

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

Jaehyuk Choi

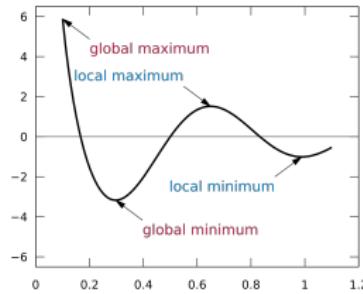
2017-18 Module 3 (Spring 2018)

AlphaGo vs Humans (LEE Sedol, KE Jie)



We have to think again about joseki / 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): $\mathbf{x} = (x_i), \mathbf{y} = (y_i)$
- Matrix (uppercase bold): $\mathbf{X} = (X_{ij}), \mathbf{Y} = (Y_{ij})$
- The (i, j) component of \mathbf{X} : X_{ij}
- The i -th row vector of \mathbf{X} : $\mathbf{X}_{i*} = (X_{i1}, X_{i2}, \dots, X_{ip})^T$
- The j -th column vector of \mathbf{X} : $\mathbf{X}_{*j} = (X_{1j}, X_{2j}, \dots, X_{Nj})$

Examples

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

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**:
 \mathbf{X} ($N \times p$ matrix), \mathbf{Y} ($N \times K$ matrix)
- The i -th observation of the j -th feature: \mathbf{X}_{ij} v.s. $X_j^{(i)}$ in **PML**
- The i -th observation set: \mathbf{X}_{i*} ($1 \times p$ row vector) v.s. $X^{(i)}$ in **PML**
- All observation of j -th feature X_j : \mathbf{X}_{*j} ($N \times 1$ row vector) v.s. X_j in **PML**
- $\mathbf{X} = (\mathbf{X}_{*1} \cdots \mathbf{X}_{*p})$ (column-wise concatenation)
- The weight vector, β or w , are **column** vectors used interchangeably.

Simple Linear Regression (Ordinary Least Square)

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

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

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

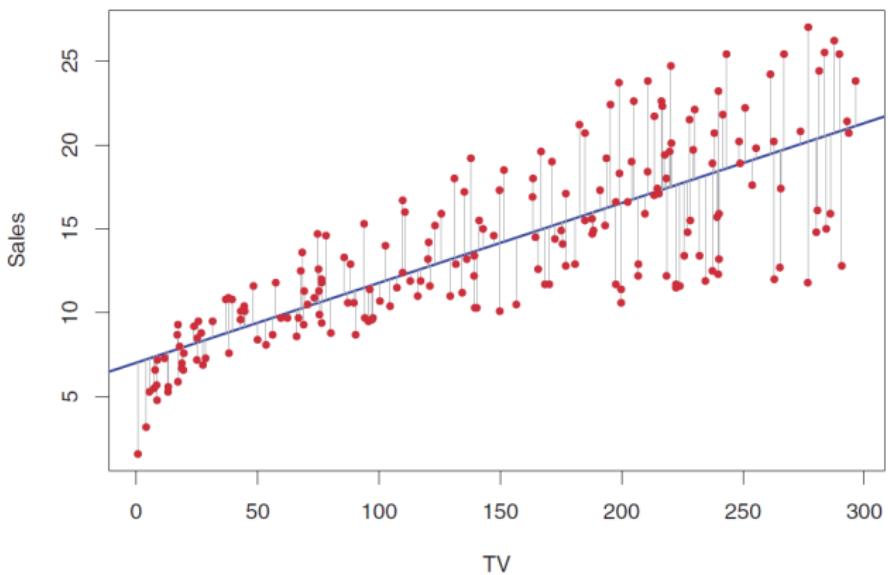
$$\text{RSS}(\beta) = \sum_{i=1}^N (y_i - \beta_0 - \beta_1 x_i)^2 = (\mathbf{y} - \beta_0 - \beta_1 \mathbf{x})^T (\mathbf{y} - \beta_0 - \beta_1 \mathbf{x})$$

is given as

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

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),

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

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

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

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

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

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

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

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

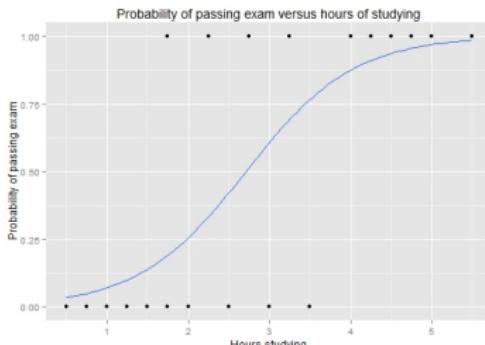
Logistic Regression (Classification)

- 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: $\sigma(\text{logit}) = \text{quantile}$

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

- 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 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 (\mathbf{X}, \mathbf{y}) where $y_i \in \{0, 1\}$,

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

$$\log L(\mathbf{w}) = \sum_i y_i \log \phi(\mathbf{X}_{i*}\mathbf{w}) + (1 - y_i) \log (1 - \phi(\mathbf{X}_{i*}\mathbf{w}))$$

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

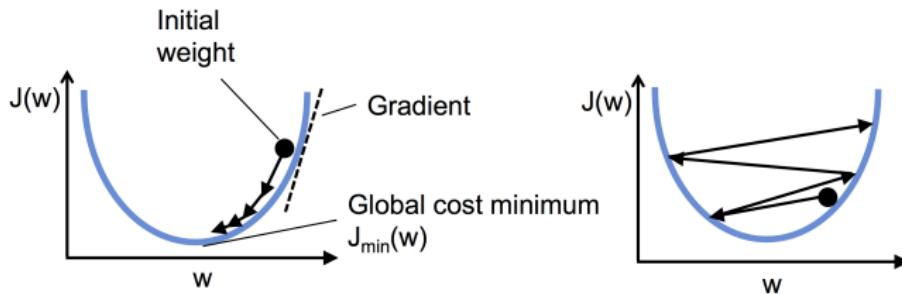
Gradient Descent

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 minimum location $J(\mathbf{x})$ can be found by the following iteration:

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

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 w to minimize the error (or cost)?

Imagine we fit a simple linear model $y = xw$ to a single observation (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

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

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

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

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

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

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

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

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

Gradient Descent of Weight: Logistic Regression

We use the properties of logistic function,

$$\frac{d}{dt}\phi(t) = \frac{e^{-t}}{(1+e^{-t})^2} = \phi(t)(1-\phi(t)), \quad \frac{\partial}{\partial w_j}\phi(\mathbf{X}_{i*}\mathbf{w}) = \phi(\cdot)(1-\phi(\cdot))X_{ij},$$

the gradient of error function is obtained as

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

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

To avoid *overfitting*, we do not want \mathbf{w} to be too big. We add *penalty* for big \mathbf{w} :

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

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

An example from polynomial curve fitting:

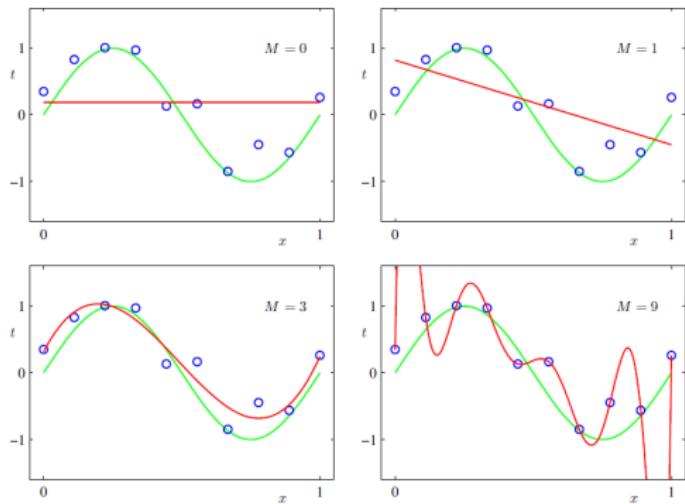


Figure 1.4 Plots of polynomials having various orders M , shown as red curves, fitted to the data set shown in Figure 1.2.

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

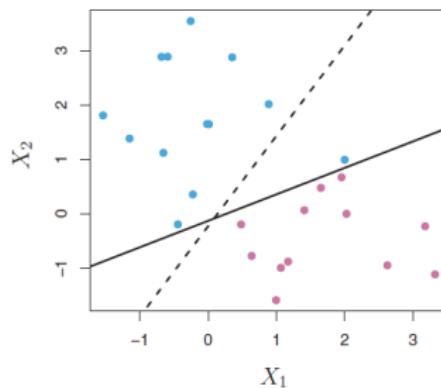
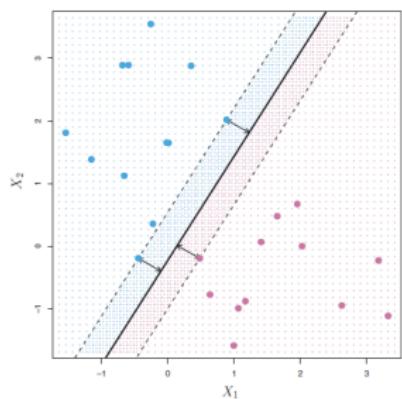


Maximal margin classifier

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

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

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 $|\mathbf{w}| = 1$,

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

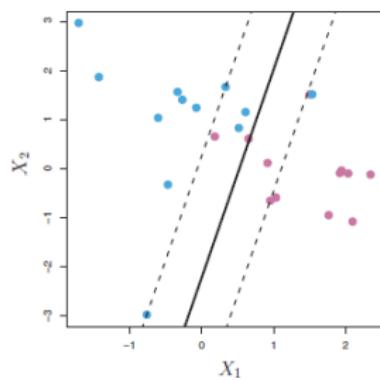
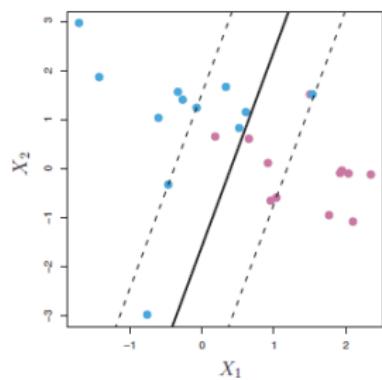
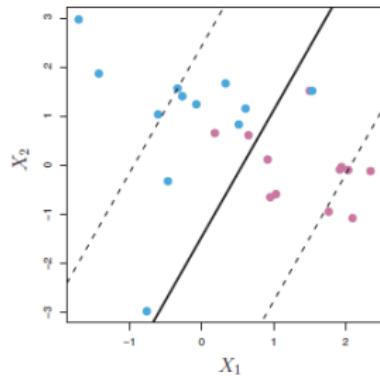
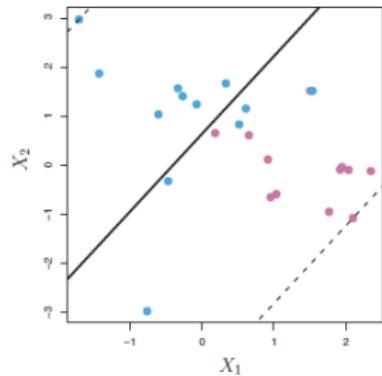
where $\epsilon_i \geq 0$ is *slack variable* indicating the degree of violation ($\epsilon_i = 0$: no violation, $\epsilon_i < 1$: margin violation, $\epsilon_i > 1$: classification violation) and C is a *budget* for the amount of violations by all observations.

Alternatively (in PML), we minimize

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

where $\{\xi_i \geq 0\}$ satisfies $y_i(w_0 + \mathbf{X}_{i*}\mathbf{w}) \geq 1 - \xi_i$ 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.

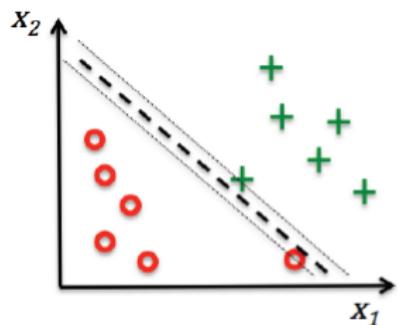
Support vector classifier: role of C



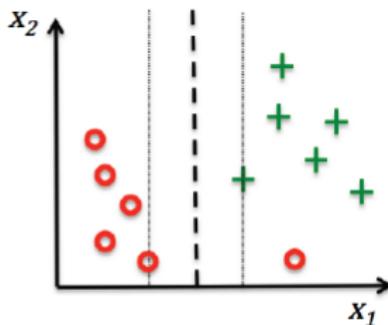
The value of C decreasing from the largest value on top left.

Support vector classifier: role of C'

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



Large value for
parameter C



Small value for
parameter C

Support vector machines (SVM)

How can we deal with non-linear decision boundary?

Enlarging feature space

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

Kernel

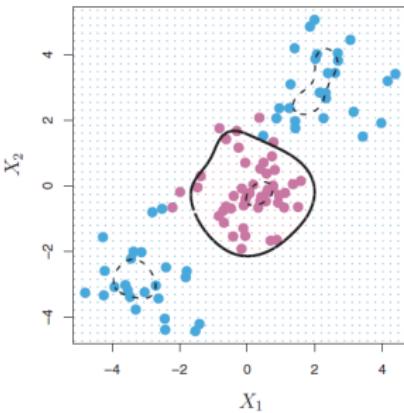
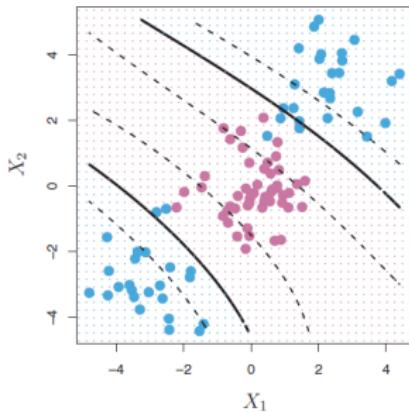
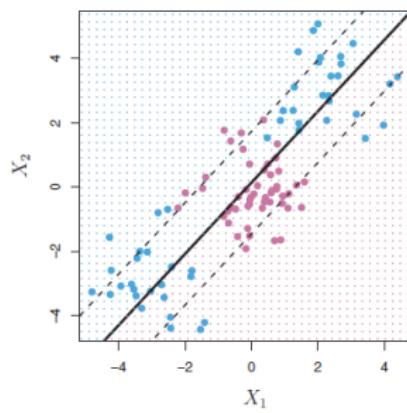
Instead we introduce kernel function, as a generalization of dot product in hyperplane:

- Linear: $K(\mathbf{X}_{i*}, \mathbf{X}_{j*}) = \mathbf{X}_{i*} \mathbf{X}_{j*}^T$
- Polynomial: $K(\mathbf{X}_{i*}, \mathbf{X}_{j*}) = (1 + \mathbf{X}_{i*} \mathbf{X}_{j*}^T)^d$ (order d)
- Radial basis: $K(\mathbf{X}_{i*}, \mathbf{X}_{j*}) = \exp(-\gamma |\mathbf{X}_{i*} - \mathbf{X}_{j*}|^2)$ ($\gamma = 1/2\sigma^2$)

Kernel $K(\mathbf{X}_{i*}, \mathbf{X}_{j*})$ can be understood as a *distance* between two observations: \mathbf{X}_{i*} and \mathbf{X}_{j*} 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)

The method is based on the set of the K nearest neighbors around x , $N_K(x)$:

- Regression:

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

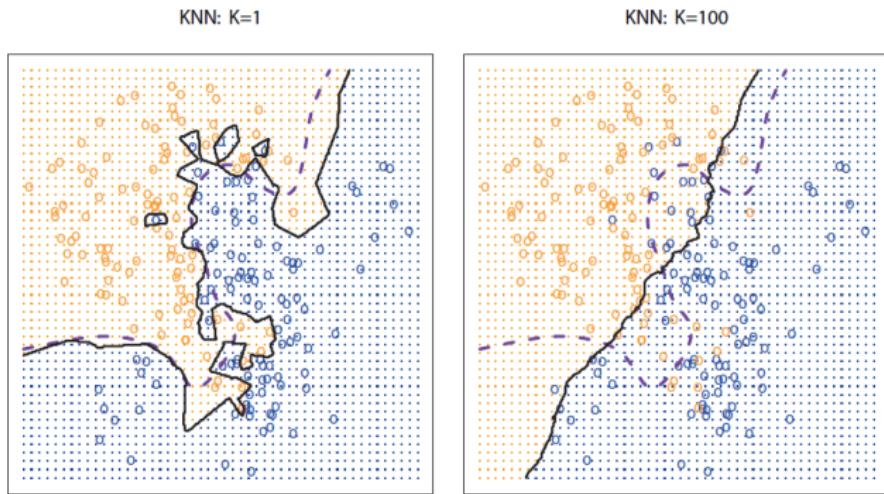
- Classifier:

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

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

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

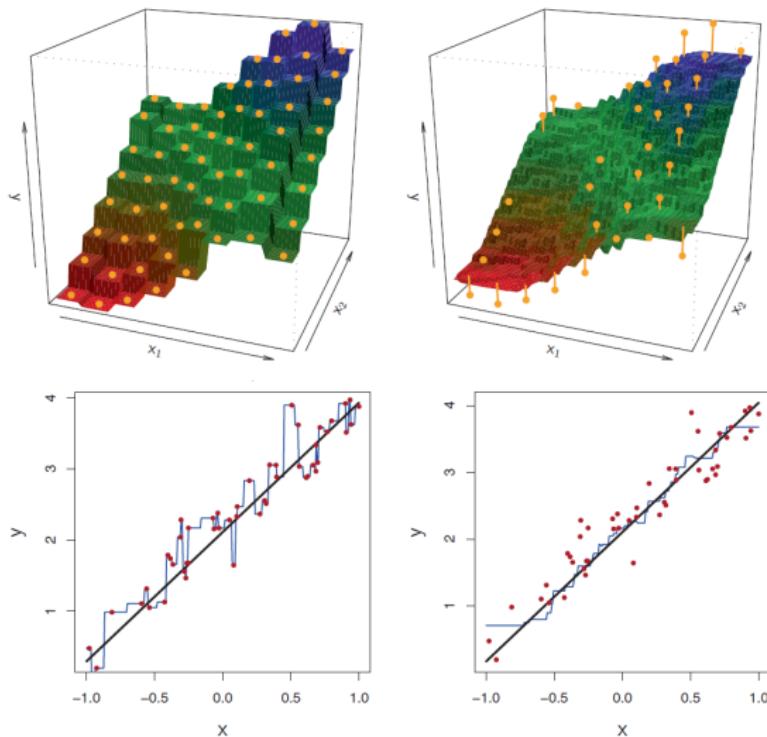
KNN: Classifier example



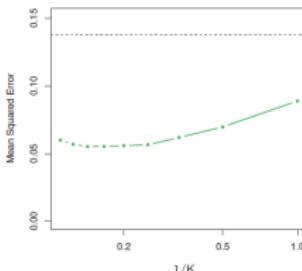
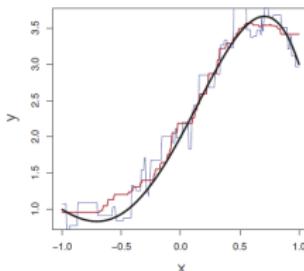
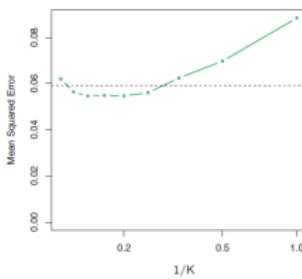
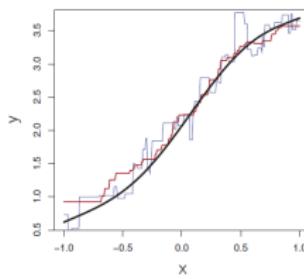
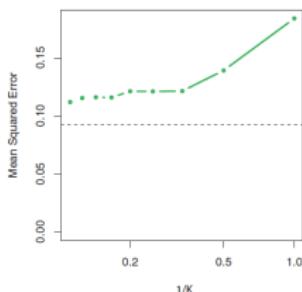
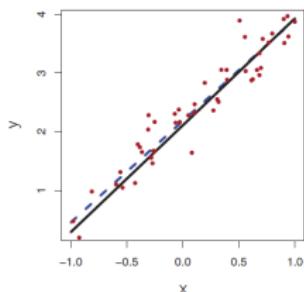
- Overfitting as $K \rightarrow 1$
- Generalization (accuracy on test set) is difficult as $K \rightarrow 1$

KNN: Regression example

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



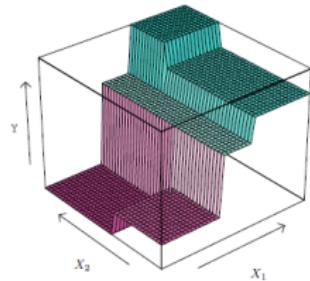
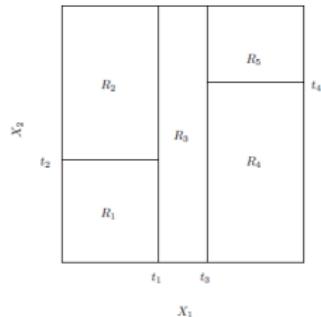
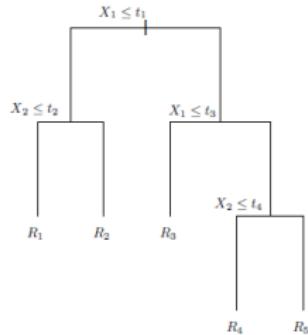
KNN: Parametric vs Non-parametric



Non-parametric regression works better as the true function deviates from the basis function (linear function in this example). The dashed line is the test set RSS from linear regression.

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

We can grow tree to the maximum level so that each leaf contains only one sample point. However, it is **over-fitting**. We need to **regularize** # of leaves or the maximum level of branching.

Growing Tree

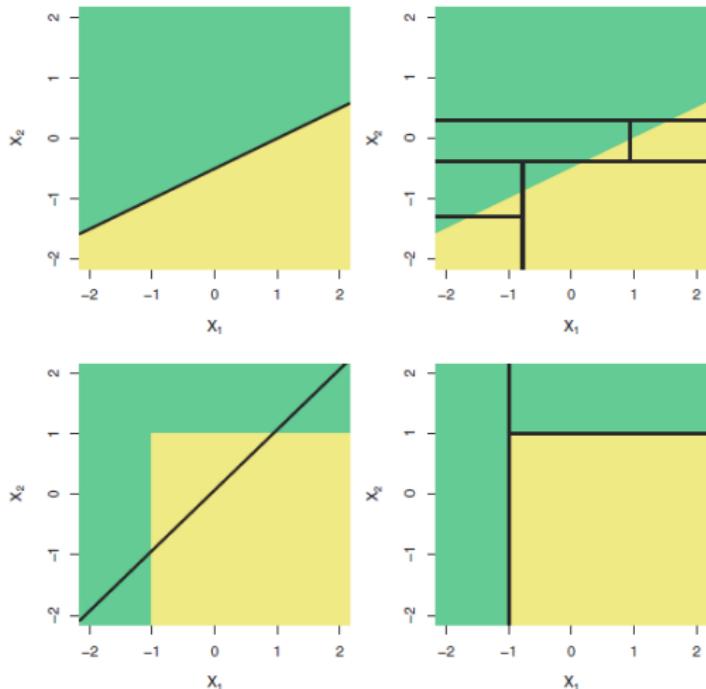
We decide the binary split condition (and feature) in such a way that maximize the information gain (or increase purity):

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

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.

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

Tree vs Linear model



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

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 (sensitive to input samples)
→ 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

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

Standardization

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

Regularization L-1 vs L-2

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

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

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

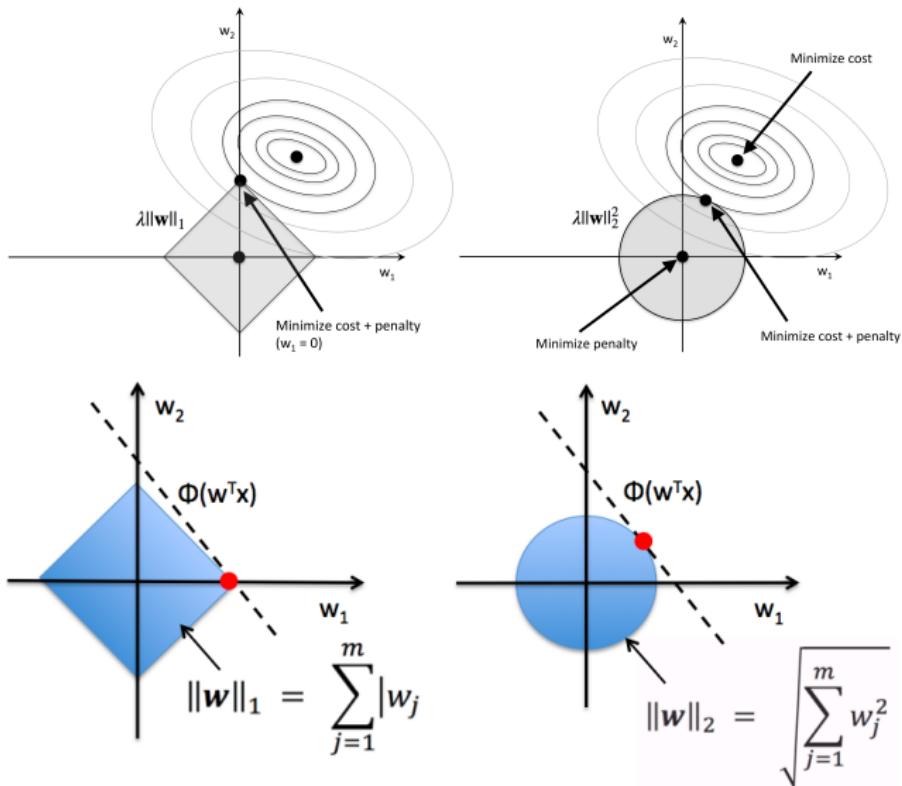
L-2 Regularization

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

L-1 Regularization

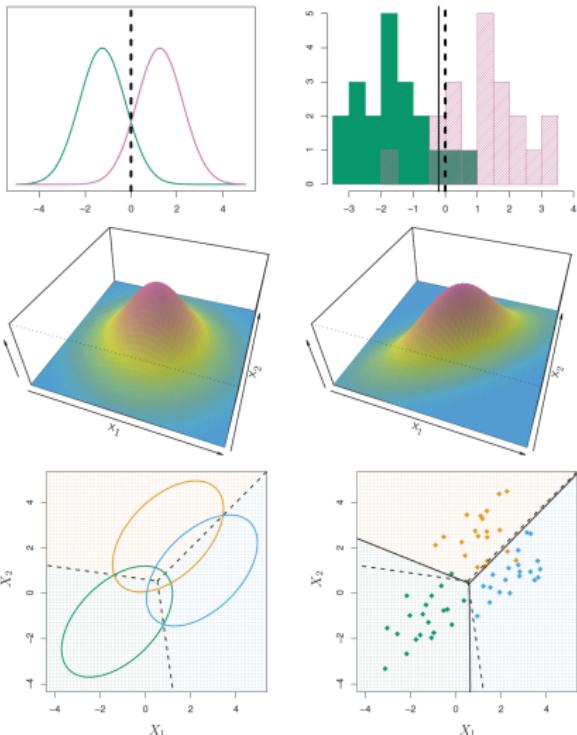
- $R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_j |w_j|$
- ‘Diamond’ boundary: leads to sparse vector (many zero components)
- Effectively works as feature selection

Regularization L-1 vs L-2



Linear Discriminant Analysis (LDA) as a classifier

- Assume the samples in each class follow normal (Gaussian) distribution.
- Estimate mean $\hat{\mu}_k$ and variance $\hat{\Sigma}_k$ of class k :
- Obtain multivariate normal PDF:
$$f_k(\mathbf{x}) = n(\mathbf{x}|\hat{\mu}_k, \hat{\Sigma}_k)$$
- LDA if $\Sigma_W = \sum_{k=1}^K \Sigma_k$ (within covariance) is used for all Σ_k .
- QDA if Σ_k is estimated for each class k
- A test sample \mathbf{x} is classified to the class k for which $f_k(\mathbf{x})$ is largest.



LDA as a dimensionality reduction

- Given the LDA assumptions, which direction \mathbf{w} best separates the feature?
- $\mathbf{w} \approx \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1$? Probably not the best.
- If $(\boldsymbol{\mu}_{1,2}, \sigma_{1,2}^2)$ is the mean and variance pair of the samples (1-D) projected on \mathbf{w} , $y = \mathbf{x}\mathbf{w}$, with $|\mathbf{w}| = 1$, we want to maximize the Fisher criterion:

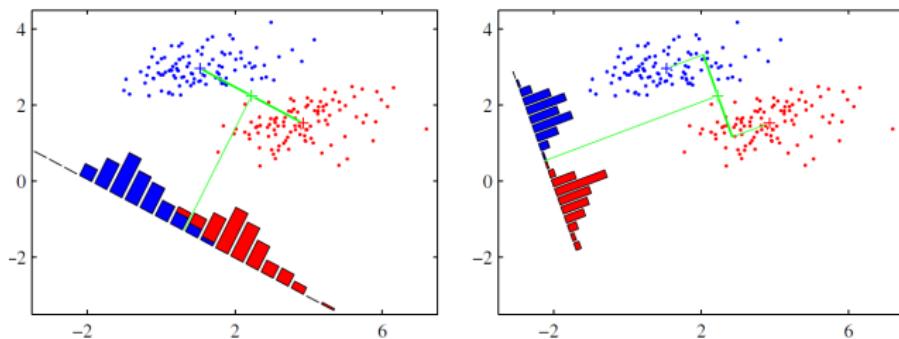
$$J(\mathbf{w}) = \frac{(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^2}{N_1\sigma_1^2 + N_2\sigma_2^2} = \frac{\mathbf{w}^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \mathbf{w}}{\mathbf{w}^T (N_1 \boldsymbol{\Sigma}_1 + N_2 \boldsymbol{\Sigma}_2) \mathbf{w}} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}},$$

where \mathbf{S}_W and \mathbf{S}_B are *within-* and *between-class* variance matrices

$$\mathbf{S}_W = \sum_{k=1,2} N_k \boldsymbol{\Sigma}_k, \quad \mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

LDA as a dimensionality reduction

- The direction w maximizing $J(w)$ is $\propto S_W^{-1}(\mu_2 - \mu_1)$.
- In general, the eigenvectors, W , of $S_W^{-1}S_B$ in the decreasing order of eigenvalue (similar to PCA) are the best directions to discriminate features.
- The transformation $z = xW$ is the extracted factors with the best separability, which can be used for other ML methods.



Measuring quality of ML method

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

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

where, $y = f(x) + \varepsilon$ (true pattern)

- By a given ML method, we mean that model (LR, SVM, etc) and hyper-parameter (C , γ , reduced dimension k for PCA/LDA, etc) are fixed. However fitted model parameters (i.e., \hat{f}) can change over training set.
- The expectation is made over repeatedly selecting different training vs test dataset. Therefore, the expectation is over \hat{f} as well as x .
- We need to minimize $\text{Bias}(\hat{f}(x))^2$ and $\text{Var}(\hat{f}(x))$ together while $\text{Var}(\varepsilon)$ is fundamentally irreducible.

Bias and Variance

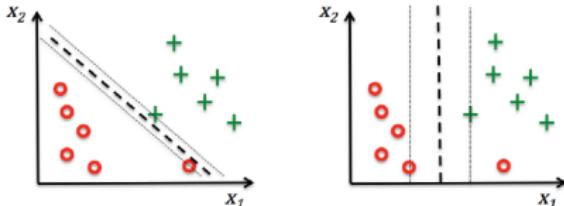
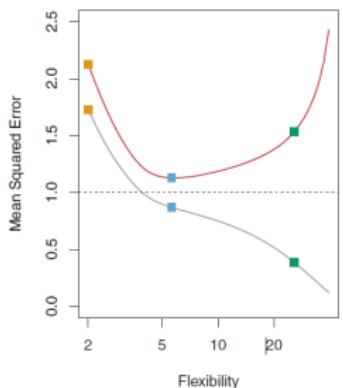
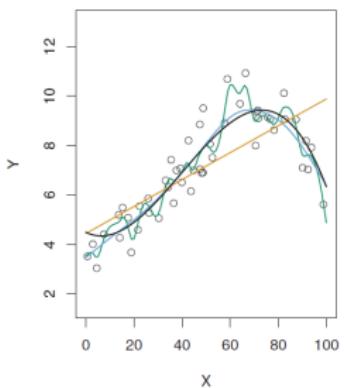
Bias

- Error from \hat{f} not correctly representing the true f (e.g. linear regression on non-linear data).
- A model has **high bias** when \hat{f} overly simply f (under-fitting), i.e., the used 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 the model is too flexible (over-fitting), i.e., there are too many parameters, e.g. KNN with $K = 1$, high-order polynomial regression, SVM/LR with large C (small λ), decision tree with many leaves, etc.

Bias and Variance (examples)

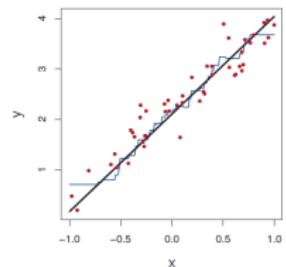
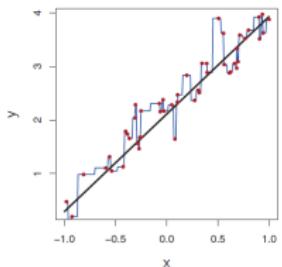
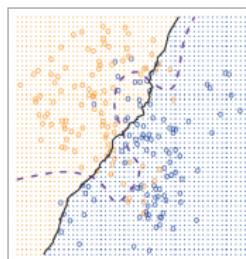
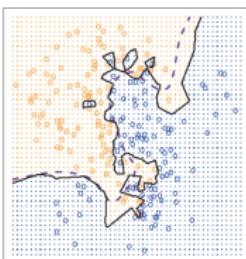


Large value for
parameter C

KNN: K=1

Small value for
parameter C

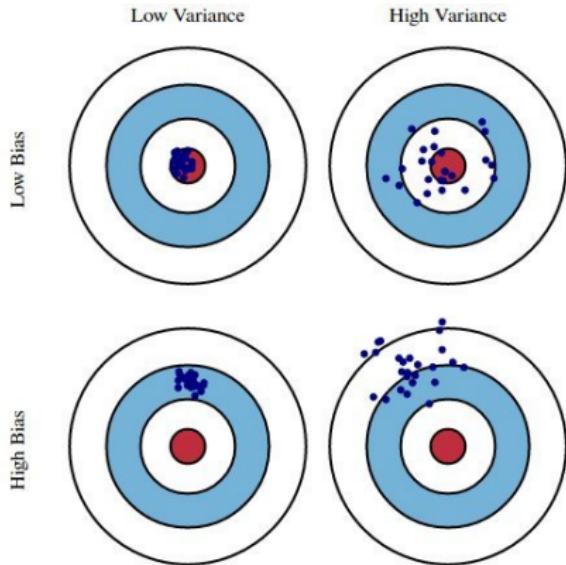
KNN: K=100



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

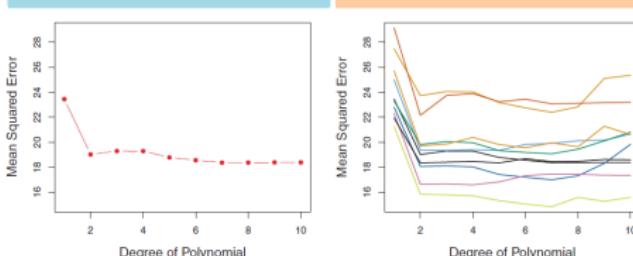
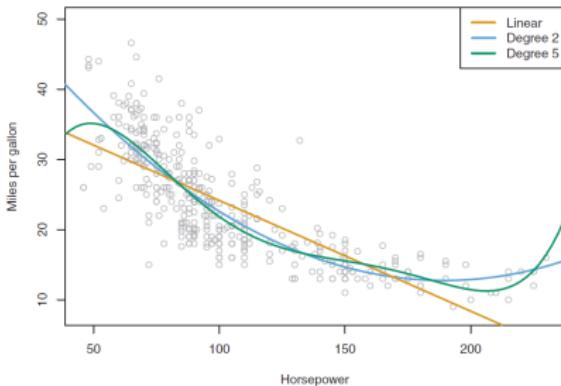
Bias-Variance Tradeoff

- It is hard to reduce both bias and variance.
- As the model flexibility increase, bias decreases but variance increases. It is important to find a right trade-off.
- Bias-variance tradeoff is one of the most important theme in ML (and other fields!).
- In real problems, the true pattern f is unknown and the dataset size is limited. How can we efficiently measure the expected test MSE?



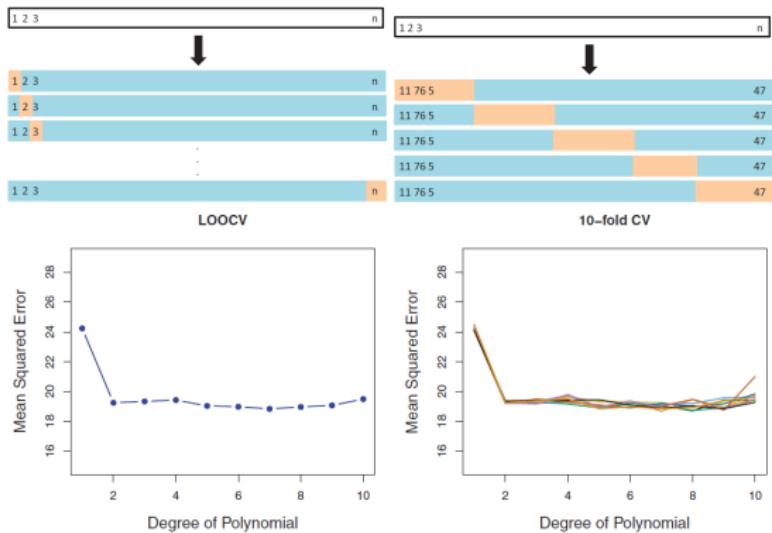
Cross-Validation(CV): Validation Set (Hold-out set)

- 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) and over-estimated than the true test error rate as the model is trained on fewer observations.
- Training set is further divided into **training** and **validation** sets. Validation set is used for model selection and hyper-parameter tuning.



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

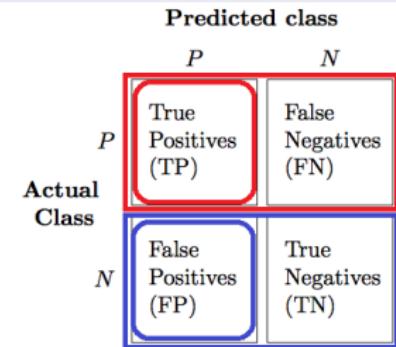
- LOOCV: train model with one sample left out and measure the error on the sample. Error is close to the true test rate but computation is heavy (train n times).
- k -fold CV: divide the samples into k (typically 5 or 10) folds. Train model on $k - 1$ **training** folds and measure error on the remaining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



Evaluation Metrics

Confusion Matrix

Credit card Default		Predicted		
		P*	N*	Total
Actual	P	40	40	80
	N	10	910	920
Total		50	950	1000



- Accuracy (ACC) = $\frac{FP + FN}{ALL} = \frac{40 + 910}{1000} = 95\%$
- Error (ERR) = $1 - ACC = \frac{10 + 40}{1000} = 5\%$
- However, accuracy/error may be misleading!

Evaluation Metrics

- Precision (PRE) = $\frac{TP}{P^*} = \frac{TP}{TP + FP} = \frac{40}{50} = 80\%$
- Recall (REC) = $\frac{TP}{P} = \frac{TP}{TP + FN} = \frac{40}{80} = 50\%$
- F1-Score (F1) = $\frac{2 \text{ PRE} \times \text{ REC}}{\text{ PRE} + \text{ REC}} = 61.5\%$

The harmonic average of PRE and REC to ensure
 $0 \leq F1 \leq 1$:

$$\frac{2}{F1} = \frac{1}{\text{PRE}} + \frac{1}{\text{REC}}$$

A widely used accuracy for binary classification with imbalanced data.

- True Positive Rate (TPR=REC) = $\frac{TP}{P} = 50\%$
- False Positive Rate (FPR) = $\frac{FP}{N} = \frac{10}{920} = 1.1\%$

	P*	N*
P	40	40
N	10	910

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

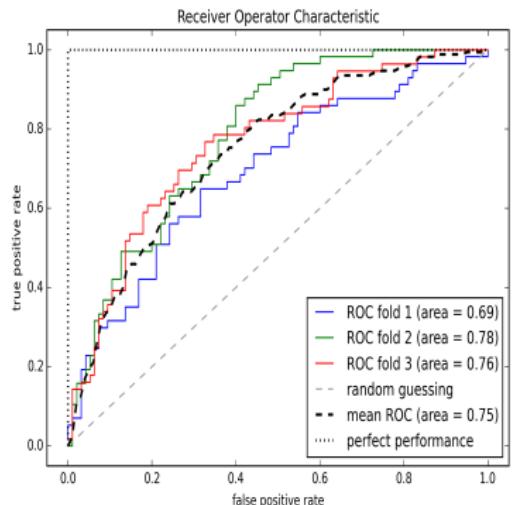
Receiver Operator Characteristic (ROC) Curve

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

		Predicted class	
		P	N
Actual Class	P	True Positive (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

→

- 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 (T-shaped lines): ROC AUC = 1.



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 —→ no clear way to measure performance

K-mean Clustering

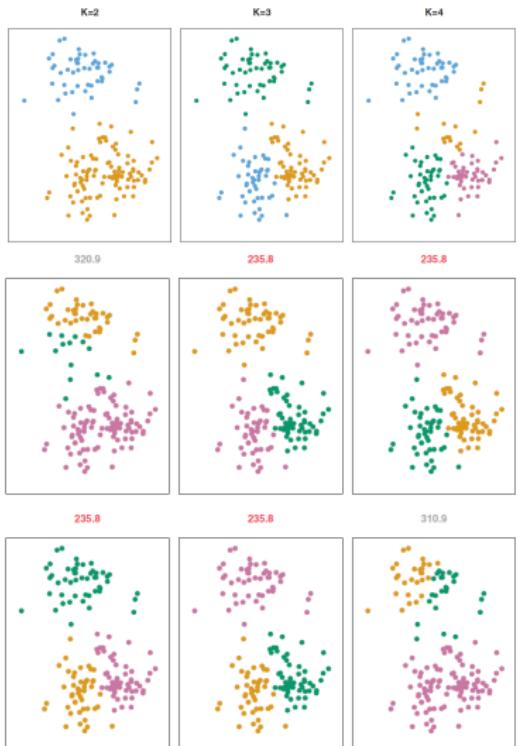
- Assign each observation X_{i*} in to (predetermined) K clusters
- Find μ_k and the clusters C_k which minimize the L2 norm to μ_k ,

$$\text{SSE} = \sum_k \sum_{i \in C_k} |X_{i*} - \mu_k|^2,$$

where μ_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.

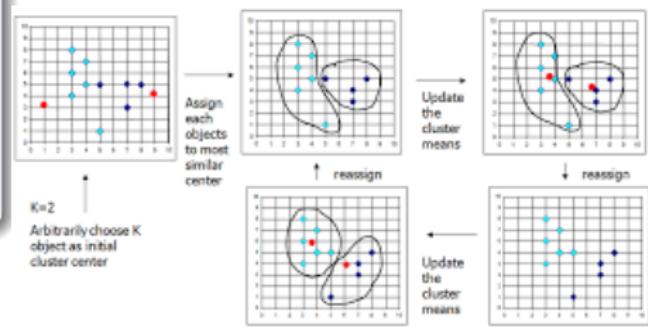
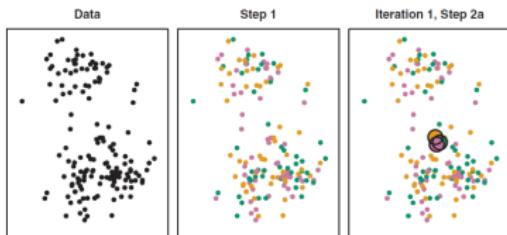


K-mean Clustering Iteration

- Given μ_k , we assign X_{i*} to the nearest center.
- Given the cluster C_k , the center should be the mean,
$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} X_{i*}.$$

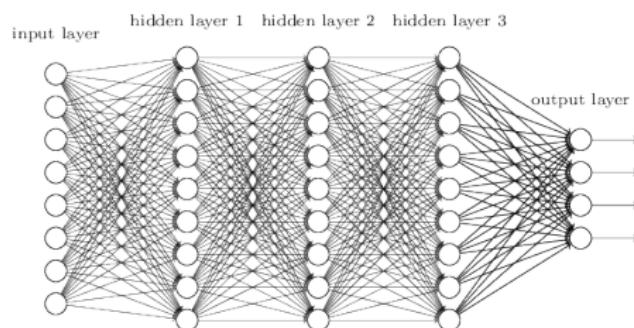
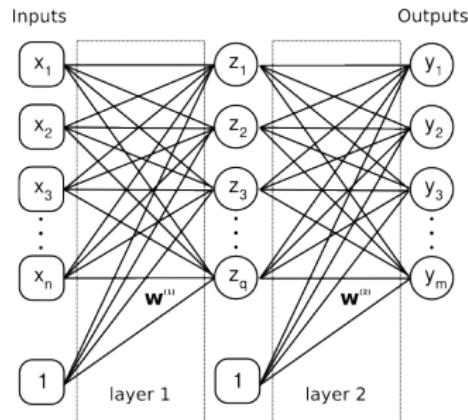
Iteration Step

- 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.



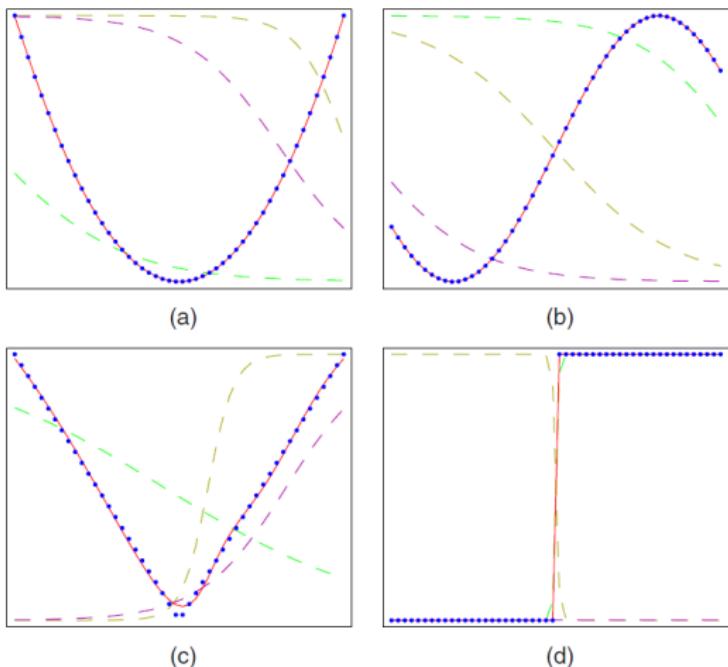
Neural Network

- 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

$$\mathbf{y} = \phi(\mathbf{xW}) \Rightarrow [\dots y_j \dots] = [\dots x_i \dots] \begin{bmatrix} \dots & \dots \\ \dots & W_{ij} & \dots \\ \dots & \dots \end{bmatrix} \quad (x_0 = 1)$$

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

Multi-layer: L layers

$$\mathbf{x}^{(1)} = \phi(\mathbf{a}^{(1)} = \mathbf{x}^{(0)}\mathbf{W}^{(1)}) \qquad \text{Input: } \mathbf{x} = \mathbf{x}^{(0)}, \text{ Output: } \mathbf{y} = \mathbf{x}^{(L)}$$

$$\mathbf{x}^{(2)} = \phi(\mathbf{a}^{(2)} = \mathbf{x}^{(1)}\mathbf{W}^{(2)}) \qquad [\dots y_k \dots] = [\dots x_i \dots]$$

...

$$\mathbf{x}^{(L)} = \phi(\mathbf{a}^{(L)} = \mathbf{x}^{(L-1)}\mathbf{W}^{(L)}) \qquad \times \begin{bmatrix} \dots \\ \dots & W_{ij}^{(1)} & \dots \\ \dots \end{bmatrix} \dots \begin{bmatrix} \dots \\ \dots \times W_{jk}^{(L)} & \dots \\ \dots \end{bmatrix}$$

$$\mathbf{y} = \phi(\dots \phi(\phi(\mathbf{x}^{(0)}\mathbf{W}^{(1)})\mathbf{W}^{(2)}) \dots)$$

Training NN

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

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

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

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

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

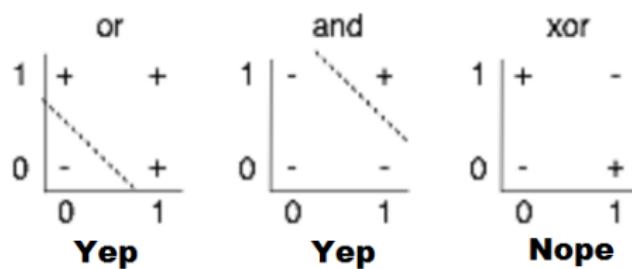
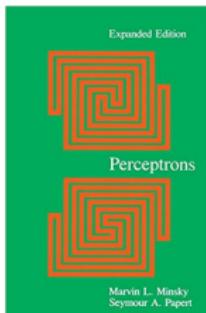
$$\frac{\partial J_n(\mathbf{W})}{\partial W_{ij}} = -(Y_{nj} - \phi(X_{nj})) X_{ni}$$

- Total loss function with regularization:

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

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 $y = f_3(f_2(f_1(x_0)))$:

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

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,
 $y = f_3(f_2(f_1(x_0w_1)w_2)w_3)$

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

$$\frac{\partial y}{\partial w_3} = f'_3(x_2w_3) \cdot x_2$$

$$\frac{\partial y}{\partial w_2} = f'_3(x_2w_3) \cdot w_3 f'_2(x_1w_2) \cdot x_1$$

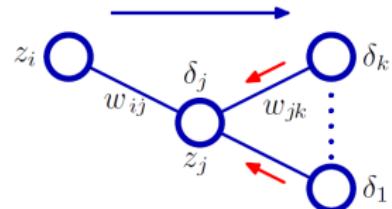
$$\frac{\partial y}{\partial w_1} = f'_3(x_2w_3) \cdot w_3 f'_2(x_1w_2) \cdot w_2 f'_1(x_0w_1) \cdot x_0$$

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

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

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



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,

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

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

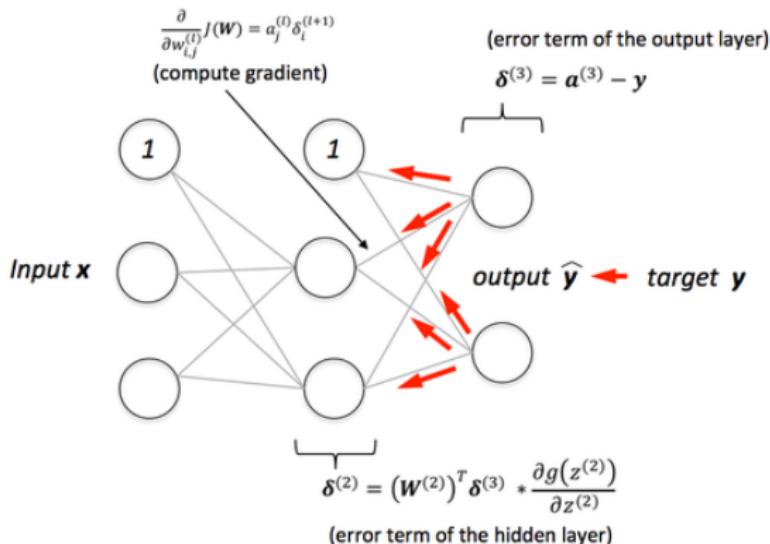
$$\text{v.s. Forward-propagation: } \mathbf{x}^{(m+1)} = \phi(\mathbf{a}^{(m)} = \mathbf{x}^{(m)} \mathbf{W}^{(m)})$$

where $*$ is the element-wise multiplication.

Backpropagation: error term δ (cont)

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

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



Backpropagation: Summary and benefit

Procedure

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

Benefits

Numerical differentiation:

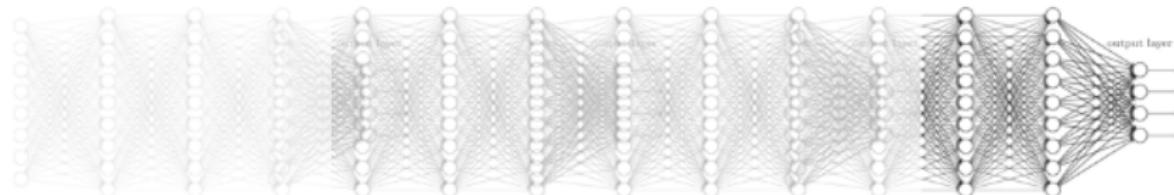
- $\frac{\partial J}{\partial W_{ij}} = \frac{J(W_{ij} + \varepsilon) - J(W_{ij} - \varepsilon)}{2\varepsilon}$
- require $O(N_w^2)$ operations for the total number of weights N_w .
- can be used to validate the backpropagation

Back-propagation:

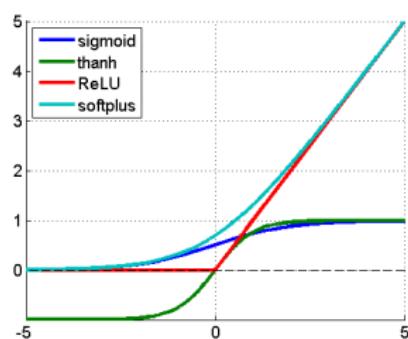
- require $O(N_w)$ operations.

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

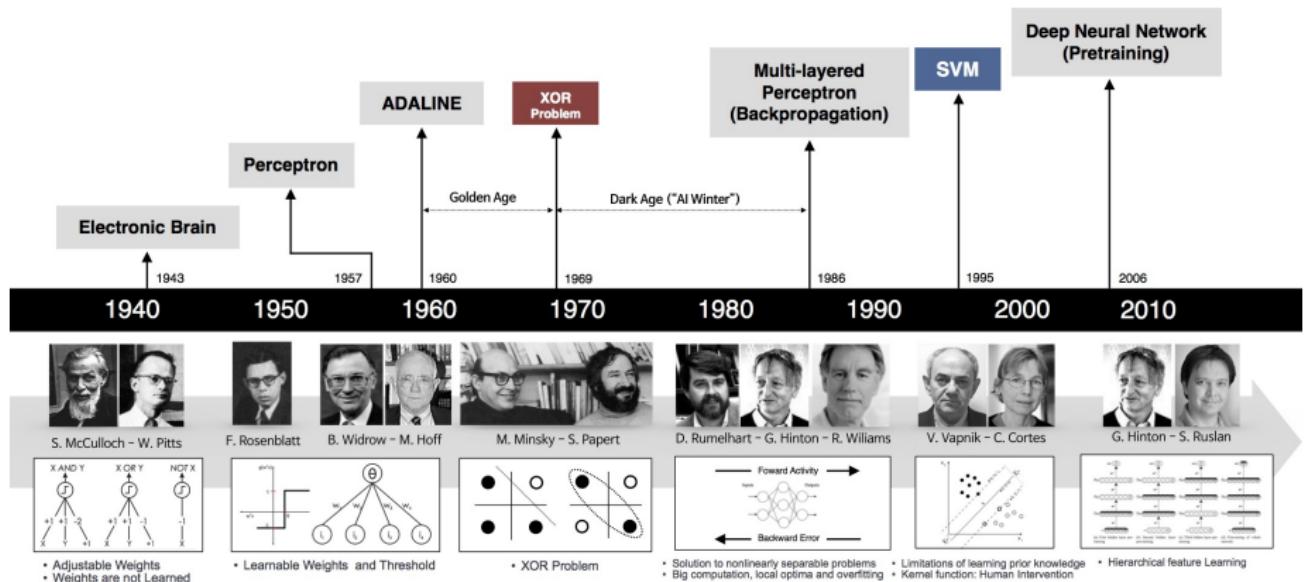
ReLU (Rectified linear unit)
 $\max(0, x)$

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

ReLU or Softplus activations have non-vanishing gradient.

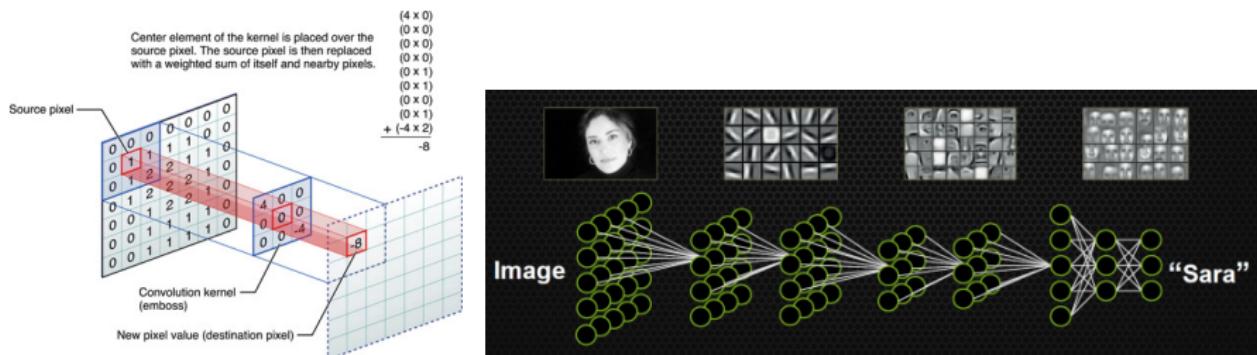
Timeline of NN

From Andrew Beam's Deep Learning 101



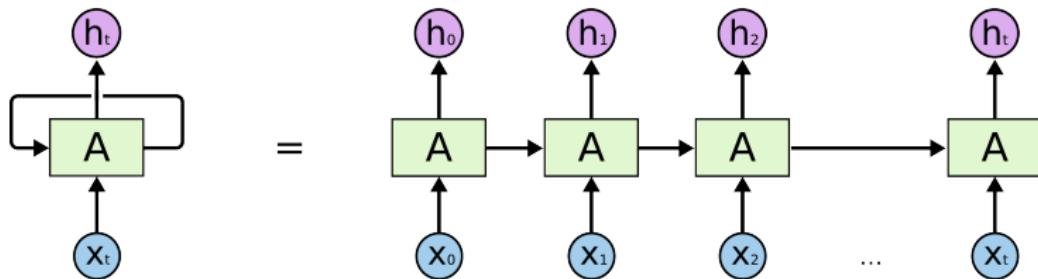
Convolutional Neural Network (CNN / ConvNet)

- 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 \mathbf{a} , we generalize the logistic function:

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

- 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.

