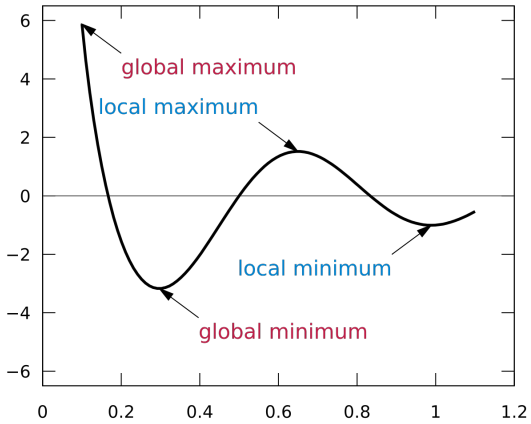
Topics in Quantitative Finance (FIN 528): Machine Learning for Finance

Jaehyuk Choi

2017-18 Module 3 (Spring 2018)

AlphaGo vs Humans (LEE Sedol, KE Jie)

We have to think again about joseki / dìngshì (定石/ 定式) .
In Go (围棋), a joseki is the studied sequences of moves for which the result is considered balanced for both black and white sides.



LEE Sedol afterwards · · ·

- Became more popular and richer
- Perhaps titled as the last human who beat the machine in Go

ML/AI: Rise of the machines?

Probably not like this!

What and why now?

What is ML?

- Prediction based on data (data into knowledge)
- Extended linear/logistic regression
- Pattern recognition

Why now?

- Abundant Data (Big Data)
- Faster computer (Graphics Processing Unit: GPU)
- Advances in research: Geoffrey Hinton (Google), Yann LeCun (Facebook).

Recent applications of ML

- Automated driving system (Google, Apple, etc)
- Suggestion engine (Amazon, Taobao)
- Cancer diagnosis (IBM Watson)
- Digitizing images (Facebook, Google)
- Shopping without checkout (Amazon Go: big data)
- ...

ML in finance?

Prediction

- Asset management / investment
- Trading algorithm (alpha)
- Earnings prediction: e.g., Prediction Valley

Cost cut / labor reduction

- Automated accounting / tax
- Automated analyst report
- Chat-bot (trading and sales)
- Data analytics: e.g., Kensho

Softwares to use

Python

- Anaconda (Python distribution + Environment management)
- Python Language Tutorial
- Sci-Kit Learn
- TensorFlow (wrapped by Keras)

Github.com

- Distributed version control system
- Clone a repository to create a local copy
- `https://github.com/PHBS/2017.M3.TQF-ML` (our course)
- `https://github.com/rasbt/python-machine-learning-book-2nd-edition` (**PML**)

Other resources for ML

- Coursera ML course (**CML**) by Andrew Ng (Baidu)
- Stanford CS229 Machine Learning: course notes, student projects, etc
- The Elements of Statistical Learning (**ESL**)
- An Introduction to Statistical Learning (with R) (**ISLR**)
- Pattern Recognition and Machine Learning by Bishop (Microsoft)

Notations and conventions: vector and matrix

General rules (guess from the context)

- Scalar (non-bold): x, y, X, Y
- Vector (lowercase bold): $\boldsymbol{x} = (x_i), \boldsymbol{y} = (y_i)$
- Matrix (uppercase bold): $\boldsymbol{X} = (X_{ij}), \boldsymbol{Y} = (Y_{ij})$
- The (i, j) component of $\boldsymbol{X}$: X_{ij}
- The i-th row vector of $\boldsymbol{X}$: $\boldsymbol{X}_{i*} = (X_{i1}, X_{i2}, \cdots, X_{ip})^T$
- The j-th column vector of $\boldsymbol{X}$: $\boldsymbol{X}_{*j} = (X_{1j}, X_{2j}, \cdots, X_{Nj})$

Examples

- Dot product: $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y}$
- Vector norm: $|\boldsymbol{x}| = \sqrt{\boldsymbol{x}^T \boldsymbol{x}}$
- Matrix multiplication: $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{Y} \rightarrow Z_{ij} = \boldsymbol{X}_{i*}\boldsymbol{Y}_{*j}$

Notation and conventions: variables and observations

General rules

- Generic (or representative) variables (uppercase non-bold): X (input), Y (output), G (classification output)
- The predictions: $\hat{Y}$, $\hat{G}$
- X (input) may be p-dimensional (features/predictors): X_j $(j \leq p)$, row vector
- Y (output) may be K-dimensional (responses): Y_k $(k \leq K)$, row vector.
- The N observations of X or Y are stacked over as rows: $\boldsymbol{X}$ ($N \times p$ matrix), $\boldsymbol{Y}$ ($N \times K$ matrix)
- The i-th observation of the j-th feature: X_{ij} v.s. $X_j^{(i)}$ in **PML**
- The i-th observation set: $\boldsymbol{X}_{i*}$ ($1 \times p$ row vector) v.s. $X^{(i)}$ in **PML**
- All observation of j-th feature X_j: $\boldsymbol{X}_{*j}$ ($N \times 1$ row vector) v.s. X_j in **PML**
- $\boldsymbol{X} = (\boldsymbol{X}_{*1} \cdots \boldsymbol{X}_{*p})$ (column-wise concatenation)
- The weight vector, $\boldsymbol{\beta}$ or $\boldsymbol{w}$, are column vectors used interchangeably.

Simple Linear Regression (Ordinary Least Square)

For scalar predictor (X) and response (Y),

$$Y \approx \beta_0 + \beta_1 X \quad \longrightarrow \quad \hat{\boldsymbol{y}} = \beta_0 + \beta_1 \boldsymbol{x}.$$

For N observations $(x_1, y_1), \cdots, (x_N, y_N)$, the set of $(\hat{\beta}_0, \hat{\beta}_1)$ to minimize the residual sum of squares (RSS):

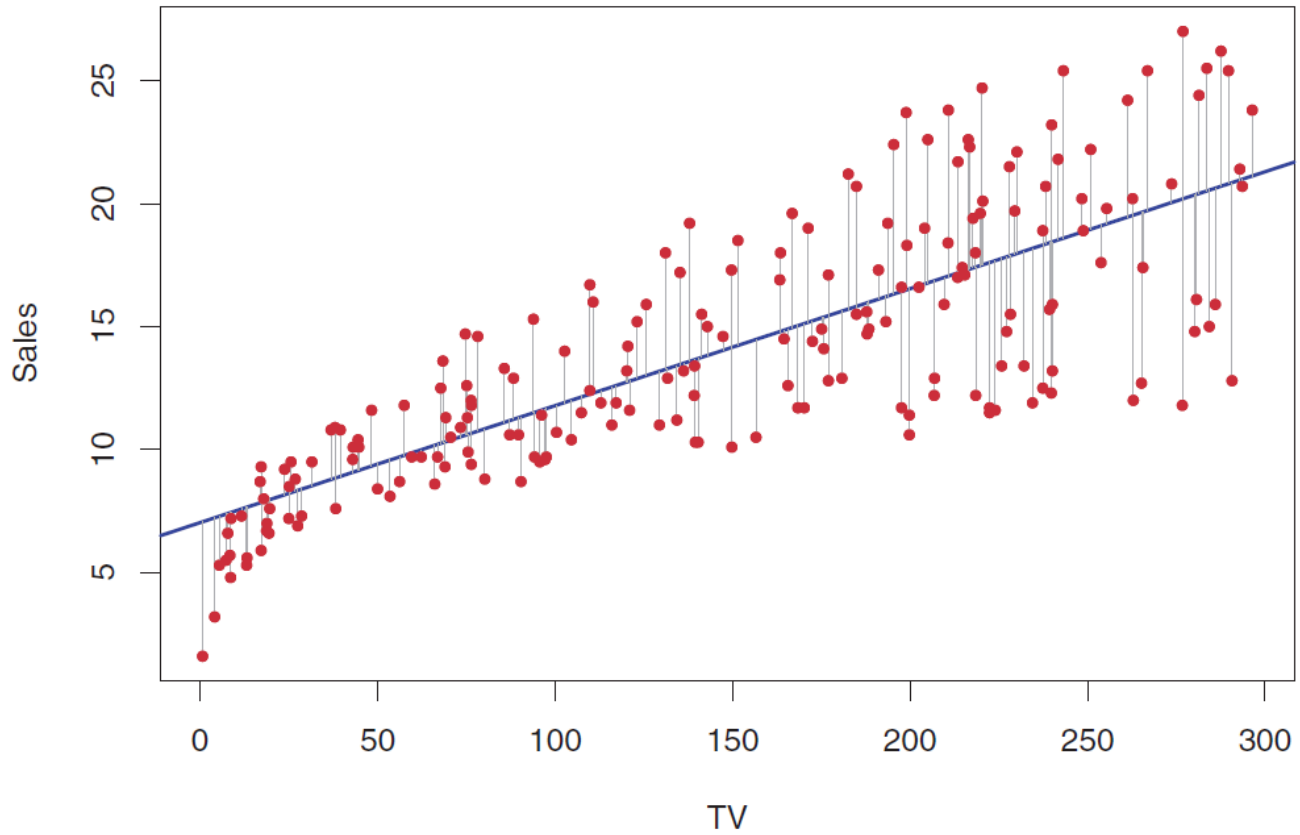
$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2 = (\boldsymbol{y} - \beta_0 - \beta_1 \boldsymbol{x})^T (\boldsymbol{y} - \beta_0 - \beta_1 \boldsymbol{x})$$

is given as

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{\mathrm{Cov}(X, Y)}{\mathrm{Var}(X)},$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

for $\bar{x} = \sum x_i / N$ and $\bar{y} = \sum y_i / N$.

Figure 3.1 (p62) from **ISLR**



Multi-dimensional Linear Regression

For $(p+1)$-vector predictor (X) and scalar response (Y),

$$Y \approx X\boldsymbol{\beta} \quad \longrightarrow \quad \hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta},$$

where $X_0 = 1$ $(\boldsymbol{X}_{*0} = \mathbf{1})$ and $\boldsymbol{\beta}$ is a $(p+1)$-column vector.

$$\text{RSS}(\boldsymbol{\beta}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

$$\frac{\partial}{\partial \boldsymbol{\beta}}\text{RSS}(\boldsymbol{\beta}) = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \quad \Rightarrow \quad \hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

For $(p+1)$-vector predictor (X) and K-vector response (Y), the result is similarly given as

$$\hat{\boldsymbol{Y}} = \boldsymbol{X}\boldsymbol{B} \quad \text{where} \quad \hat{\boldsymbol{B}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y},$$

which is the independent regressions on Y_j $(\boldsymbol{Y}_{*j})$ combined together,

$$\hat{\boldsymbol{B}}_{*j} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}_{*j}$$

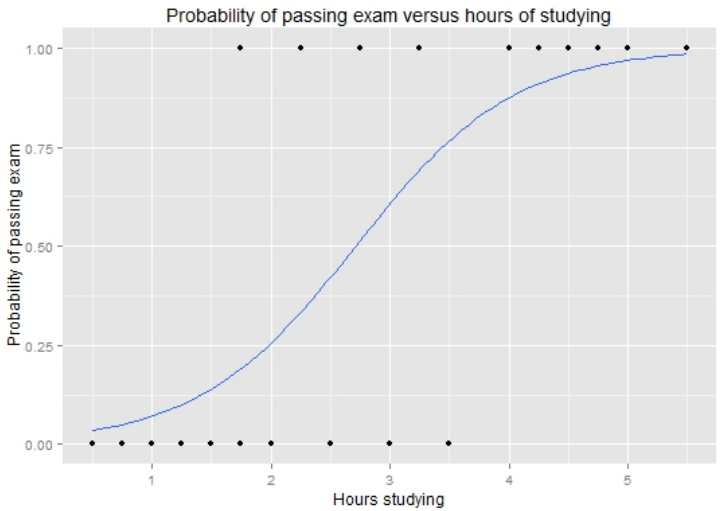
Logistic Regression (Classification)

- Qualitative (categorical) response (binary dependent variable, $Y \in \{0, 1\}$)
- Multiple categories: how to give order?
- Linear regression (quantitative) is not proper
- Logistic (sigmoid) function: $\sigma(\text{logit}) = \text{quantile}$

$$p = \phi(t) = \frac{e^t}{1 + e^t} = \frac{1}{1 + e^{-t}} \quad \text{for} \quad t = Xw \ (X_0 = 1)$$

- Logit function (the inverse): log odds

$$\phi^{-1}(p) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p)$$



Fitting of logistic regression

Likelihood function

- For a given the prediction model, measures the likelihood of a data set.
- The best prediction model/weight is the one that maximizes the likelihood of the dataset.

For a data set $(\boldsymbol{X}, \boldsymbol{y})$ where $y_i \in \{0, 1\}$),

$$
\begin{aligned}
L(\boldsymbol{w}) &= \prod_i P(y_i = \hat{y}_i) = \prod_{i:y_i=1} \phi(\boldsymbol{X}_{i*}\boldsymbol{w}) \prod_{i:y_i=0} \left(1 - \phi(\boldsymbol{X}_{i*}\boldsymbol{w})\right) \\
&= \prod_i \phi(\boldsymbol{X}_{i*}\boldsymbol{w})^{y_i} \left(1 - \phi(\boldsymbol{X}_{i*}\boldsymbol{w})\right)^{1-y_i} \\
\log L(\boldsymbol{w}) &= \sum_i y_i \log \phi(\boldsymbol{X}_{i*}\boldsymbol{w}) + (1 - y_i) \log \left(1 - \phi(\boldsymbol{X}_{i*}\boldsymbol{w})\right)
\end{aligned}
$$

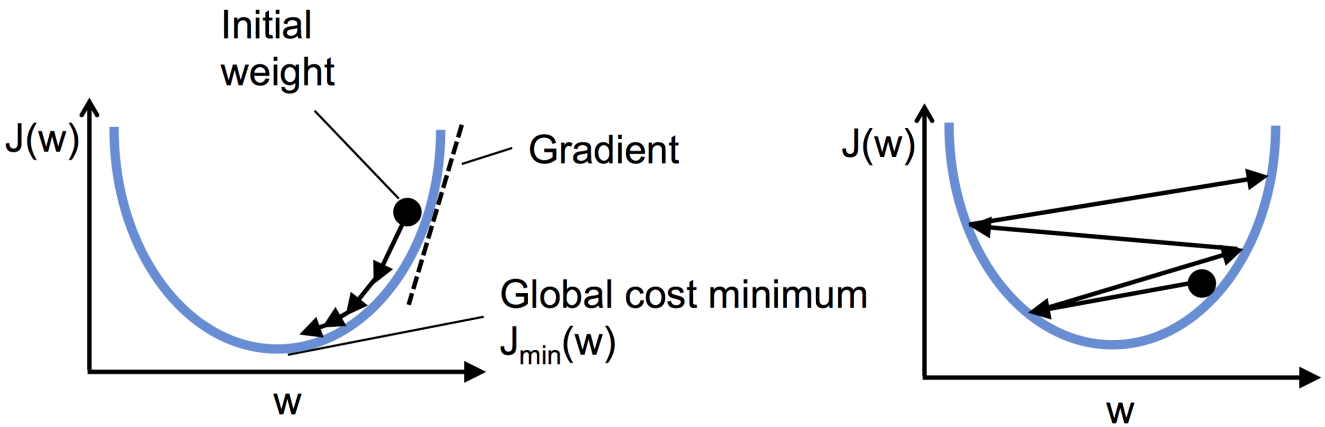
The cost function (to minimize) is $J(\boldsymbol{w}) = -\log L(\boldsymbol{w})$

Gradient Descent (Newton's method)

A first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point
The minumum location $J(\boldsymbol{x})$ can be found by the following iteration:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \eta \boldsymbol{\nabla} J(\boldsymbol{x}_n)$$

The constant η is called *learning rate*. Typically we use $0 < \eta < 1$ to avoid *overshooting*.



Gradient Descent of Weight: A Simple Case

How can we search for the right weight $\boldsymbol{w}$ to minize the error (or cost)?
Imagine we fit a simple linear model $y = xw$ to a single observations (x_1, y_1). We need find w to minimize the RSS:

$$J(w) = \frac{1}{2}(y_1 - x_1 w)^2$$

Although we know the answer ($w = y_1/x_1$), let's pretend that we need to improve w iteratively, such that

$$w := w + \Delta w \quad \text{with an initial guess} \quad w^{(0)}.$$

The amount of update should be proportional to the derivative

$$\Delta w = -\eta \frac{d}{dw} J(w) = \eta(y_1 - x_1 w)\, x_1 = \eta(y_1 - \hat{y}_1)\, x_1.$$

The result is intuitive: the update amount is proportional to
(i) the *error*, $(y_1 - x_1 w)$ and (ii) the x_1 value (at least the sign).
You will see this equation over and over!

Gradient Descent of Weight: Linear Regression (Adaline)

Remind that the error (RSS) function and the gradient is given as

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_i (y_i - \boldsymbol{X}_{i*}\boldsymbol{w})^2,$$

$$\frac{\partial}{\partial w_j} J(\boldsymbol{w}) = -\sum_i (y_i - \boldsymbol{X}_{i*}\boldsymbol{w}) X_{ij}.$$

The weight update rule is given as (η is *learning rate*)

$$\boldsymbol{w} := \boldsymbol{w} + \Delta\boldsymbol{w}$$

$$\Delta w_j = -\eta \frac{\partial}{\partial w_j} J(\boldsymbol{w}) = \eta \sum_i (y_i - \boldsymbol{X}_{i*}\boldsymbol{w}) X_{ij} = \eta \sum_i (y_i - \hat{y}_i) X_{ij}.$$

Note that one observation $(y_i, \boldsymbol{X}_{i*})$ contribute $(y_i - \hat{y}_i)X_{ij}$. In **batch gradient descent**, $\boldsymbol{w}$ is updated from all i's. Wehereas, in **stochastic gradient descent** (iterative/online), $\boldsymbol{w}$ is updated from randomly selected single i.

Gradient Descent of Weight: Logistic Regression

We use the properties of logistic function,

$$\frac{d}{dt}\phi(t) = \frac{e^{-t}}{1+e^{-t}} = \phi(t)(1-\phi(t)), \quad \frac{\partial}{\partial w_j}\phi(\boldsymbol{X}_{i*}\boldsymbol{w}) = \phi(\cdot)(1-\phi(\cdot))X_{ij},$$

the gradient of error function is obtained as

$$\begin{aligned}
\frac{\partial}{\partial w_j}J(\boldsymbol{w}) &= \sum_i \left(-\frac{y_i}{\phi(\boldsymbol{X}_{i*}\boldsymbol{w})} + \frac{1-y_i}{1-\phi(\boldsymbol{X}_{i*}\boldsymbol{w})} \right) \frac{\partial}{\partial w_j}\phi(\boldsymbol{X}_{i*}\boldsymbol{w}) \\
&= \sum_i \Big(-y_i(1-\phi(\cdot)) + (1-y_i)\phi(\cdot) \Big) X_{ij} \\
&= -\sum_i (y_i - \phi(\cdot))X_{ij} = -\sum_i (y_i - \hat{y}_i)X_{ij}, \\
\Delta w_j &= \eta \sum_i (y_i - \hat{y}_i)X_{ij}.
\end{aligned}$$

We get the exactly same weight updating rule as that of linear regression and Adaline! Perceptron's updating rule is also based on this result.

Regularization

Regularization

We avoid $\boldsymbol{w}$ being too big.

$$J(\boldsymbol{w}) = -\log L(\boldsymbol{w}) + \frac{\lambda}{2}|\boldsymbol{w}|^2$$

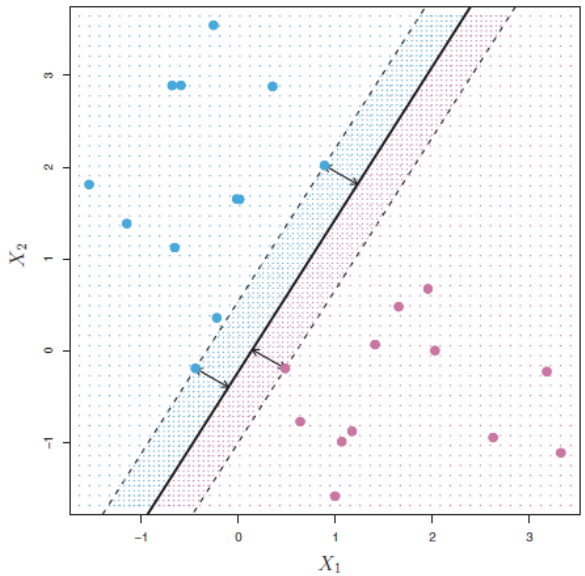
$$J(\boldsymbol{w}) = -C \log L(\boldsymbol{w}) + \frac{1}{2}|\boldsymbol{w}|^2 \quad (C = \frac{1}{\lambda}, \text{SciKit-Learn})$$

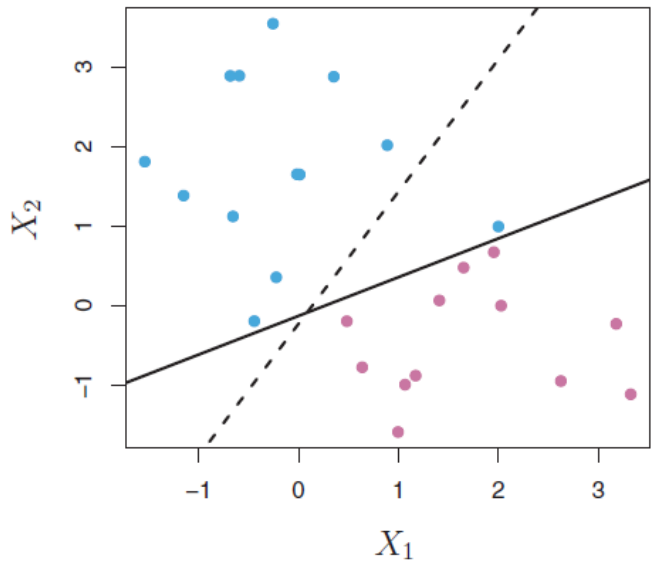
Maximal margin classifier

For $y_i \in \{-1, 1\}$, maximize the margin of the separating hyperplane M,

$$y_i(w_0 + \sum_{j=1}^{p} X_{ij} w_j) = y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq M > 0 \text{ for all } i, \text{ with } |\boldsymbol{w}| = 1$$

Maximal margin classifier only works for the separable data set and is sensitive to the change in the *support vectors*.





Support vector classifier

We make maximal margin classifier flexible: maximize the margin of the separating hyperplane M with $|\boldsymbol{w}| = 1$,

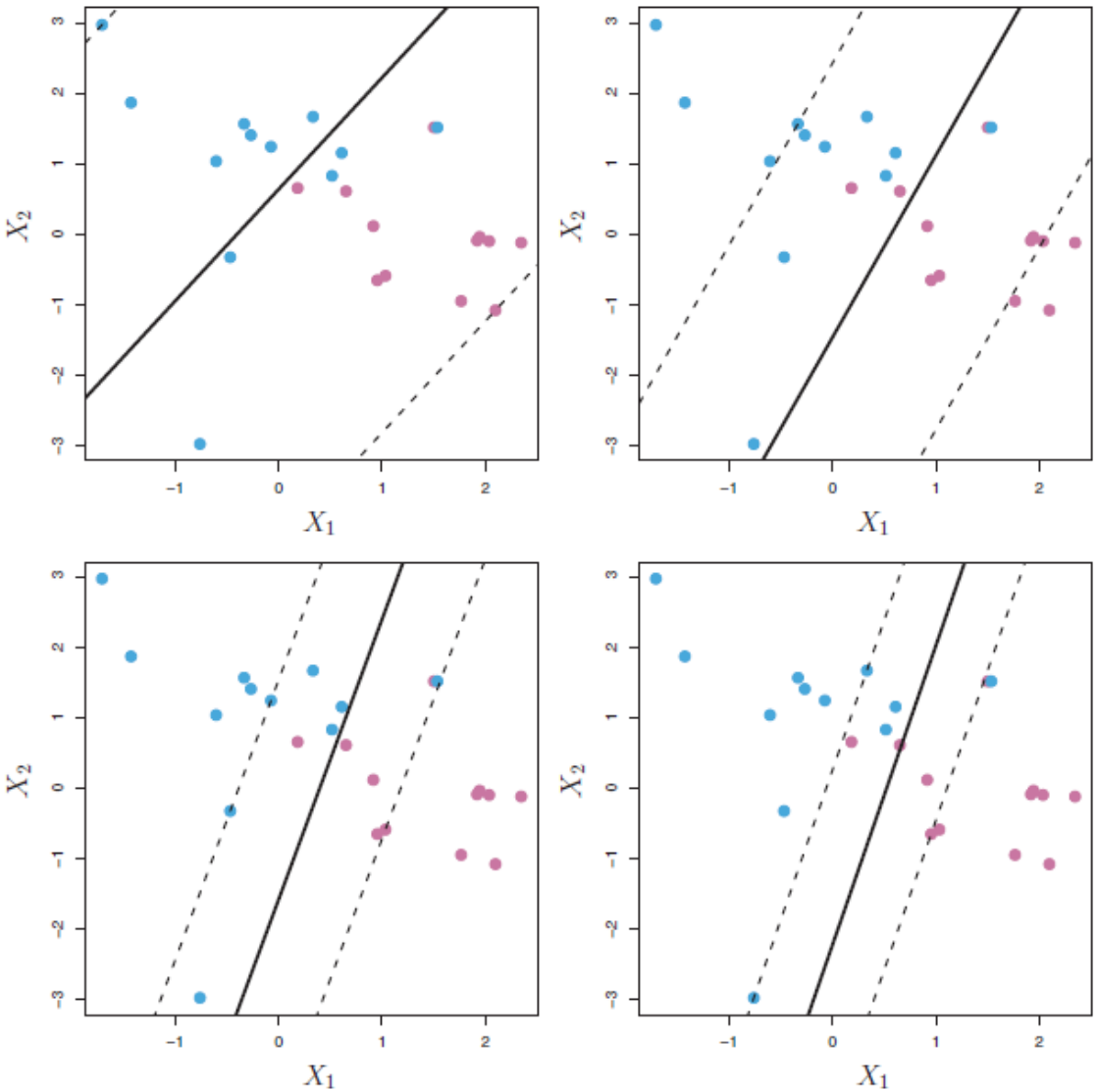
$$y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq M(1 - \epsilon_i) \text{ for all } i, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

where $\epsilon_i \geq 0$ is *slack variable* indicating the degree of violation ($\varepsilon_i = 0$: no violation, $\varepsilon_i < 1$: margin violation, $\varepsilon_i > 1$: classification violation) and C is a *budget* for the amount of violations by all observations.
Alternatively (in PML), we minimize

$$\frac{1}{2}|\boldsymbol{w}|^2 + C' \sum_{i=1}^{n} \xi_i,$$

where $\{\xi_i \geq 0\}$ satisfies $y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq 1 - \xi_i$ for all i. The model converges to maximal margin classifier if C' is very large, so the role of C' is opposite to that of C in the original formulation.

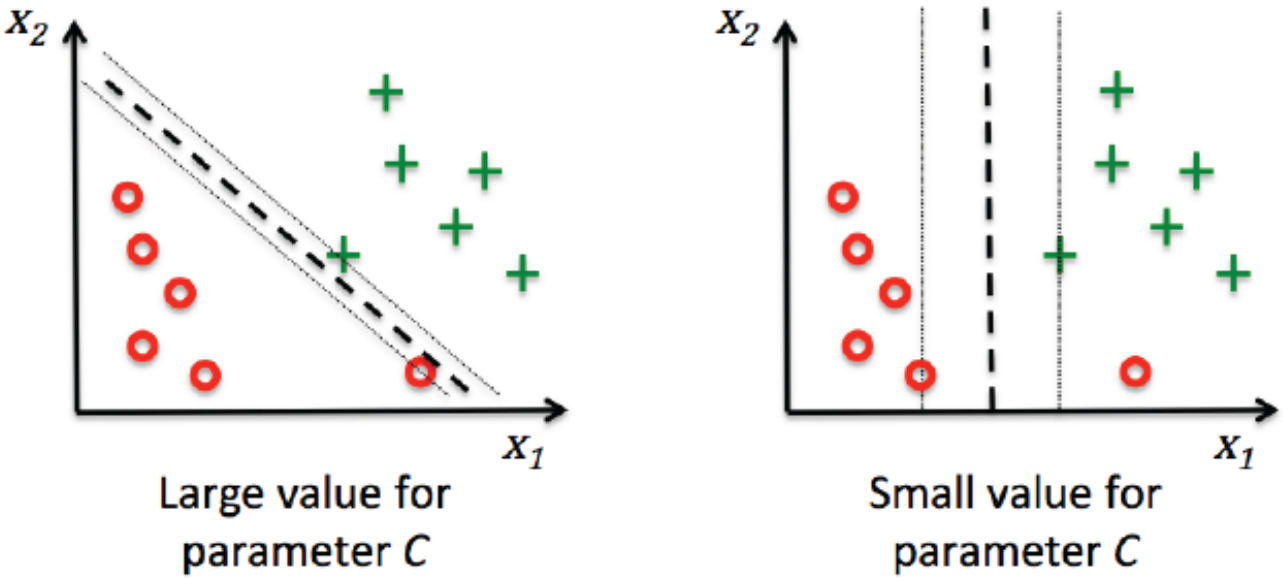
Support vector classifier: role of C



The value of C decreasing from the largest value on top left.

Support vector classifier: role of C'

Large C' (left) vs small C' (right)



Support vector machines (SVM)

How can we deal with non-linear decision boundary?

Enlarging feature space

Including high-order terms, $1, X_j, \cdots, X_j^2, \cdots, X_i X_j, \cdots$, can be helpful, but the computation becomes very heavy.

Kernel

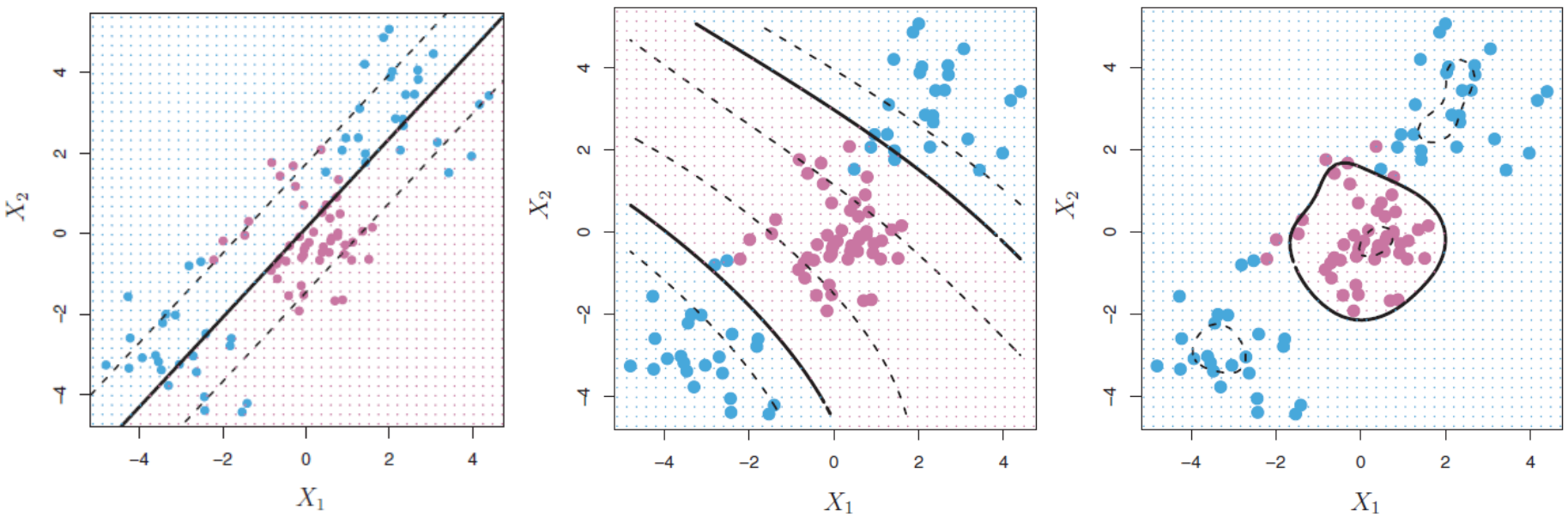
Instead we introduce kernel function, as a generalization of dot product in hyperplane:

- Linear: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{i'*}) = \boldsymbol{X}_{i*}\, \boldsymbol{X}_{i'*}^T$
- Polynomial: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{i'*}) = (1 + \boldsymbol{X}_{i*}\, \boldsymbol{X}_{i'*}^T)^d$
- Radial basis: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{i'*}) = \exp(-\gamma |\boldsymbol{X}_{i*} - \boldsymbol{X}_{i'*}|^2)$

Kernel $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{i'*})$ can be understood as a *distance* between two observations: $\boldsymbol{X}_{i*}$ and $\boldsymbol{X}_{i'*}$ are similar if the kernel value is high (low) whereas they are different if low (high).

SVM: non-linear decision boundary

SVM classification with linear kernel (left) polynomial kernel of degree 3 (middle) and radial basis kernel (right)



K Nearest Neighbor (KNN)

If $N_K(x)$ is the set of the K nearest neighbors around x,

$$\text{Regression:} \hat{y} = f(x) = \frac{1}{K} \sum_{x_i \in N_K(x)} y_i$$

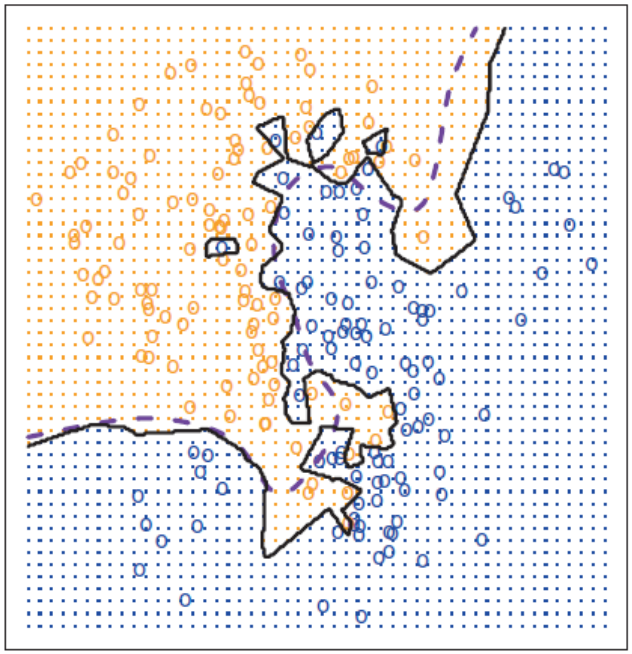
$$\text{Classifier: } \hat{y} = \text{majority of } \{y_i\} \text{ for } x_i \in N_K(x)$$

$$\text{Prob}(y = j|x) = \frac{1}{K} \sum_{x_i \in N_K(x)} I(y_i = j)$$

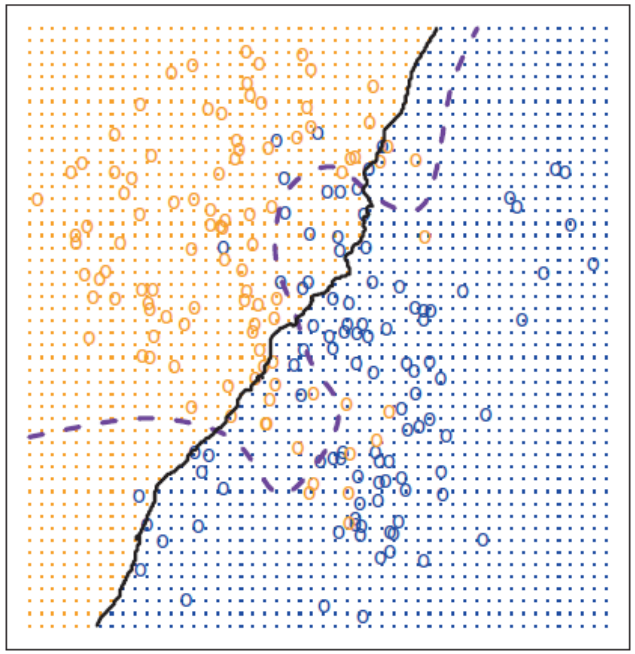
- Parametric vs Non-parametric model
- Learning step is not required, but KNN is intensive in both computation and storage (*memorize* training data set for prediction)

KNN: Classfier example

KNN: K=1

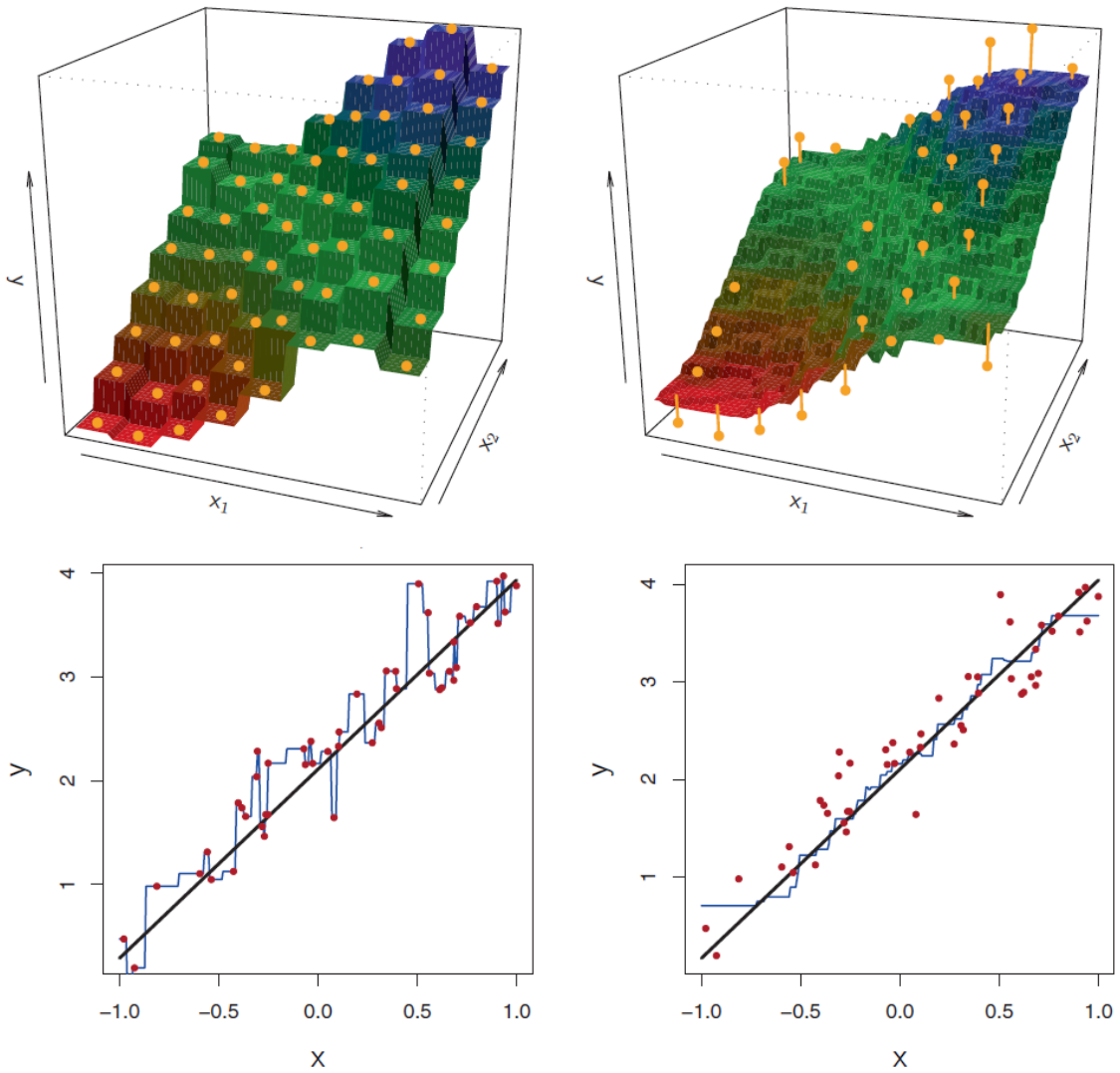


KNN: K=100

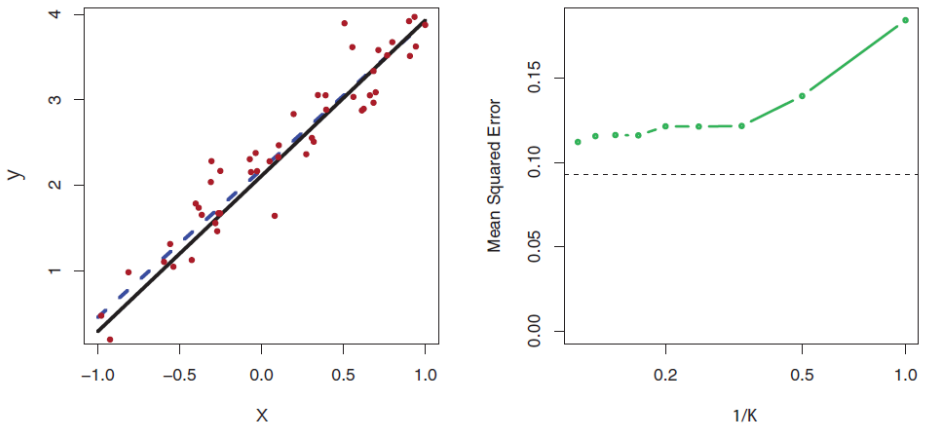


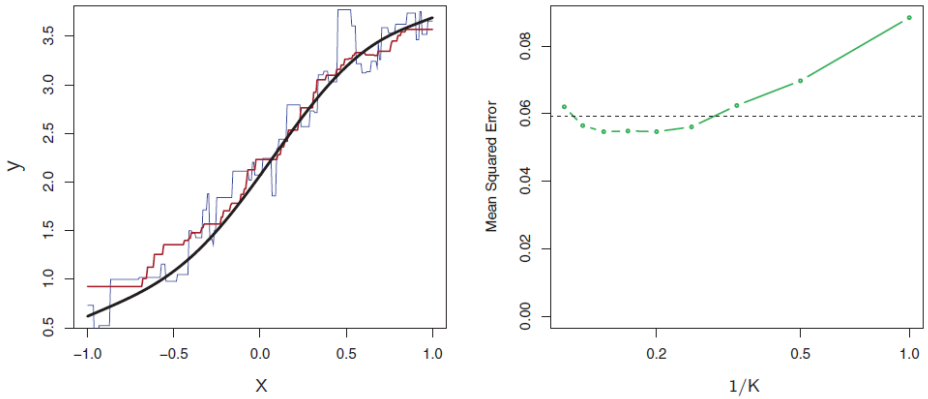
KNN: Regression example

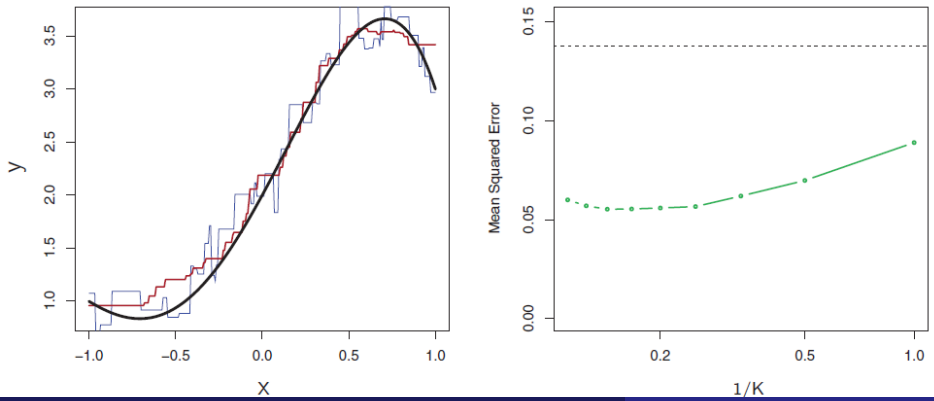
$K = 1$ (left) vs $K = 9$ (right)



KNN: Parametric vs Non-parametric



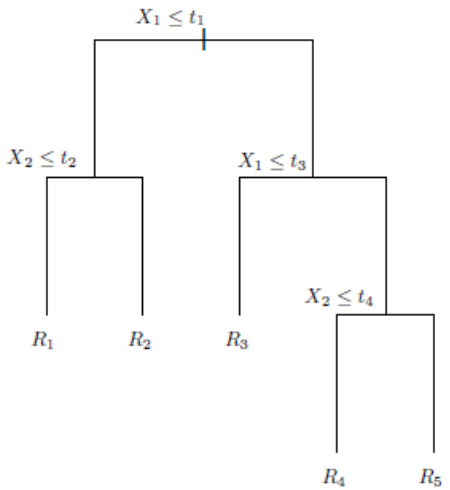


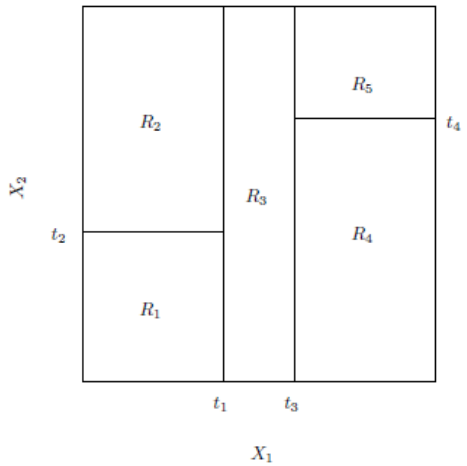


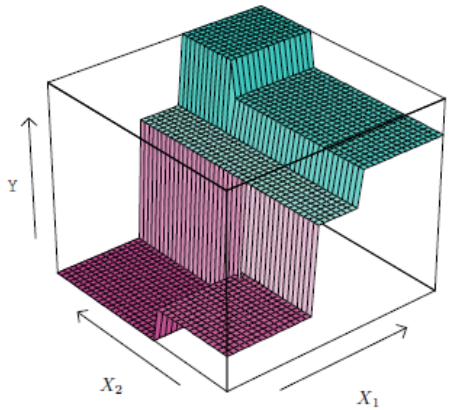
Non-parametric regression works better as the true function deviates from the basis function (linear function in this example).

Decision Tree

- Regression/classification is made on a series of conditions on input variables
- Final conclusion on each *terminal node* or *leaf* of the (upside down) tree.
- The predictor space is broken down to boxes
- Regression: the average value is assigned to each box
- Classification: the majority class is assigned to each box







Growing Tree

We want to minimize the error in each leaf. For a given leaf of tree, t,
Regression Error:

$$RSS(t) = \sum_{i \in t} (y_i - \hat{y}_t)^2$$

Classification Error (measure of impurity):

- Gini index: $I_G(t) = \sum_{k=1}^{K} p(k|t)\big(\boxed{1} - p(k|t)\big) = 1 - \sum_{k=1}^{K} p^2(k|t)$
- (Cross-)Entropy: $I_H(t) = -\sum_{i=1}^{K} p(k|t) \log_2 p(k|t)$
- Classification Error: $I_E(t) = 1 - \max_{1 \le k \le K} p(k|t)$
- $I_G(t) = I_H(t) = I_E(t) = 0$ if the composition is pure, i.e., $p(k|t) = 1$ for some k. Otherwise > 0.
- I_E is less sensitive for branching options, so not recommended for growing tree.

Ideally, we can grow tree such that each leaf contains one sample point, but it is over-fitting. Thus we need to regularize the number of leaves or the maximum level of branching.

Growing Tree

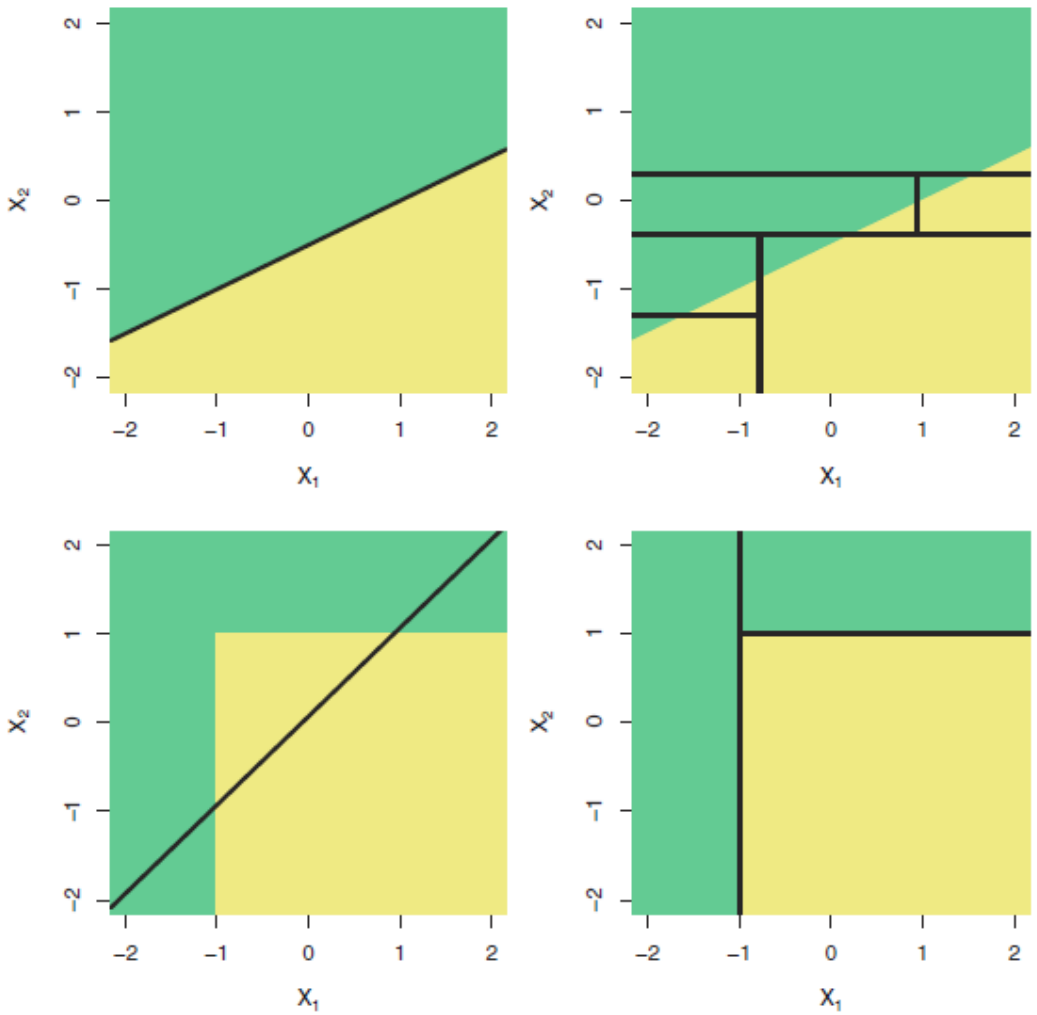
We decide the binary split condition (and feature) in such a way that maximize the information gain (or increase purity):

$$\mathsf{IG}(D_P) = I(D_P) - \frac{N_L}{N_p} I(D_L) - \frac{N_R}{N_P} I(D_R)$$

where D_P, D_L and D_R are the parent, left and right data set and N is the number of the samples in the corresponding sets.

$$I(D) = \sum_{t \in D} I(t) \quad \text{where} \quad I = I_E, I_H \text{ or } I_G$$

Tree vs Linear model



Two classification problems (top vs bottom) approached by linear model (left) and decision tree (right)
Linear model outperforms decision tree on the top problem, whereas

Decision Tree

Pros

- Intuitive and easy to explain. (Even easier than linear regression)
- Closely mirror human decision making
- Can be displayed graphically and easily interpreted by non-experts

Cons

- Prediction is less accurate than other ML methods
- Model variance is high-variant (sensitive to input samples)
  $\longrightarrow$ Bagging, Random Forests, Boosting

Bagging, Random Forest, etc (Ensemble Learning, Ch 7)

Build an ensemble of different classifiers/regressors: the result is interpreted as majority vote/average.

$$\hat{f}_E(X) = \text{Avg}_{e \in E}\{\hat{f}_e(X)\} \quad \text{or} \quad \hat{f}_E(X) = \text{Majority}_{e \in E}\{\hat{f}_e(X)\}$$

Bagging

- Build different trees out of subsets (*bags*) of training set.
- Increase prediction accuracy and reduce model variance

Random Forest (RF)

- Build different trees out of subsets of training set.
- At each split, randomly select m ($\approx \sqrt{p}$) out of p features.
- By random selection of features, RF reduces the correlation between the trained trees.

Both bagging and RF can be applied to any ML methods.

Data Pre-processing (Ch 4)

Handling missing data

- Remove the sample
- Fill in `NaN` with the mean of the average of the feature

Normalization

$$X'_{ij} = \frac{X_{ij} - \min(\boldsymbol{X}_{*j})}{\max(\boldsymbol{X}_{*j}) - \min(\boldsymbol{X}_{*j})}$$

Standardization

$$X'_{ij} = \frac{X_{ij} - E(\boldsymbol{X}_{*j})}{\sigma(\boldsymbol{X}_{*j})}$$

Regularization L-1 vs L-2

Give a penalty for complexity or over-fitting. The cost function to minimize:

$$J(\boldsymbol{w}) = J_0(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) \quad (= C\, J_0(\boldsymbol{w}) + R(\boldsymbol{w})),$$

where $J_0(\boldsymbol{w})$ is the un-regularized cost function, e.g., log-maximum likelihood (logistic) or RSS/SEE (linear).

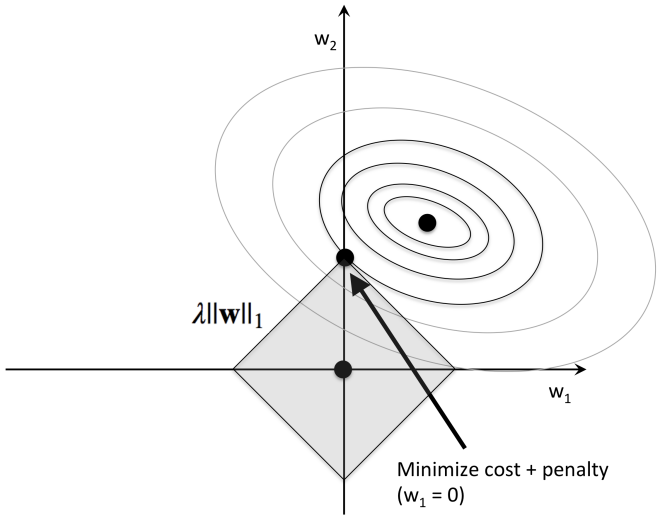
L-2 Regularization

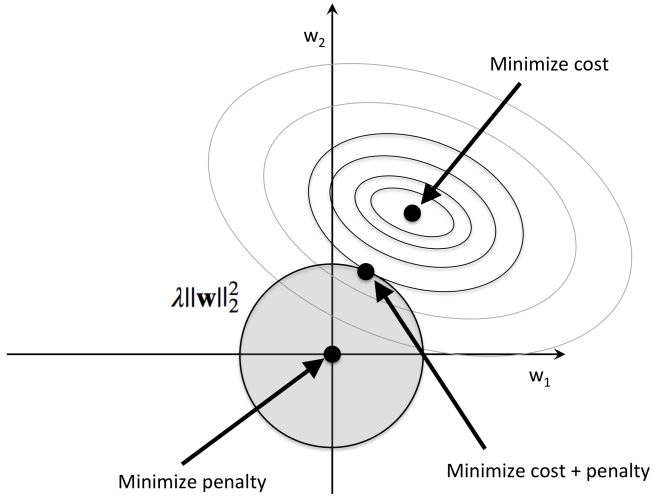
- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2 = \sum_j w_j^2$
- N-sphere boundary (e.g., circle or sphere). Easy to solve the minimum.

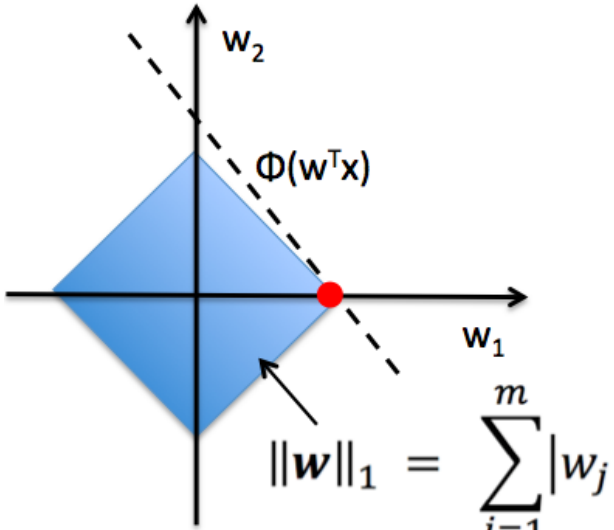
L-1 Regularization

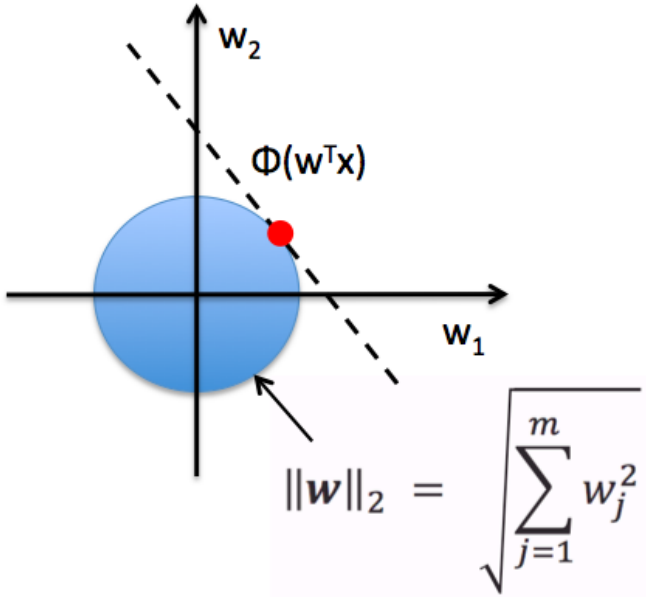
- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_1 = \sum_j |w_j|$
- 'Diamond' boundary: leads to sparse vector (many zero components)
- Effectively works as feature selection

Regularization L-1 vs L-2





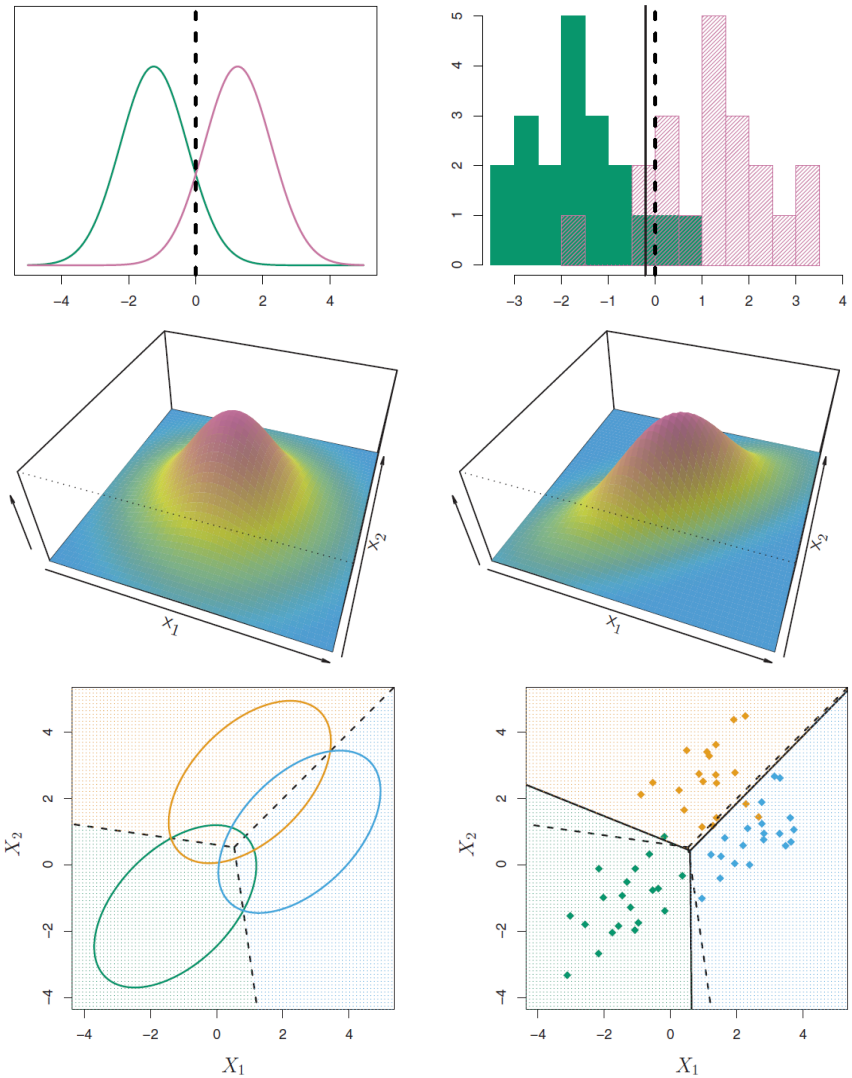




Feature selection

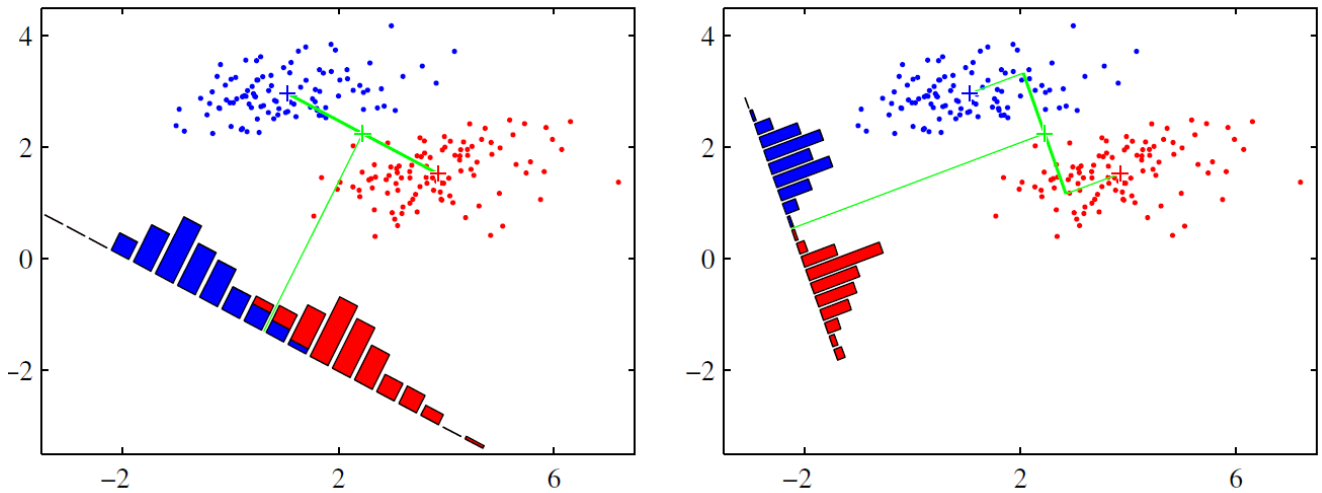
Linear Discriminant Analysis (LDA) as a classifier

- Assume the samples in each class follow normal (Gaussian) distribution.
- Estimate mean $\hat{\boldsymbol{\mu}}_k$ and variance $\hat{\boldsymbol{\Sigma}}_k$ of class k:
- Obtain multi-variate normal PDF: $f_k(\boldsymbol{x}) = n(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$
- LDA if $\boldsymbol{\Sigma}_W = \sum_{k=1}^{K} \boldsymbol{\Sigma}_k$ (within covariance)is used for all $\boldsymbol{\Sigma}_k$.
- QDA if $\boldsymbol{\Sigma}_k$ is estimated for each class k
- A test sample $\boldsymbol{x}$ is classified to the class k for which $f_k(\boldsymbol{x})$ is largest.



LDA as a dimensionality reduction

- Given the LDA assumptions, which direction $\boldsymbol{w}$ best separates the feature?
- $\boldsymbol{w} \approx \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1$? Probably not the best
- Want to minimize $J(\boldsymbol{w}) = \frac{(\mu_2 - \mu_1)^2}{\sigma_1^2 + \sigma_2^2}$, where $(\mu_1,\, \sigma_1^2)$ and $(\mu_2,\, \sigma_2^2)$ are mean variance of the samples (1-D) projected on $\boldsymbol{w}$.
- The best directions $\boldsymbol{w}$ are the first eigenvectors of $\Sigma_W^{-1}\Sigma_B$, where Σ_W and Σ_B are within and between covariance matrices.
- The transformation $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}$ is the extracted factors with the best separability, which can be used for other ML methods.



Bias-Variance Trade-Off

Given a ML method, we want to minimize the mean squared error (MSE) on **test data set** (test error rate).

$$E(y - \hat{f}(x))^2 = \text{Bias}(\hat{f}(x))^2 + \text{Var}(\hat{f}(x)) + Var(\varepsilon)$$

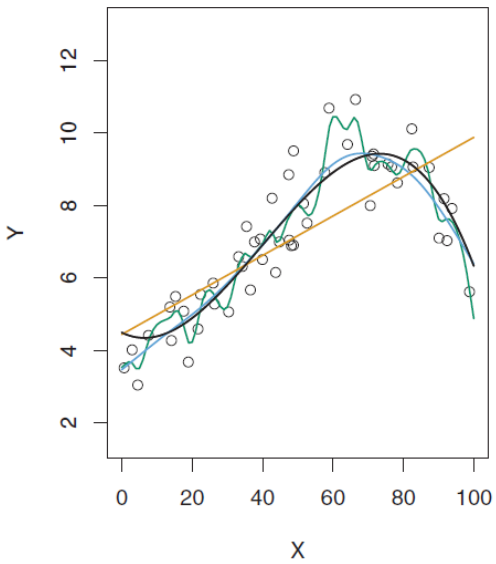
Bias

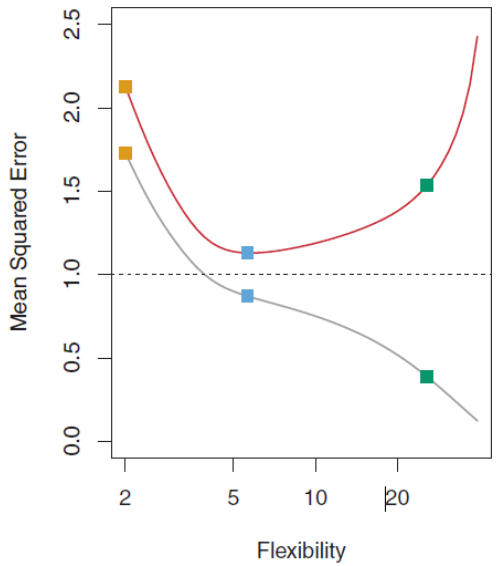
- Error from $\hat{f}$ not correctly representing the true f.
- A model has **high bias** when $\hat{f}$ overly simply f (e.g. linear regression on non-linear data), the parameters are too few.

Variance

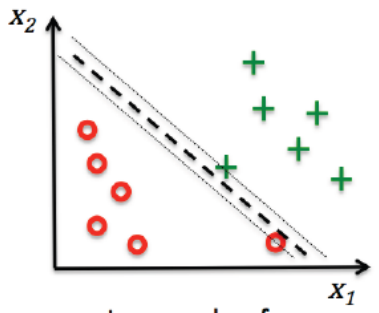
- Error from variability or sensitivity (vs consistency) of the trained model $\hat{f}$ against the selection of training dataset.
- A model has **high variance** when there are too many parameters (over-fitting), e.g. KNN with $K = 1$, high-order polynomial regression, SVM with large C, logistic regression with large C (or small λ), decision tree with many leaves, etc.

Bias-Variance Trade-Off

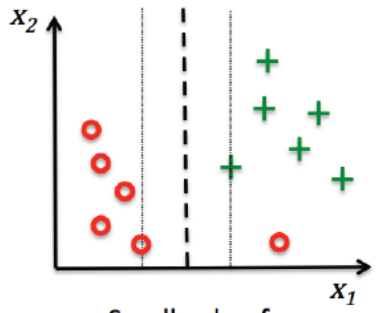




- Grey line: Bias vs the number of parameters
- Red line: MSE measured with the true f (black line).

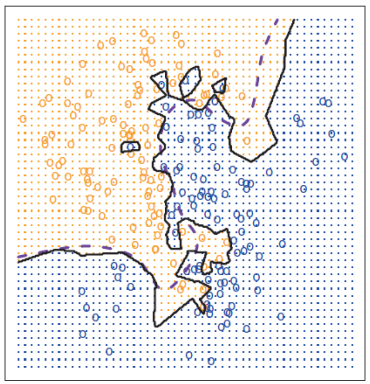


Large value for parameter C

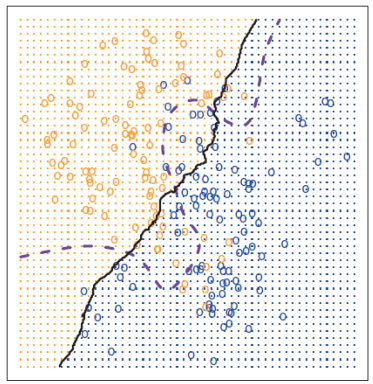


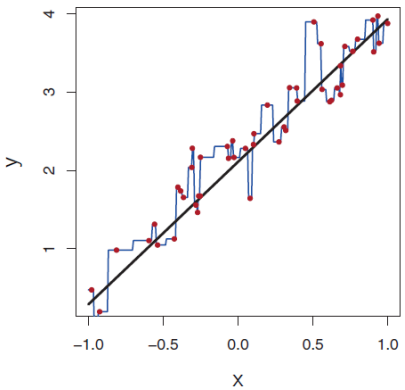
Small value for parameter C

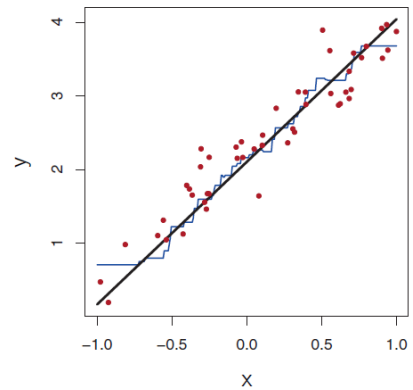
KNN: K=1



KNN: K=100

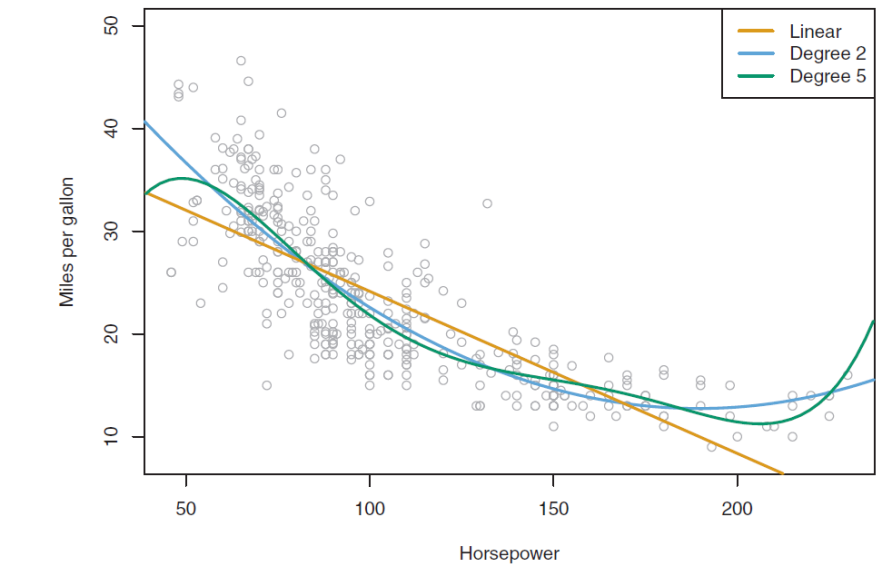


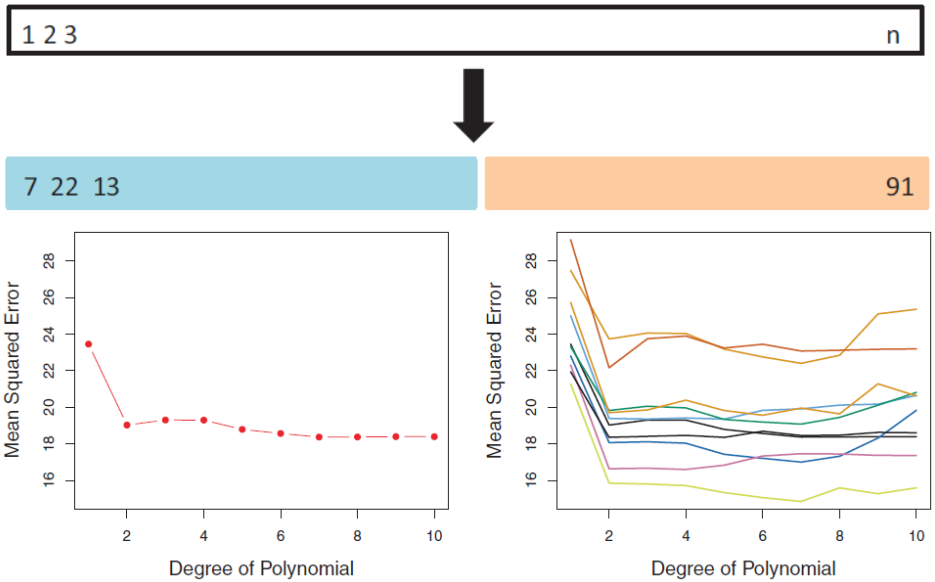




Cross-Validation(CV): Validation Set (Hold-out set)

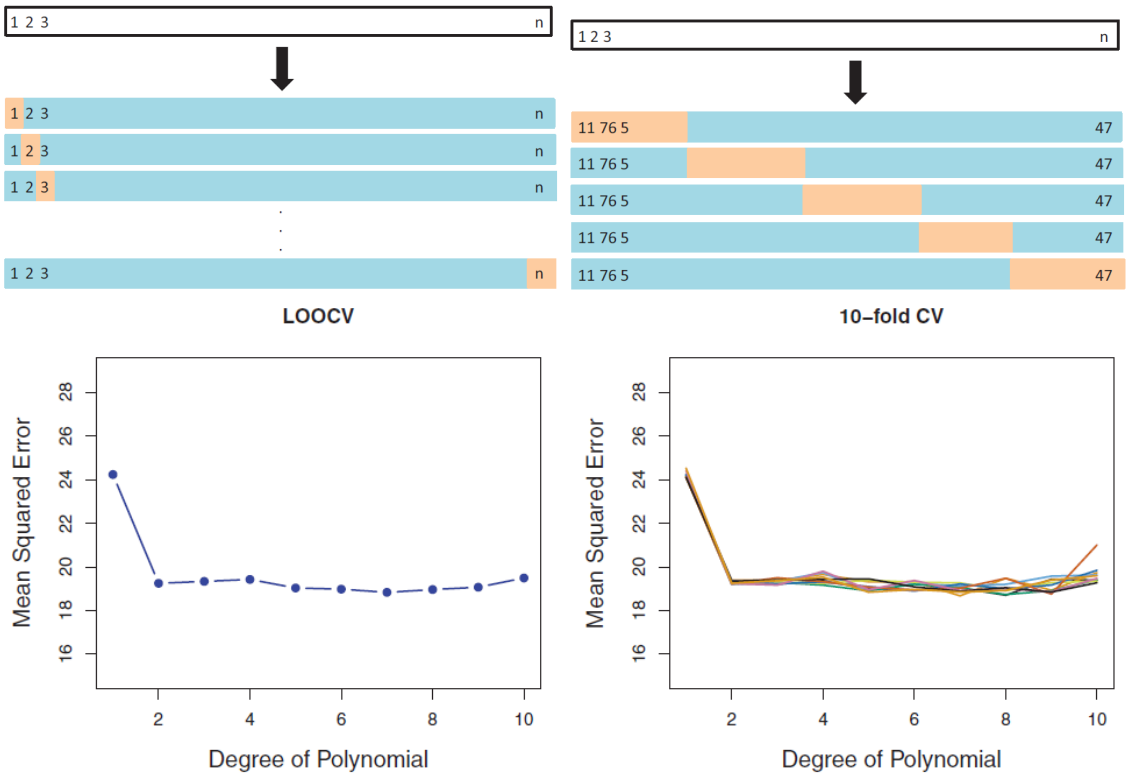
- Divide observations into a **training set** and a **validation (hold-out) set**.
- Fit model on the training set and measure error on the validation set.
- Error rate is highly variable (sensitive to division)
- Error rate tends to over-estimate the true test error rate as the model is trained on fewer observations.





Cross-Validation: Leave-One-Out (LOOCV) and k-Fold CV

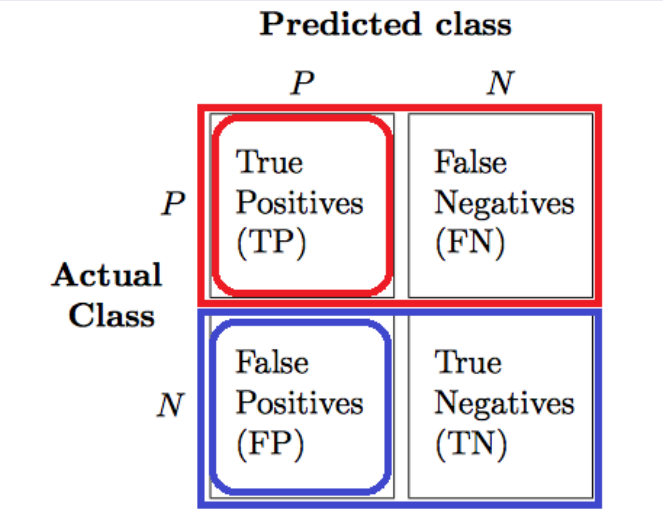
- LOOCV: train model with one sample left out and measure the error on the sample. Error is close to the true test rate but computation is heavy (train n times).
- k-fold CV: divide the samples into k (typically 5 or 10) folds. Train model on $k-1$ **training** folds and measure error on the remaining **test** fold.



Evaluation Metrics

Confusion Matrix

- Error rate: $\text{ERR} = \frac{\text{FP}+\text{FN}}{\text{FP}+\text{FN}+\text{TP}+\text{TN}}$
- True Positive rate: $\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}}$
- False Positive rate: $\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}$
- Precision(PRE), Recall(REC), F1-Score($F1$), $\cdots$.



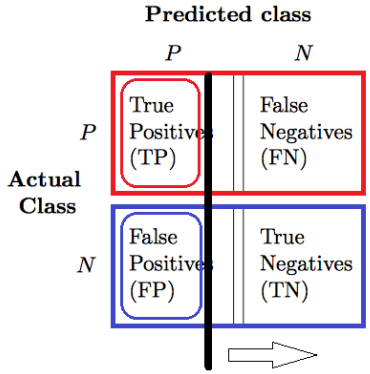
Credit card default example

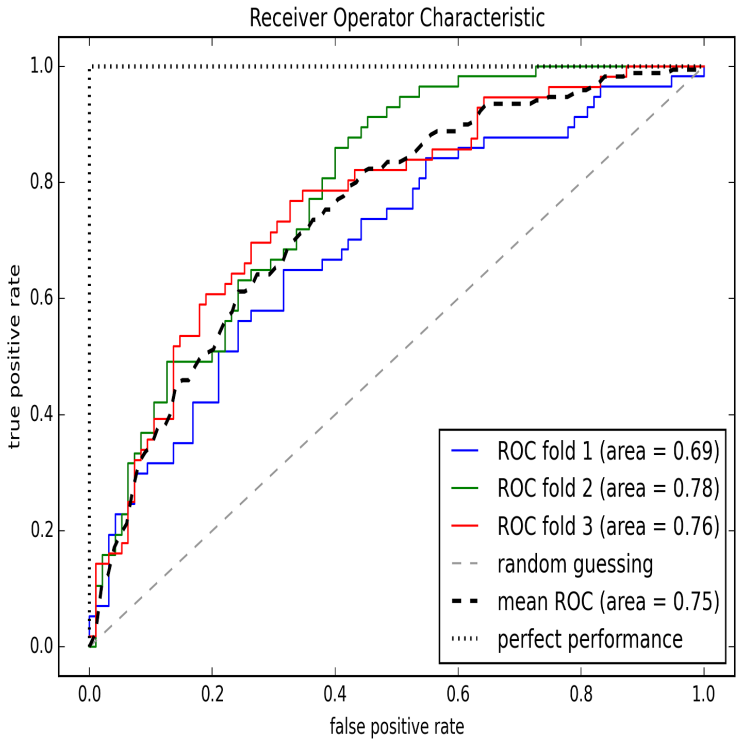
- $\text{ERR} = \frac{23+252}{10{,}000} = 2.75\%$
- $\text{FPR} = \frac{23}{9{,}667} = 0.2\%$
- $1 - \text{TPR} = \frac{252}{333} = 75.7\%$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

Receiver Operator Characteristic (ROC) Curve

- Graph of (FPR, TPR) for varying classification threshold of the binary classification.





- Area Under Curve (AUC) give an overall accuracy of a classifier, summarizing over all possible threshold
- The diagonal line is from random-guessing: ROC AUC $= 0.5$. A model with lower AUC than 0.5 is worthless.
- A perfect classifier (Γ-shaped lines): ROC AUC $= 1.0$.

Unsupervised Learning

No response variable (y) associated to feature variable (X), thus no prediction.

Goals

- Find interesting things about the observations X.
- How to visualize the data?
- How to subgroup (cluster) the observations?

Challenges

- No clear answer $\longrightarrow$ no clear way to measure performance

K-mean Clustering

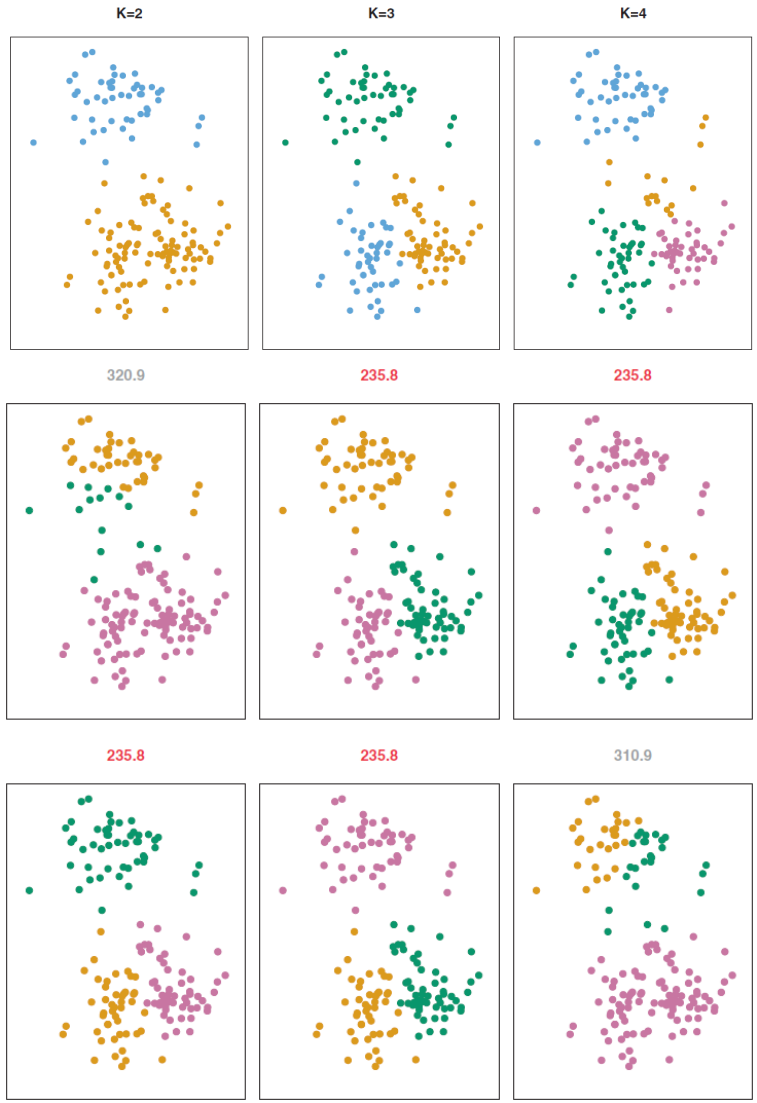
- Assign each observation $\boldsymbol{X}_{i*}$ in to (predetermined) K clusters
- Find $\boldsymbol{\mu}_k$ and the clusters C_k which minimize the L2 norm to $\boldsymbol{\mu}_k$,

$$\text{SSE} = \sum_k \sum_{i \in C_k} |\boldsymbol{X}_{i*} - \boldsymbol{\mu}_k|^2,$$

where $\boldsymbol{\mu}_k$ is the center of the k-th cluster C_k.

Drawback:

- The number of clusters K has to be manually selected.
- Difficult to solve. The results are sensitive to initial guess.

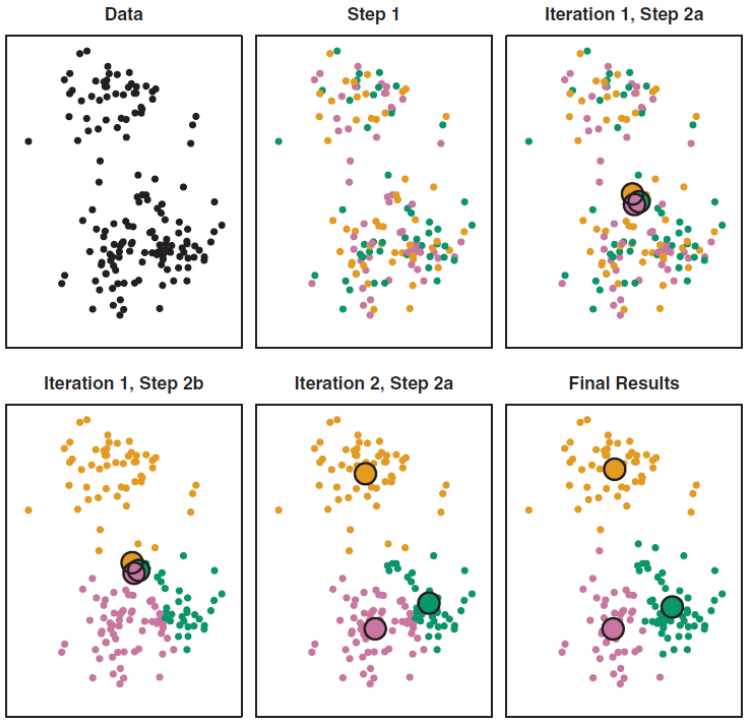


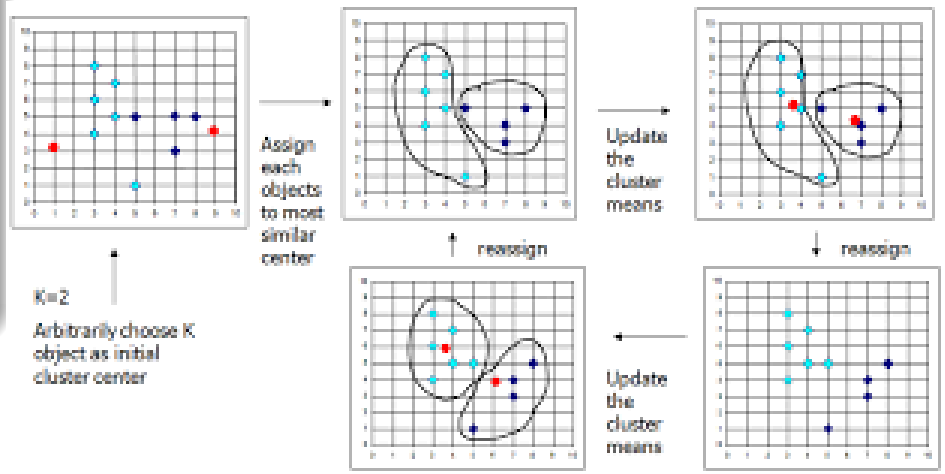
K-mean Clustering Iteration

1. Given $\boldsymbol{\mu}_k$, we assign $\boldsymbol{X}_{i*}$ to the nearest center.
2. Given the cluster C_k, the center should be the mean, $\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \boldsymbol{X}_{i*}$.

Iteration Step

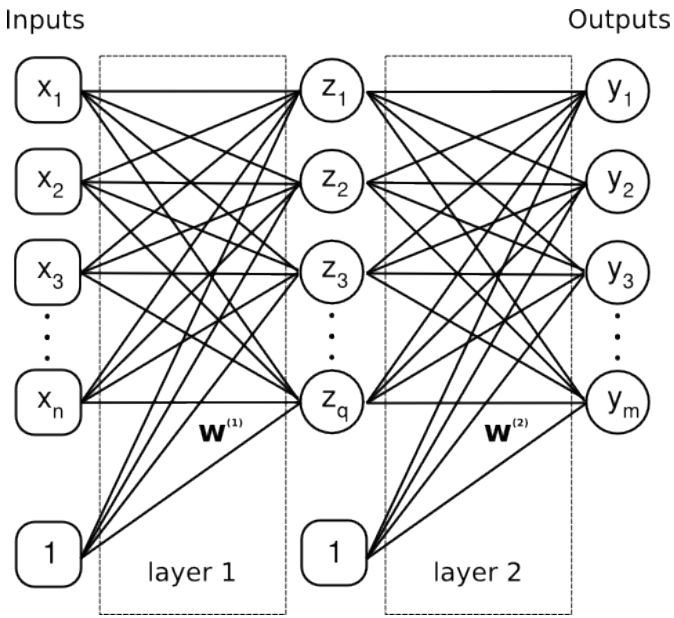
- Randomly pick K centers (or randomly assign samples to C_k).
- Repeat (1) and (2) until $\{C_k\}$ no-longer changes or the maximum iteration is reached.

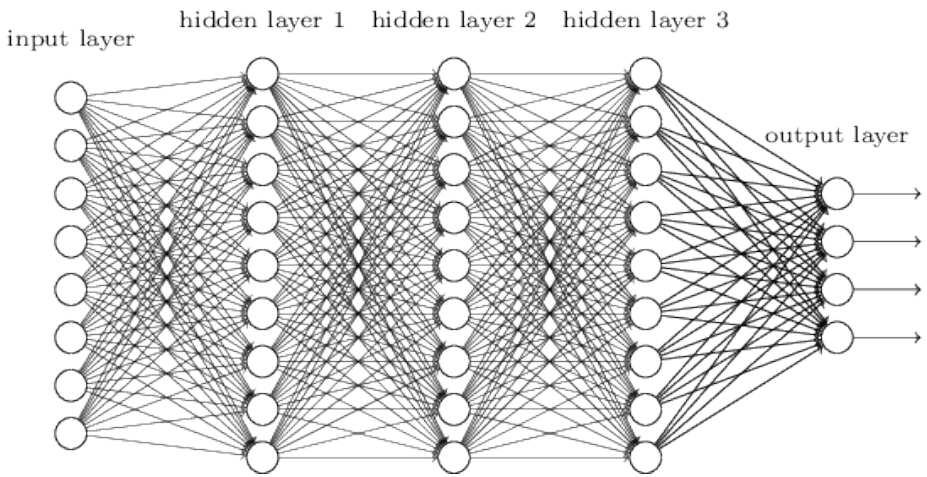




Neural Network

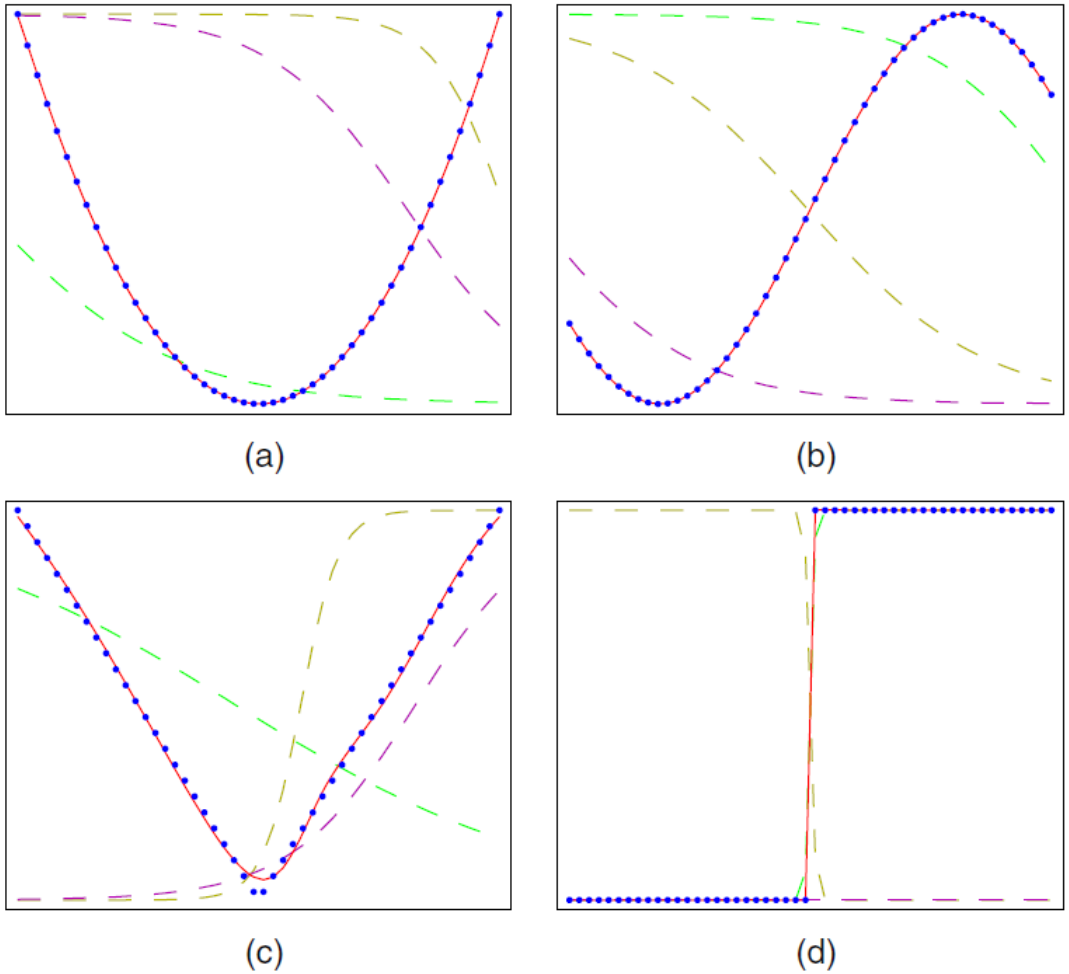
- Multi-layer 'perceptron' (MLP): logistic regression
- **Deep learning**: a set of methods to efficiently train multi-layer NN





Neural Network

Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.



NN Notations

The conventions are **different** from other books.

Single layer

$$\boldsymbol{y} = \phi(\boldsymbol{x}\boldsymbol{W}) \quad \Rightarrow \quad [\cdots\ y_j\ \cdots] = [\cdots\ x_i\ \cdots] \begin{bmatrix} & \cdots & \\ \cdots & W_{ij} & \cdots \\ & \cdots & \end{bmatrix} \quad (x_0 = 1)$$

W_{ij} connects i-th input x_i to j-th output y_j. $\phi(x)$ is the sigmoid function.

Multi-layer: L layers

$$\begin{aligned}
\boldsymbol{x}^{(1)} &= \phi(\boldsymbol{a}^{(1)} = \boldsymbol{x}^{(0)}\boldsymbol{W}^{(1)}) \\
\boldsymbol{x}^{(2)} &= \phi(\boldsymbol{a}^{(2)} = \boldsymbol{x}^{(1)}\boldsymbol{W}^{(2)}) \\
&\cdots \\
\boldsymbol{x}^{(L)} &= \phi(\boldsymbol{a}^{(L)} = \boldsymbol{x}^{(L-1)}\boldsymbol{W}^{(L)}) \\
\boldsymbol{y} &= \phi(\cdots\phi(\phi(\boldsymbol{x}^{(0)}\boldsymbol{W}^{(1)})\boldsymbol{W}^{(2)})\cdots)
\end{aligned}$$

Input: $\boldsymbol{x} = \boldsymbol{x}^{(0)}$, Output: $\boldsymbol{y} = \boldsymbol{x}^{(L)}$

$$[\cdots\ y_k\ \cdots] = [\cdots\ x_i\ \cdots] \times \begin{bmatrix} & \cdots & \\ \cdots & W_{ij}^{(1)} & \cdots \\ & \cdots & \end{bmatrix} \cdots \begin{bmatrix} & \cdots & \\ \cdots & \times W_{jk}^{(L)} & \cdots \\ & \cdots & \end{bmatrix}$$

Training NN

- Multi-variate response variable (n: samples, j: responses)

$$\boldsymbol{Y} = \begin{bmatrix} \cdots \\ \cdots\ Y_{nj}\ \cdots \\ \cdots \end{bmatrix} = \boldsymbol{X}_{n*}\boldsymbol{W}_{*j}$$

- Error (loss) function for the n-th sample:

$$J_n(\boldsymbol{W}) = -\sum_j Y_{nj} \log \phi(X_{nj}) + (1 - Y_{nj}) \log \big(1 - \phi(X_{nj})\big)$$

- Gradient: (i: input, j: output)

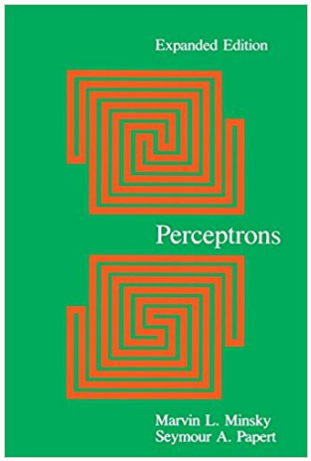
$$\frac{\partial J_n(\boldsymbol{W})}{\partial W_{ij}} = -(Y_{nj} - \phi(X_{nj}))X_{ni}$$

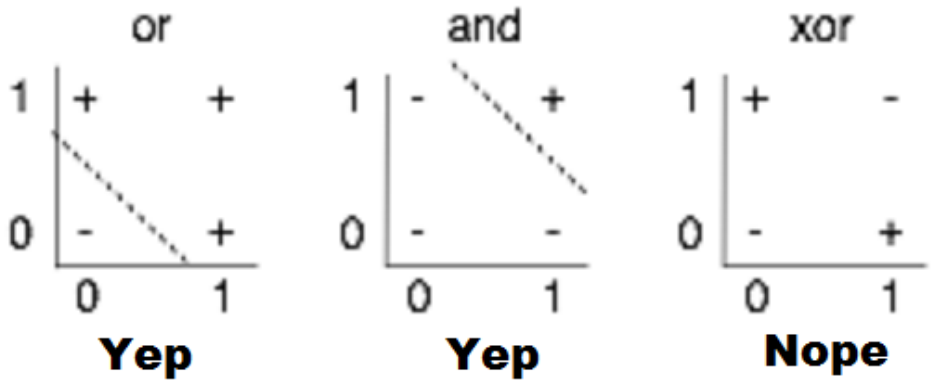
- Total loss function with regularization:

$$J(\boldsymbol{W}) = \sum_n J_n(\boldsymbol{W}) + \lambda_1 \sum_k \sum_{ij} |W_{ij}^{(k)}| + \frac{\lambda_2}{2} \sum_k \sum_{ij} (W_{ij}^{(k)})^2$$

The 1st AI winter (1969∼1986)

- *Perceptrons* by Minsky and Papert (1969) mathematically proved that the single layer can not learn exclusive OR (XOR) gate due to its nonlinear feature. MLP is needed but it's too complicated to train the parameters.
- The criticism from the leading AI figure effectively killed all AI research.
- The difficulty is resolved by backpropagation by Werbos (1974,1982), Hinton(1986).





Backpropagation: chain rule

It's a fancy name of chain rule in calculus. Imagine input x_0 goes through functions f_1, f_2 and f_3 to reach the output $y = f_3(f_2(f_1(x_0)))$:

$$x_0 \to f_1(x_0) = x_1 \to f_2(x_1) = x_2 \to f_3(x_2) = x_3 = y$$

The chain rule is to multiply the derivative of each function:

$$\frac{\partial y}{\partial x_0} = \frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x_1}\frac{\partial x_1}{\partial x_0} = f_3'(x_2)f_2'(x_1)f_1'(x_0),$$

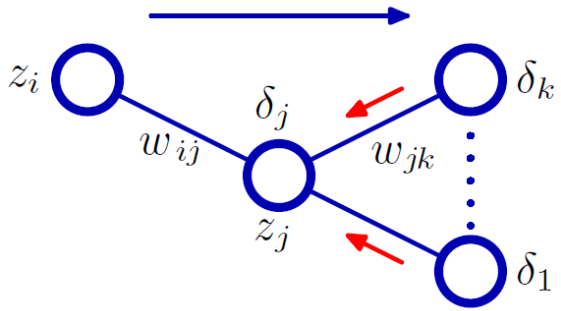
Now imagine the weight w is multiplied in the function,
$y = f_3(f_2(f_1(x_0w_1)w_2)w_3)$

$$x_0 \to f_1(x_0w_1) = x_1 \to f_2(x_1w_2) = x_2 \to f_3(x_2w_3) = x_3 = y$$

$$\frac{\partial y}{\partial w_3} = f_3'(x_2w_3) \cdot x_2$$
$$\frac{\partial y}{\partial w_2} = f_3'(x_2w_3) \cdot w_3 f_2'(x_1w_2) \cdot x_1$$
$$\frac{\partial y}{\partial w_1} = f_3'(x_2w_3) \cdot w_3 f_2'(x_1w_2) \cdot w_2 f_1'(x_0w_1) \cdot x_0$$

Backpropagation: error term δ

Imagine W_{ij} connects unit x_i (layer $m-1$) to x_j (layer m) and the unit x_j is connected to the units x_k in the next layer $m+1$. Remind that



$$a_j = \sum_i x_i W_{ij}, \quad x_j = \phi(a_j)$$

$$\frac{\partial J_n}{\partial W_{ij}} = \frac{\partial J_n}{\partial a_j}\frac{\partial a_j}{\partial W_{ij}} = \delta_j x_i, \quad \text{where} \quad \delta_j := \frac{\partial J_n}{\partial a_j}$$

The term δ_j is referred to as error because $\delta_j = \hat{y}_j - y_j$ for the output units. Now apply chain rule to obtain the backpropagation,

$$\delta_j := \frac{\partial J_n}{\partial a_j} = \sum_k \frac{\partial J_n}{\partial a_k} \cdot \frac{\partial a_k}{\partial x_j} \cdot \frac{\partial x_j}{\partial a_j} = \phi'(a_j) \sum_k W_{jk}\delta_k$$

Back-propagation: $\boldsymbol{\delta}^{(m)} = \phi'(\boldsymbol{a}^{(m)}) * \left(\boldsymbol{\delta}^{(m+1)}(\boldsymbol{W}^{(m+1)})^T\right)$

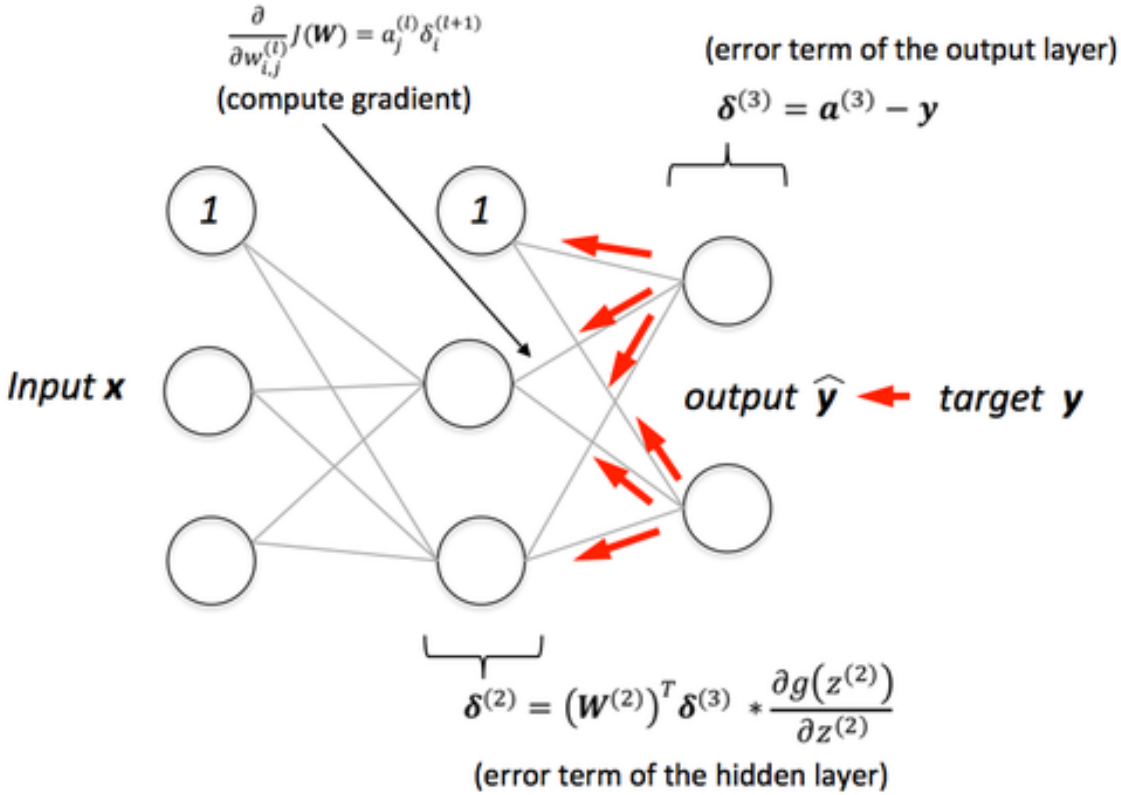
v.s. Forward-propagation: $\boldsymbol{x}^{(m+1)} = \phi(\boldsymbol{a}^{(m)} = \boldsymbol{x}^{(m)}\boldsymbol{W}^{(m+)})$

where $*$ is the element-wise multiplication.

Backpropagation: error term δ (cont)

$$x = \phi(a) = \frac{1}{1+e^{-a}} \quad \rightarrow \quad \frac{\partial x}{\partial a} = \phi'(a) = \frac{e^{-a}}{(1+e^{-a})^2} = \phi(a)(1-\phi(a)) = x(1-x)$$

$$x = \phi(a) = \tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}} \quad \rightarrow \quad \frac{\partial x}{\partial a} = \phi'(a) = (1-\phi^2(a)) = (1-x^2)$$



Backpropagation: Summary and benefit

Procedure

1. Forward-propagate $\boldsymbol{x}$'s and $\boldsymbol{a}$'s: $\boldsymbol{x}^{(L+1)} = \phi(\boldsymbol{a}^{(L)} = \boldsymbol{x}^{(L)}\boldsymbol{W})$
2. Evaluate $\boldsymbol{\delta}^{(L)} = \hat{\boldsymbol{y}} - \boldsymbol{y}$ at the output units
3. Backpropagate $\boldsymbol{\delta}$'s: $\boldsymbol{\delta}^{(m)} = \phi'(\boldsymbol{a}^{(m)}) * \left(\boldsymbol{\delta}^{(m+1)}(\boldsymbol{W}^{(m+1)})^T\right)$
4. Obtain derivatives: $\dfrac{\partial J_n}{\partial W_{ij}^{(m)}} = \delta_j^{(m)} X_{ni}^{(m-1)}, \quad \dfrac{\partial J}{\partial W_{ij}^{(m)}} = \sum \delta_j^{(m)} X_{ni}^{(m-1)}$

Benefits

Numerical differentiation:

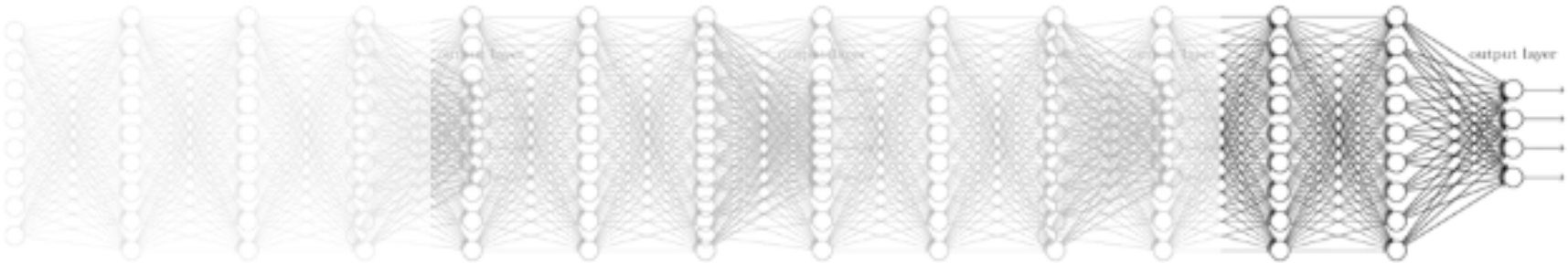
- $\frac{\partial J}{\partial W_{ij}} = \frac{J(W_{ij}+\varepsilon) - J(W_{ij}-\varepsilon)}{2\varepsilon}$
- require $O(N_w^2)$ operations for the total number of weights N_w.
- can be used to validate the backpropagation

Back-propagation:

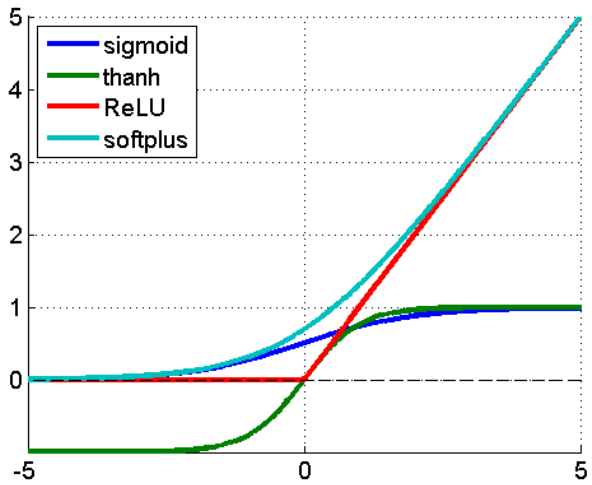
- require $O(N_w)$ operations.

The 2nd AI winter (1986-2006)

- Even the backpropagation is not enough for 'deep' layers as the gradient vanishes.



- Other 'better' activation functions used: $\phi(x) =$



Sigmoid (logistic) $\frac{1}{1+e^{-x}}$

Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

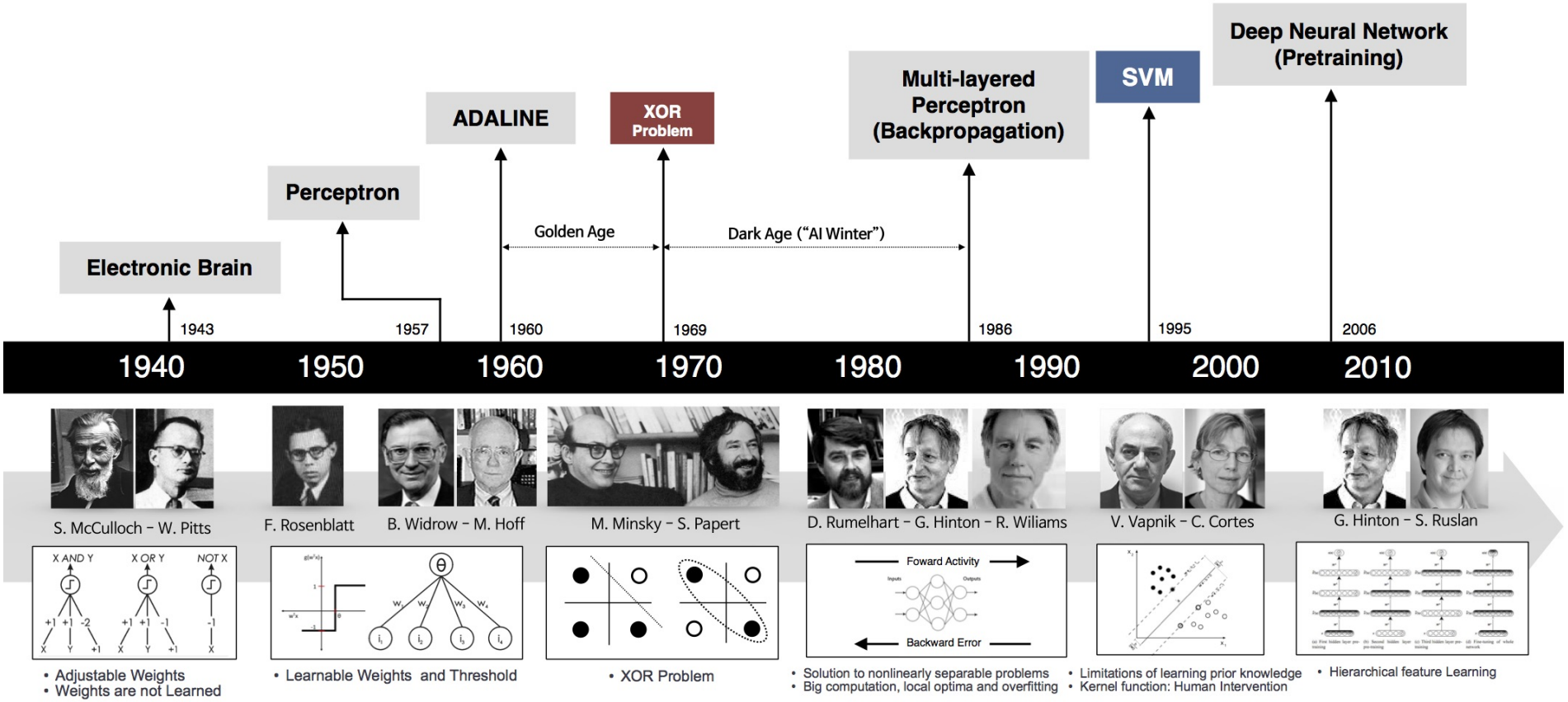
ReLU (Rectified linear unit) $\max(0, x)$

Softplus $\log(1 + e^x)$

ReLU or **Softplus** activations have non-vanishing gradient.

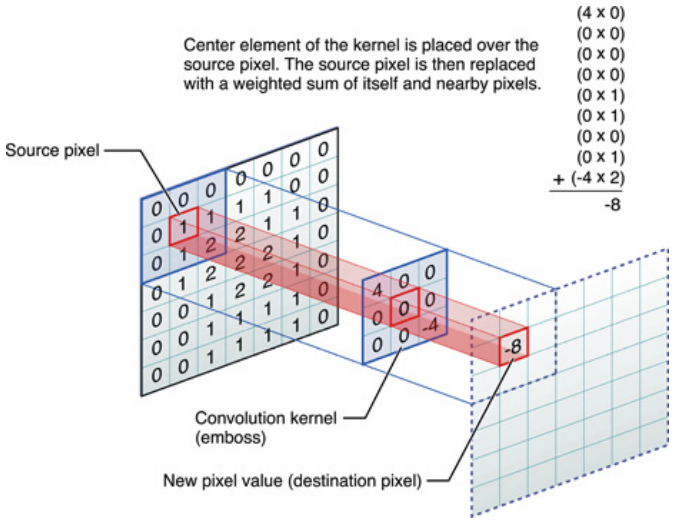
Timeline of NN

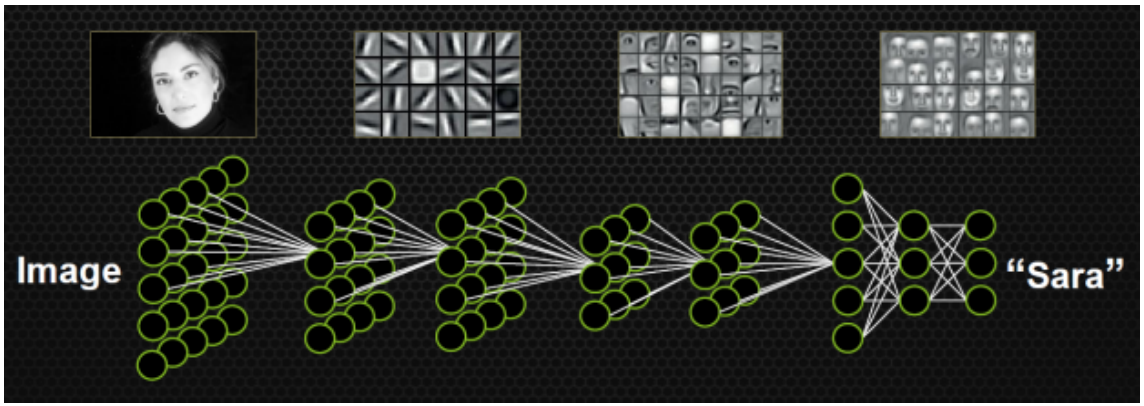
From Andrew Beam's Deep Learning 101



Convolutional Neural Network (CNN / ConvNet)

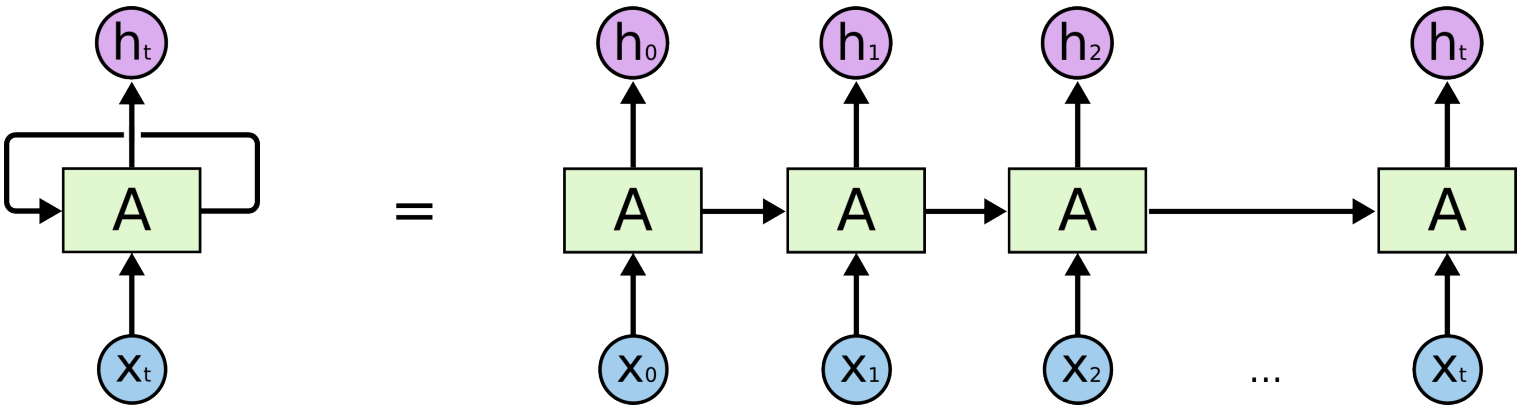
- Image classification
- Apply spatial filters to create feature maps / convolutional layers
- Pooling/sampling layers (max, min, average, etc)
- Multi-layer NN
- Demo: `http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`





Recurrent Neural Network (RNN)

- Sequence data: speech recognition, translation, connected hand-writing, etc
- Recurring hidden layer
- Example training sequence: "HELLO"



Soft-max classification function

- In multi-class classification, we need to *normalize* the outcome for the last unit to the probability on each class for a given sample.
- Given the output $\boldsymbol{a}$, we generalize the logistic function:

$$P(y = i|\boldsymbol{a}) = \phi_{\text{softmax}}(\boldsymbol{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}} \qquad \left(\sum_i P(y = i|\boldsymbol{a}) = 1\right)$$

- The probability has the largest value for the largest value of a_i.
- The logistic function is a special case with $a_0 = 0$: $\phi(a) = e^a/(e^0 + e^a)$.
- In NN for multi-class classification, soft-max is applied to the last units to match the 'one-hot' encoding of the response variable.