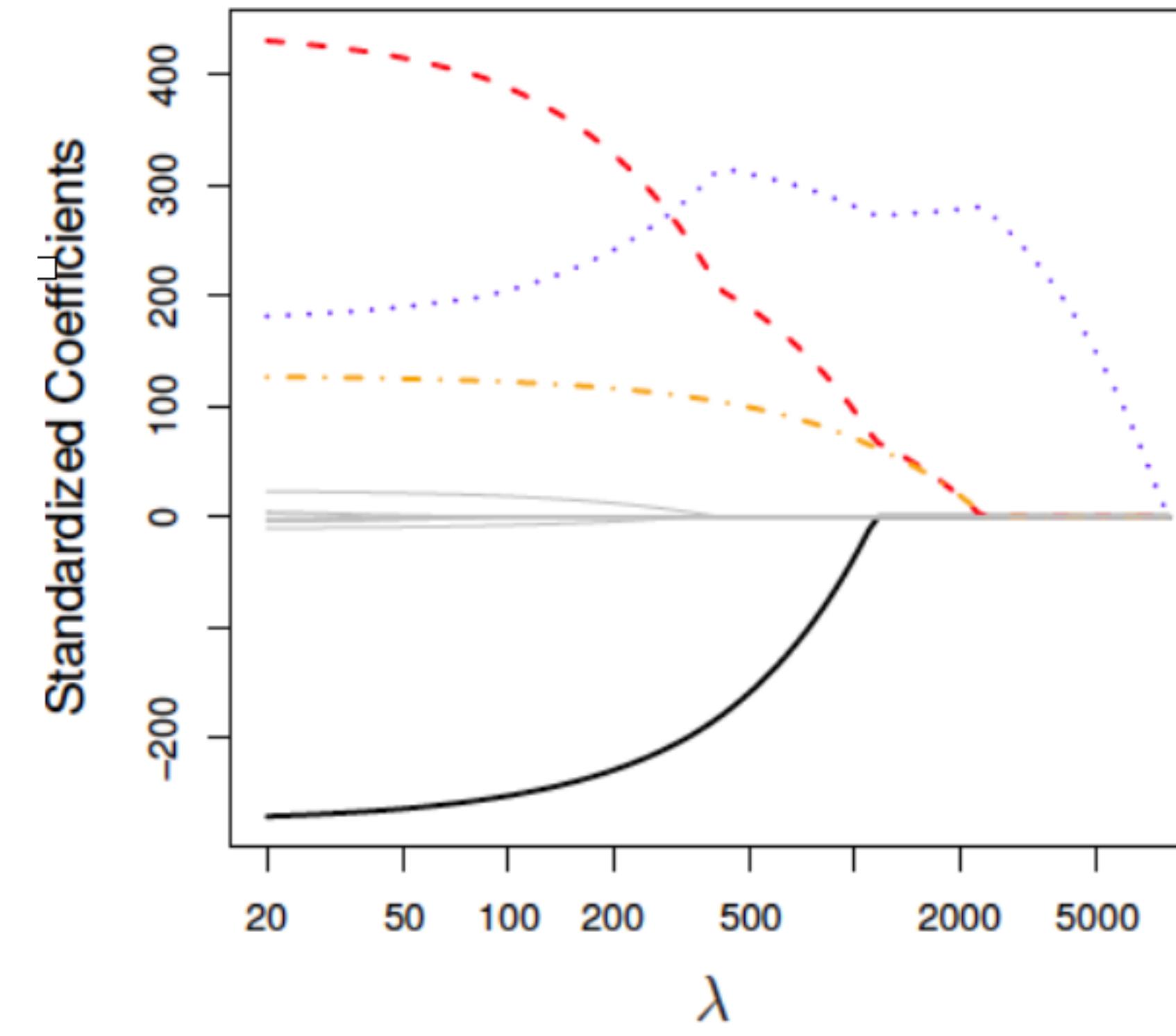
Therefore, it is best to apply ridge regression after *standardizing the predictors*, using the formula

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}}$$



The right-hand panel displays the same ridge coefficient estimates as the left-hand panel, but instead of displaying λ on the x -axis, we now display $\| \hat{\beta}_\lambda^R \|_2 / \| \hat{\beta} \|_2$, where $\hat{\beta}$ denotes the vector of least squares coefficient estimates.

Credit Data Example of Lasso



- However, in the case of the lasso, the L_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter is sufficiently large.
- much like best subset selection, the lasso performs variable selection.
- We say that the lasso yields **sparse** models | that is, models that involve only a subset of the variables.

Alternatives to Squared Error

Huber M-estimator:

$$\min J_h(\Theta) = \rho_\lambda(\delta_0 \Theta - Y) \quad (14)$$

where the Huber's loss function $\rho_\lambda(x)$ is defined as

$$\rho_\lambda(x) = \begin{cases} x^2/2, & \text{if } |x| \leq \lambda \\ \lambda|x| - \lambda^2/2, & \text{if } |x| > \lambda. \end{cases}$$



Chap 4

Linear

Classification

- Logistic Regression
- Linear Discriminant Analysis
- Naïve Bayes



Logistic Regression

Optional subtitle

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$$

where the sigmoid is defined as

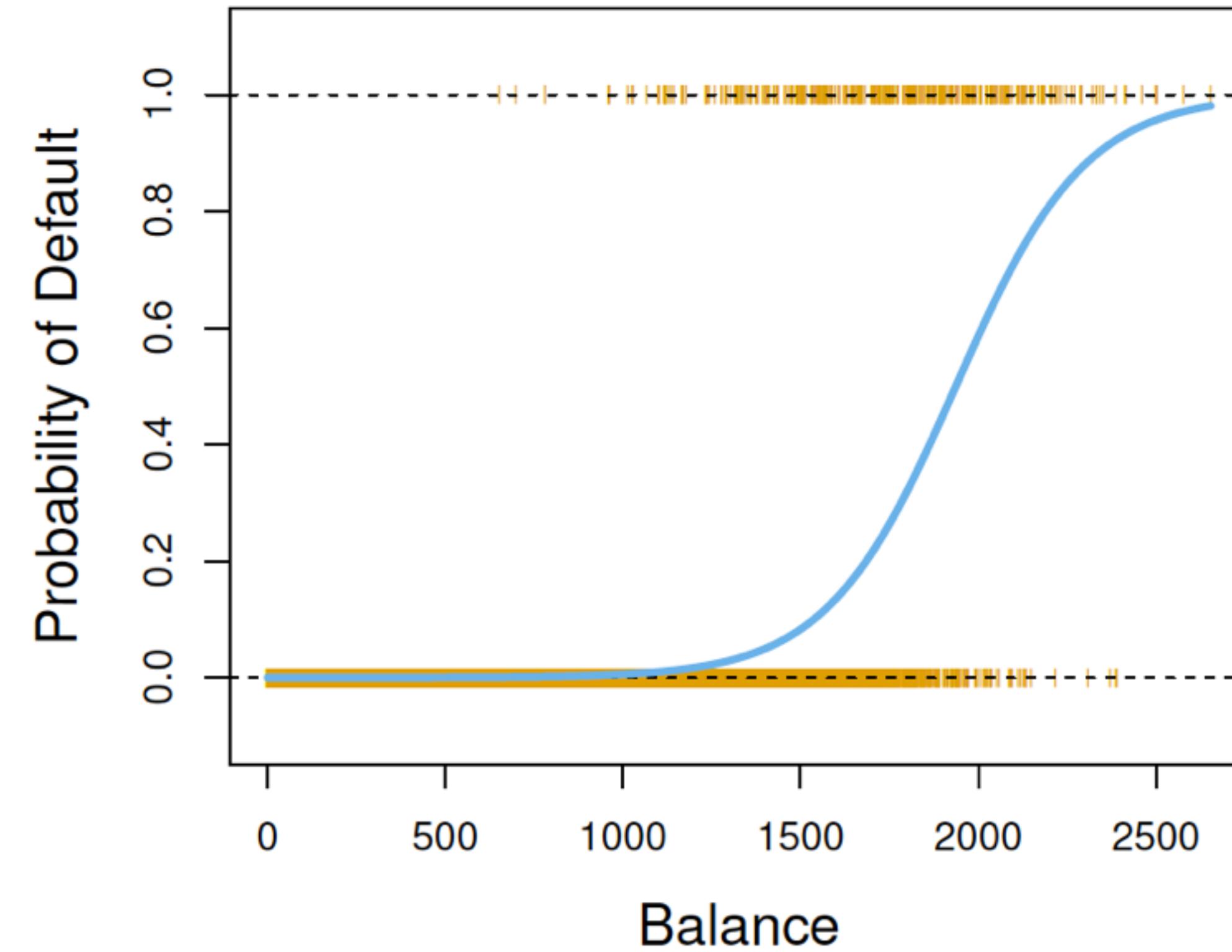
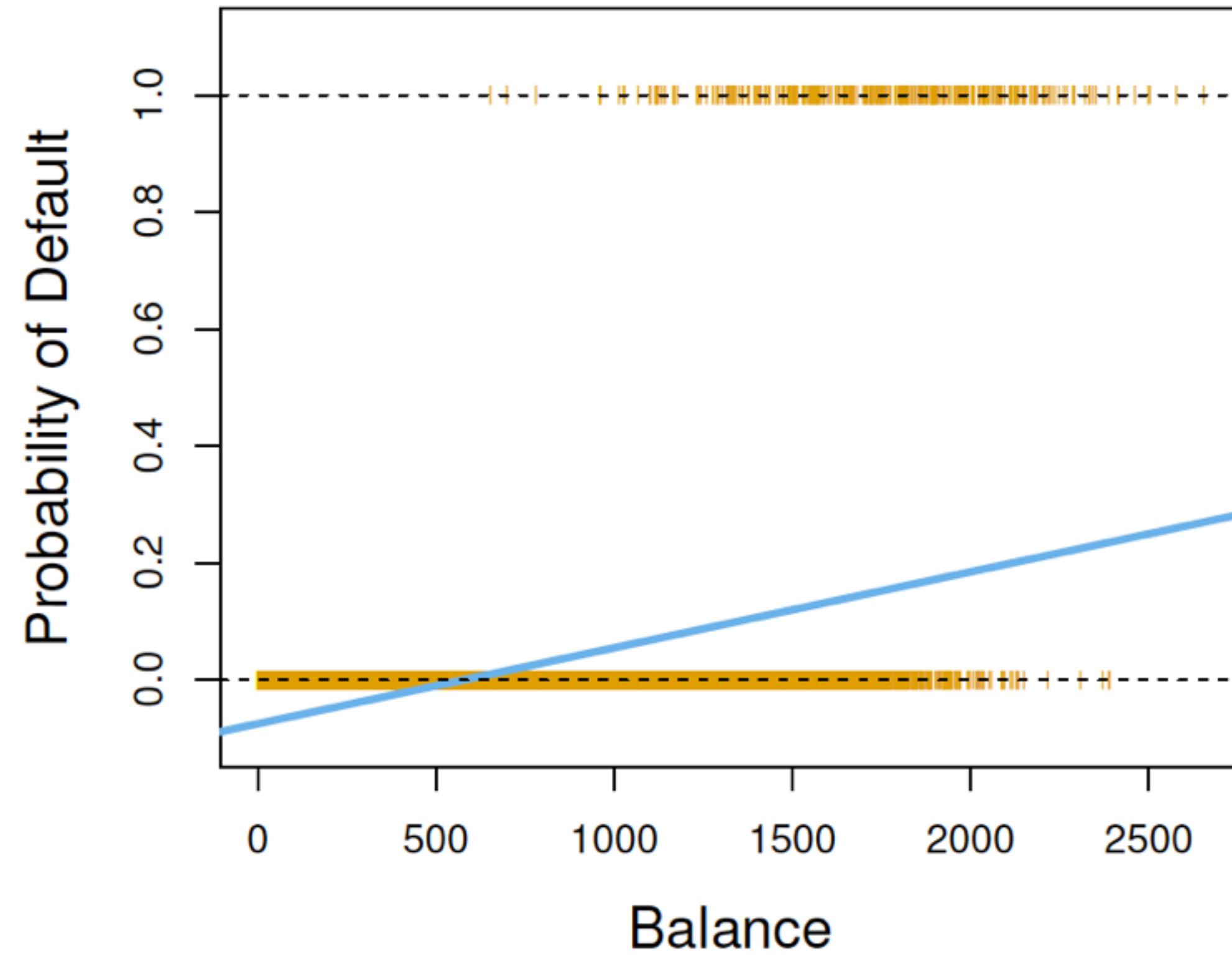
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- ▶ One parameter per data dimension (feature)
- ▶ Features can be discrete or continuous
- ▶ Output of the model: value $y \in [0, 1]$
- ▶ This allows for gradient-based learning of the parameters: smoothed version of the $\text{sign}(\cdot)$



Linear versus Logistic Regression

Optional subtitle



Logistic regression ensures that our estimate for $p(X)$ lies between 0 and 1.

Conditional likelihood(MLE)

Optional subtitle

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \dots, t^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})$
- Assuming that the training examples are **sampled IID**: independent and identically distributed

$$p(t^{(1)}, \dots, t^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) = \prod_{i=1}^N p(t^{(i)} | \mathbf{x}^{(i)})$$

- We can write each probability as

$$\begin{aligned} p(t^{(i)} | \mathbf{x}^{(i)}) &= p(C = 1 | \mathbf{x}^{(i)})^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}} \\ &= \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}} \end{aligned}$$

- We might want to learn the model, by **maximizing the conditional likelihood**

$$\max_{\mathbf{w}} \prod_{i=1}^N p(t^{(i)} | \mathbf{x}^{(i)})$$

- Convert this into a minimization so that we can write the **loss function**



Optional subtitle

$$\begin{aligned} p(t^{(1)}, \dots, t^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) &= \prod_{i=1}^N p(t^{(i)} | \mathbf{x}^{(i)}) \\ &= \prod_{i=1}^N \left(1 - p(C = 0 | \mathbf{x}^{(i)})\right)^{t^{(i)}} p(C = 0 | \mathbf{x}^{(i)})^{1-t^{(i)}} \end{aligned}$$

- It's convenient to take the logarithm and convert the maximization into minimization by changing the sign

$$\ell_{log}(\mathbf{w}) = - \sum_{i=1}^N t^{(i)} \log(1 - p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})) - \sum_{i=1}^N (1 - t^{(i)}) \log p(C = 0 | \mathbf{x}^{(i)}, \mathbf{w})$$

- Why is this equivalent to maximize the conditional likelihood?
- Is there a closed form solution?
- It's a convex function of \mathbf{w} . Can we get the global optimum?

Logistic Regression with Regularisation

Optional subtitle

- We can also look at

$$p(\mathbf{w}|\{t\}, \{\mathbf{x}\}) \propto p(\{t\}|\{\mathbf{x}\}, \mathbf{w}) p(\mathbf{w})$$

with $\{t\} = (t^{(1)}, \dots, t^{(N)})$, and $\{\mathbf{x}\} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})$

- We can define priors on parameters \mathbf{w}
- This is a form of regularization
- Helps avoid large weights and **overfitting**

$$\max_{\mathbf{w}} \log \left[p(\mathbf{w}) \prod_i p(t^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \right]$$

- What's $p(\mathbf{w})$?



Linear Discriminant Analysis

Optional subtitle

The Gaussian density has the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}$$

Here μ_k is the mean, and σ_k^2 the variance (in class k). We will assume that all the $\sigma_k = \sigma$ are the same.

Plugging this into Bayes formula, we get a rather complex expression for $p_k(x) = \Pr(Y = k|X = x)$:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_l}{\sigma}\right)^2}}$$

Happily, there are simplifications and cancellations.



Discriminant functions

Optional subtitle

To classify at the value $X = x$, we need to see which of the $p_k(x)$ is largest. Taking logs, and discarding terms that do not depend on k , we see that this is equivalent to assigning x to the class with the largest *discriminant score*:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

Note that $\delta_k(x)$ is a *linear* function of x .

If there are $K = 2$ classes and $\pi_1 = \pi_2 = 0.5$, then one can see that the *decision boundary* is at

$$x = \frac{\mu_1 + \mu_2}{2}.$$

(See if you can show this)



Bayes decision rule

Optional subtitle

- If we know the conditional probability $P(X | y)$ we can determine the appropriate class by using Bayes rule:

$$P(y = i | X) = \frac{P(X | y = i)P(y = i)}{P(X)} \stackrel{\text{def}}{=} q_i(X)$$

But how do we determine $p(X|y)$?



Naïve Bayes Classifier

Optional subtitle

- Naïve Bayes classifiers assume that given the class label (Y) the attributes are **conditionally independent** of each other:

$$p(X | y) = \prod_j p_j(x^j | y)$$

Product of probability terms

Specific model for attribute j

- Using this idea the full classification rule becomes:

$$\begin{aligned}\hat{y} &= \arg \max_v p(y=v | X) \\ &= \arg \max_v \frac{p(X | y=v)p(y=v)}{p(X)} \\ &= \arg \max_v \prod_j p_j(x^j | y=v)p(y=v)\end{aligned}$$

v are the classes we have



Conditional likelihood: Full version

Optional subtitle

$$L(X_i | y_i = 1, \Theta) = \prod_j p(x_i^j | y_i = 1, \theta_1^j)$$



Vector of binary attributes for sample i



The set of all parameters in the NB model



The specific parameters for attribute j in class 1

Note the following:

1. We assume conditional independence between attributes **given** the class label
2. We learn a **different** set of parameters for the two classes (class 1 and class 2).

Chap 5、6

Support Vector Machine

- Logistic Regression
- Linear Discriminant Analysis
- Naïve Bayes



SVM in a Nutshell

Optional subtitle

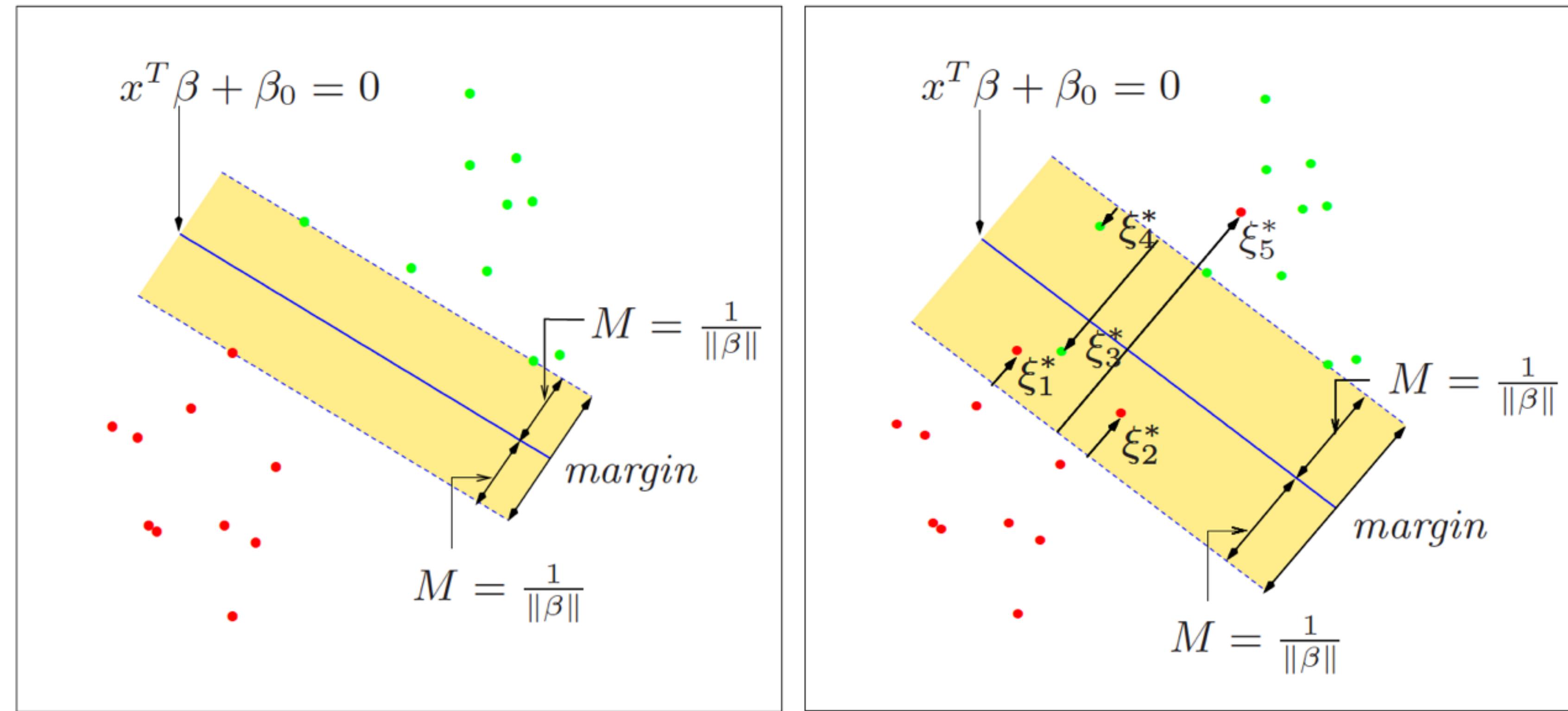
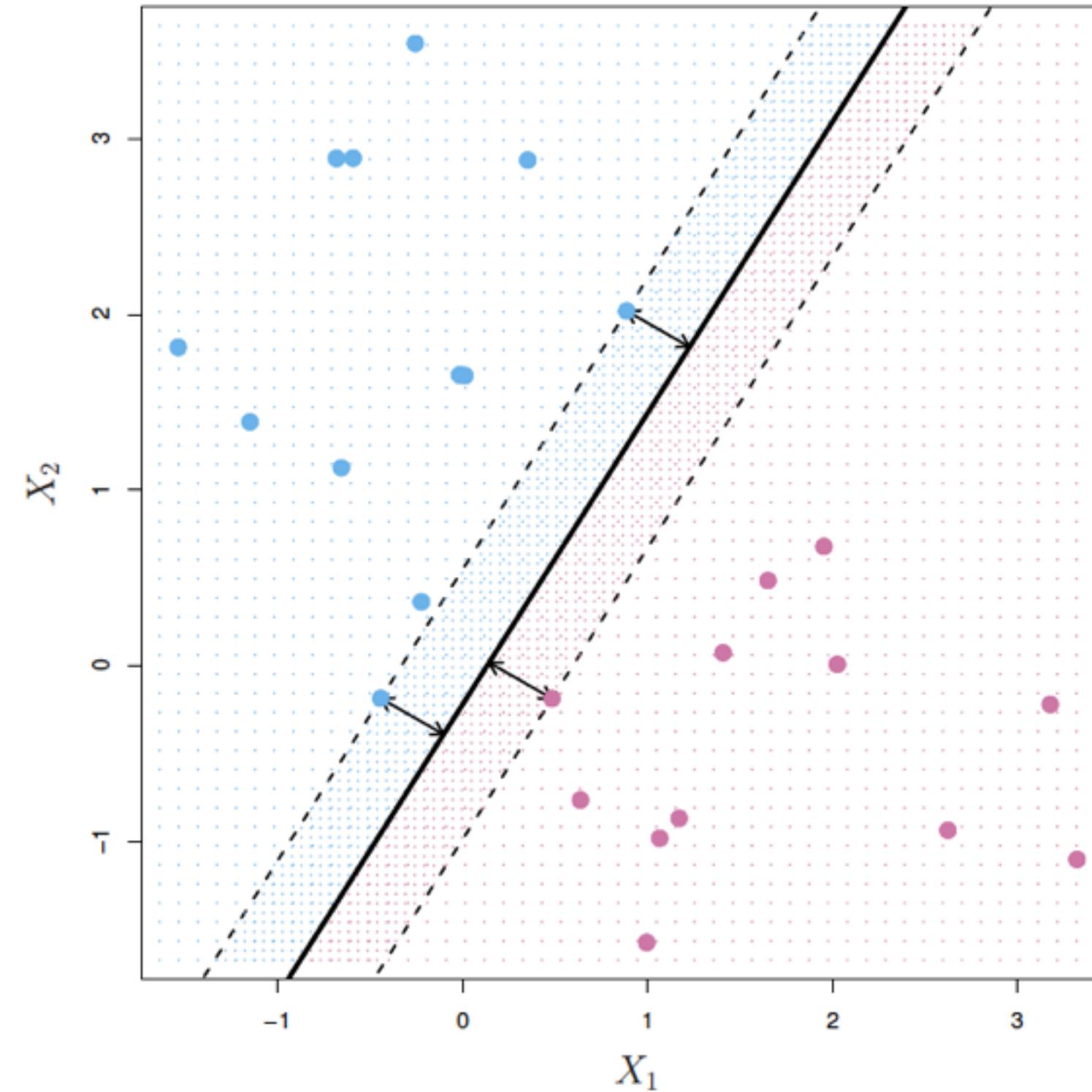


FIGURE 12.1. Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/\|\beta\|$. The right panel shows the nonseparable (overlap) case. The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$; points on the correct side have $\xi_j^* = 0$. The margin is maximized subject to a total budget $\sum \xi_i \leq \text{constant}$. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.

Maximal Margin Classifier

Optional subtitle

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

$$\begin{array}{ll}\text{maximize} & M \\ \text{subject to} & \beta_0, \beta_1, \dots, \beta_p\end{array}$$

$$\sum_{j=1}^p \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M \\ \text{for all } i = 1, \dots, N.$$



This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently

Different Forms of SVM (separated cases)

Optional s

$$\begin{aligned} & \max_{\beta, \beta_0, \|\beta\|_2=1} M \\ & \text{s.t. } y_i (x_i^T \beta + \beta_0) \geq M, \quad i = 1, \dots, n \end{aligned} \quad (1)$$

which is equivalent to

$$\begin{aligned} & \min \| \beta \|_2 \\ & \text{s.t. } y_i (x_i^T \beta + \beta_0) \geq 1, \quad i = 1, \dots, n \end{aligned}$$

A natural way to modify the constraint in Eq(1) is by introducing the slack variable $\xi = (\xi_1, \dots, \xi_n)$:

$$y_i (x_i^T \beta + \beta_0) \geq M(1 - \xi_i)$$

$$\forall i, \xi_i \geq 0, \sum_i \xi_i \leq \text{constant}$$

Remark: $M \sum_i \xi_i$ measures the total amount distance of points on the wrong side of their margin.



Different Forms of SVM (non-seperable cases)

Optional subtitle

$$\begin{aligned} & \min \| \beta \|_2^2 \\ & s.t. y_i (x_i^T \beta + \beta_0) \geq 1 - \xi_i, \quad i = 1, \dots, n \end{aligned} \quad (2)$$

$$\xi_i \geq 0, \sum_i \xi_i \leq constant$$

$$\begin{aligned} & \min \frac{1}{2} \| \beta \|_2^2 + C \sum_i \xi_i \\ & s.t. y_i (x_i^T \beta + \beta_0) \geq 1 - \xi_i, \quad i = 1, \dots, n, \xi_i \geq 0 \end{aligned} \quad (3)$$

$$\min \sum_{i=1}^n [1 - y_i (x_i^T \beta + \beta_0)]_+ + \frac{\lambda}{2} \| \beta \|_2^2 \quad (4)$$

where x_+ indicates the positive part of x . If $\lambda = C/2$, then Eq(3) and Eq(4) are equivalent.



Primal Classifier in transformed feature space

Optional subtitle

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- Simply map \mathbf{x} to $\Phi(\mathbf{x})$ where data is separable
- Solve for \mathbf{w} in high dimensional space \mathbb{R}^D
- If $D \gg d$ then there are many more parameters to learn for \mathbf{w} . Can this be avoided?



Dual Classifier in transformed feature space

Optional subtitle

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector α needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a **Kernel Classifier**:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$



Example kernels

Optional subtitle

- Linear kernels $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- Polynomial kernels $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^d$ for any $d > 0$
 - Contains all polynomials terms up to degree d
- Gaussian kernels $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/2\sigma^2)$ for $\sigma > 0$
 - Infinite dimensional feature space



Chap 7

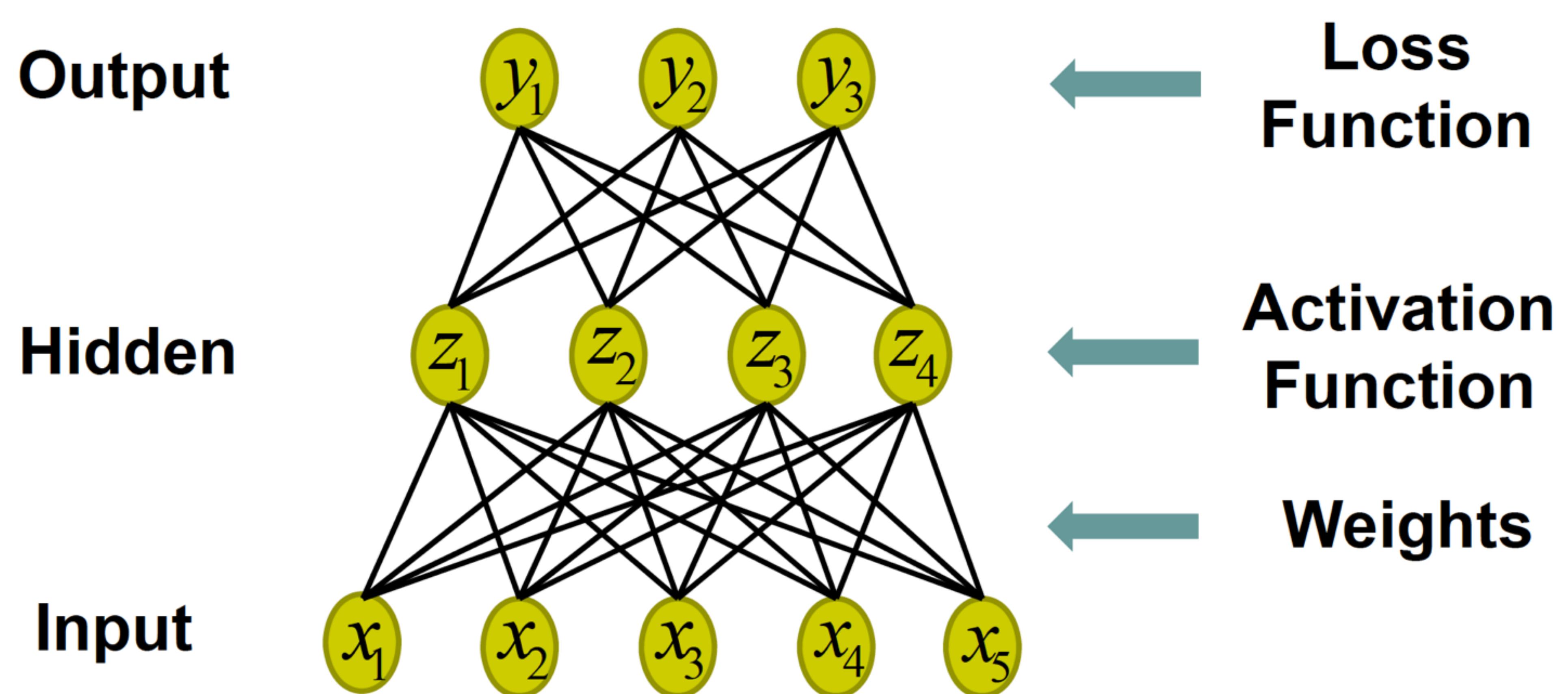
Neural Network

- Perceptron
- Convolutional Neural Network



Neural Network

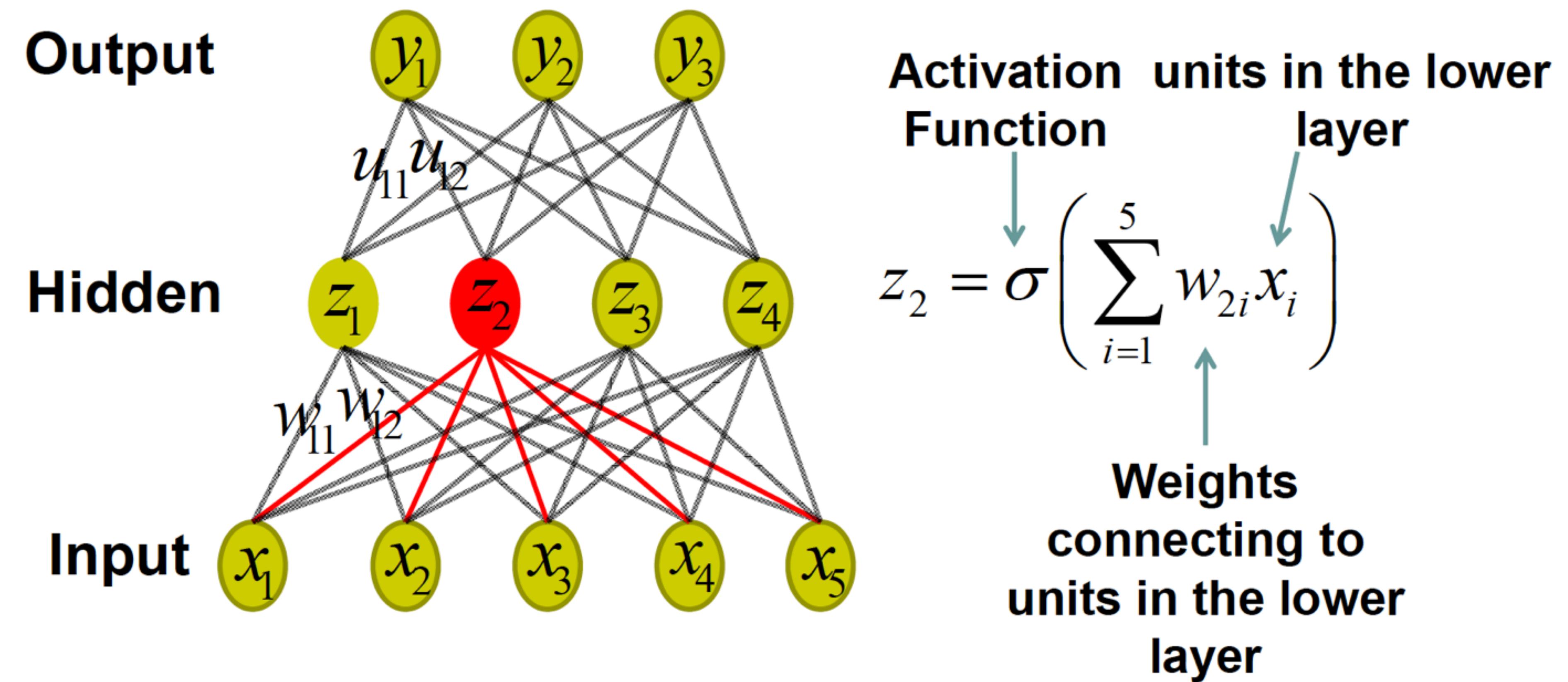
Optional subtitle



Local Computation At Each Unit

Optional subtitle

Linear Combination + Nonlinear Activation



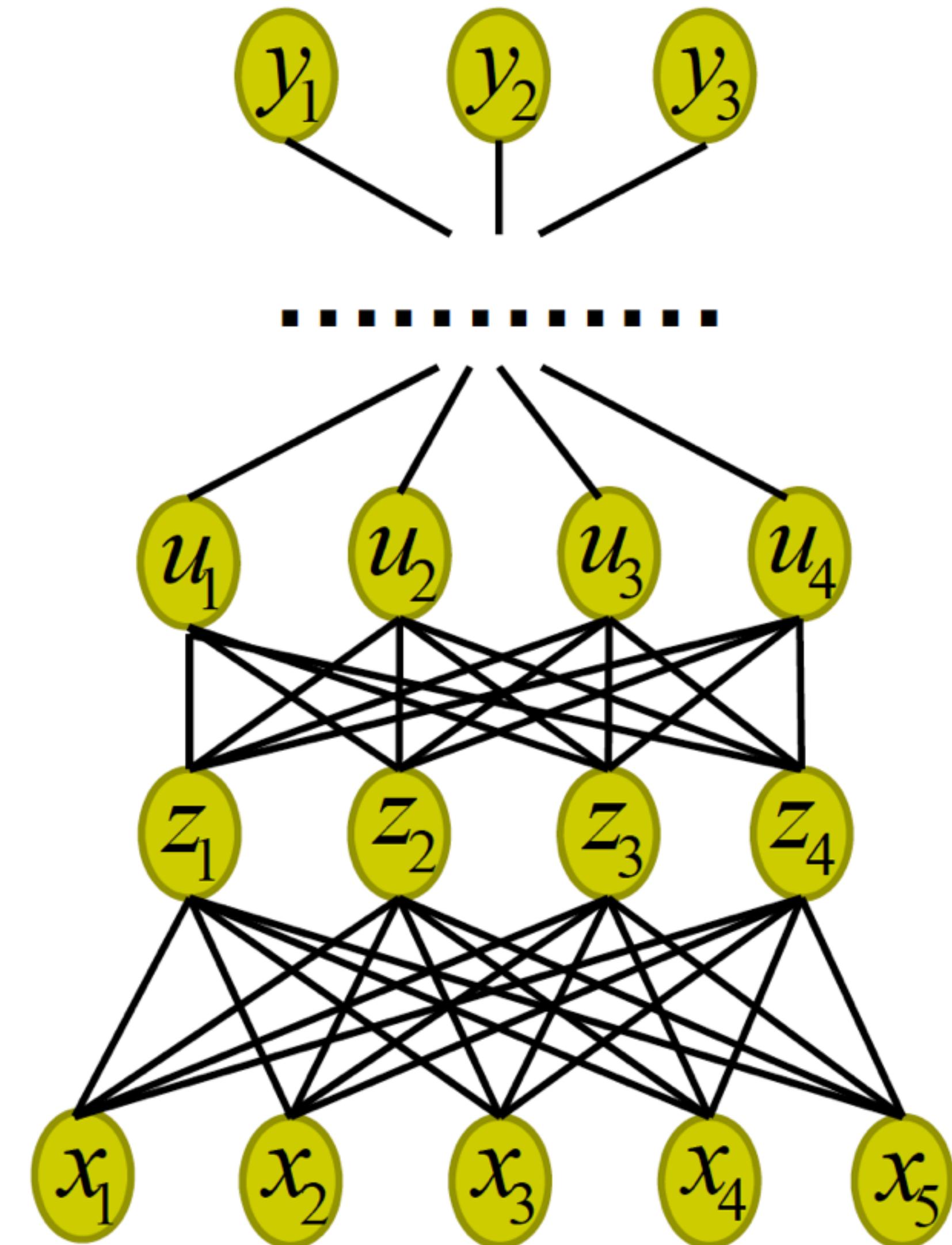
Deep Neural Network

Optional subtitle

Output

**More
Hidden
Layers**

Input

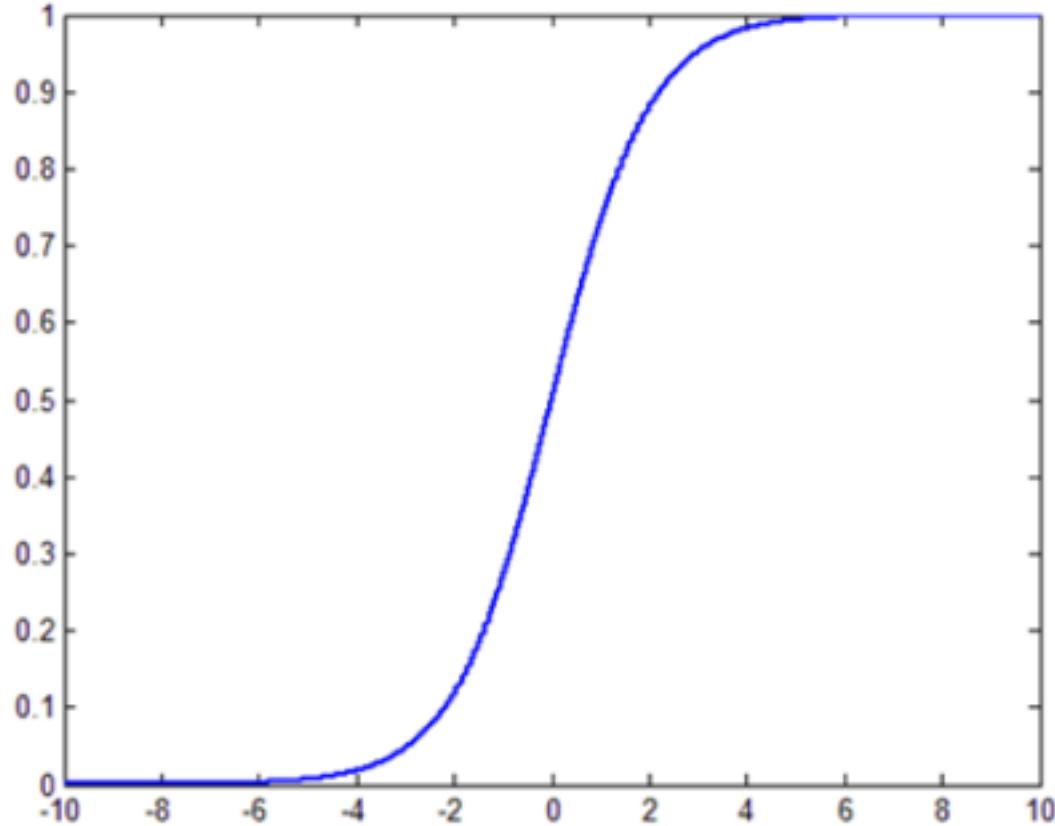


Activation Functions

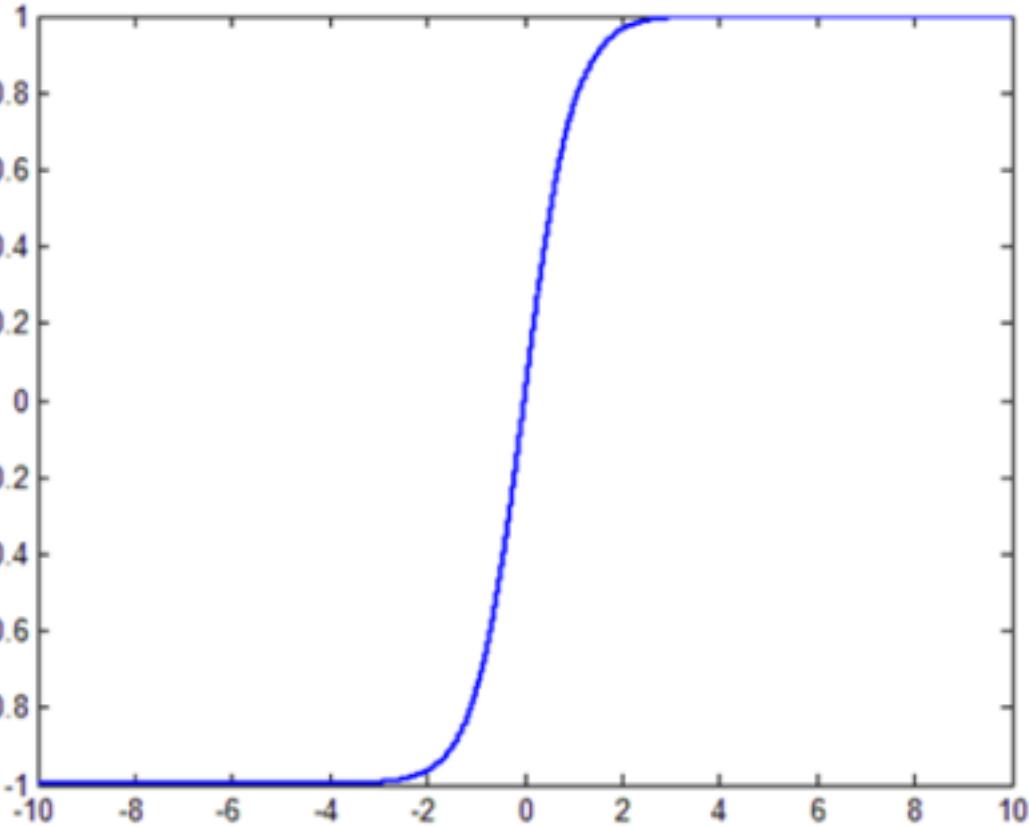
Optional subtitle

- Applied on the hidden units
- Achieve nonlinearity
- Popular activation functions

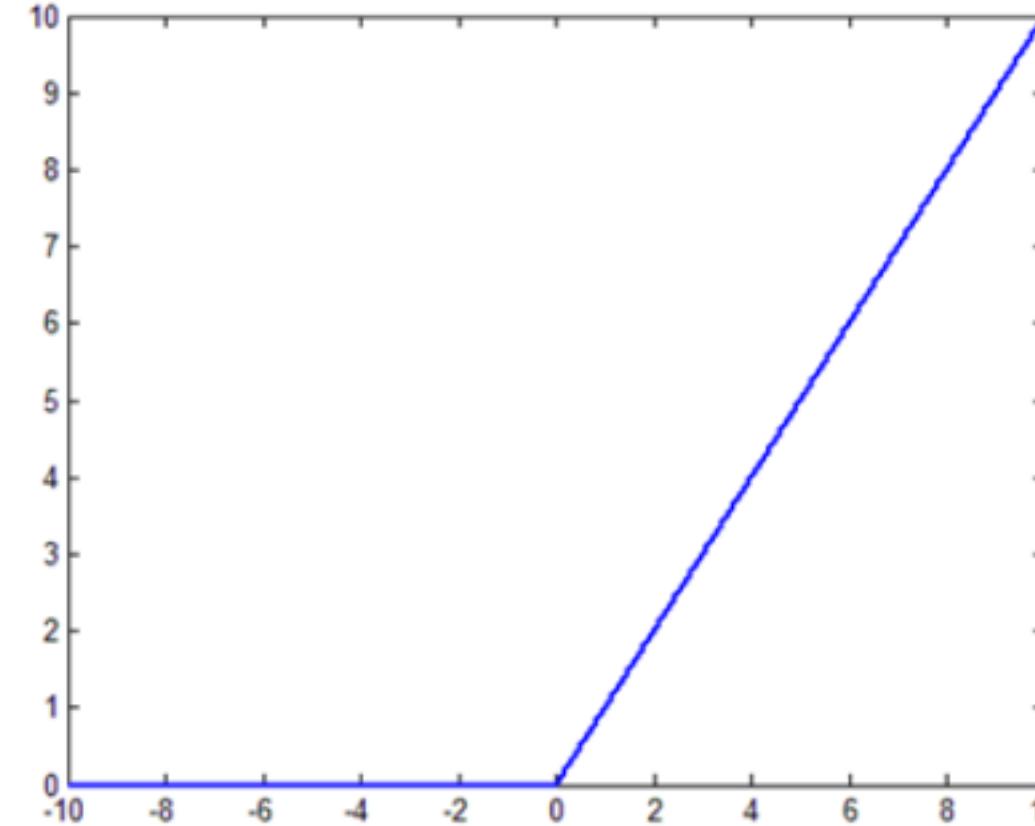
Sigmoid



Tanh



Rectified Linear



More Activation functions

Popular choice of $f(\cdot)$

- Sigmoid function

$$f(s) = \frac{1}{1 + e^{-s}}$$

- Tanh function (shift the center of Sigmoid to the origin)

$$f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

- Hard tanh

$$f(s) = \max(-1, \min(1, x))$$

- Rectified linear unit (ReLU)

$$f(s) = \max(0, x)$$

- Softplus: smooth version of ReLU

$$f(s) = \log(1 + e^s)$$

- Softmax: mostly used as output non-linearity for predicting discrete probabilities

$$f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^C e^{s_{k'}}}$$

- Maxout: it generalizes the rectifier assuming there are multiple netactivations

$$f(s_1, \dots, s_n) = \max_i(s_i)$$



Loss Functions

Optional subtitle

- Squared loss for regression

$$(y - t)^2$$

Prediction True value



A diagram showing the squared loss function $(y - t)^2$. It consists of two teal arrows pointing towards each other from below, forming a V-shape. Below the left arrow is the text "Prediction" and below the right arrow is the text "True value".

- Cross entropy loss for classification

$$-\sum_{k=1}^K t_k \ln a_k \quad a_k = \frac{\exp(y_k)}{\sum_{j=1}^K \exp(y_j)}$$

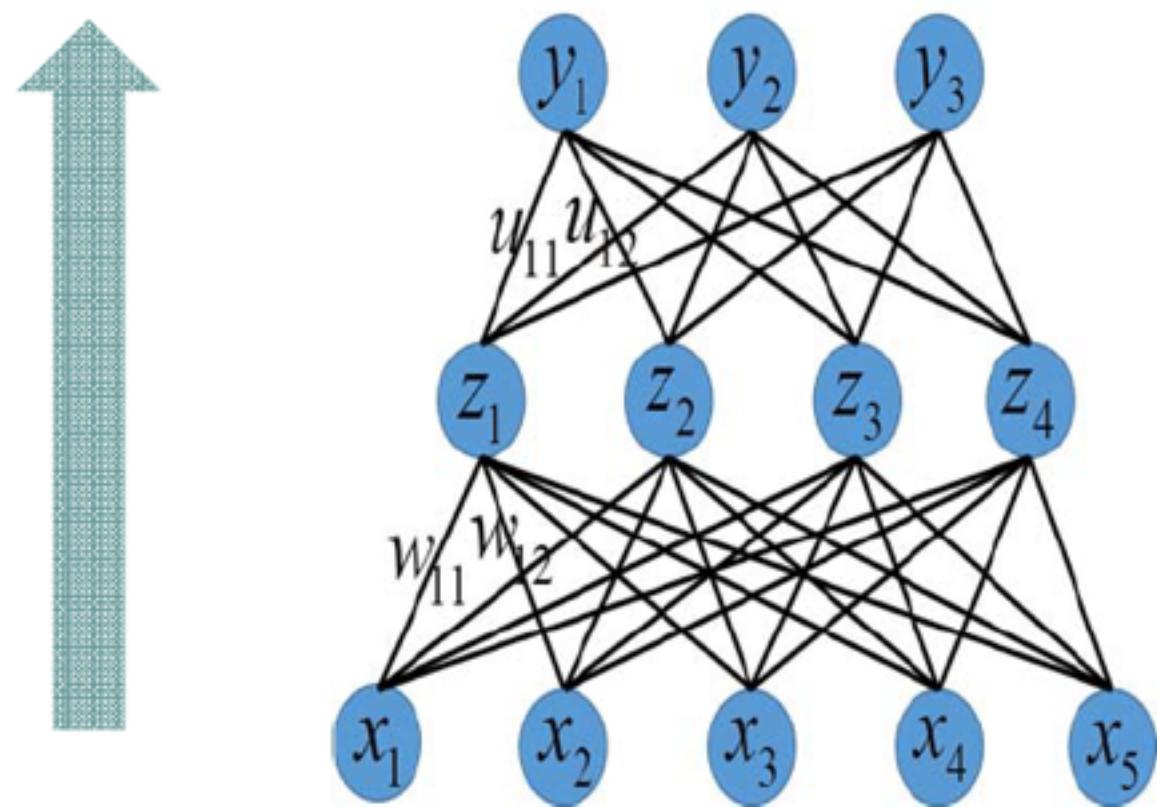
Class label Prediction



A diagram showing the cross entropy loss function. It consists of two teal arrows pointing upwards from below, forming a V-shape. Below the left arrow is the text "Class label" and below the right arrow is the text "Prediction".

Neural Network Prediction

- Compute unit values layer by layer in a forward manner



- Prediction function

Activation Function

Input

Output

Weights

$$y_k = \sum_{j=1}^4 u_{kj} \sigma\left(\sum_{i=1}^5 w_{ji} x_i\right)$$

Neural Network Training

- Goal: compute gradient

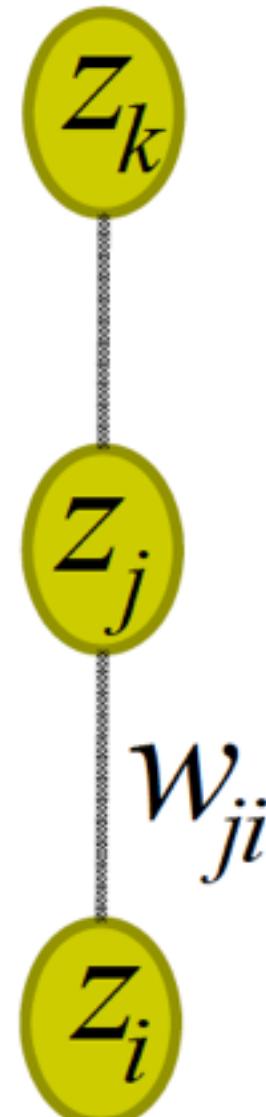
$$\frac{\partial L}{\partial w_{ij}} \leftarrow \begin{array}{l} \text{Training loss} \\ \text{Weight between unit } i \text{ and } j \end{array}$$

- Apply chain rule

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \leftarrow \begin{array}{l} \text{Linear combination} \\ \text{value } a_j = \sum_i w_{ji} z_i \end{array}$$

$$\boxed{\frac{\partial L}{\partial a_j}} = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

Called error, computed
recursively in a
backward manner



Neural Network Training

Optional subtitle

- Apply chain rule (cont'd)

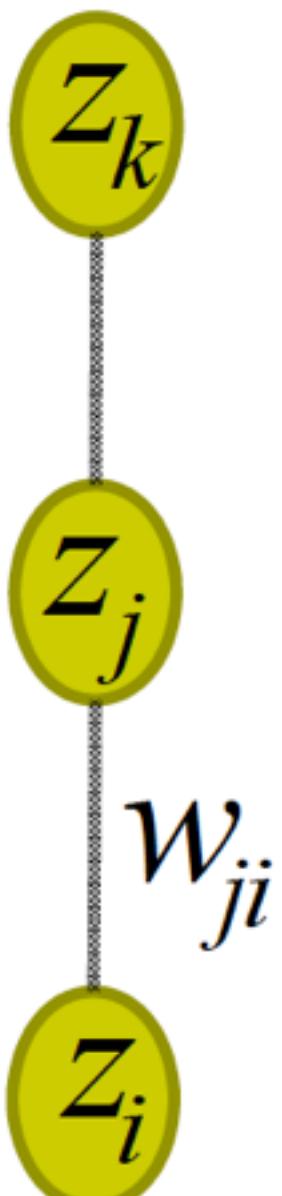
$$\frac{\partial a_j}{w_{ji}} = z_i \leftarrow \text{Derivative of activation function}$$
$$\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = w_{kj} \sigma'(a_j)$$

gradient = $\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{w_{ji}}$ = backward error x forward activation

- Pseudo code of BP

While not converge

1. compute forward activations
2. compute backward errors
3. compute gradients of weights
4. perform gradient descent



Local connectivity

Optional subtitle

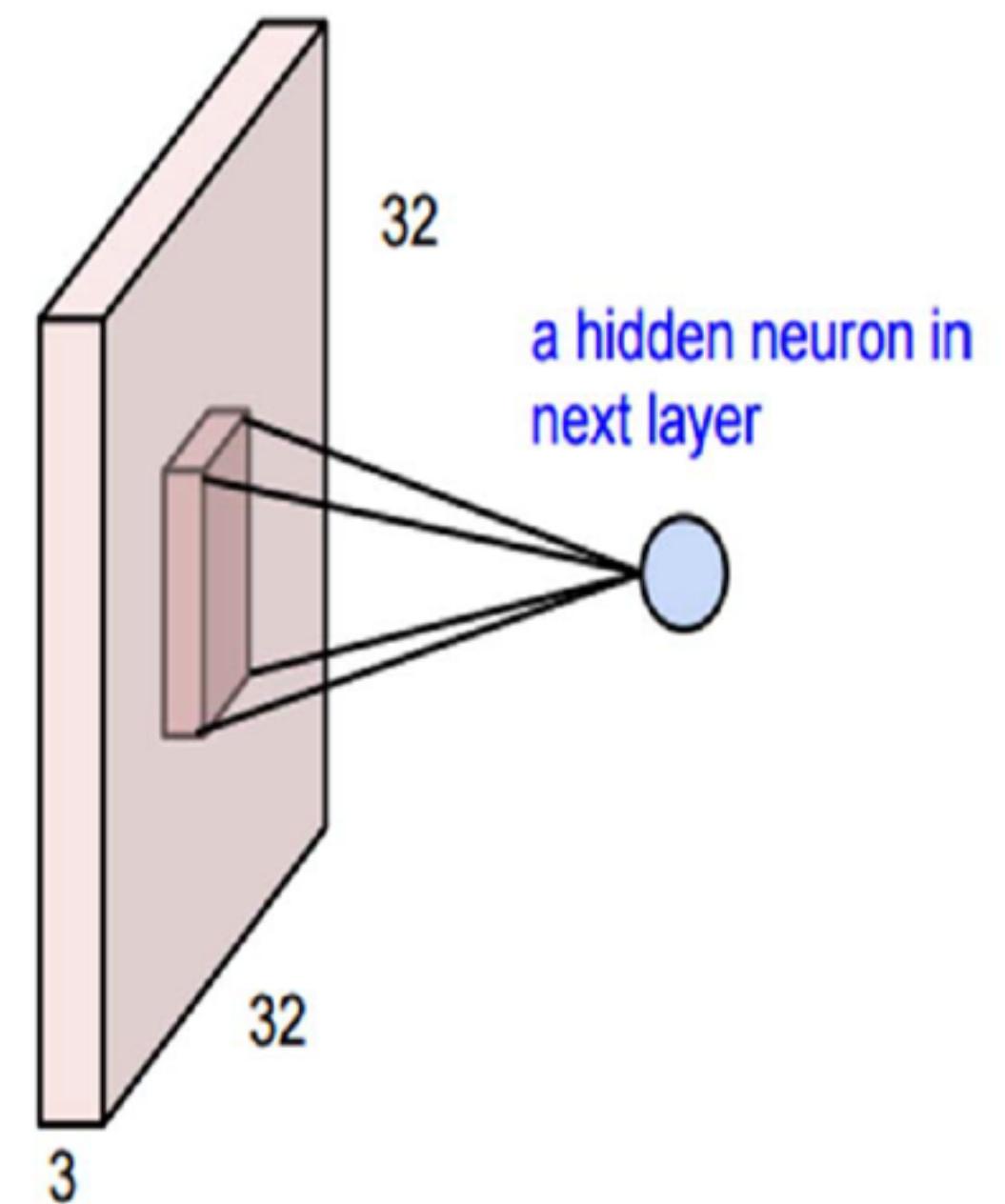
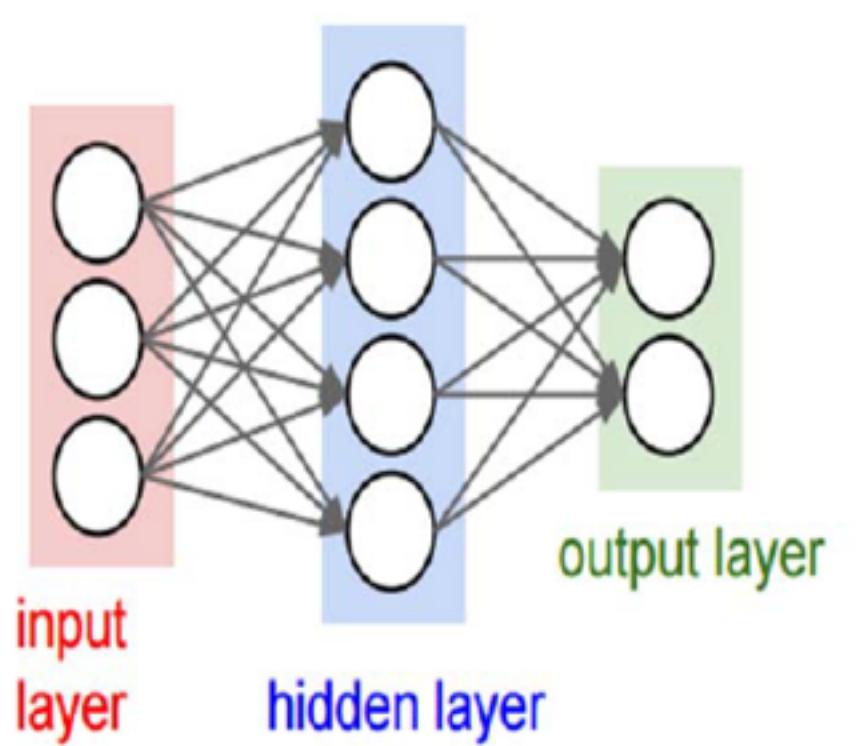


image: $32 * 32 * 3$ volume

**before: full connectivity:
 $32 * 32 * 3$ weights for each
neuron**

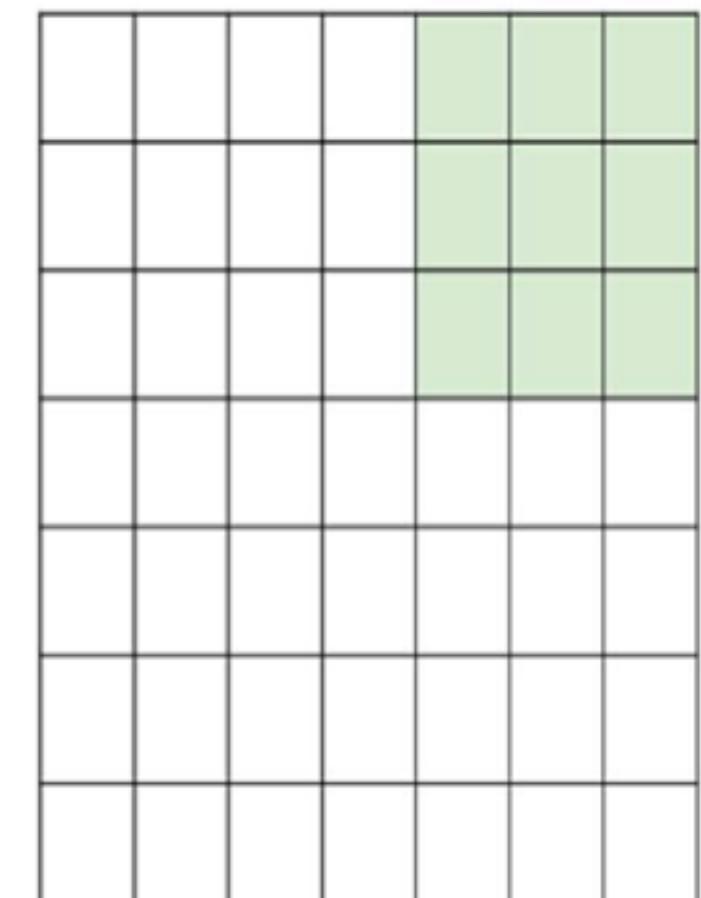
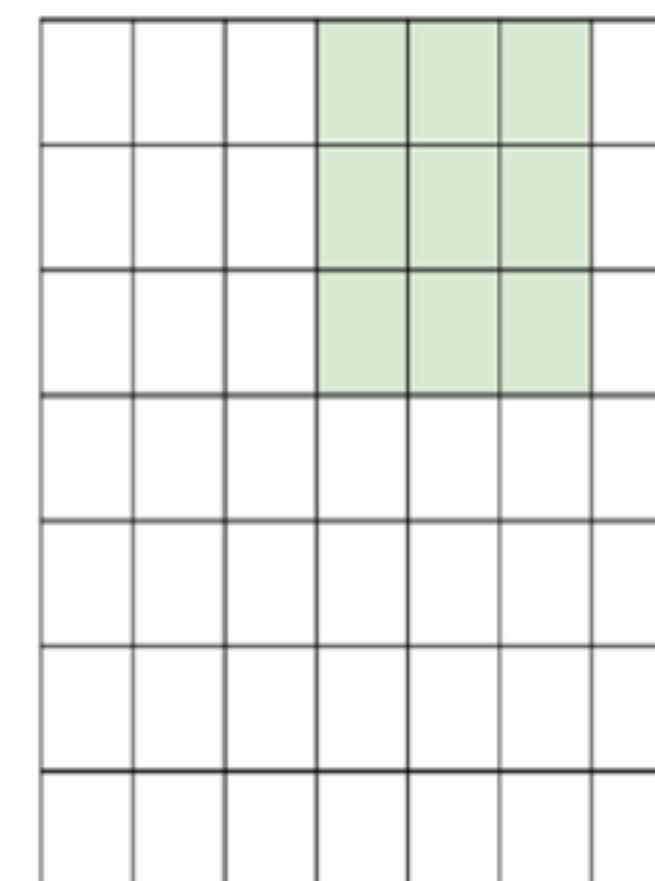
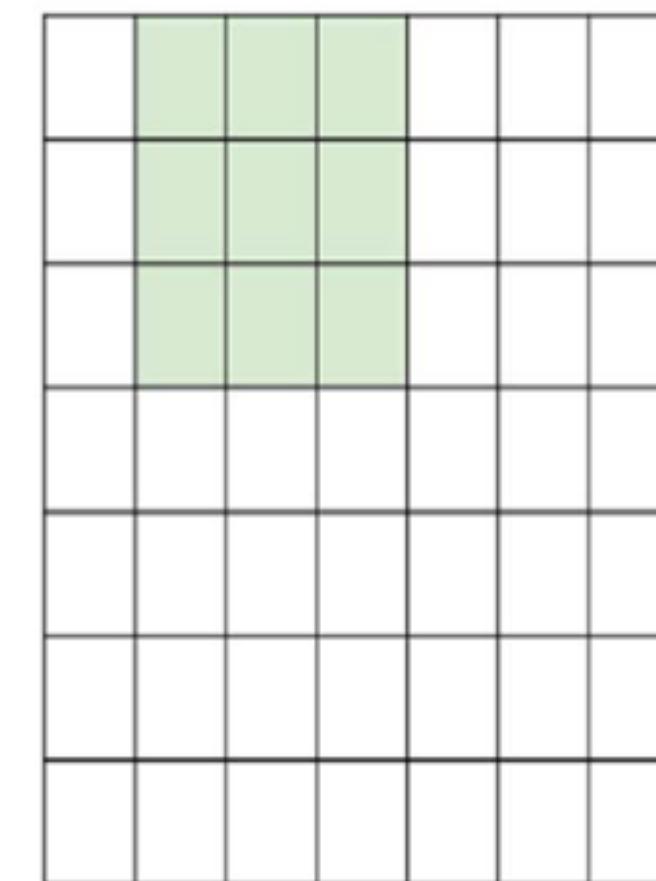
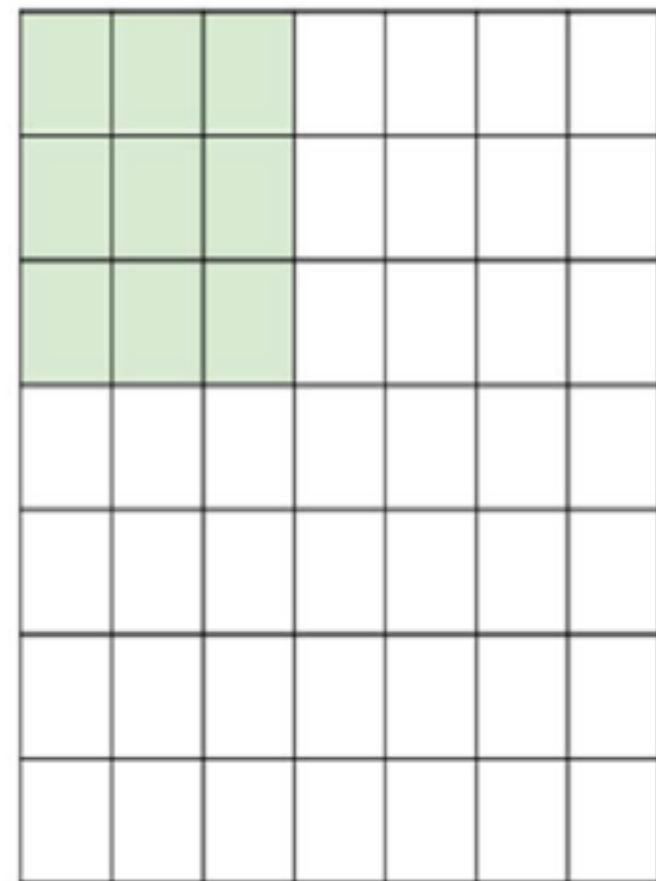
**now: one unit will connect
to, e.g. $5*5*3$ chunk and
only have $5*5*3$ weights**

Note the connectivity is:

- **local in space**
- **full in depth**

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride



Stride,
Zero-padding

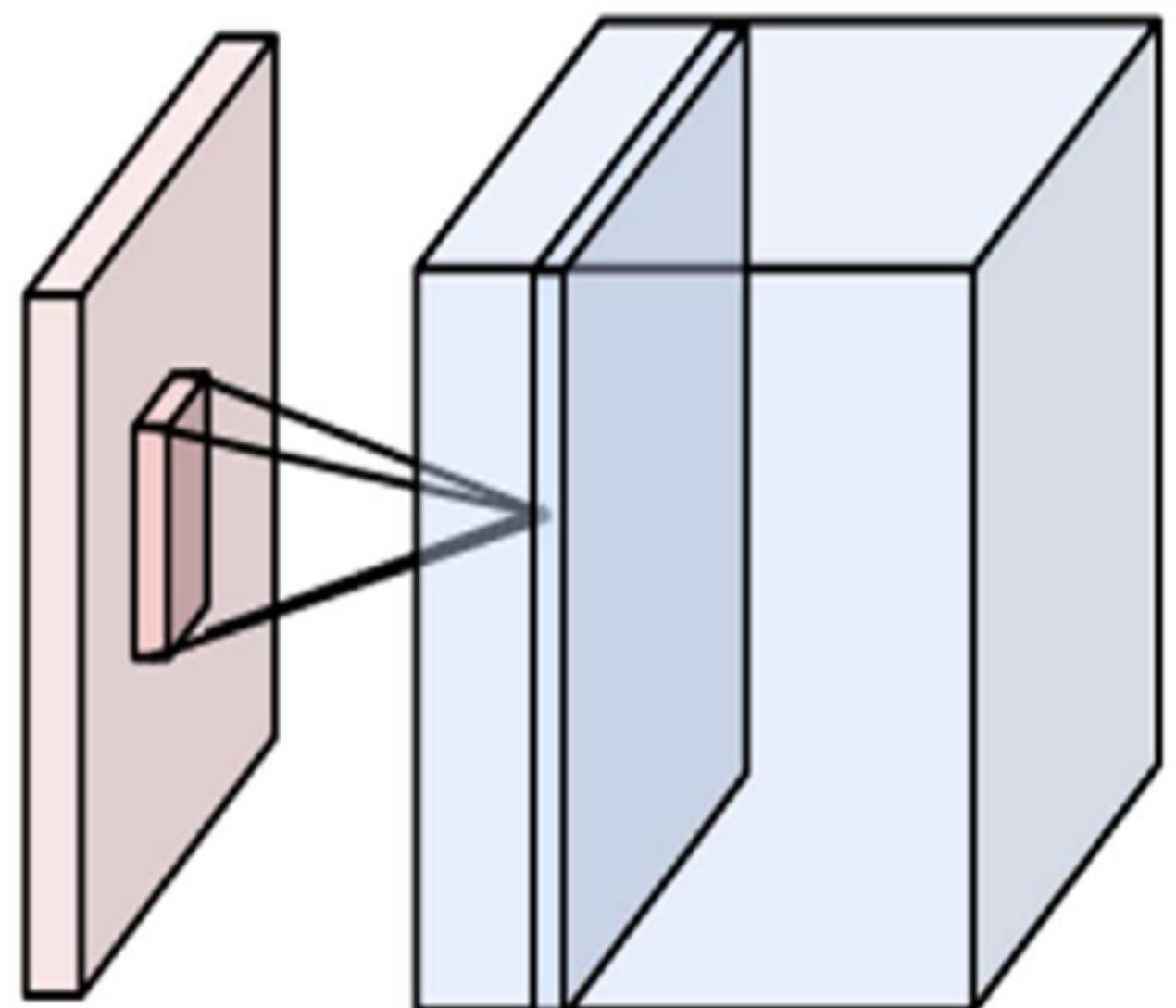
- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1



- Produce a map
- What's the size of the map?
 $5 * 5$

Convolutional Layers

- Connect units only to local receptive fields
- Use the same unit weight parameters for units in each “depth slice” (i.e. across spatial positions)



one activation map (a depth slice),
computed with one set of weights

Can call the units “**filters**”

We call the layer convolutional because
it is related to convolution of two signals

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x-n_1, y-n_2]$$

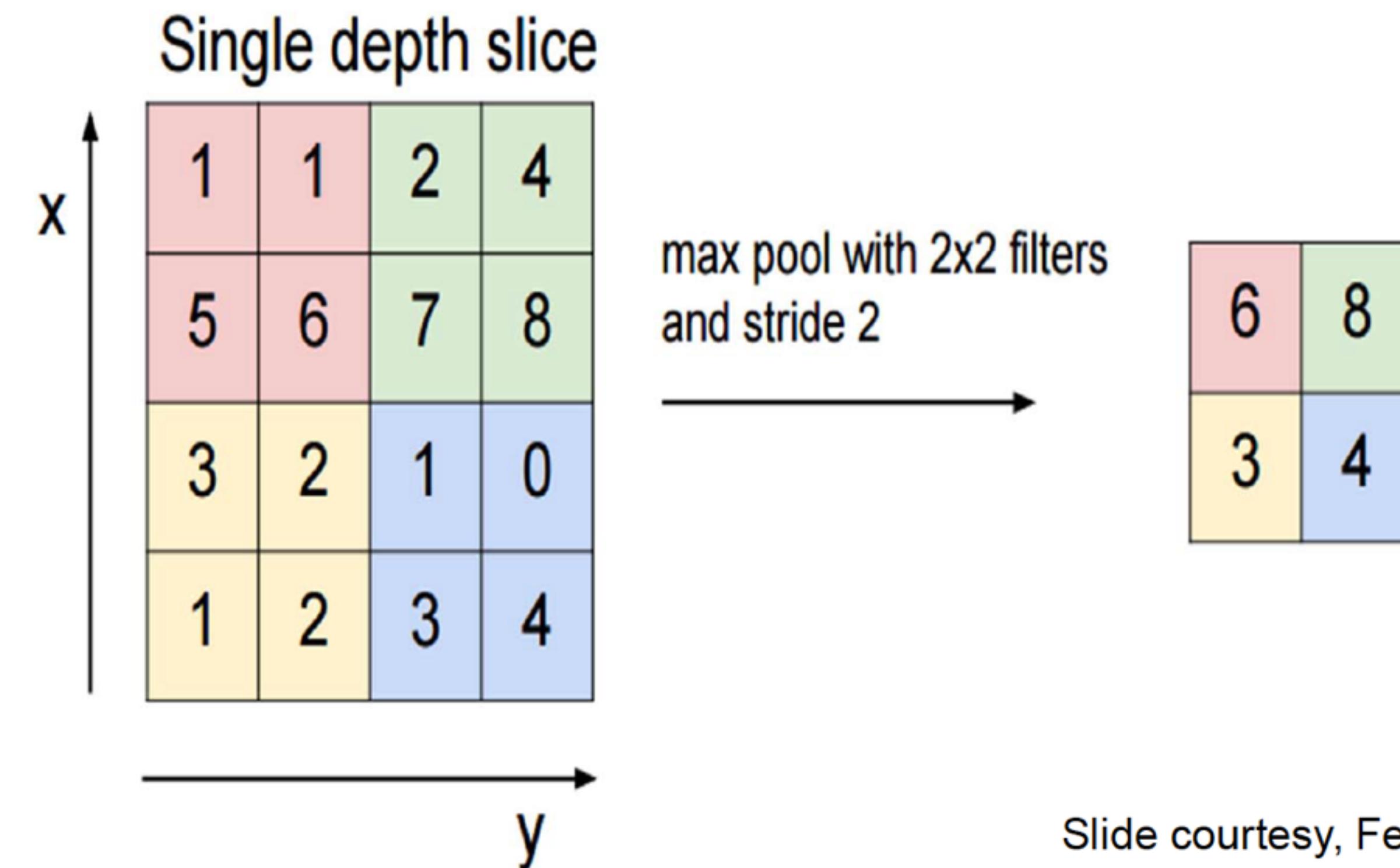
Sometimes we also add a bias term b , $y = Wx + b$,
like what we have done for ordinary NN

**Short question: Will convolution layers
introduce nonlinearity?**

Pooling Layers

Optional subtitle

- In ConvNet architectures, Conv layers are often followed by Pool layers
 - makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



Chap 8

PAC

- PAC



Choosing among several classifiers

A fictional example

Suppose Alibaba holds a competition for the best face recognition classifier (+1 if image contains a face, -1, otherwise)

Lots of teams compete ...

Alibaba get back 20,000 recognition algorithm

They evaluate all 20,000 algorithm on m labelled images which is not previously shown to the competitors) and chooses a winner.

The winner obtains 98% accuracy on m labelled images!

Alibaba has a face recognition algorithm that is known to be 95% accurate,

- Should they deploy the winner's algorithm instead?
- Can't risk doing worse ... would be a disaster for Alibaba.



How likely is learner to pick a bad hypothesis?

Optional subtitle

$$\Pr(h \text{ gets } m \text{ iid data points right} \mid \text{error}_{\text{true}}(h) \geq \varepsilon) \leq e^{-\varepsilon m}$$

Suppose there are $|H_c|$ hypotheses consistent with the training data

- How likely is learner to pick a bad one, i.e. with *true error* $\geq \varepsilon$?
- We need to a bound that holds for all of them!

$$\begin{aligned} & P(\text{error}_{\text{true}}(h_1) \geq \varepsilon \text{ OR } \text{error}_{\text{true}}(h_2) \geq \varepsilon \text{ OR } \dots \text{ OR } \text{error}_{\text{true}}(h_{|H_c|}) \geq \varepsilon) \\ & \leq \sum_k P(\text{error}_{\text{true}}(h_k) \geq \varepsilon) \quad \leftarrow \text{Union bound} \\ & \leq \sum_k (1-\varepsilon)^m \quad \leftarrow \text{bound on individual } h_j \text{s} \\ & \leq |H| (1-\varepsilon)^m \quad \leftarrow |H_c| \leq |H| \\ & \leq |H| e^{-m\varepsilon} \quad \leftarrow (1-\varepsilon) \leq e^{-\varepsilon} \text{ for } 0 \leq \varepsilon \leq 1 \end{aligned}$$



Using a PAC bound

Typically, 2 use cases:

- 1: Pick ϵ and δ , compute m
- 2: Pick m and δ , compute ϵ

Argument: Since for all h we know that

$$P(\text{error}_{\text{true}}(h) > \epsilon) \leq |H|e^{-m\epsilon}$$

... with probability $1-\delta$ the following holds... (either case 1 or case 2)

$$p(\text{error}_{\text{true}}(h) \geq \epsilon) \leq |H|e^{-m\epsilon} \leq \delta \quad]$$

Says: we are willing to tolerate a δ probability of having $\geq \epsilon$ error

$$\ln(|H|e^{-m\epsilon}) \leq \ln \delta$$

$$\ln |H| - m\epsilon \leq \ln \delta$$

Case 1

$$m \geq \frac{\ln |H| + \ln \frac{1}{\delta}}{\epsilon}$$

Log dependence on $|H|$,

ϵ has stronger influence than δ

Case 2

$$\epsilon \geq \frac{\ln |H| + \ln \frac{1}{\delta}}{m}$$

ϵ shrinks at rate $O(1/m)$

Generalisation bound for $|H|$ hypothesis

Optional subtitle

Theorem: Hypothesis space H finite, dataset D with m i.i.d. samples, $0 < \varepsilon < 1$: for any learned hypothesis h :

$$P(\text{error}_{\text{true}}(h) - \text{error}_{\text{train}}(h) > \epsilon) \leq |H|e^{-2m\epsilon^2}$$

Why? Same reasoning as before. Use the Union bound over individual Chernoff bounds



PAC bound and Bias-Variance tradeoff

Optional subtitle

for all h , with probability at least $1-\delta$:

$$\text{error}_{\text{true}}(h) \leq \text{error}_{\text{train}}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}}$$

“bias” “variance”

- For large $|H|$
 - low bias (assuming we can find a good h)
 - high variance (because bound is looser)
- For small $|H|$
 - high bias (is there a good h ?)
 - low variance (tighter bound)

PAC bound: How much data?

Optional subtitle

$$P(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon) \leq |H|e^{-2m\epsilon^2}$$

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}}$$

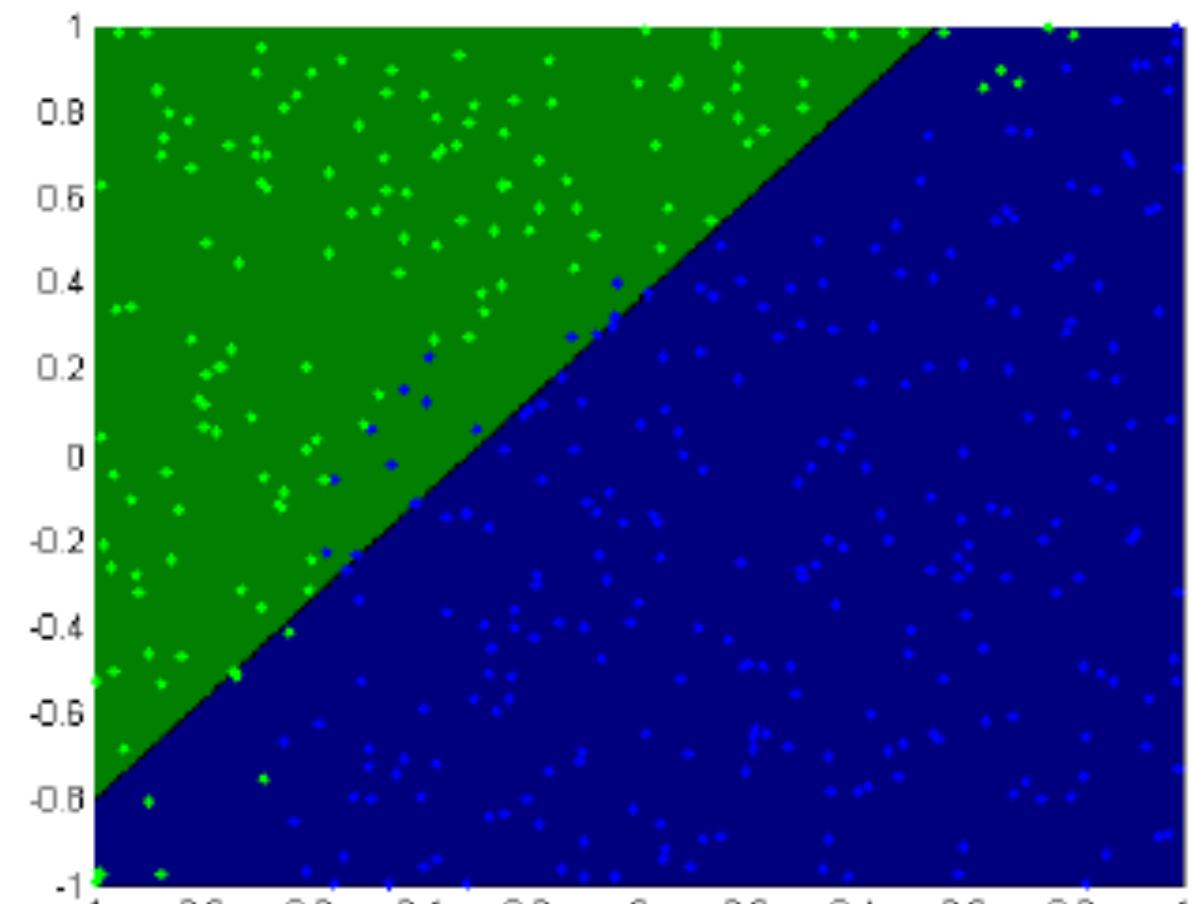
- Given δ, ϵ how big should m be?

$$m \geq \frac{1}{2\epsilon^2} \left(\ln |H| + \ln \frac{1}{\delta} \right)$$

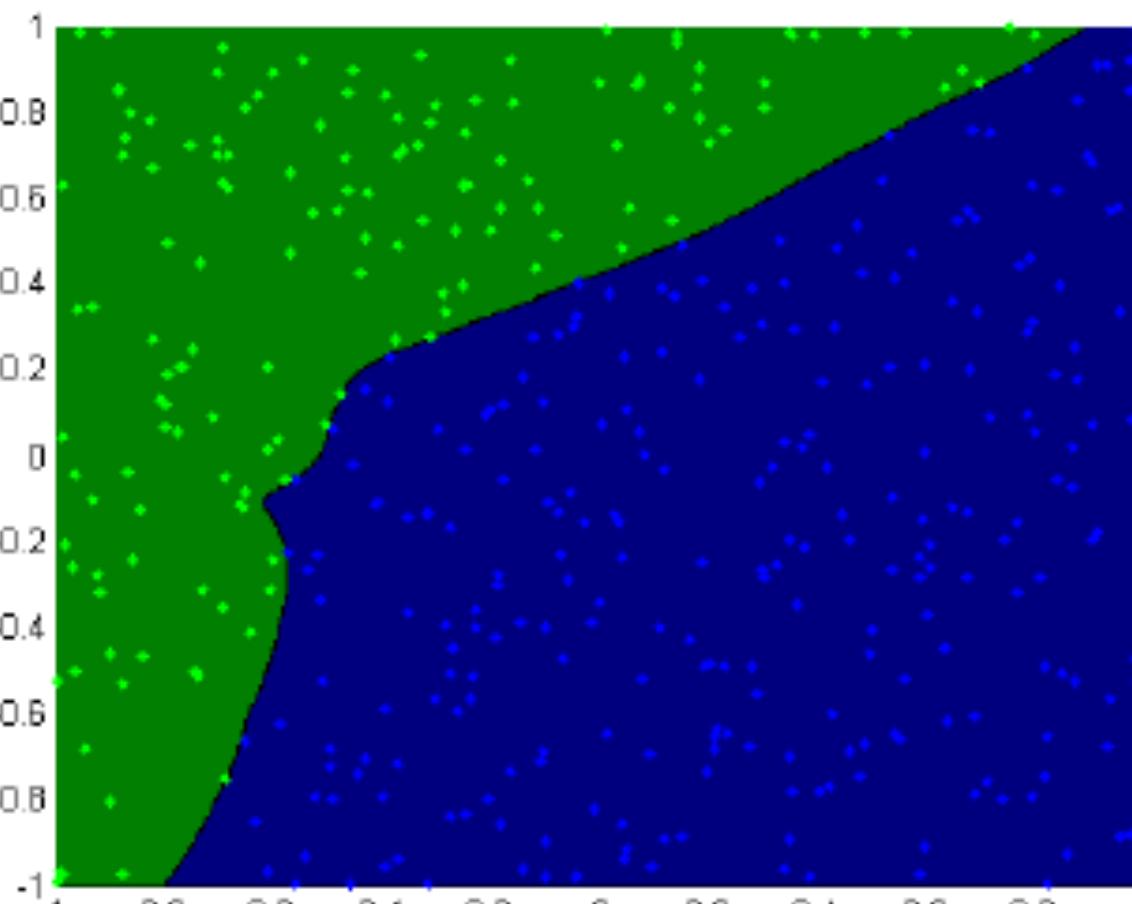


Questions?





(a)



(b)

a. is logistic regression, b. is neural network
 Logistic regression classifiers have linear decision boundaries .

$$p(y = 1|x) = \sigma(\mathbf{w}^T x + b).$$

Neural networks have nonlinear decision boundaries due to nonlinearities in the hidden layers.

- One of the curves is from a logistic regression classifier and the other one is from a neural network.
- Which one do you think is which, and why?

Why is it more difficult to learn the parameters of a neural network than a logistic regression classifier?

Optimization

- Learning the parameters for a neural network is a non-convex optimization problem. Easy to get stuck in local minima!
- Learning for logistic regression is a convex optimization problem We can always find the globally optimal solution.

Computation

- Computing gradients for a neural network is more expensive.
- We must first forward propagate the input to compute the error...
- then back-propagate the error to compute gradients for each parameter.



What kinds of data are expected to be (in)appropriate for a K-nearest neighbor classifier?

- KNN handles non-linearly separable classes much better than logistic regression.
- Notion of distance becomes important.
 - Features with larger ranges normalize scale.
 - Irrelevant or correlated features may have to eliminate or weight.
 - Distances become larger for higher dimensions.
- Must store all training cases becomes an issue for large training set size
- Sensitive to class noise



Using R in interactive environment (ipython, Jupiter)

Optional subtitle

1, <https://irkernel.github.io>

(<https://irkernel.github.io/installation/>)

2, Using Anaconda

- Create an environment
- Install R environment: **conda install -c r r-essentials**

1/2) Installing via supplied binary packages
(default on Windows + Mac OS X)

You can install all packages using the following lines in an R console:

```
install.packages(c('repr', 'IRdisplay', 'evaluate', 'crayon', 'pbZIP4', 'devtools', 'u
devtools::install_github('IRkernel/IRkernel')
# Don't forget step 2/2!
```

To update the IRkernel package, which is not yet on CRAN, you have to rerun the `devtools::` line. For the other packages, a simple `update.packages()` is sufficient.

2/2) Making the kernel available to Jupyter

If you haven't done this already, you will have to make Jupyter see the newly installed R kernel by installing a kernel spec.

The kernel spec can be installed for the current user with the following line from R:

```
IRkernel::installspec()
```

To install system-wide, set `user` to `False` in the `installspec` command:

```
IRkernel::installspec(user = FALSE)
```

