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**
  - **T**ask: playing checkers games;
  - **E**xperience: obtained by playing games against itself;
  - **P**erformance 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.

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

Performance:

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

$l\left(y, y^{'}\right)$ 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,

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

- Regression Vs. Classification problems,
  - ***Regression***: *Y* is 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:

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

So if presented with a training set $\mathcal{D}$, it provides a decision rule/function $f : \mathcal{X} \to \mathcal{Y}$

Let *L* be a learning system.

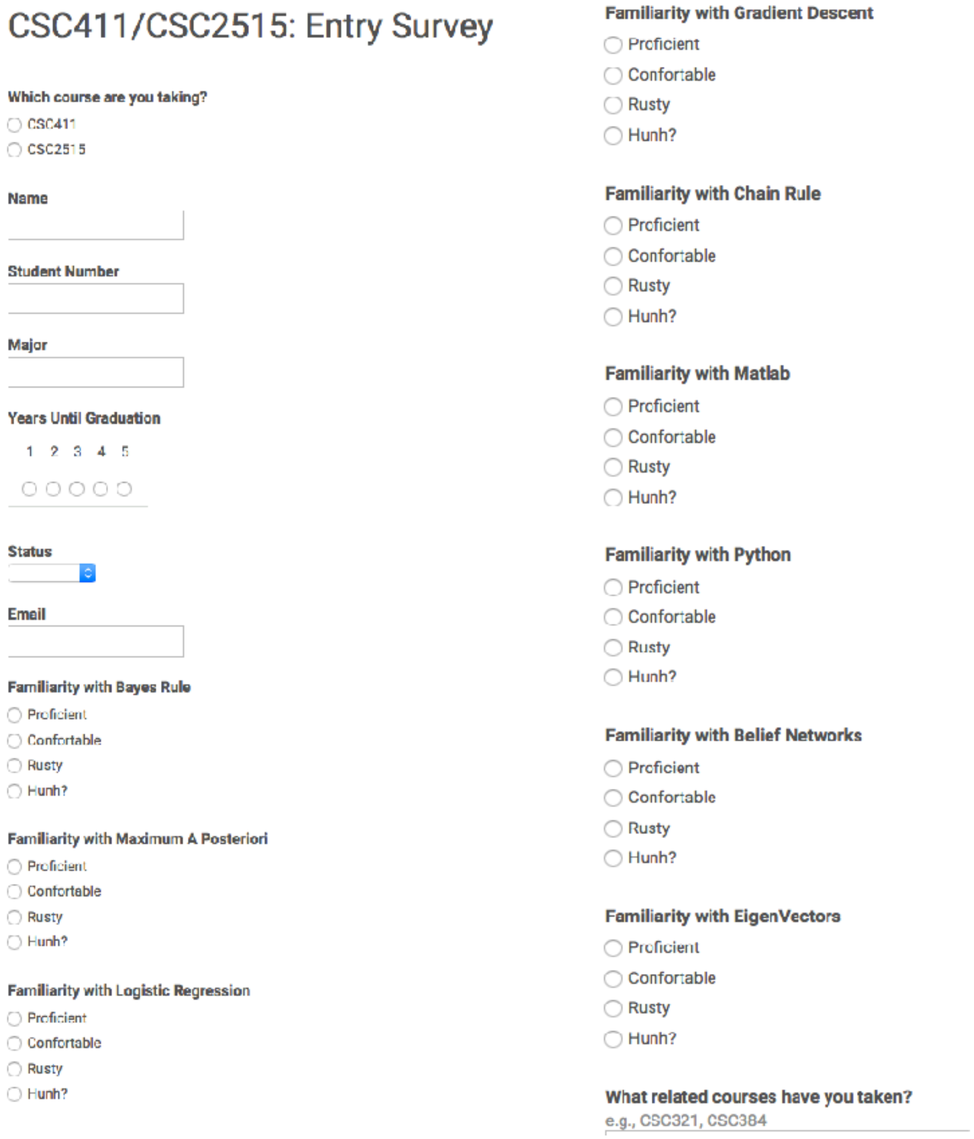
- Process of computing is $f = L(\mathcal{D})$ called training (phase).

- 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)

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

Settings:

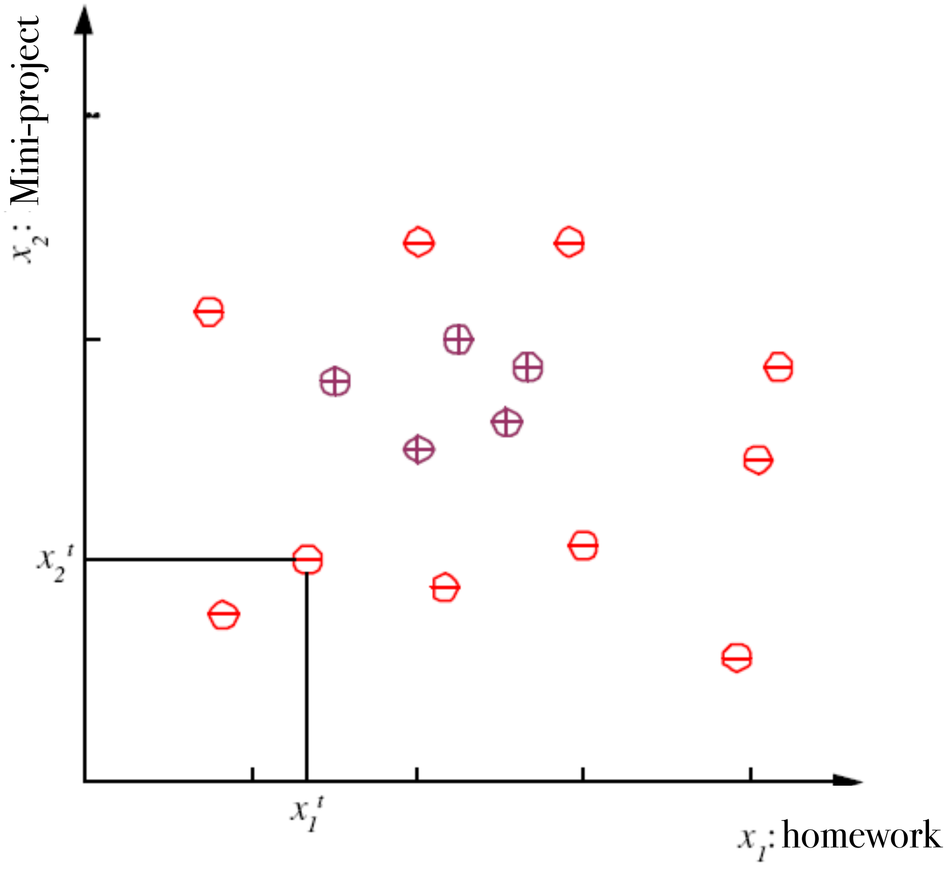
- Class *C* of a “good score”
  - Knowledge extraction: What do people expect from a good score?
- Output:

  Positive (+) and negative (–) examples
- Input representation:

  x1: homework, x2 : Mini-projects

Training Set

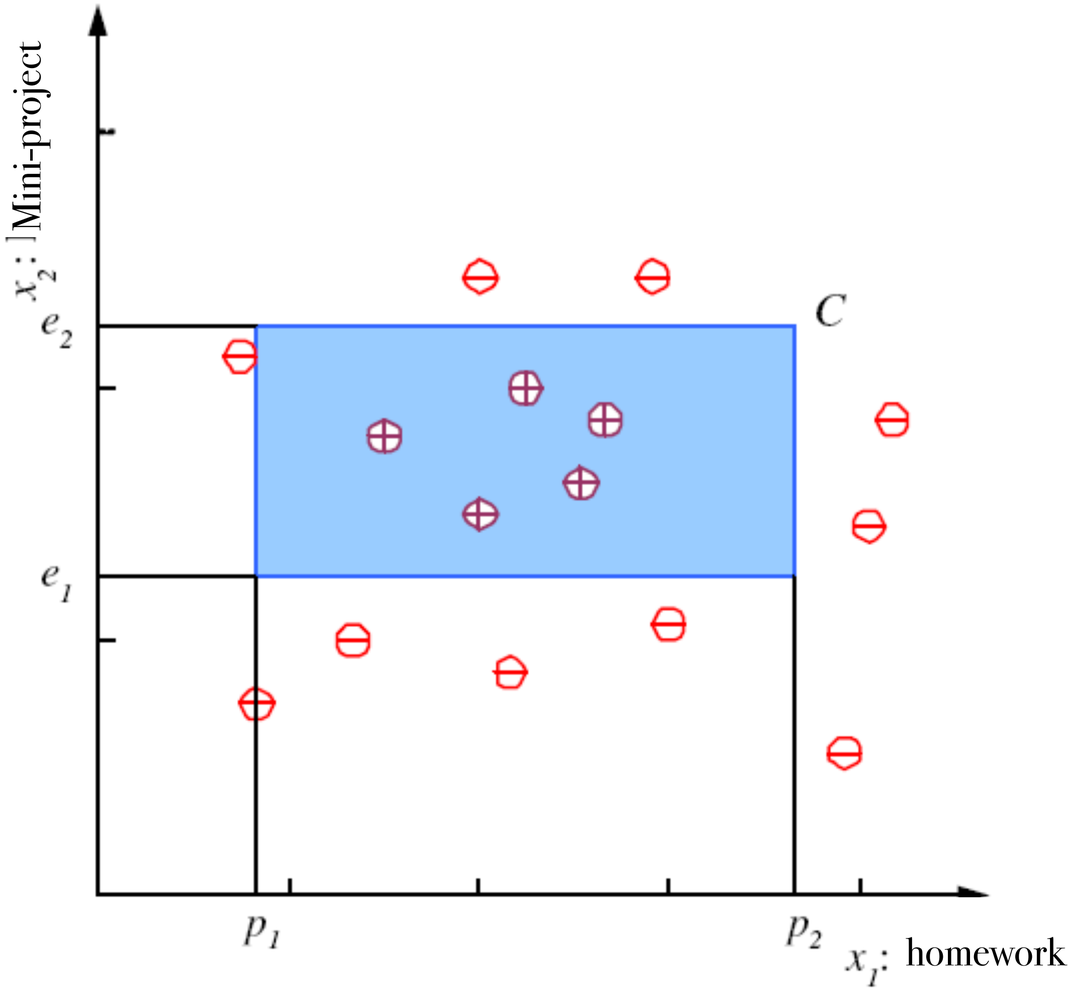
$$\mathcal{D} = \left\{ \left(x^1, y^1 \right), \left(x^2, y^2 \right), \cdots, \left(x^N, y^N \right) \right\} \subset \mathcal{X} \times \mathcal{Y}$$



$$y = \begin{cases} 1 & if\ x\ is\ positive \\ 0 & if\ x\ is\ negative \end{cases}$$

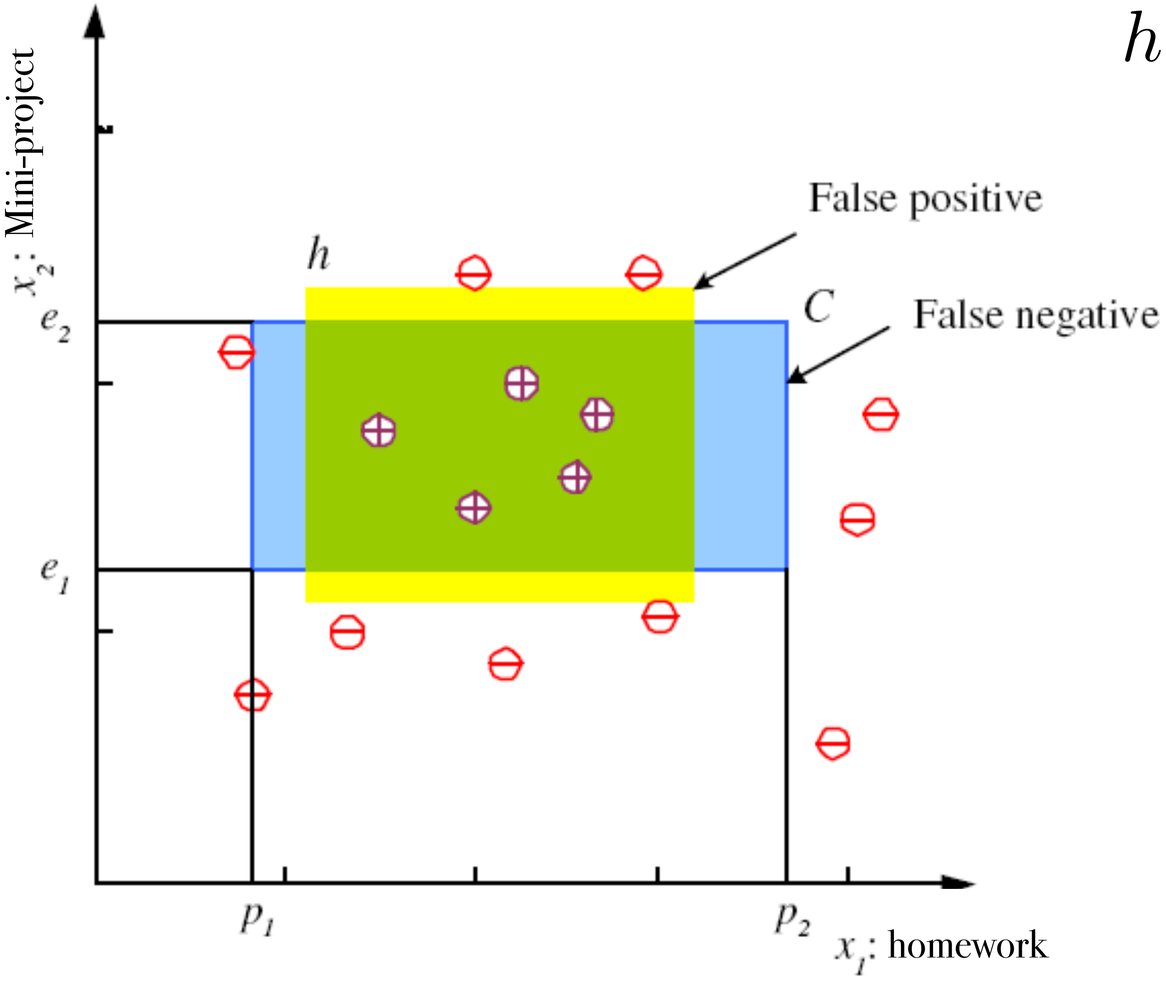
$$x^t = \begin{bmatrix} x_1^t \\ x_2^t \end{bmatrix}$$

Class *C*



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

Hypothesis class $\mathcal{H}$



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

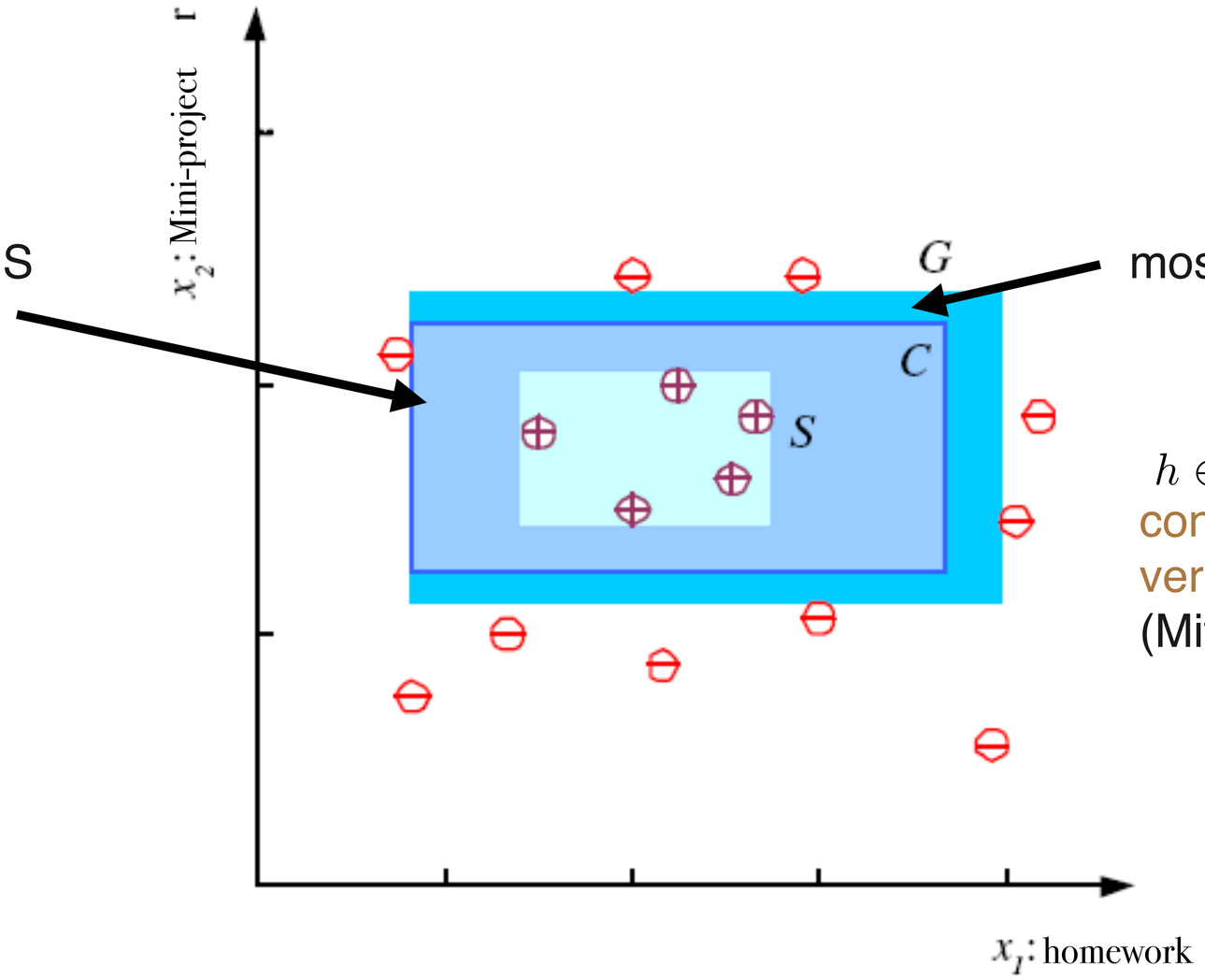
Error of h on $\mathcal{H}$

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

Version Space (Mitchell, 1997)

most specific hypothesis, S

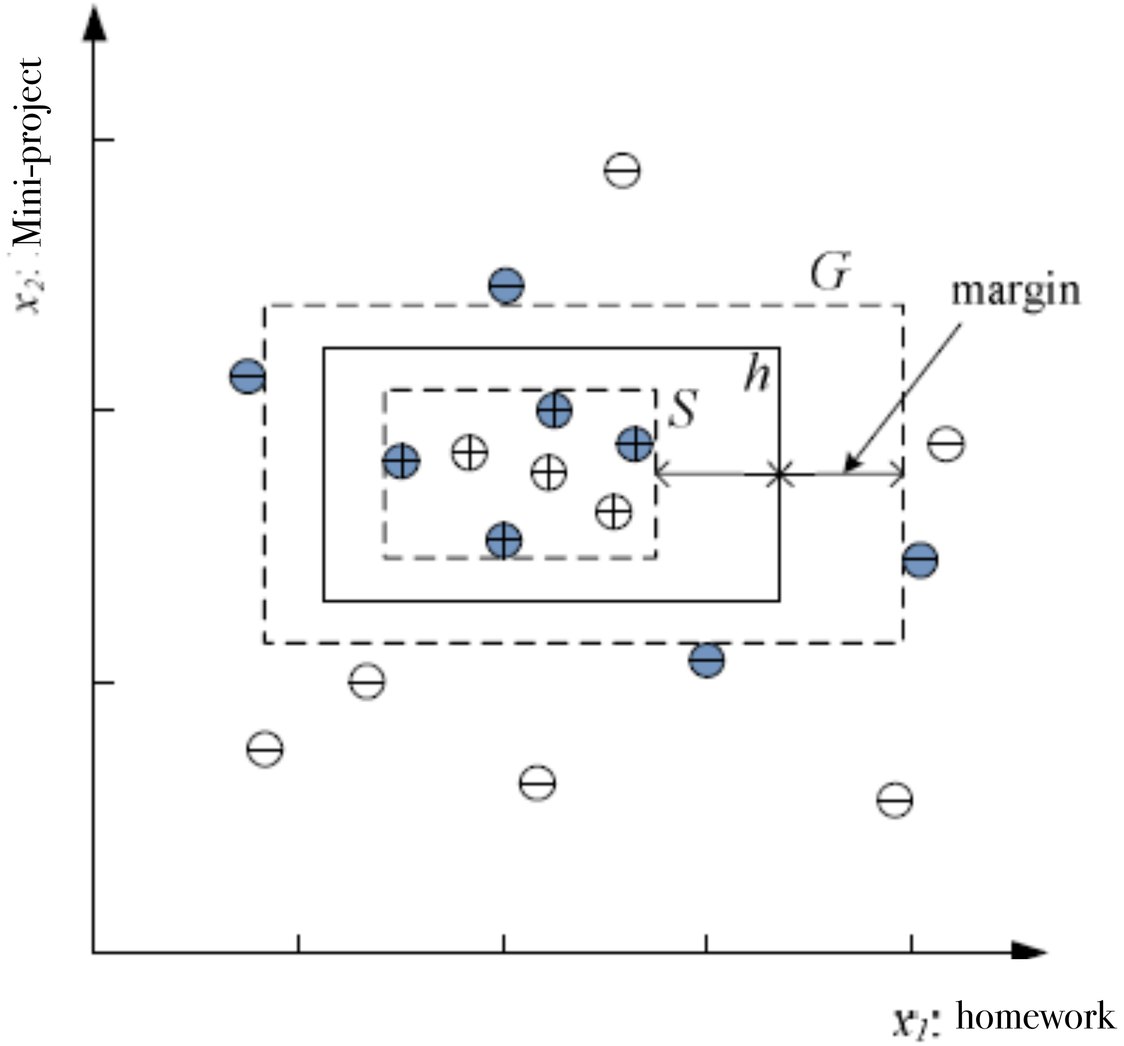
most general hypothesis, G



$h \in \mathcal{H}$ between S and G is consistent and make up the version space
(Mitchell, 1997)

Margin

One solution:
Choose *h* with
largest margin



Chap 2
Linear Regression

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

Simple Linear Regression

Parametric method

- ***Simple Linear Regression***: *Y* is 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 "≈" 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, ˆ , 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?

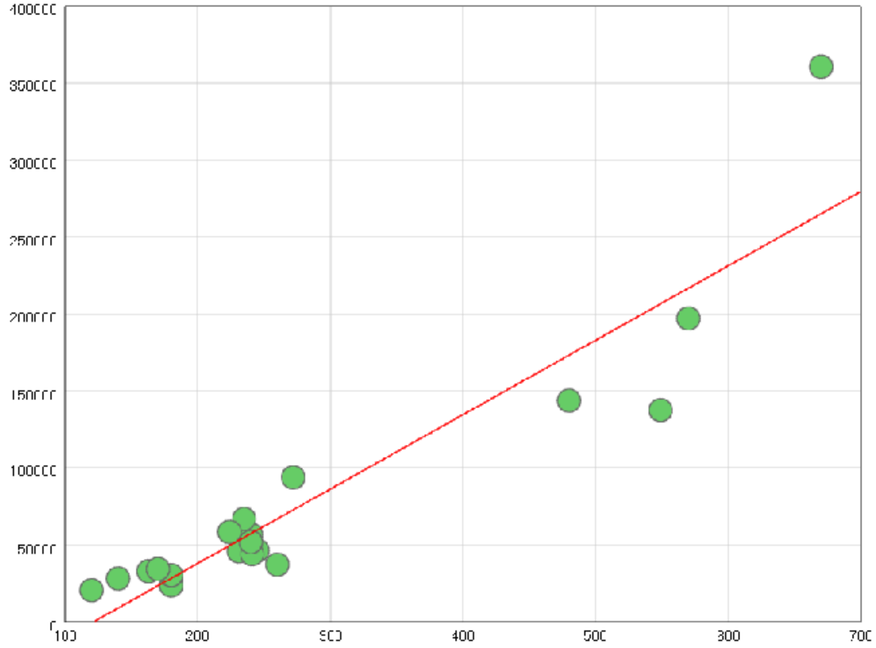
Optimizing the Objective

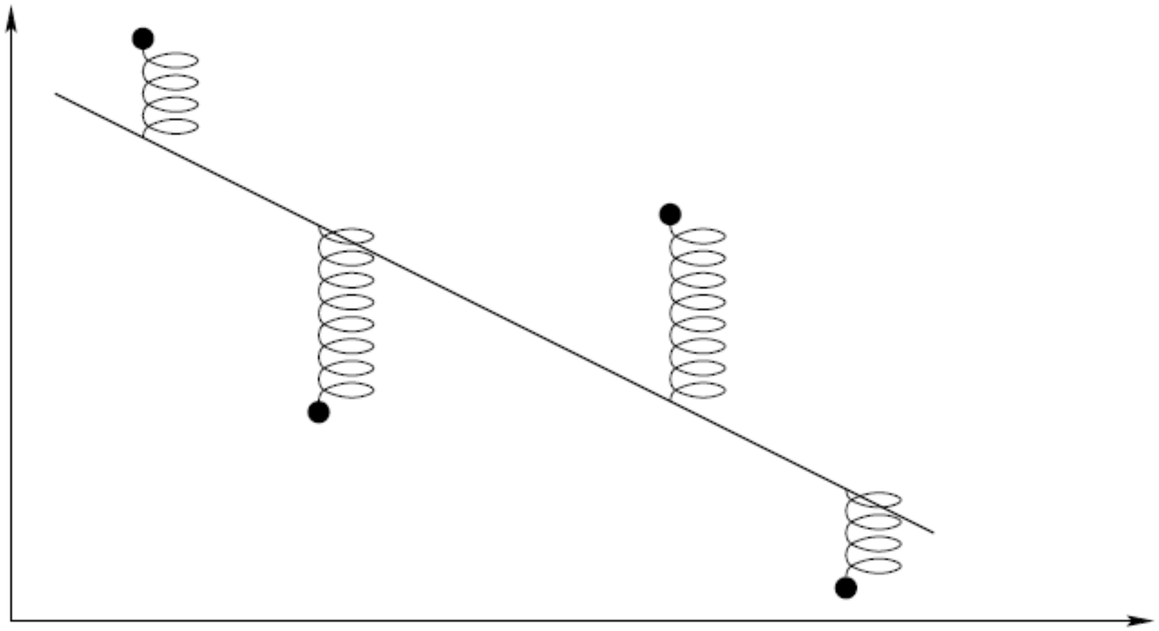
$$Y = \beta_0 + \beta_1 X + \epsilon.$$

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

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

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

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

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

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

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?

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

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:

Data:
X, y, X_{test}, y_{test}

1. Train:
$model = fit(X, y)$

2. Predict test set labels
$\hat{y} = predict(model, X_{test})$

3. Evaluate
$error = diff(\hat{y}, y_{test})$

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.

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

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

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

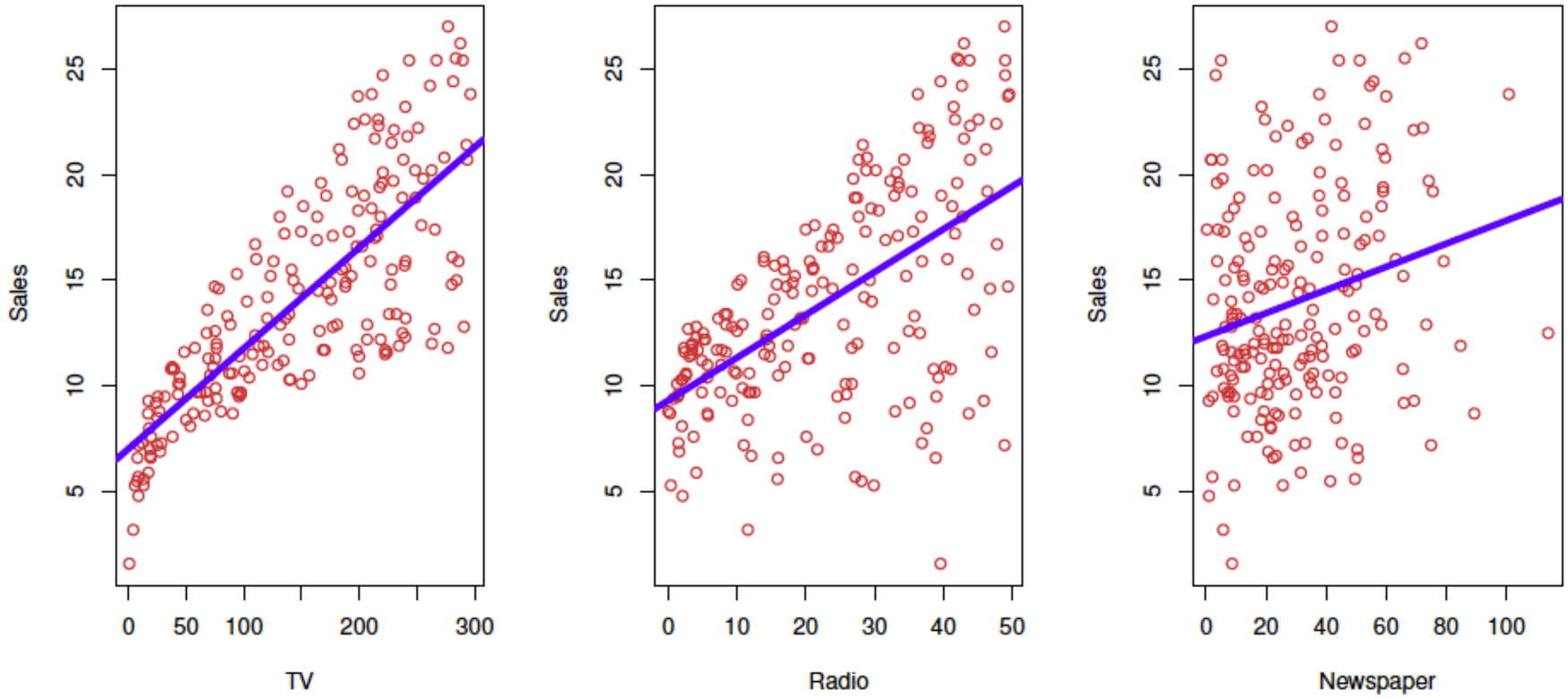
Optional subtitle

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

Multiple Linear Regression

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

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



Multiple Linear Regression

Sec 3.2 of “The Elements of Statistical Learning”

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

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:

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

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

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

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

Note that: For a unique solution, the matrix X^TX must be full rank.

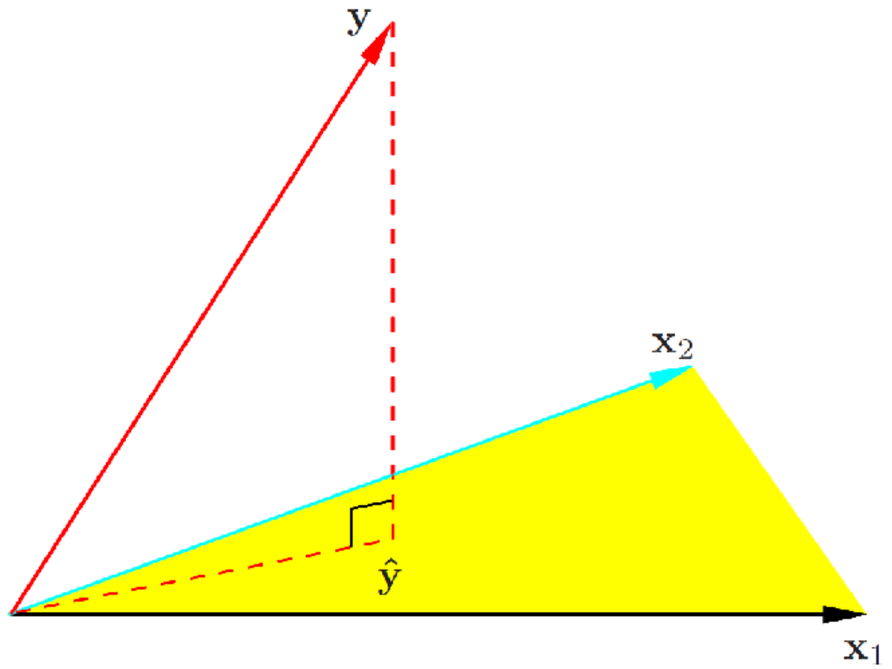
$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

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

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

Orthogonal Projection of Y on the space spanned by the columns of X.

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



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 shrunken 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 t a linear regression model by least squares.

Shrinkage Methods(1)

- Ridge Regression

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

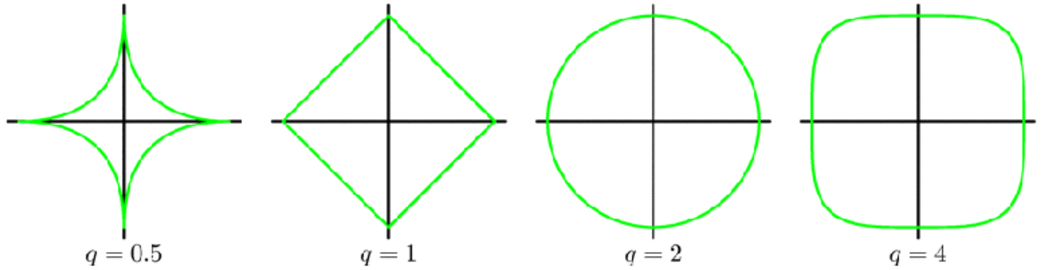
where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately.

- Lasso

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

Shrinkage Methods in Matrix Form

$$\underset{\beta}{\operatorname{argmin}} \parallel Y - X\beta \parallel_2^2 + \lambda \parallel \beta \parallel_q$$



q=0, L_0-norm; —> 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., $\parallel \beta \parallel_0 = \sharp\sigma(\beta)$, $\sharp$ stands for cardinality; $\sigma(\beta)$ is the support of β

q<1, **hard-thresholding**

q=1, L_1-norm —> Lasso (convex), a.k.a., **soft-thresholding**.

q<=1 used for outlier detection [2,3].

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

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

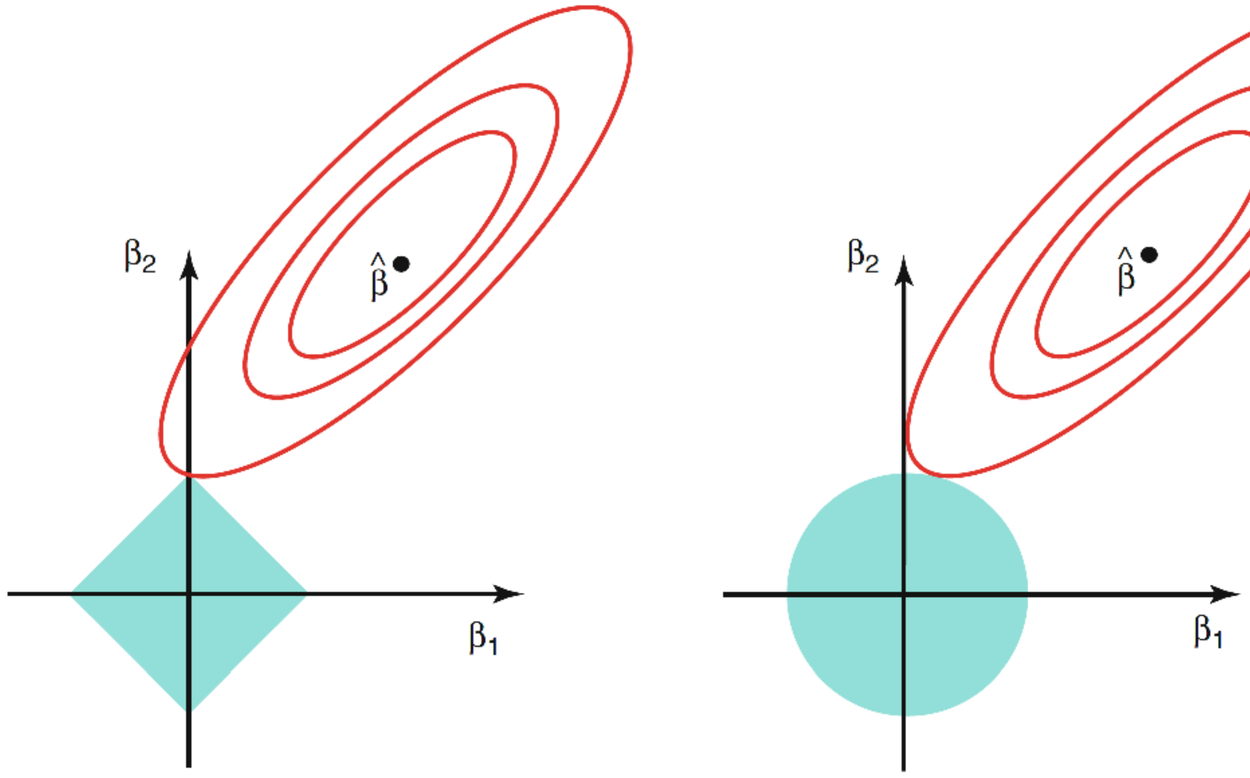
$$\parallel \beta \parallel_q = \left(\sum_{i=1}^{p} |\beta_i|^q \right)^{\frac{1}{q}}$$

[1] Mila Nikolova, Description of the minimizers of least squares regularized with ℓ0-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

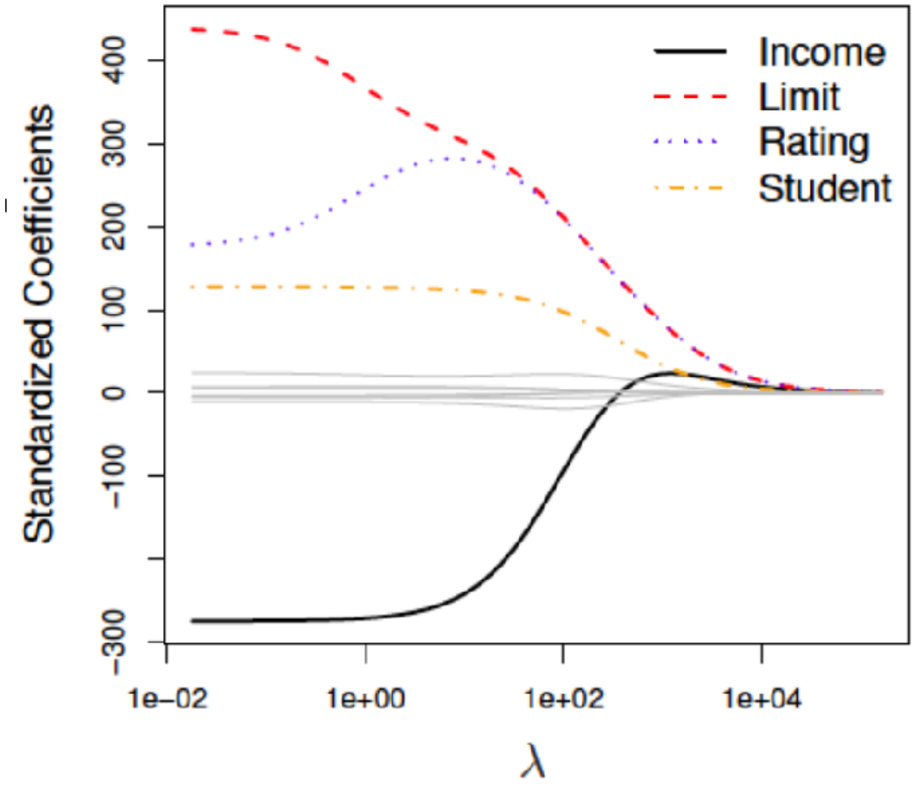
Regularized Least Squares

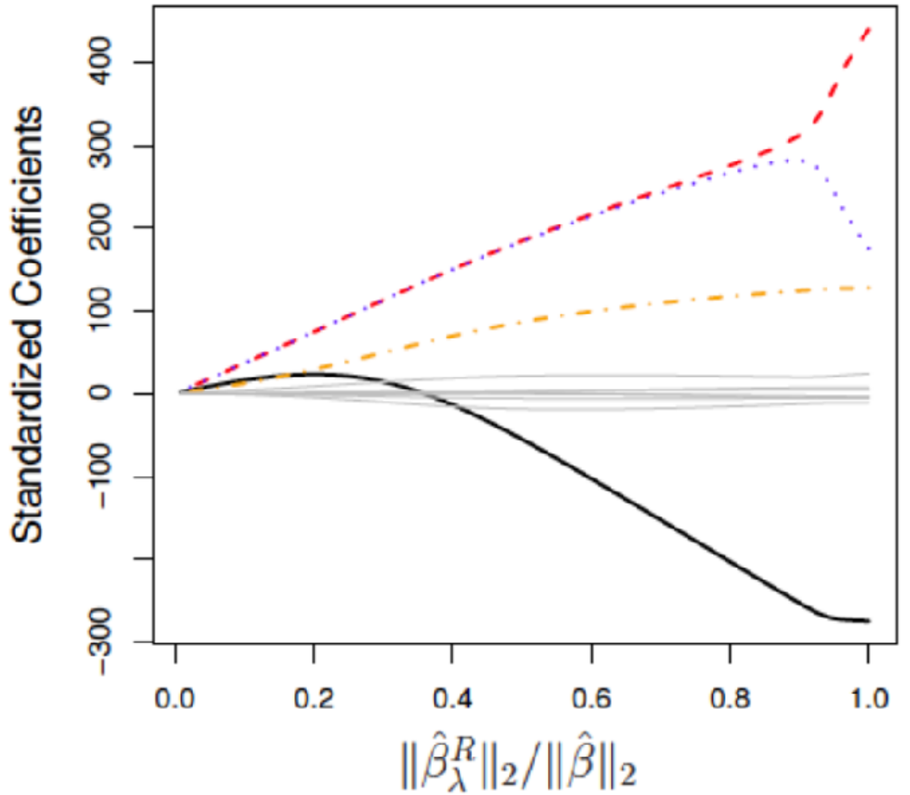


ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero.

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





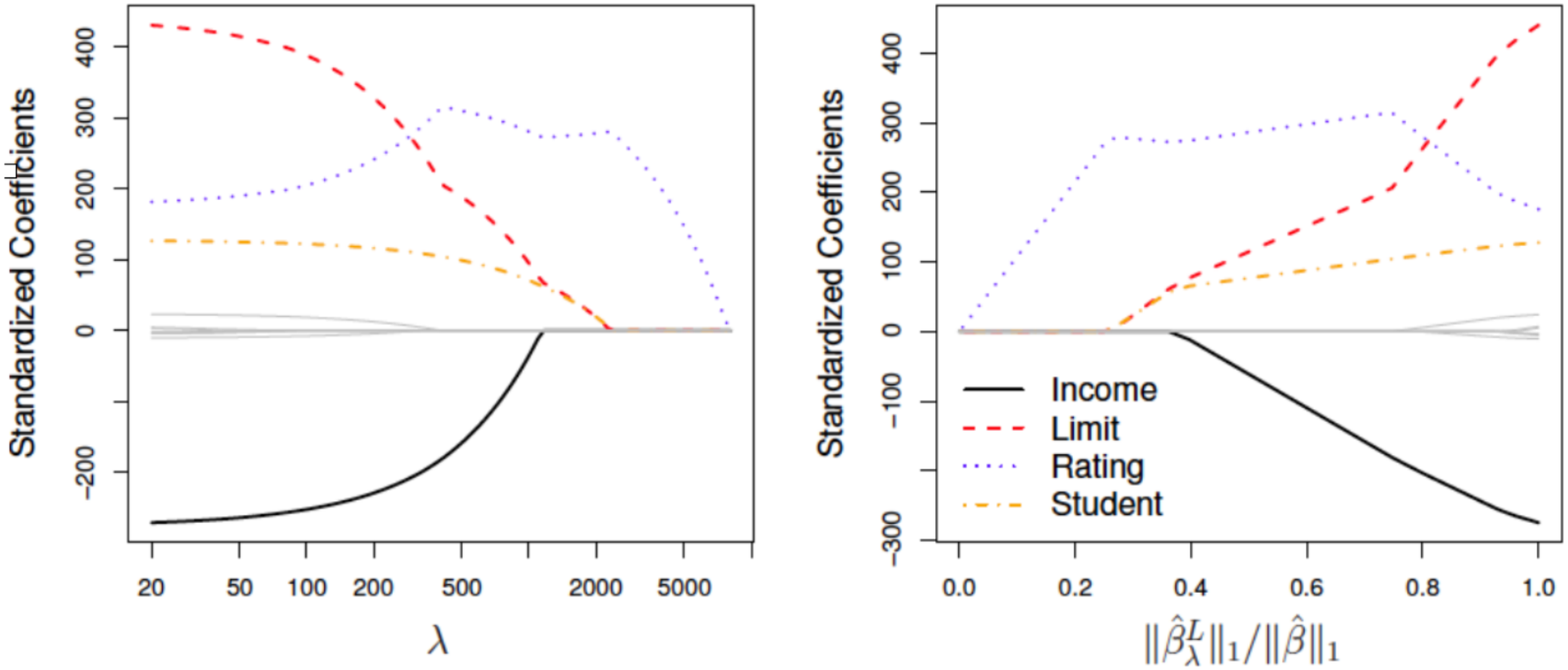
$$\beta\,(\lambda) = \underset{\beta}{\operatorname{argmin}} \parallel Y - X\beta \parallel_2^2 + \lambda \parallel \beta \parallel_2$$

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

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

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 I that is, models that involve only a subset of the variables.

Alternatives to Squared Error

Huber M-estimator:

$$\min J_h(\Theta) = \rho_\lambda(\delta_0\Theta - Y) \tag{14}$$

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

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

Chap 4 Linear Classification

- Logistic Regressin
- Linear Discriminant Analysis
- Naïve Bayes

Logistic Regression

Optional subtitle

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

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

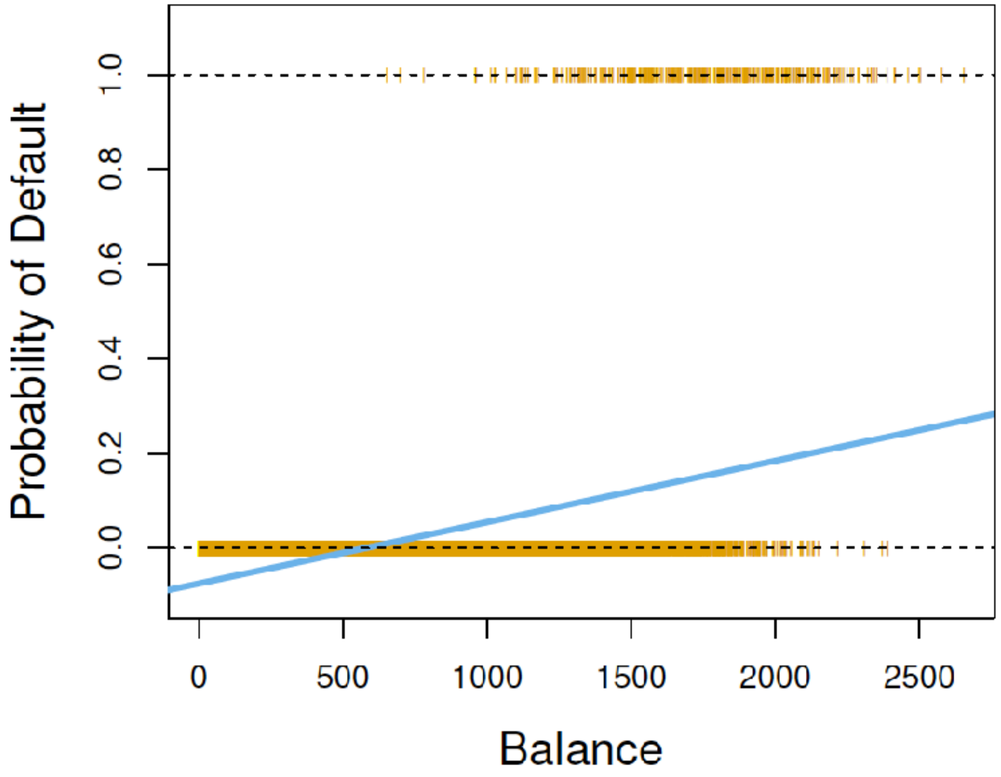
where the sigmoid is defined as

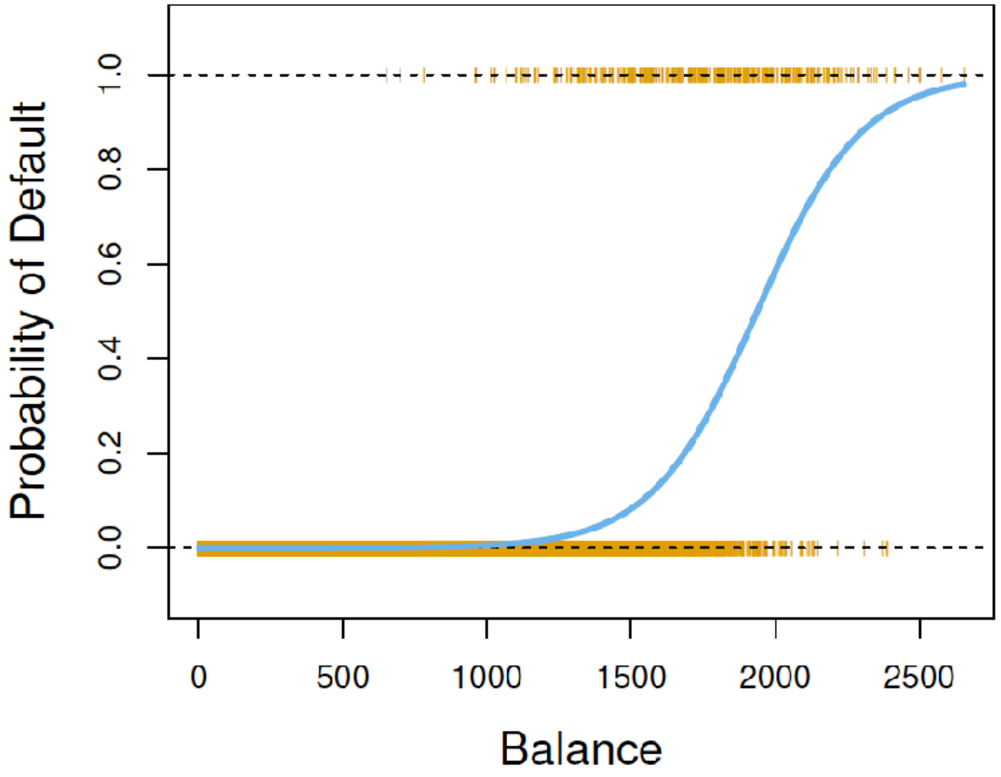
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

  - 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 $sign(\cdot)$

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 $p(t^{(1)}, \cdots, t^{(N)} | \mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)})$
- Assuming that the training examples are sampled IID: independent and identically distributed

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

- 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

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

- 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 $\{t\} = (t^{(1)}, \cdots, t^{(N)})$, and $\{\mathbf{x}\} = (\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)})$

- 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:

$$P(y=i \mid X) = \frac{P(X \mid y=i)P(y=i)}{P(X)} \overset{def}{=} q_i(X)$$

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:

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

Product of probability terms

Specific model for attribute *j*

- Using this idea the full classification rule becomes:

$$\hat{y} = \arg\max_v \; p(y = v \mid X)$$

$$= \arg\max_v \; \frac{p(X \mid y = v)\, p(y = v)}{p(X)}$$

$$= \arg\max_v \; \prod_j p_j(x^j \mid y = v)\, p(y = v)$$

v are the classes we have

Conditional likelihood: Full version

Optional subtitle

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

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 assumes 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 Regressin
- Linear Discriminant Analysis
- Naïve Bayes

SVM in a Nutshell

Optional subtitle

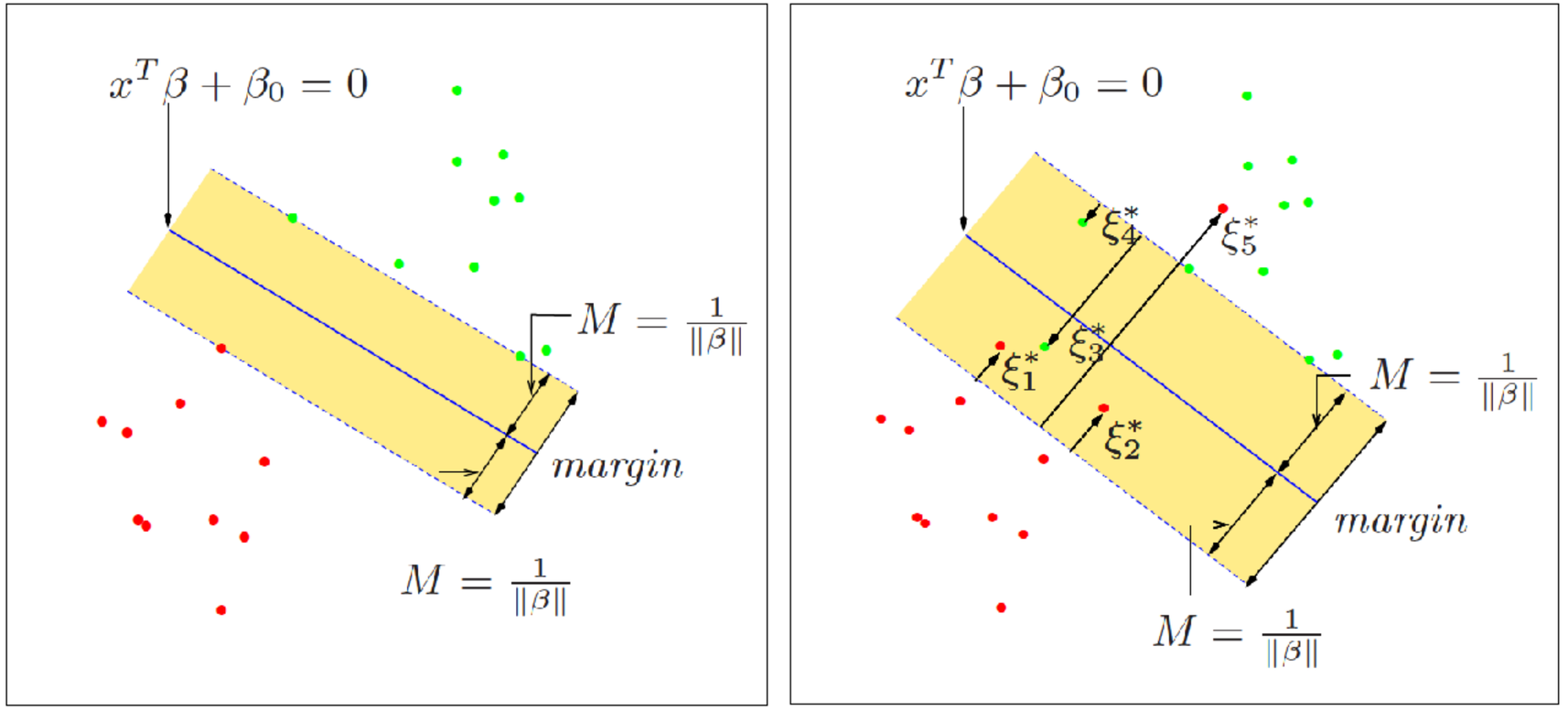
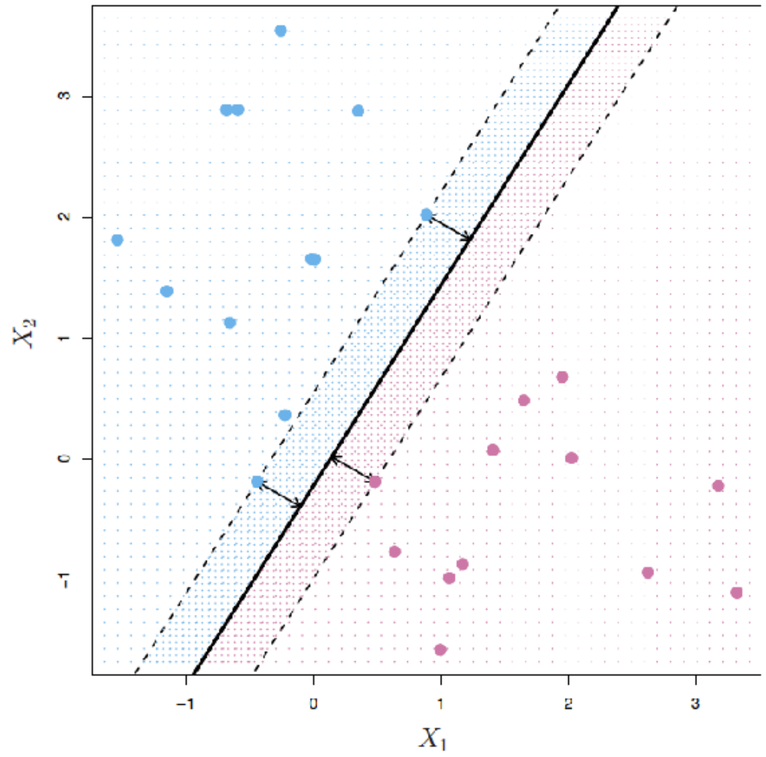


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$. *The right panel shows the nonseparable (overlap) case. The points labeled* ξ_j^* *are on the wrong 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* $\sum \xi_i \leq$ *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

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

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

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{for all } i = 1, \ldots, N.$$

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

$$\max_{\beta,\beta_0,\|\beta\|_2=1} M$$

$$s.t. y_i\left(x_i^T\beta+\beta_0\right)\geqslant M,\quad i=1,\cdots,n \qquad (1)$$

which is equivalent to

$$\min \parallel \beta \parallel_2$$

$$s.t. y_i\left(x_i^T\beta+\beta_0\right)\geqq 1,\quad i=1,\cdots,n$$

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

$$y_i\left(x_i^T\beta+\beta_0\right)\geqslant M\left(1-\xi_i\right)$$

$\forall i,\xi_i\geqslant 0,\sum_i\xi_i\leqslant constant$

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-seperatable cases)

Optional subtitle

$$\begin{aligned} &\min \parallel \beta \parallel_2^2 \\ &s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqslant 1 - \xi_i, \quad i = 1, \cdots, n \qquad (2) \\ &\xi_i \geqslant 0, \sum_i \xi_i \leqslant constant \end{aligned}$$

$$\begin{aligned} &\min \frac{1}{2} \parallel \beta \parallel_2^2 + C \sum_i \xi_i \\ &s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqslant 1 - \xi_i, \quad i = 1, \cdots, n, \xi_i \geqslant 0 \qquad (3) \end{aligned}$$

$$\min \sum_{i=1}^{n} \left[1 - y_i \left(x_i^T \beta + \beta_0\right)\right]_+ + \frac{\lambda}{2} \parallel \beta \parallel_2^2 \qquad (4)$$

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 $D >> d$ 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 $\boldsymbol{\alpha}$ 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

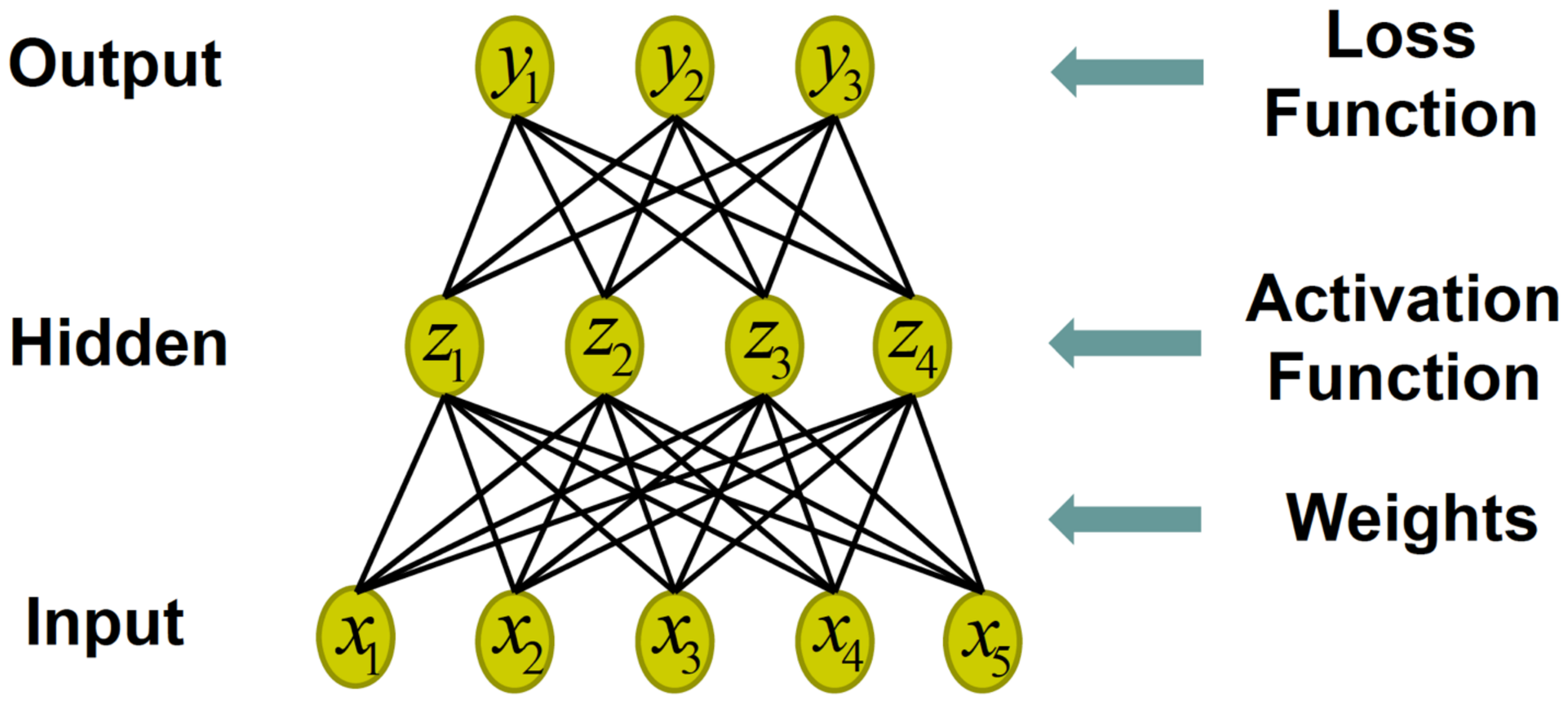
- Linear kernels $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- Polynomial kernels $k(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x}^\top \mathbf{x}'\right)^d$ for any $d > 0$
  - Contains all polynomials terms up to degree d
- Gaussian kernels $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x}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

Chap 7
Neural Network

- Perceptron
- Convolutional Neural Network

Neural Network

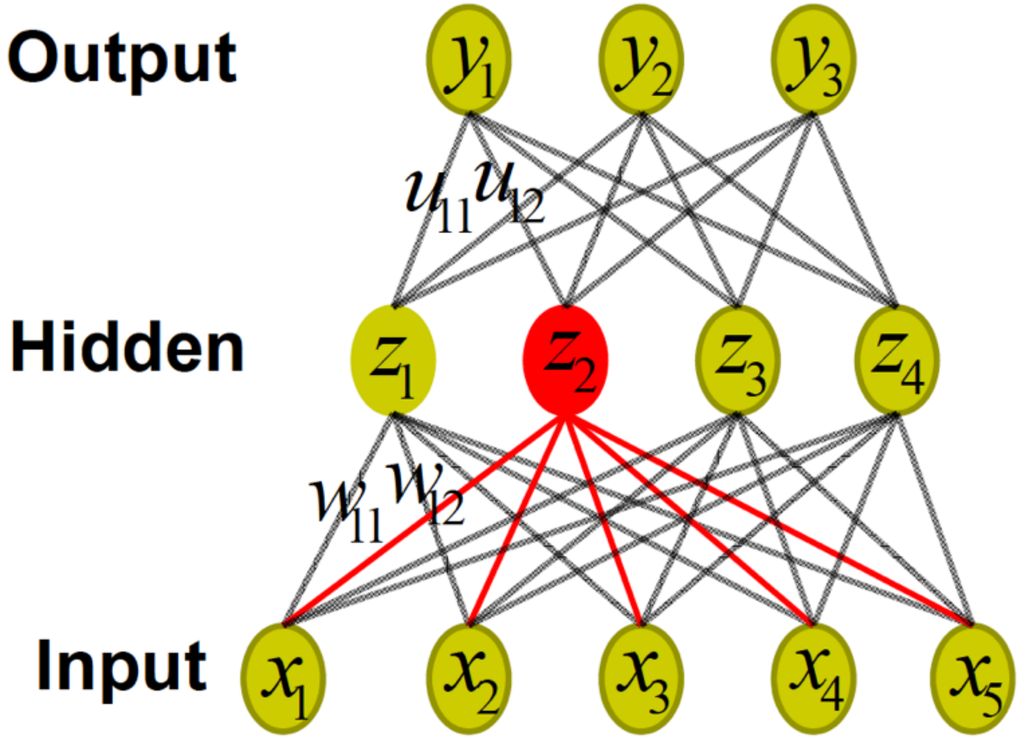
Optional subtitle



Local Computation At Each Unit

Optional subtitle

Linear Combination + Nonlinear Activation



Activation Function

units in the lower layer

$$z_2 = \sigma\left(\sum_{i=1}^{5} w_{2i} x_i\right)$$

Weights connecting to units in the lower layer

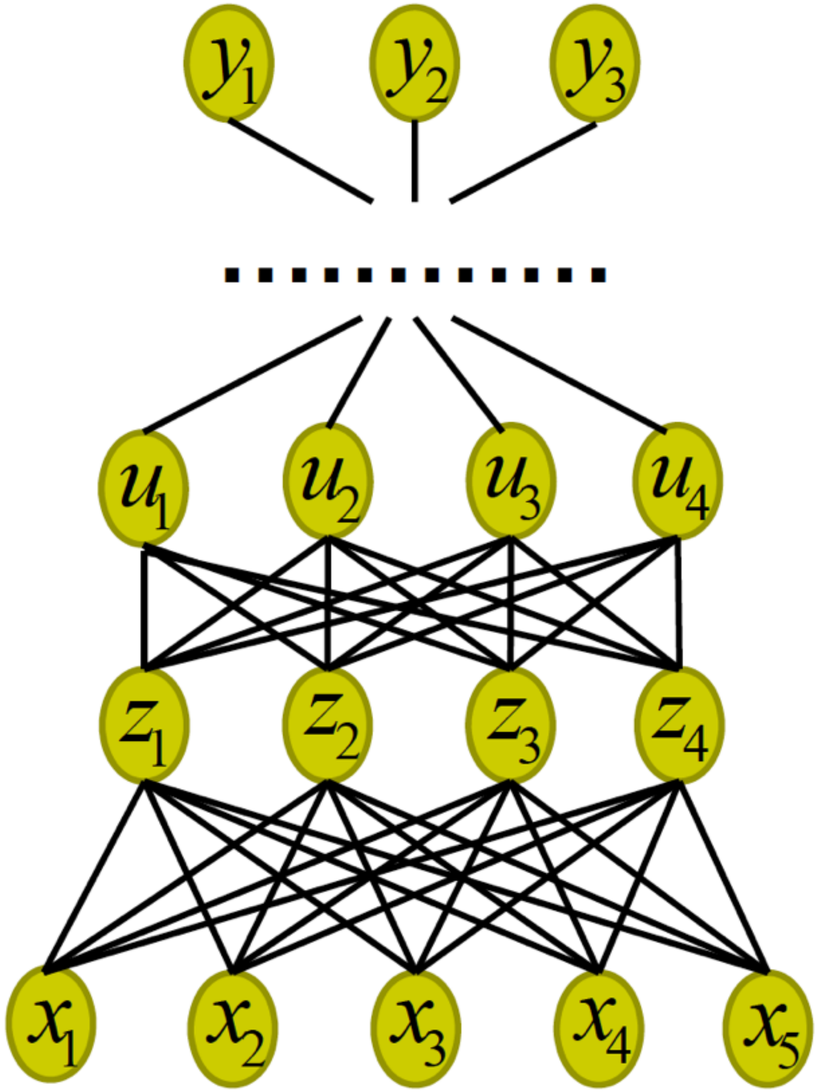
Deep Neural Network

Optional subtitle

Output

More Hidden Layers

Input

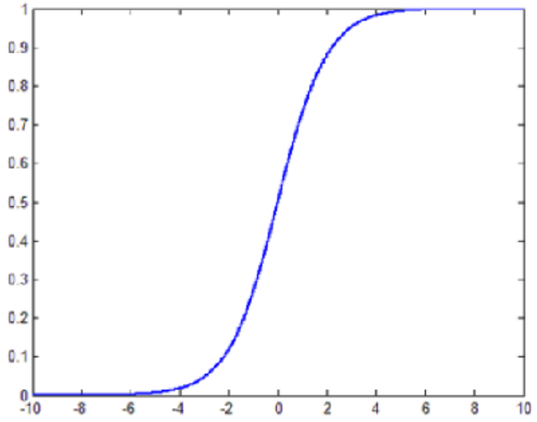


Activation Functions

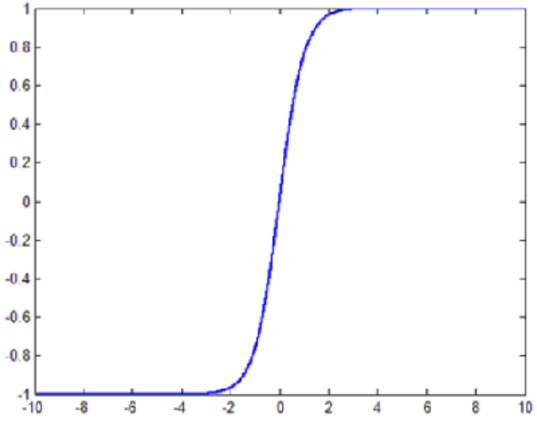
Optional subtitle

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

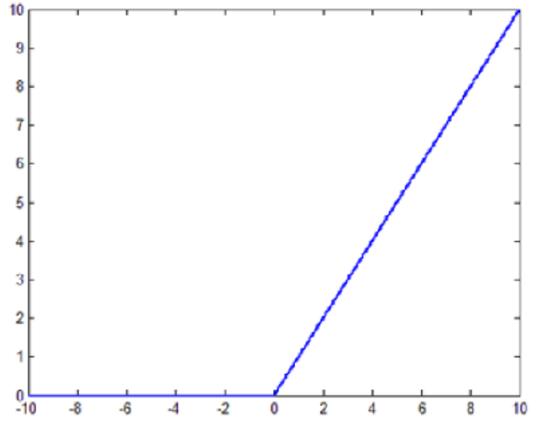
Sigmoid



Tanh



Rectified Linear



More Activation functions

Popular choice of f (·)

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

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

Loss Functions

Optional subtitle

- Squared loss for regression

$$(y - t)^2$$

Prediction　　True value

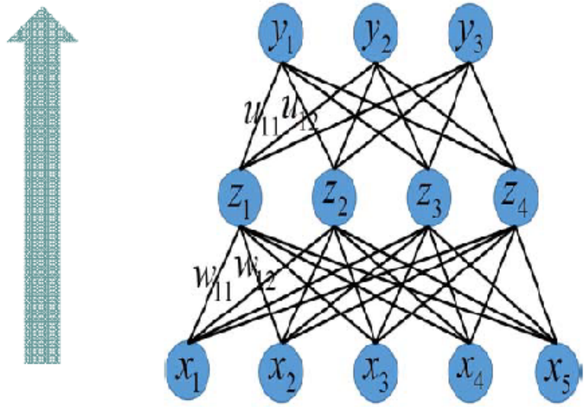
- Cross entropy loss for classification

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

Class label　　Prediction

Neural Network Prediction

- Compute unit values layer by layer in a forward manner



- Prediction function

Activation Function

Input

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

Output

Weights

Neural Network Training

- Goal: compute gradient

$$\frac{\partial L}{\partial w_{ij}}$$

∂L ← Training loss

∂w_{ij} ← Weight between unit i and j

- Apply chain rule

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{w_{ji}}$$

← Linear combination value $a_j = \sum_i w_{ji} z_i$

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

Called error, computed recursively in a backward manner

Neural Network Training

Optional subtitle

- Apply chain rule (cont'd)

$$\frac{\partial a_j}{w_{ji}} = z_i$$ ← 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

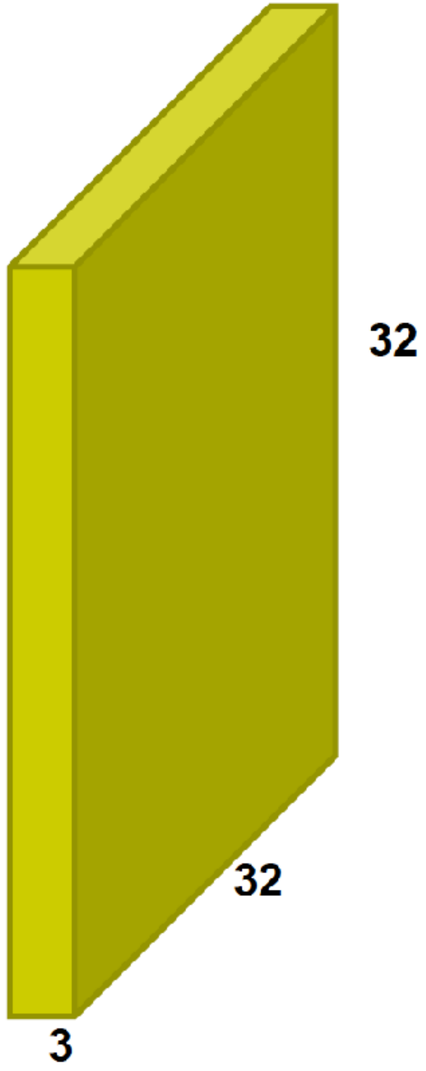
- Pseudo code of BP

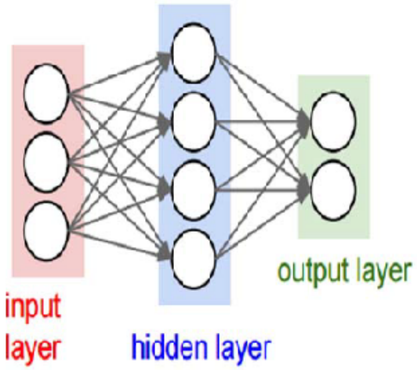
While not converge

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

Local connectivity

Optional subtitle





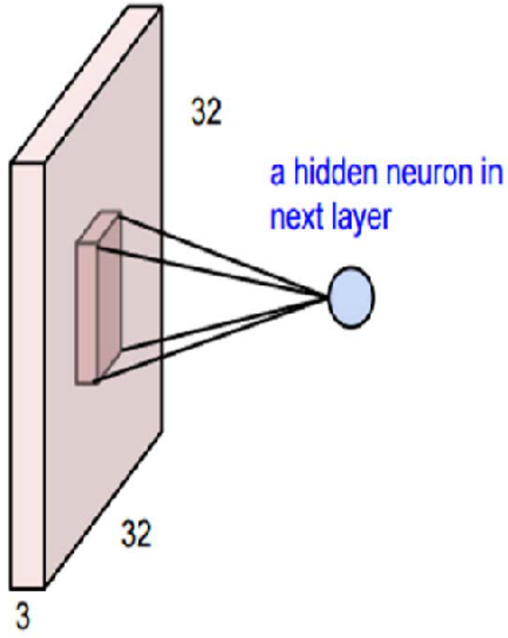


image: 32 * 32 * 3 volume

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

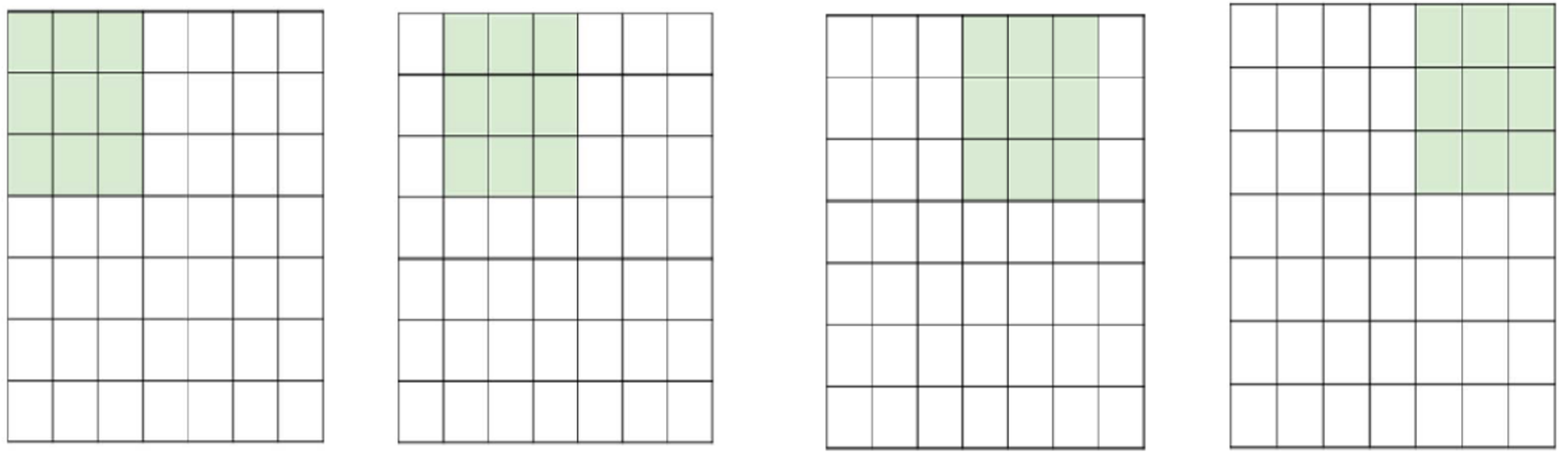
now: one unit will connect to, e.g. 5*5*3 chunk and only have 5*5*3 weights

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



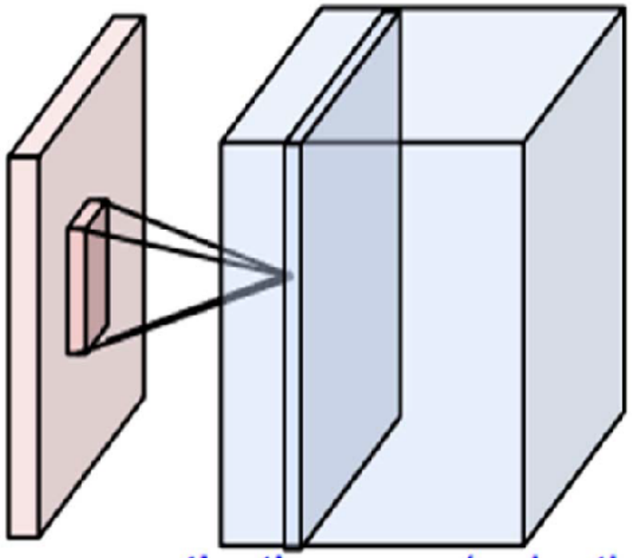
Stride,
Zero-padding

- 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

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

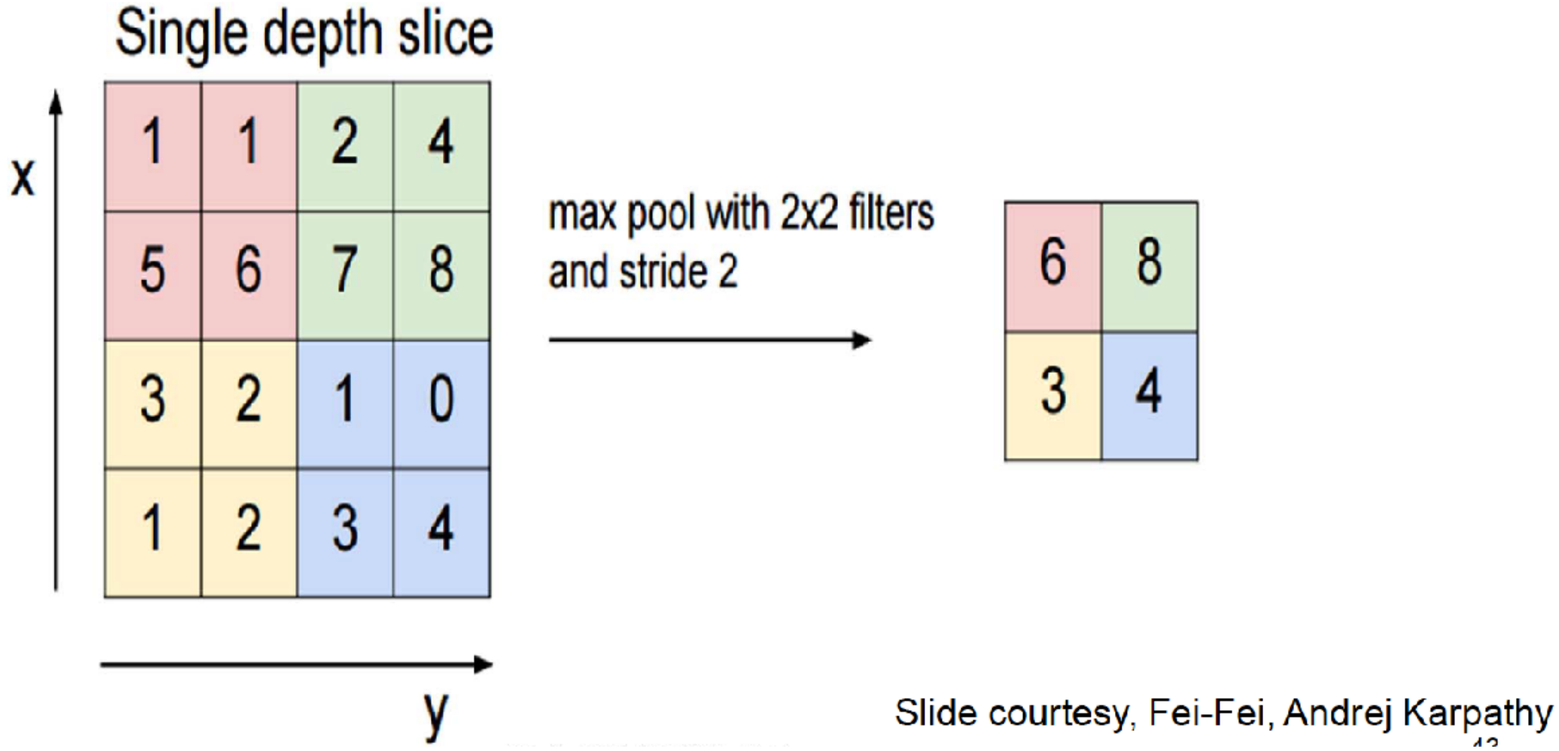
Sometimes we also add a bias term b, y = Wx + b, like what we have done for ordinary NN

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)



Slide courtesy, Fei-Fei, Andrej Karpathy

Chap 8 PAC

- PAC

Empirical Risk Minimization Paradigm

Optional subtitle

- Choose a *Hypothesis Class* H of subsets of X.
- For an input sample S, find some h in H that fits S "well".
- For a new point x, predict a label according to its membership in h.

$$\hat{h} = \arg\min_{h \in H} \hat{\epsilon}_S(h)$$

- Example:
  - Consider linear classification, and let $h_\theta(x) = 1\{\theta^T x \geq 0\}$

    Then $H = \{h_\theta \ : \ h_\theta(x) = 1\{\theta^T x \geq 0\}, \ \theta \in R^{n+1}\}$

$$\hat{\theta} = \arg\min_{\theta} \hat{\epsilon}_S(h_\theta)$$

  - We think of ERM as the most "basic" learning algorithm, and it will be this algorithm that we focus on in the remaining.
  - In our study of learning theory, it will be useful to abstract away from the specific parameterization of hypotheses and from issues such as whether we're using a linear classier or an ANN

The Case of Finite H

Optional subtitle

- H = $\{h_1, \ldots, h_k\}$ consisting of k hypotheses.
- We would like to give guarantees on the generalization error of $\hat{h}$.
- First, we will show that $\hat{\epsilon}(h)$ is a reliable estimate of $\varepsilon(h)$ for all h.
- Second, we will show that this implies an upper-bound on the generalization error of $\hat{h}$.

Misclassification Probability

Optional subtitle

- The outcome of a binary classifier can be viewed as a Bernoulli random variable Z: $Z = 1\{h_i(x) \neq c(x)\}$

- For each sample: $Z_j = 1\{h_i(x_j) \neq c(x_j)\}$

$$\hat{\epsilon}(h_i) = \frac{1}{m}\sum_{j=1}^{m} Z_j$$

- Hoeffding inequality

$$P(|\epsilon(h_i) - \hat{\epsilon}(h_i)| > \gamma) \leq 2\exp(-2\gamma^2 m)$$

  - This shows that, for our particular h_i, training error will be close to generalization error with high probability, assuming m is large.

Uniform Convergence

Optional subtitle

- But we don't just want to guarantee that $\hat{\epsilon}(h_i)$ will be close $\epsilon(h_i)$ (with high probability) for just only one particular h_i. We want to prove that this will be true for simultaneously for all $h_i \in H$
- For k hypothesis:

$$\begin{aligned} P(\exists h \in H,\ |\epsilon(h_i) - \hat{\epsilon}(h_i)| > \gamma) &= P(A_1 \cup \ldots \cup A_k) \\ &< \sum_{i=1}^{k} P(A_i) \\ &= \sum_{i=1}^{k} 2\exp(-2\gamma^2 m) \\ &= 2k\exp(-2\gamma^2 m) \end{aligned}$$

- This means:

$$\begin{aligned} P(\neg\exists h \in H,\ |\epsilon(h_i) - \hat{\epsilon}(h_i)| > \gamma) &= P(\forall h \in H,\ |\epsilon(h_i) - \hat{\epsilon}(h_i)| \leq \gamma) \\ &= 1 - 2k\exp(-2\gamma^2 m) \end{aligned}$$

- In the discussion above, what we did was, for particular values of m and γ, given a bound on the probability that:

  for some $h_i \in H$

$$|\epsilon(h_i) - \hat{\epsilon}(h_i)| > \gamma$$

- There are three quantities of interest here: m and γ, and probability of error; we can bound either one in terms of the other two.

Sample Complexity

Optional subtitle

- How many training examples we need in order make a guarantee?

$$P(\exists h \in H,\ |\epsilon(h) - \hat{\epsilon}(h)| > \gamma) \quad = \quad 2k \exp(-2\gamma^2 m)$$

- We find that if $m \geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta}$

  then with probability at least 1-δ, we have that $|\epsilon(h_i) - \hat{\epsilon}(h_i)| \leq \gamma$ for all $h_i \in H$

- The key property of the bound above is that the number of training examples needed to make this guarantee is only logarithmic in k, the number of hypotheses in H. This will be important later.

Generalization Error Bound

Optional subtitle

- Similarly, we can also hold m and δ fixed and solve for γ in the previous equation, and show [again, convince yourself that this is right!] that with probability 1- δ , we have that for all $h_i \in H$

$$|\hat{\epsilon}(h) - \epsilon(h)| \leq \sqrt{\frac{1}{m} \log \frac{2k}{\delta}}$$

- Define $h^* = \arg\min_{h \in H} \epsilon(h)$ to be the best possible hypothesis in H.

$$\begin{aligned} \epsilon(\hat{h}) &\leq \hat{\epsilon}(\hat{h}) + \gamma \\ &\leq \hat{\epsilon}(\hat{h}^*) + \gamma \\ &\leq \epsilon(\hat{h}^*) + 2\gamma \end{aligned}$$

  - If uniform convergence occurs, then the generalization error of $\hat{\epsilon}(h)$ is at most 2γ worse than the best possible hypothesis in H!

Summary

Optional subtitle

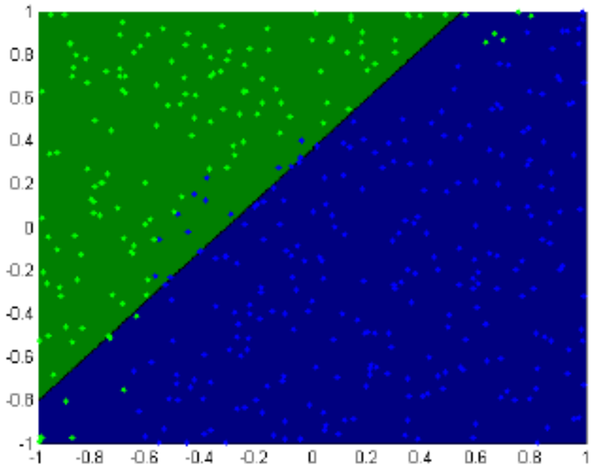
Theorem. Let $|\mathcal{H}| = k$, and let any m, δ be fixed. Then with probability at least $1 - \delta$, we have that

$$\varepsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \varepsilon(h)\right) + 2\sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

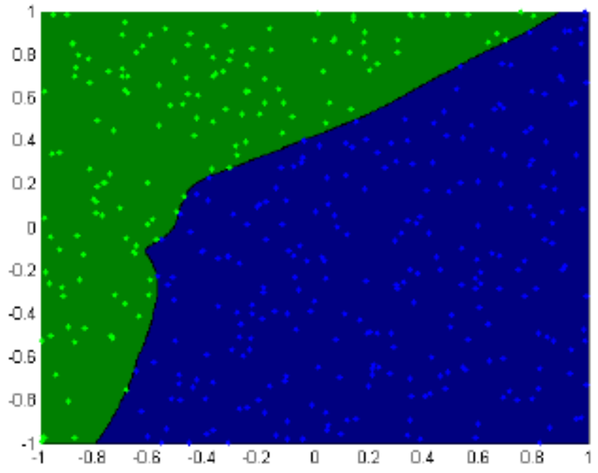
Corollary. Let $|\mathcal{H}| = k$, and let any δ, γ be fixed. Then for $\varepsilon(\hat{h}) \leq \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$ to hold with probability at least $1 - \delta$, it suffices that

$$\begin{aligned} m &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right), \end{aligned}$$

Questions?



(a)



(b)

- 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?

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.

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', 'pbdZMQ', '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)
```