CS 412 Intro. to Data Mining

Chapter 9. Classification: Advanced Methods

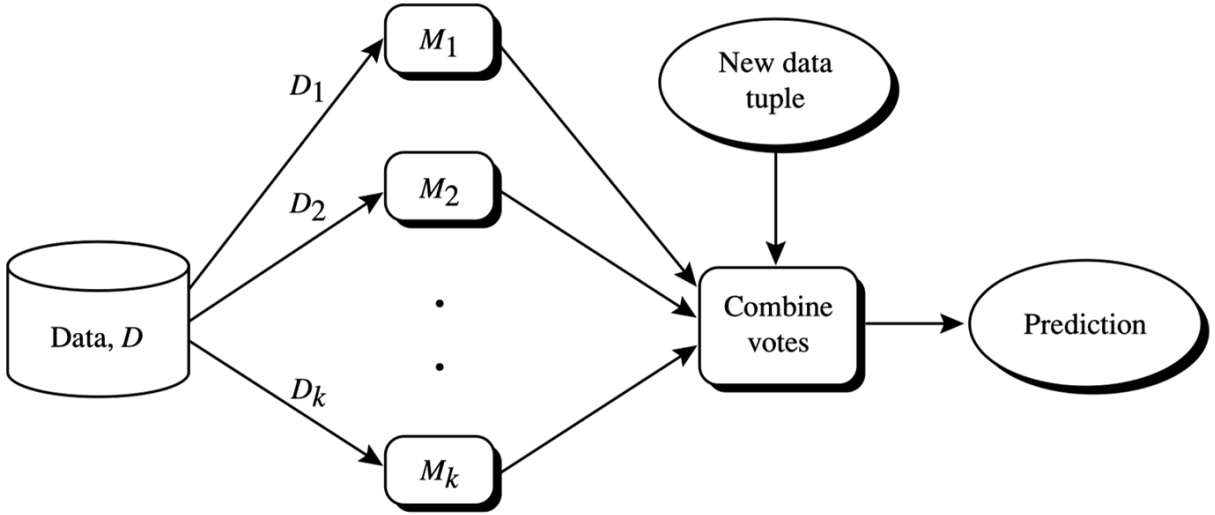
Qi Li, Computer Science, Univ. Illinois at Urbana-Champaign, 2018

Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

Ensemble Methods: Increasing the Accuracy

- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of *k* learned models, M_1, M_2, …, M_k, with the aim of creating an **improved** model M*



Ensemble Methods: Increasing the Accuracy

- What are the requirements to generate an improved model?
  - Example: majority voting

Base model performance →

Ensemble performance →

	x_1	x_2	x_3
M_1	✓	✓	✗
M_2	✗	✓	✓
M_3	✓	✗	✓
Voting Ensemble	✓	✓	✓

Case 1:
Ensemble has positive effect

	x_1	x_2	x_3
M_1	✓	✓	✗
M_2	✓	✓	✗
M_3	✓	✓	✗
Voting Ensemble	✓	✓	✗

Case 2:
Ensemble has no effect

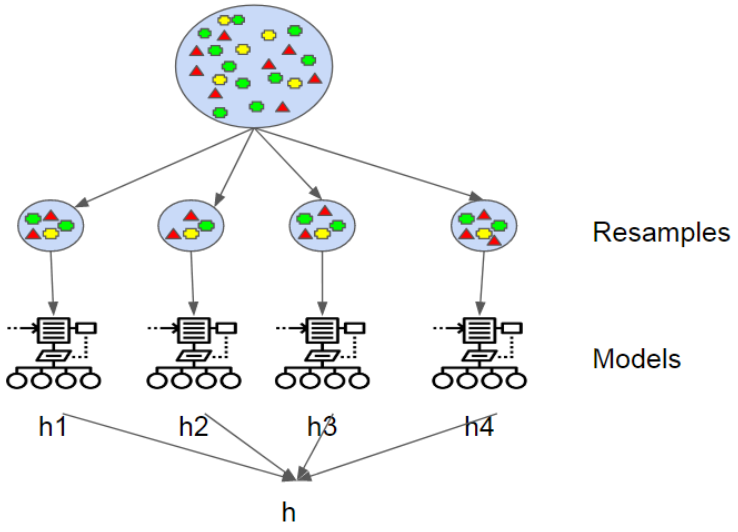
	x_1	x_2	x_3
M_1	✓	✗	✗
M_2	✗	✓	✗
M_3	✗	✗	✓
Voting Ensemble	✗	✗	✗

Case 3:
Ensemble has negative effect

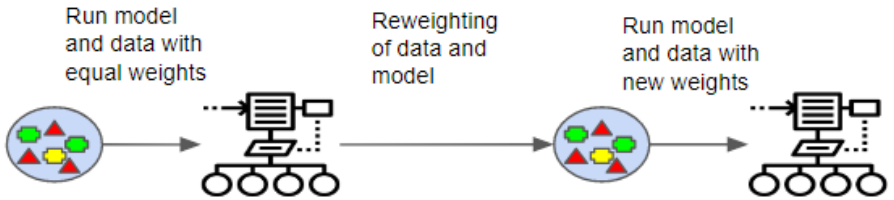
- Base models should be
  - Accurate
  - Diverse

Ensemble Methods: Increasing the Accuracy

- Popular ensemble methods
  - Bagging: Trains each model using a subset of the training set, and models learned in parallel
  - Boosting: Trains each new model instance to emphasize the training instances that previous models mis-classified, and models learned in order



Bagging



Boosting

Bagging: Bootstrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote

- Training
  - For i = 1 to k
    - create bootstrap sample, D_i, by sampling D with replacement;
    - use D_i and the learning scheme to derive a model, M_i ;

- Classification: classify an unknown sample **X**
  - let each of the k models classify X and return the majority vote

- Prediction:
  - To predict continuous variables, use average prediction instead of vote

Random Forest: Basic Concepts

- Random Forest (first proposed by L. Breiman in 2001)
  - Bagging with **decision trees** as base models
  - *Data bagging*
    - Use a **subset of training data** by sampling with replacement for each tree
  - *Feature bagging* ← Advantage of decision trees – more diversity
    - At each node use a **random selection of attributes** as candidates and split by the best attribute among them
  - During classification, each tree votes and the most popular class is returned

Random Forest

- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than typical bagging or boosting

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - A series of k classifiers are iteratively learned
  - After a classifier M_i is learned, set the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction

Adaboost (Freund and Schapire, 1997)

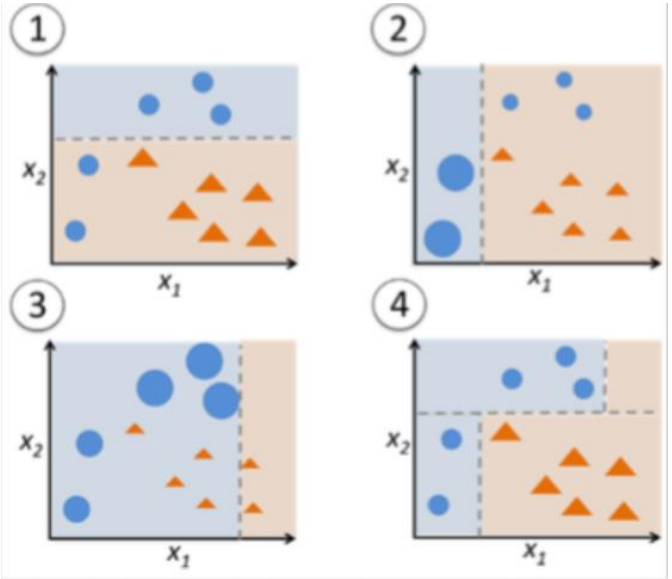
1. Assign initial weights to each training tuple

3. Update weights based on current model

$\{w_n^{(1)}\}$ $\{w_n^{(2)}\}$ … $\{w_n^{(k)}\}$

M_1 M_2 … M_k

2. Train base classifier on weighted dataset



Two 'weighting' strategy:

1. Assign weights to each training example
2. Sample dataset based on weight distribution

$$M^*(x) = \text{sign}\left(\sum_{i=1}^{k} \alpha_i M_i(x)\right)$$

4. After base classifiers are trained, they are combined to give the final classifier

Adaboost (Freund and Schapire, 1997)

- **Input**: Training set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- Initialize all weights $\{w_n^{(1)}\}$ to 1/N
- For round i = 1 to k,
  - Fit a classifier M_i based on weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(i)} I(M_i(x_n) \neq y_n)$$

  - Evaluate error rate $\epsilon_i = J_m / \sum w_n^{(i)}$ (stop iteration if ϵ_i < threshold)

    and the base model M_i's vote $\alpha_i = \frac{1}{2} \ln\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$

  - Update weights

$$w_n^{(i+1)} = w_n^{(i)} \exp\{\alpha_i \cdot I(M_i(x_n) \neq y_n)\}$$

- The final model is given by voting based on $\{\alpha_n\}$

Gradient Boosting

- Operates on:
  - A differentiable loss function
  - A weak learner to make predictions (usually trees)
  - An additive model to add weak learners to minimize the loss function
- Each time adds an additional weak learner

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

Previous model

New weak learner

- Scalable implementation: XGBoost

Ensemble Methods Recap

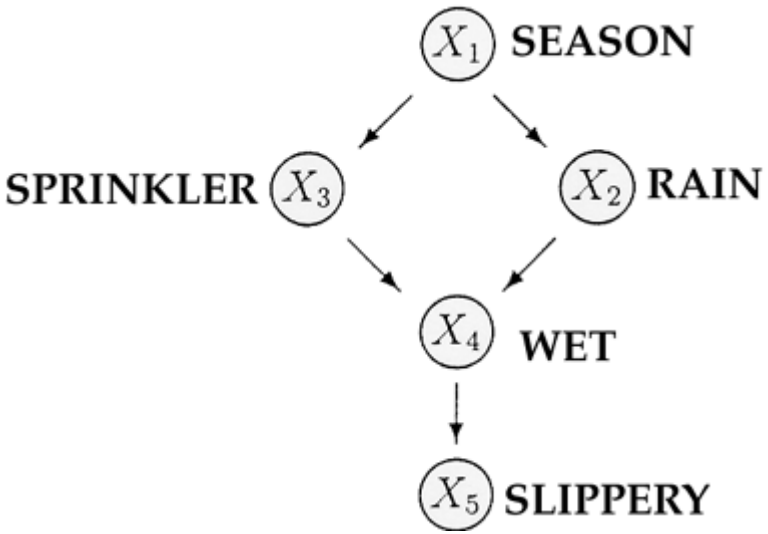
- Random forest and XGBoost are the most commonly used algorithms for tabular data
- Pros
  - Good performance for tabular data, requires no data scaling
  - Can scale to large datasets
  - Can handle missing data to some extent
- Cons
  - Can overfit to training data if not tuned properly
  - Lack of interpretability (compared to decision trees)

Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

From Naïve Bayes to Bayesian Networks

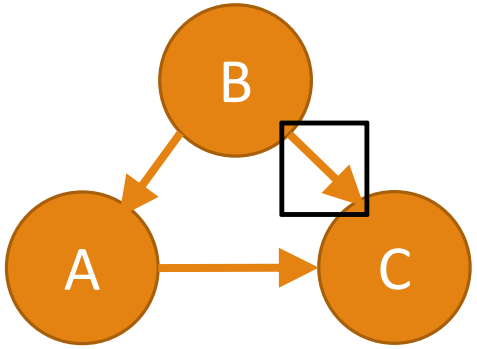
- Naïve Bayes classifiers assume that the value of a particular feature is **independent** of the value of any other feature, given the class variable
  - This assumption is often too simple to model the real world well
- Bayesian network (or Bayes network, belief network, Bayesian model or probabilistic directed acyclic graphical model) is a **probabilistic graphical model**
  - Represented by a set of *random variables* and *their conditional dependencies* via a *directed acyclic graph* (DAG)
  - E.g. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases



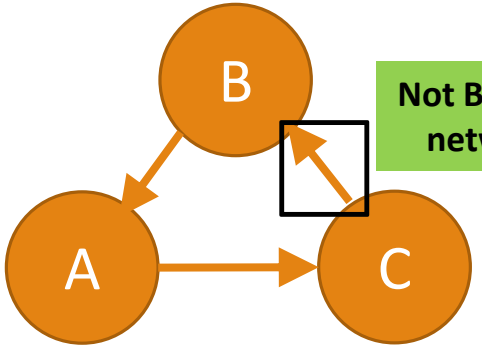
Bayesian Belief Networks

- **Bayesian belief network** (or **Bayesian network, probabilistic network**):
  - Allows *class conditional independencies* between *subsets* of variables
- Two components:
  - A *directed acyclic graph* (called a structure)
  - A set of *conditional probability tables* (CPTs)

Nodes: random variables Links: dependency



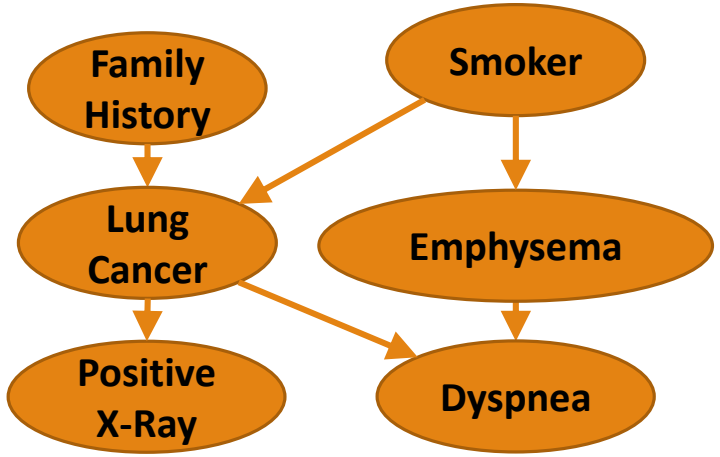
directed **acyclic** graphical model



Not Bayesian networks

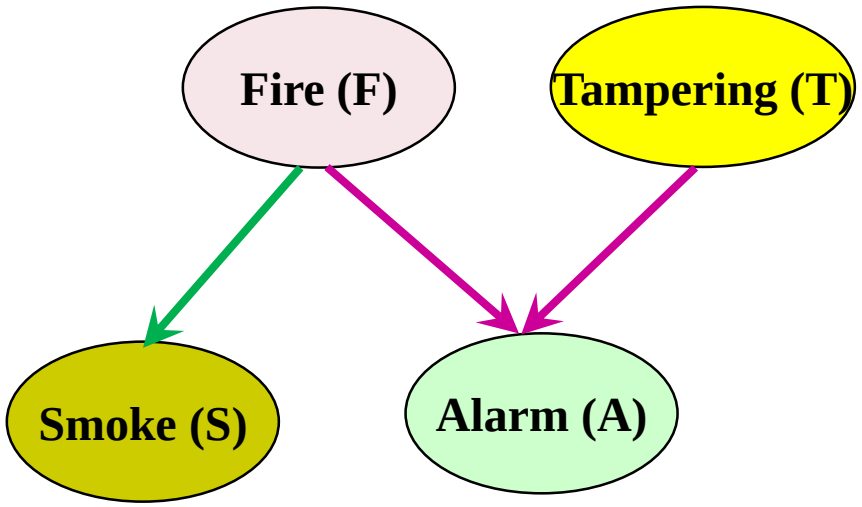
directed **cyclic** graphical model

$$p(A, B, C) = p(B) \cdot p(A|B) \cdot p(C|A, B)$$



	FH, S	*FH, ~S*	*~FH, S*	*~FH, ~S*
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

A Bayesian Network and Its CPTs



Conditional **P**robability **T**ables (CPT)

Fire	Smoke	$\Theta_{s\|f}$
True	True	.90
False	True	.01

Fire	Tampering	Alarm	$\Theta_{a\|f,t}$
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

CPT shows the conditional probability for each possible combination of its parents:

$$p(X) = \prod_k p(x_k | \text{Parents}(x_k))$$

$$p(F, S, A, T) = p(F) \cdot p(T) \cdot p(S|F) \cdot p(A|F, T)$$

Training Bayesian Networks: Several Scenarios

- Scenario 1: Given both the network structure and all variables observable: *compute only the CPT entries*

- Scenario 2: Network structure known, some variables hidden: *gradient descent* (greedy hill-climbing) method, i.e., search for a solution along the steepest descent of a criterion function
  - Weights are initialized to random probability values
  - At each iteration, it moves towards what appears to be the best solution at the moment, without backtracking
  - Weights are updated at each iteration & converge to local optimum

Training Bayesian Networks: Several Scenarios

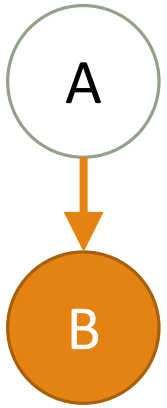
- Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*

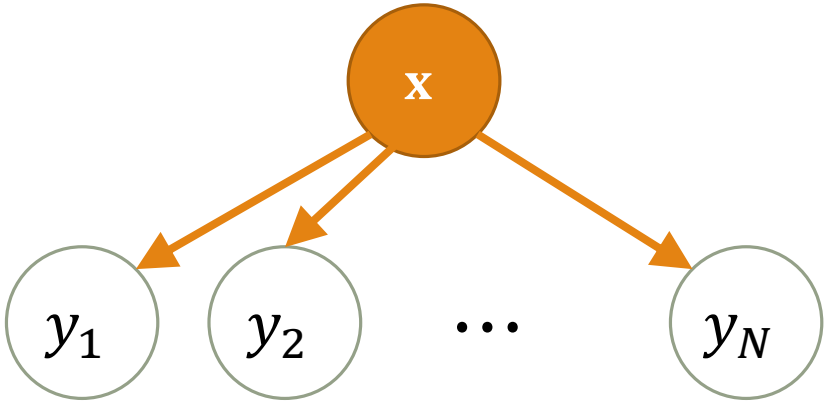
- Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose

- D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models,* M. Jordan, ed. MIT Press, 1999

Probabilistic Graphic Model: Plate Notations

- Represent variables that repeat in a graphical model
- Variables
  - A solid (or shaded) circle means the corresponding variable is *observed*; otherwise it is *hidden*
- Dependency among variables:
  - A Directed Acyclic Graphical (DAG) model
- Using plate notation instead of flat notation





Flat notation

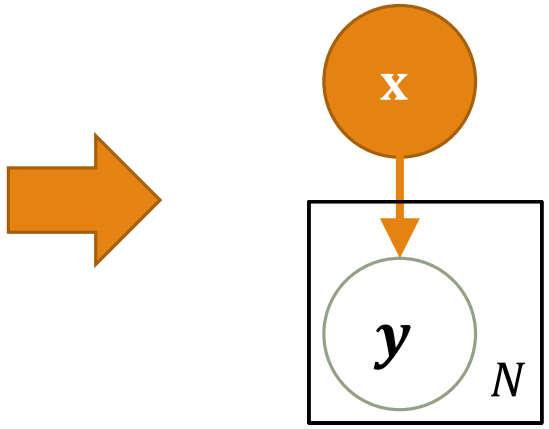


Plate notation

An Example of Plate Notation

Flat notation

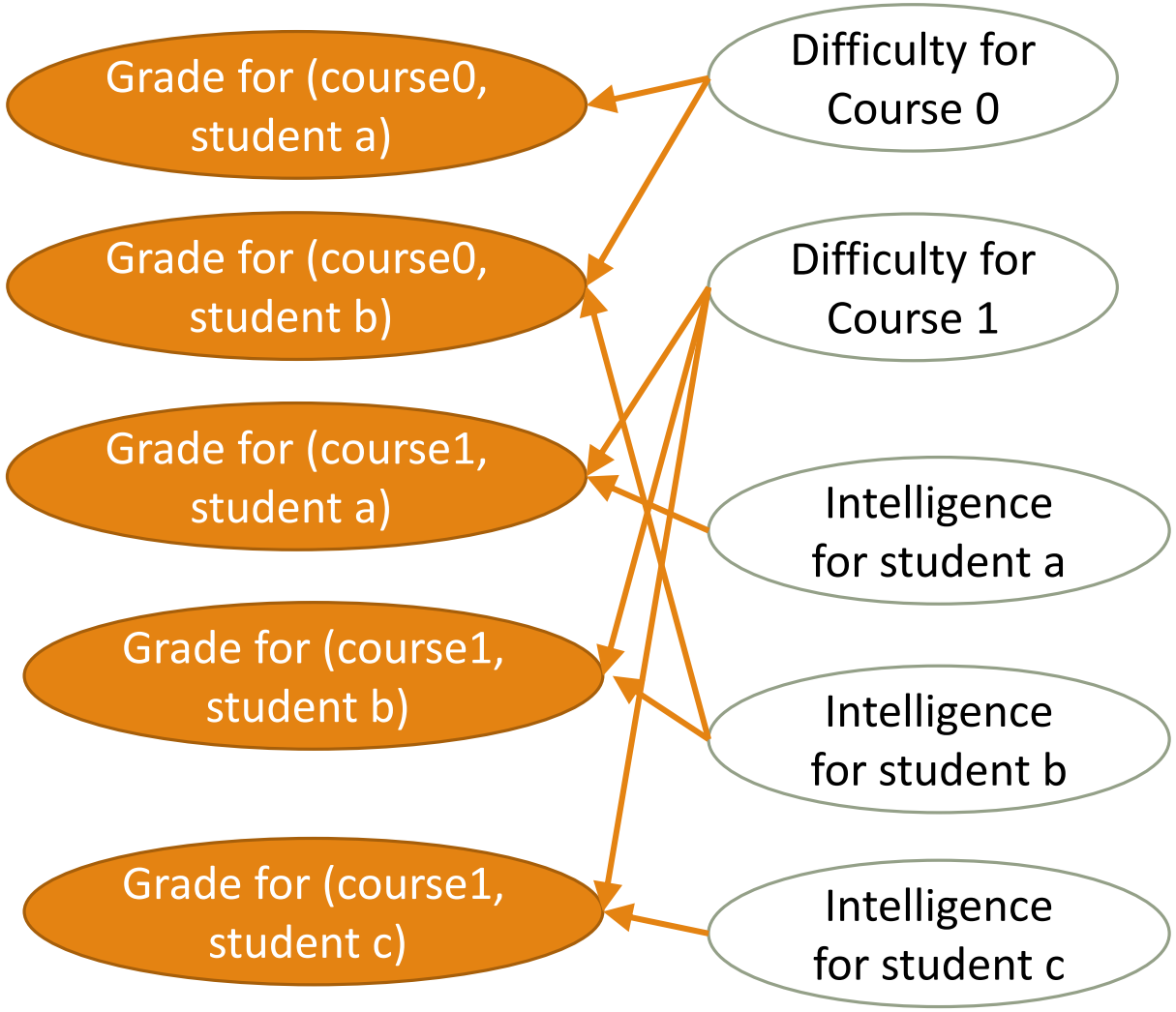
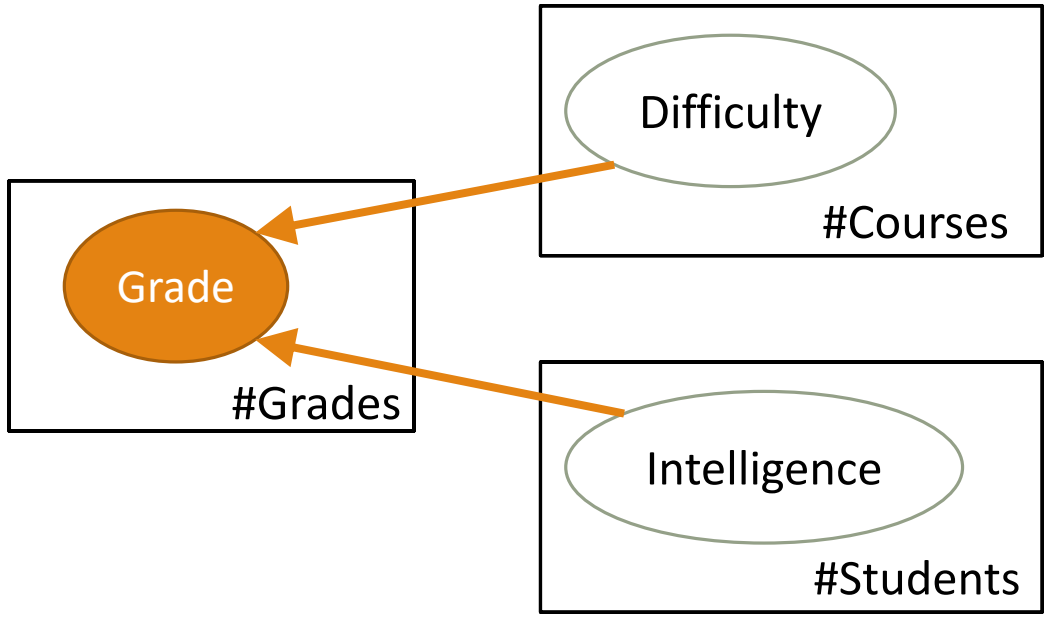


Plate notation



Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

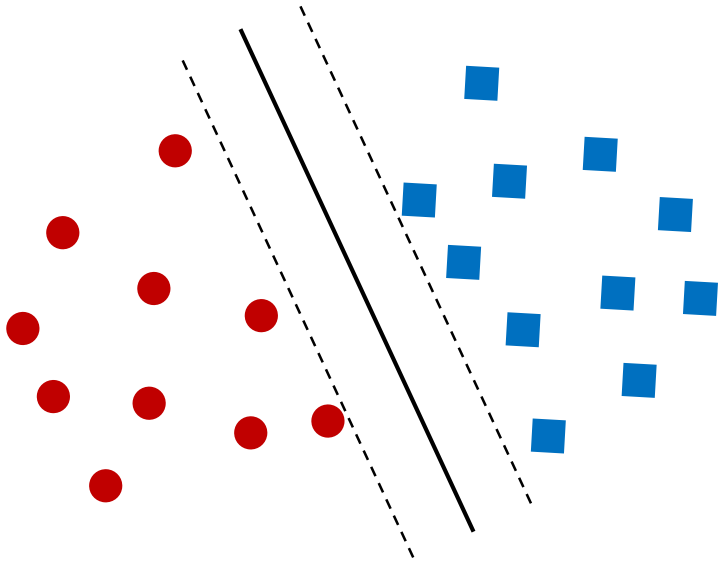
Classification: A Mathematical Mapping

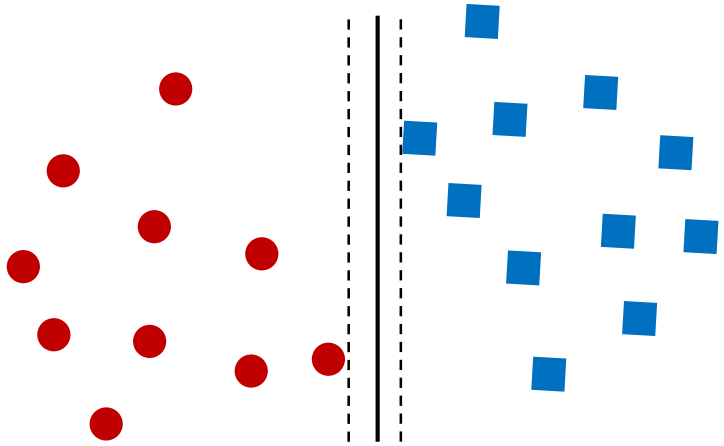
- **The binary classification problem:**
  - E.g., Movie review classification
    - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1 (positive, negative)
    - x_1 : # of word "awesome"
    - x_2 : # of word "disappointing"
- Mathematically, $\boldsymbol{x} \in X = \Re^n, y \in Y = \{+1, -1\}$
  - We want to derive a function $f\colon X \rightarrow Y$
  - which maps input examples to their correct labels

SVM—Support Vector Machines

- <u>linear and nonlinear</u>
  - Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- It uses a <u>nonlinear mapping</u> to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., "decision boundary")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** ("essential" training tuples) and **margins** (defined by the support vectors)

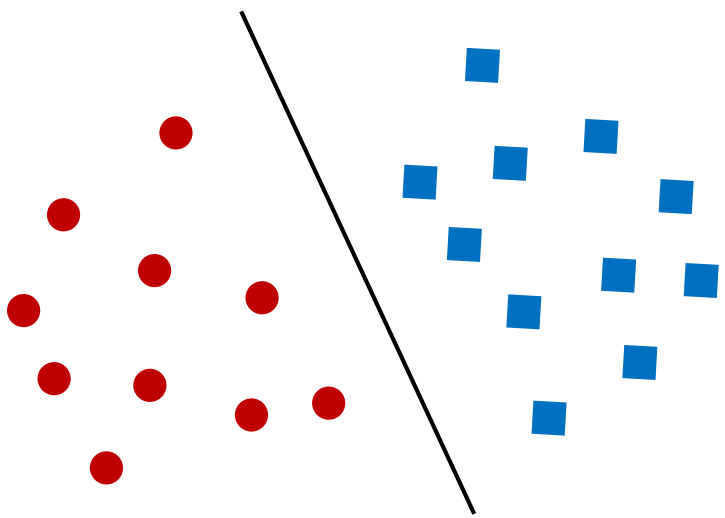
SVM—General Philosophy



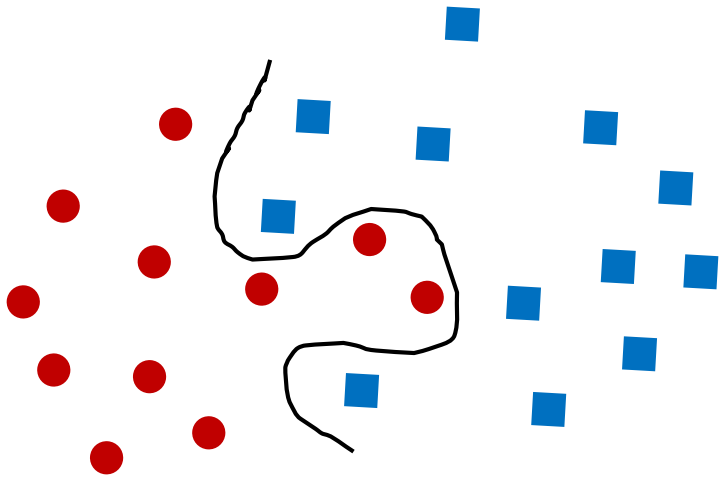


- Learning a max-margin classifier
  - From the infinite set of lines (hyperplanes) separating two classes
  - Find the one which separates two classes with the **largest margin**
  - i.e. a **maximum marginal hyperplane (MMH)**

SVM—When Data Is Linearly Separable



Linearly Separable



Linearly Inseparable

- The simplest case: When data is **linearly separable**
  - Data sets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable

Linear SVM for Linearly Separable Data

- A separating hyperplane can be written as

$$\boldsymbol{w}^T \boldsymbol{x} + b = 0$$

where $w = (w_1, w_2, \dots, w_n)^T$ is a weight vector and b a scalar (bias)

Model parameters to learn

  - For 2-D, it can be written as: $w_1 x_1 + w_2 x_2 + b = 0$
- The hyperplane defining the sides of the margin:

  H_1: $w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

  H_2: $w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**

Linear SVM for Linearly Separable Data

- The distance from any data point $\boldsymbol{x}$ to the separating hyperplane is

$$\text{distance}(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \qquad r = \frac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}$$

- Our objective is to maximize the distance of the closest data point to the hyperplane

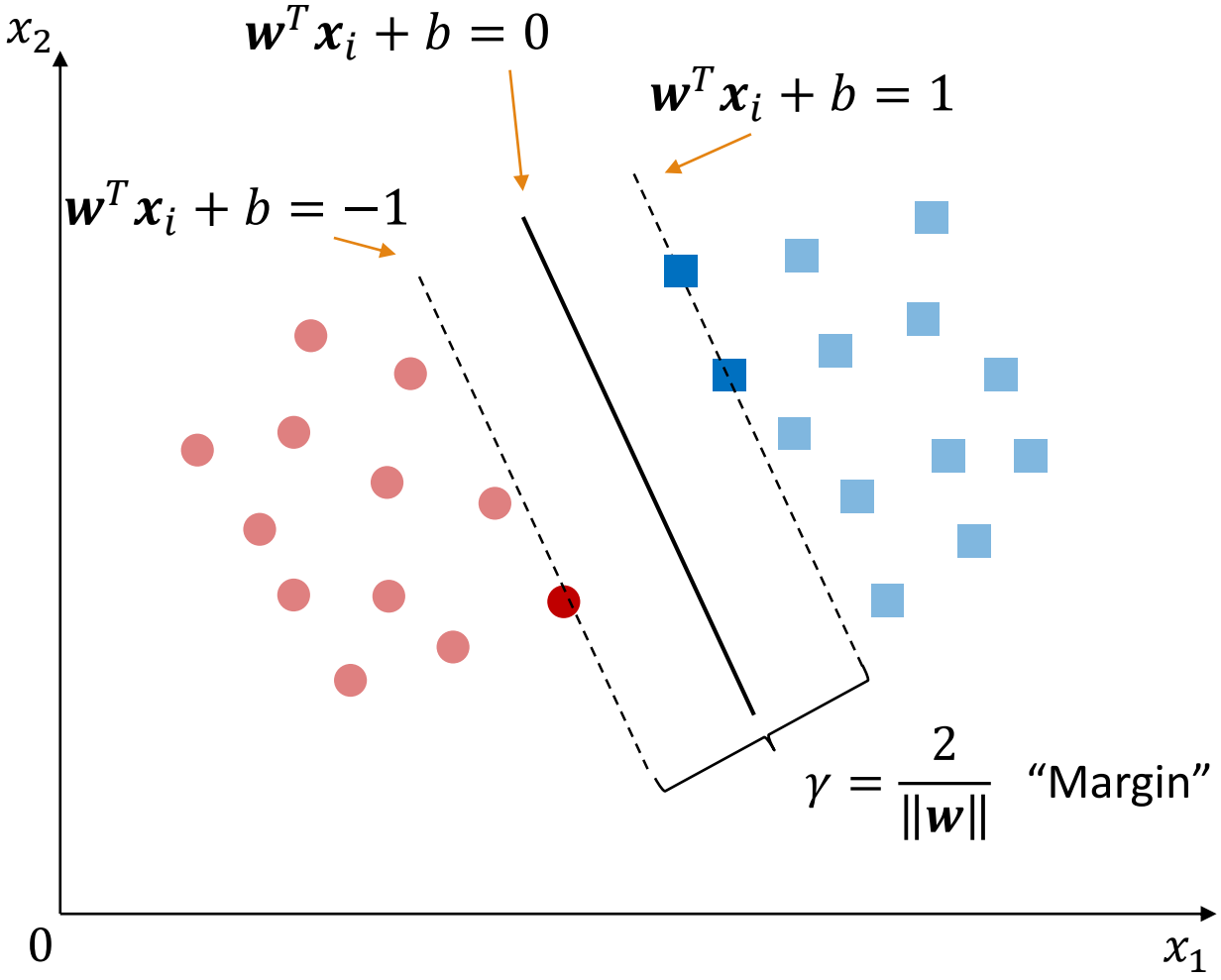
$$\arg\max_{w,b} \left\{ \frac{1}{\|\boldsymbol{w}\|} \min[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)] \right\}$$

- This is hard to solve, we shall convert it to an easier problem

$$\boxed{\begin{aligned} \arg\min_{w,b} \quad & \|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \qquad i = 1,2,\ldots,n \end{aligned}}$$

- This is the basic form of SVM, and it can be solved by using *quadratic programming*

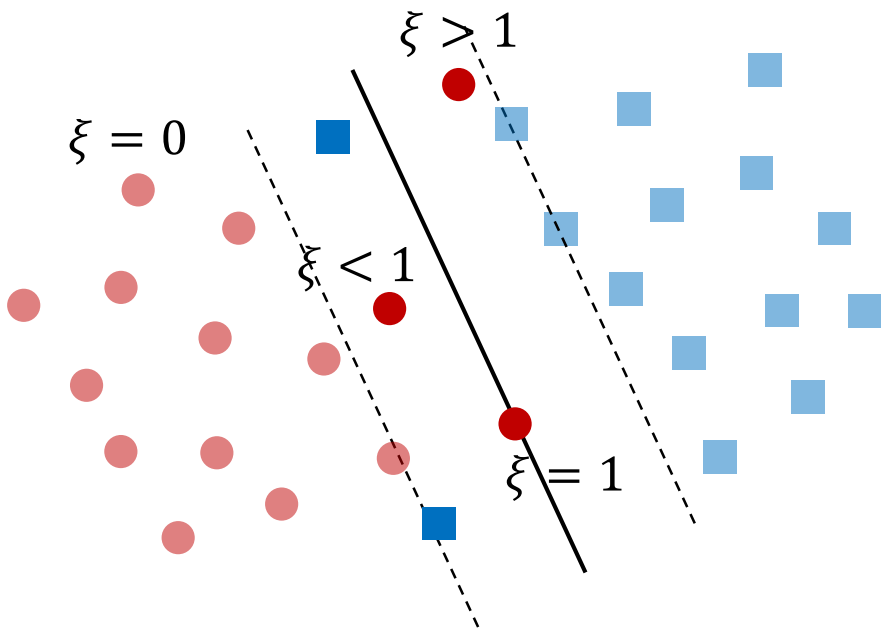
Linear SVM for Linearly Separable Data



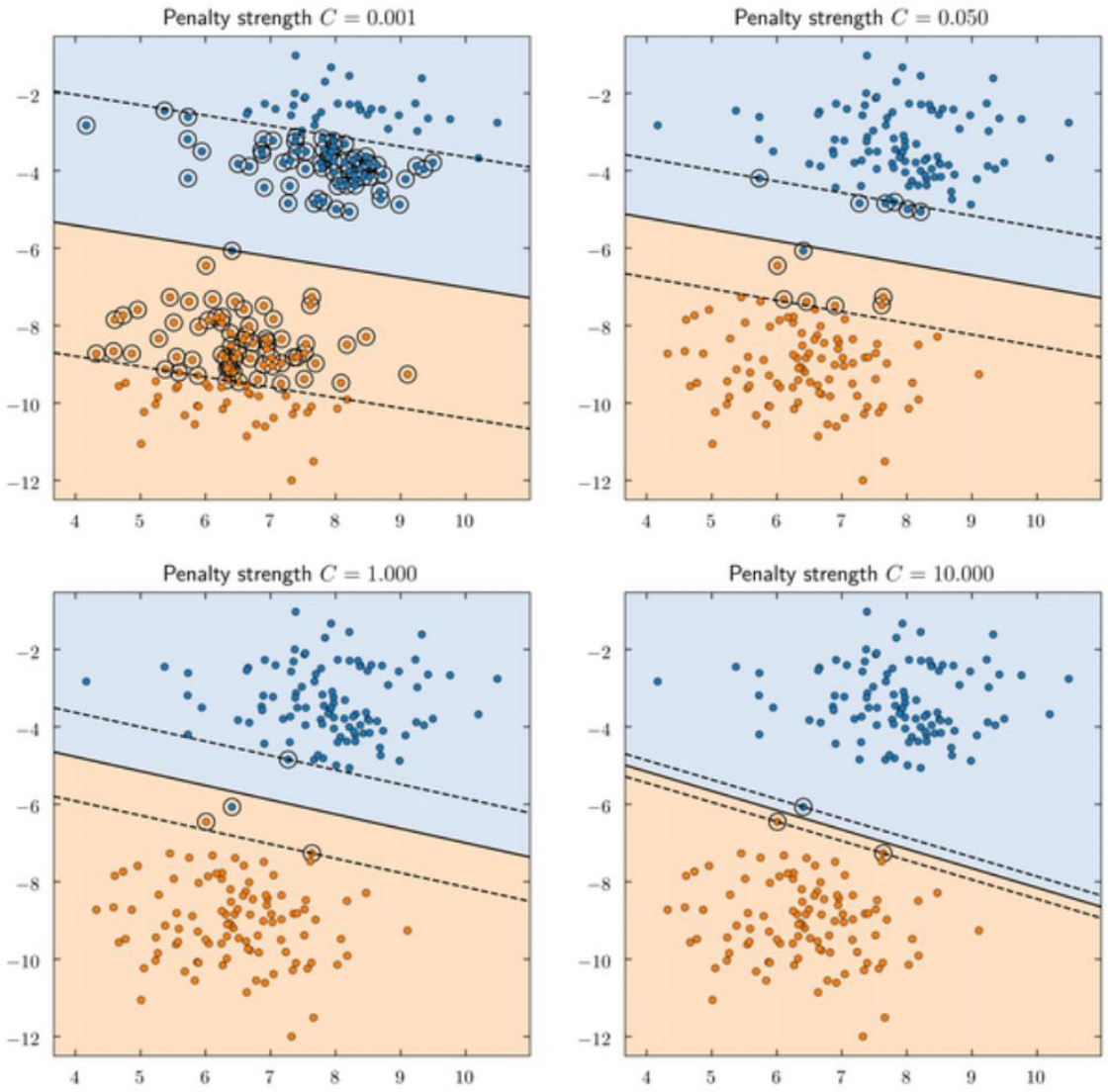
- The data points closest to the separating hyperplane are called **support vectors**

SVM for Linearly Inseparable Data

- We allow data points to be on the "wrong side" of the **margin boundary**
- Penalize points on the wrong side according to its distance to the margin boundary
- ξ : slack variable
- C (> 0): Controls the trade-off between the penalty and the margin
- Smaller C: allow more mistake
- Larger C: allow less mistake
- This is the widely used *soft-margin SVM*



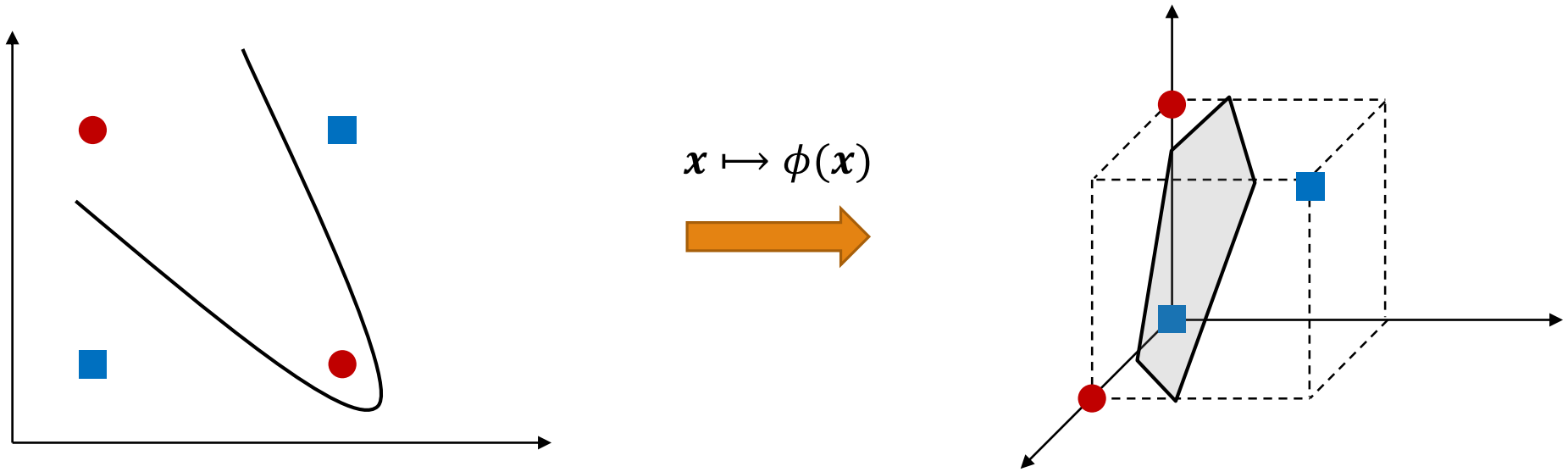
Effect of slack variable



https://www.quora.com/What-is-the-purpose-for-using-slack-variable-in-SVM

SVM for Linearly Inseparable Data

- Alternatively, for linearly inseparable data, we can map them to a higher dimensional space
- We search for a linear separating hyperplane in the new space
  - Example: The XOR problem



Kernel Functions for Nonlinear Classification

- Instead of computing the dot product on the transformed data, it is mathematically equivalent to applying a kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to the original data, i.e.,
  - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)\, \phi(\boldsymbol{x}_j)$
- Typical Kernel Functions

$$\text{Polynomial kernel of degree } h: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = (\boldsymbol{X_i} \cdot \boldsymbol{X_j} + 1)^h$$

$$\text{Gaussian radial basis function kernel}: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = e^{-\|\boldsymbol{X_i} - \boldsymbol{X_j}\|^2 / 2\sigma^2}$$

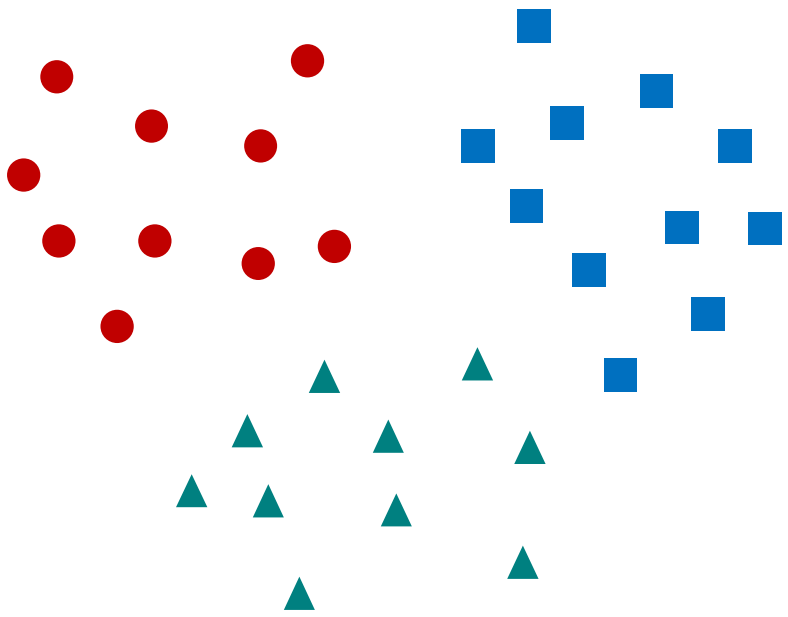
$$\text{Sigmoid kernel}: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \tanh(\kappa \boldsymbol{X_i} \cdot \boldsymbol{X_j} - \delta)$$

- SVMs can efficiently perform a non-linear classification using kernel functions, implicitly mapping their inputs into high-dimensional feature spaces

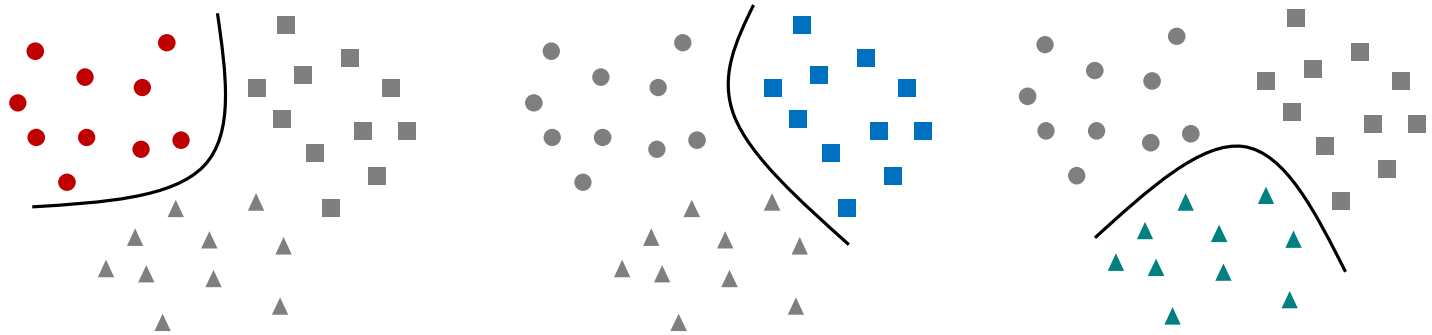
https://www.youtube.com/watch?time_continue=42&v=3liCbRZPrZA

http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/

Multi-class Classification with SVM

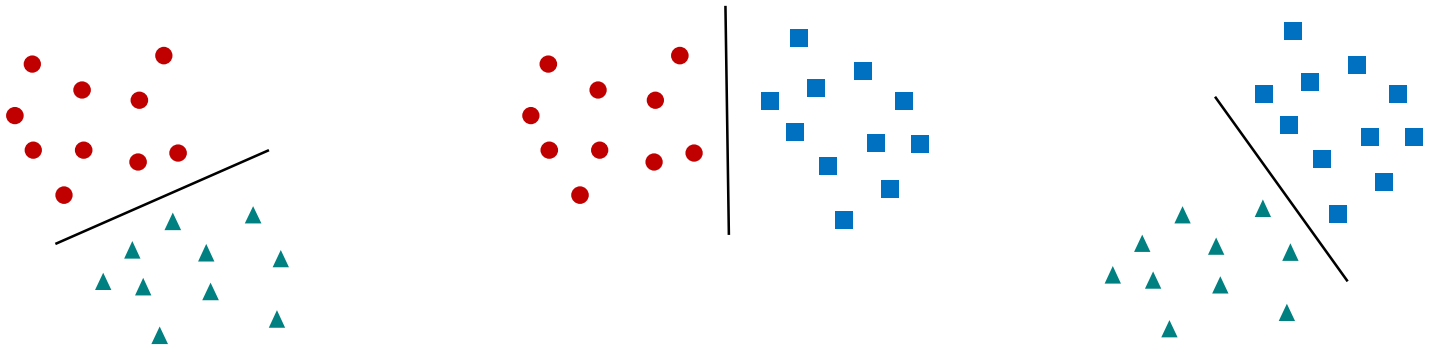


N classes



One-vs-Rest

Requires N classifiers



One-vs-One

Requires N(N - 1)/2 classifiers

Is SVM Scalable on Massive Data?

- SVM is effective on high dimensional data
  - The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
  - The **support vectors** are the essential or critical training examples—they lie closest to the decision boundary (MMH)
  - Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high
- SVM is not scalable to the # of data objects in terms of training time and memory usage
  - Scaling SVM by a hierarchical micro-clustering approach
    - H. Yu, J. Yang, and J. Han, "Classifying Large Data Sets Using SVM with Hierarchical Clusters", KDD'03

SVM: Applications

- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
  - SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

SVM Recap

- Pros
  - Elegant mathematical formulation, guaranteed global optimal with optimization
  - Trains well on small data sets
  - Flexibility through kernel functions
  - Conformity with semi-supervised training
- Cons
  - Not naturally scalable to large data sets

SVM Related Links

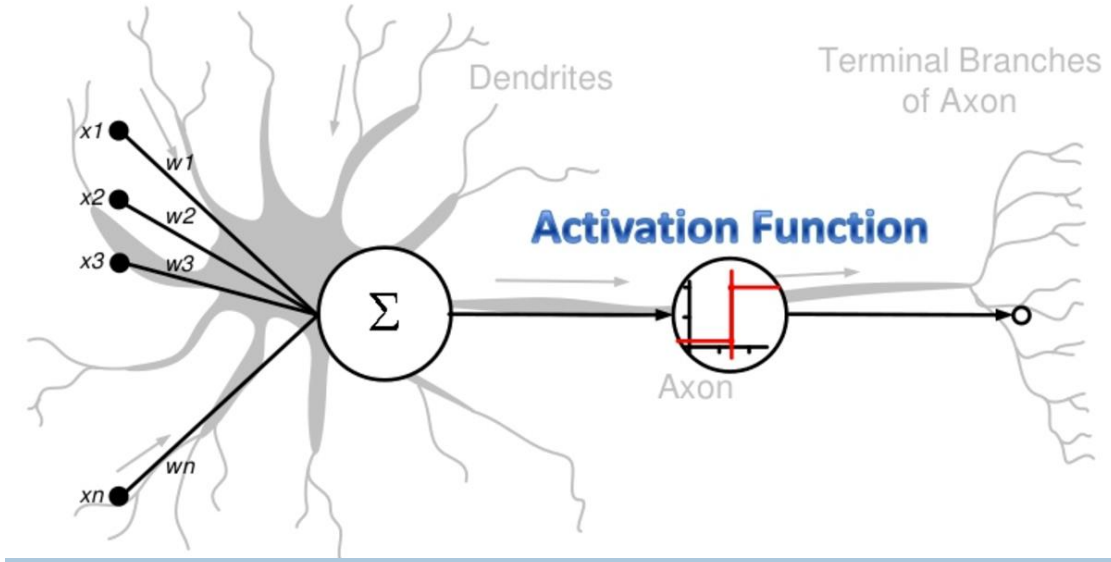
- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C

Chapter 9. Classification: Advanced Methods

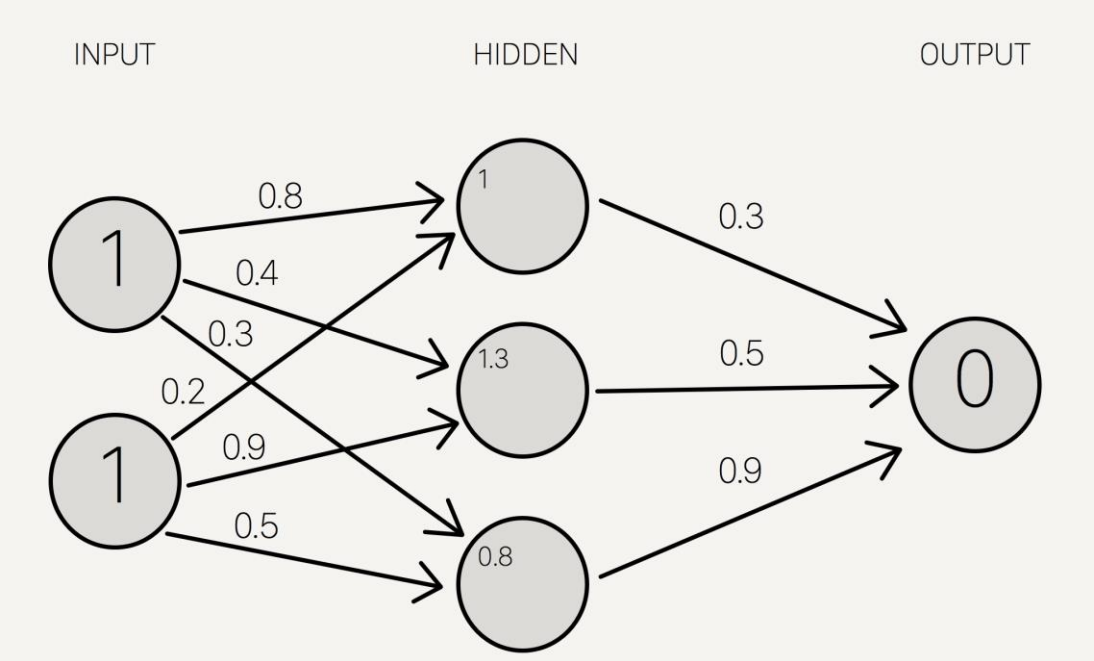
- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

Neural Network for Classification

- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
  - During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples



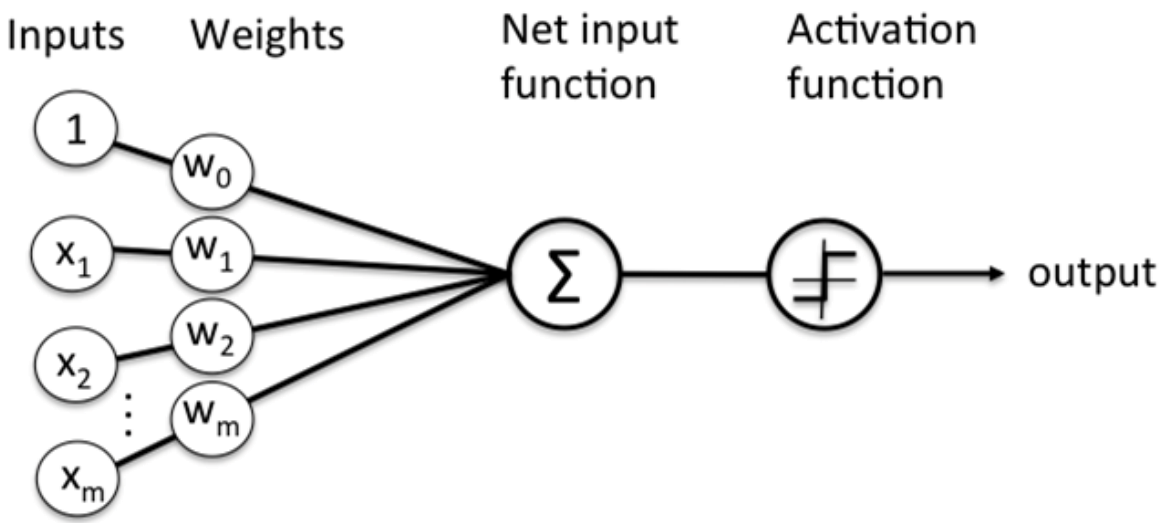
Artificial Neural Networks as an analogy of Biological Neural Networks



Learning by adjusting weights

(https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/)

Perceptron: Predecessor of a Neural Network



$$\hat{y} = f(\boldsymbol{W}^T \boldsymbol{x}) = f(\sum W_i x_i + b)$$

Activation Function
Adding non-linearity to the model

Weights

Bias
A measure of how easy it is to get the perceptron to output a 1

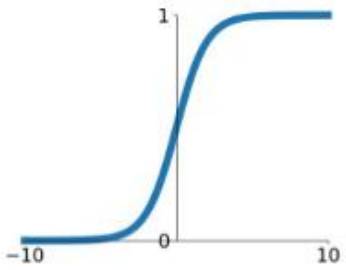
- Computes a weighted sum of inputs
- 1957 by Frank Rosenblatt - doesn't have a non-linear activation function

Perceptron: Predecessor of a Neural Network

- Examples of activation functions

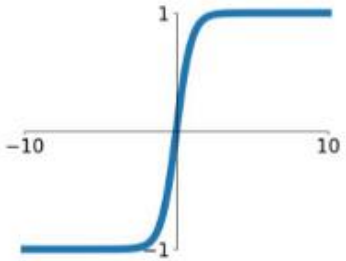
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



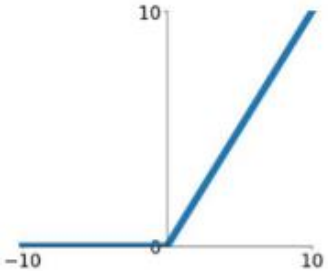
tanh

$$\tanh(x)$$



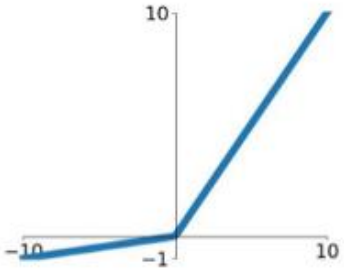
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

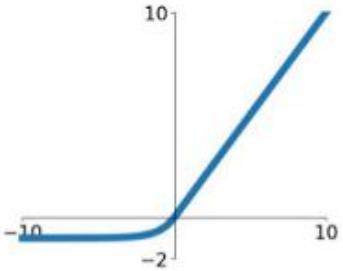


Maxout

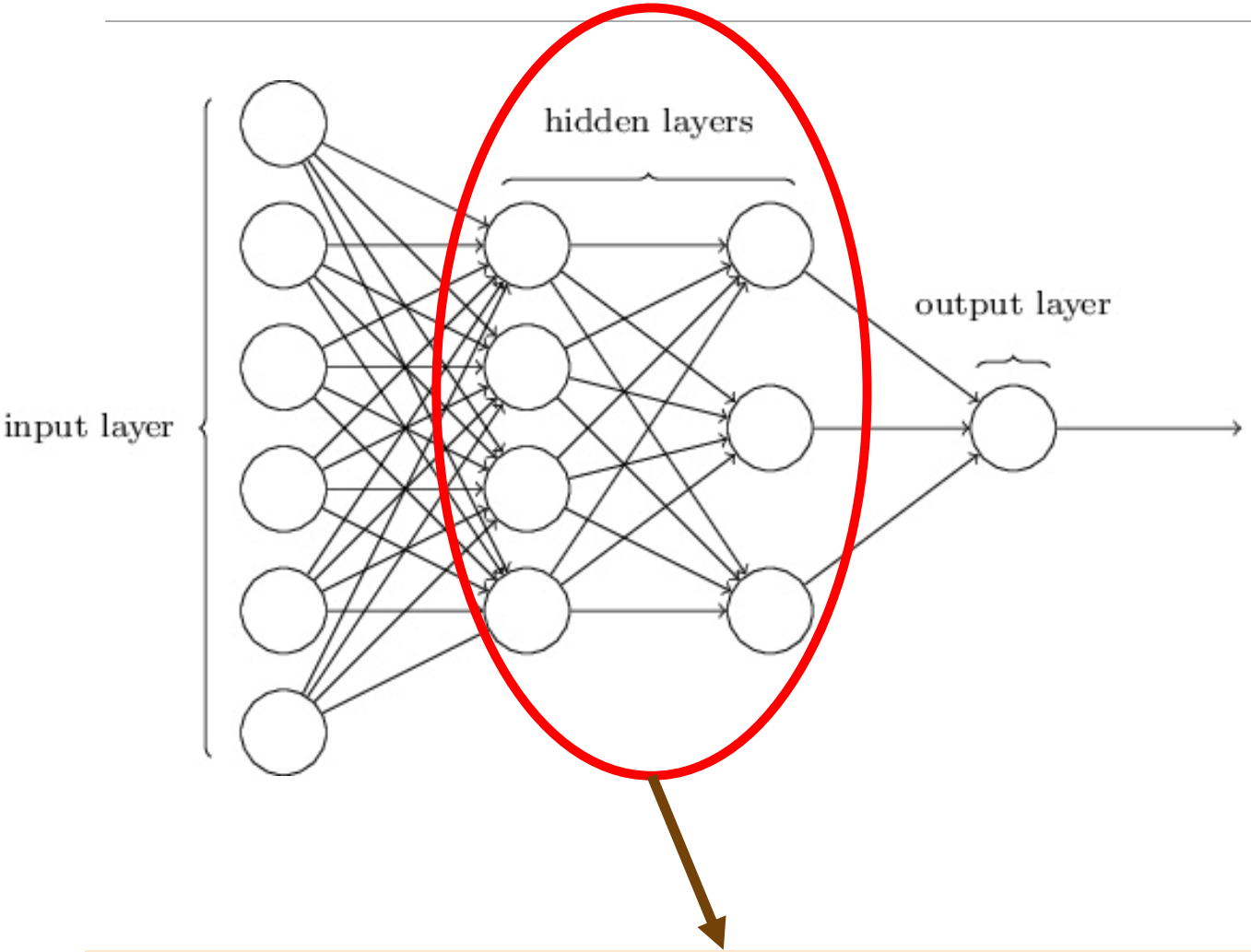
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Multilayer Perceptron (MLP)

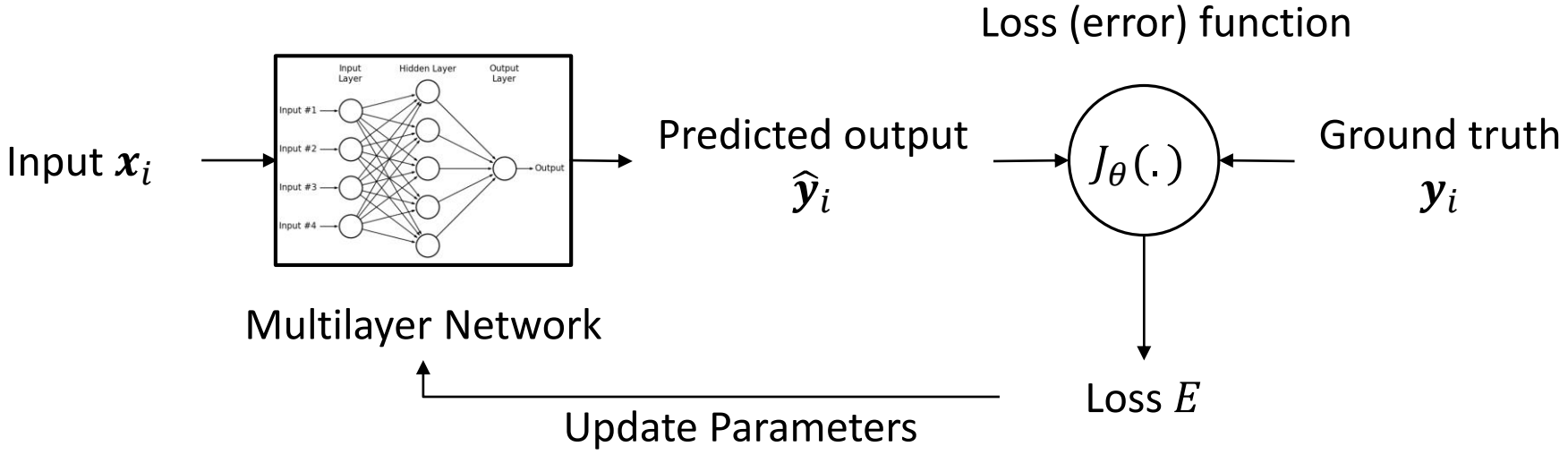


- Multilayer perceptron (MLP)
- MLP can engage in sophisticated decision making, where perceptrons fail
  - E.g. XOR problem

Play with neural network: http://playground.tensorflow.org

Stacking multiple layers of perceptrons (adding hidden layers)

Learning NN Parameters

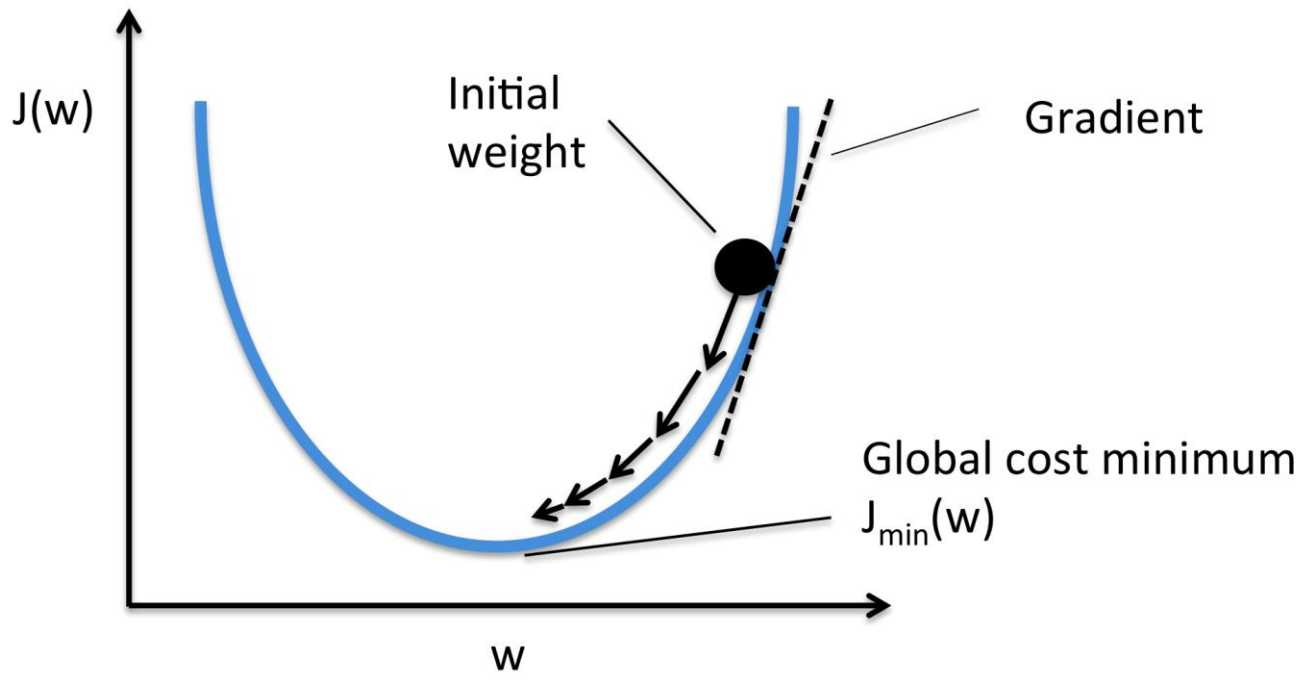


- **Gradient Descent Algorithm**
- **Input**: Training sample $\boldsymbol{x}_i$ and its label $\boldsymbol{y}_i$

1. **Feed Forward**: Get prediction $\hat{\boldsymbol{y}}_i = \mathrm{MLP}(x_i)$, and loss $E = J(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$
2. **Compute Gradient**: For each parameter θ_i (weights, bias), compute its gradient $\frac{\partial}{\partial \theta_i} J_\theta$

   Explained later
3. **Update Parameter**: $\theta_i = \theta_i - \alpha \cdot \frac{\partial}{\partial \theta_i} J_\theta$

Empirical Explanation of Gradient Descent

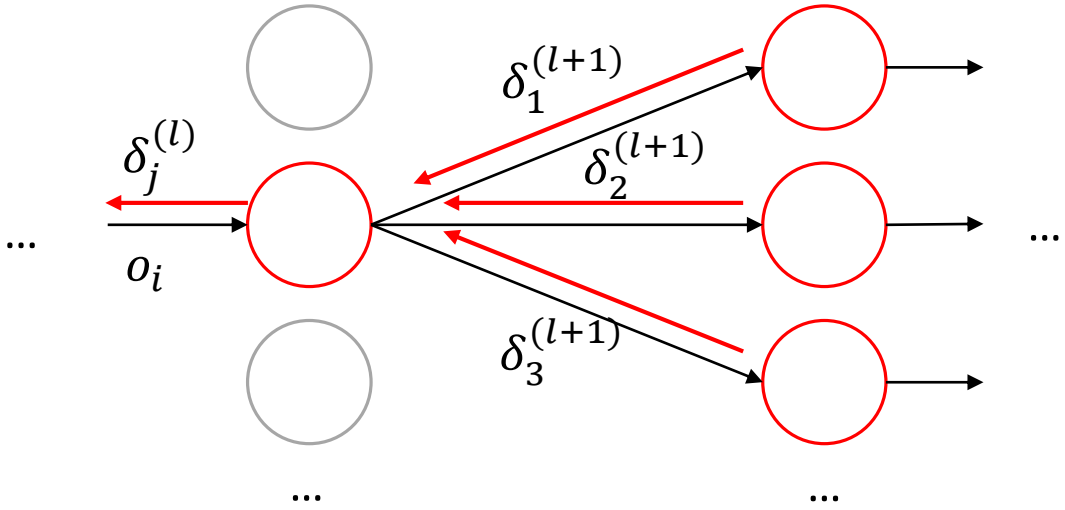


$$\theta_i = \theta_i - \alpha \cdot \frac{\partial}{\partial \theta_i} J_\theta$$

learning rate - 'step size' of the optimization

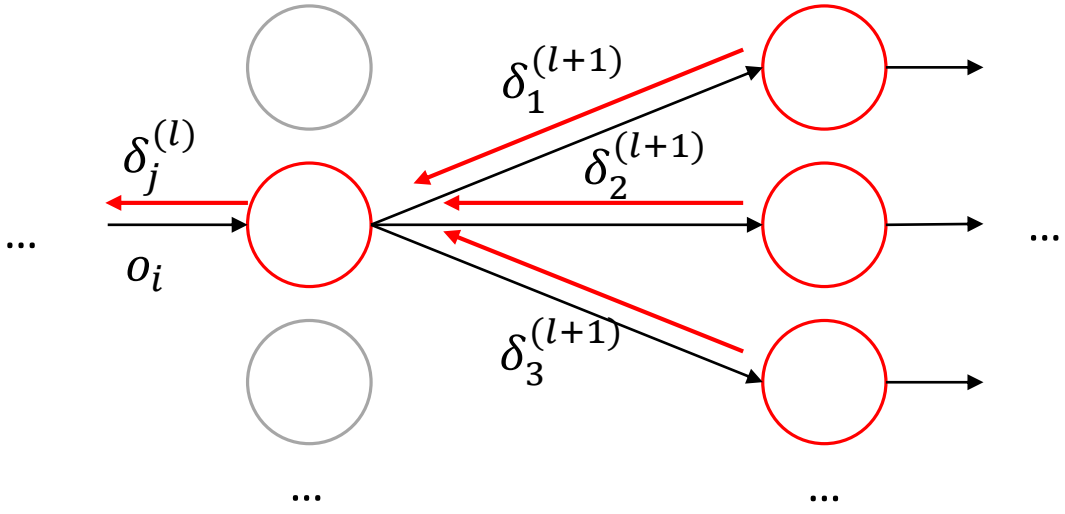
- The loss function J – a function of the model parameters
- Objective – Minimize J
- Gradient – Measures how much the output of a function changes if you change the inputs a little bit
- We update the parameters, based on their gradients, so that the loss function is going 'downhill'

Gradient Computation: Backpropagation



- The gradient of w_{ij} in the lth layer (corresponding to unit j in layer l, connected to unit i in layer l-1) is a function of
  - All 'error' terms from layer l+1 $\delta_k^{(l+1)}$ -- An auxiliary term for computation, not to be confused with gradients
  - Output from unit i in layer l-1 (input to unit j in layer l) -- Can be stored at the feed forward phase of computation

Gradient Computation: Backpropagation



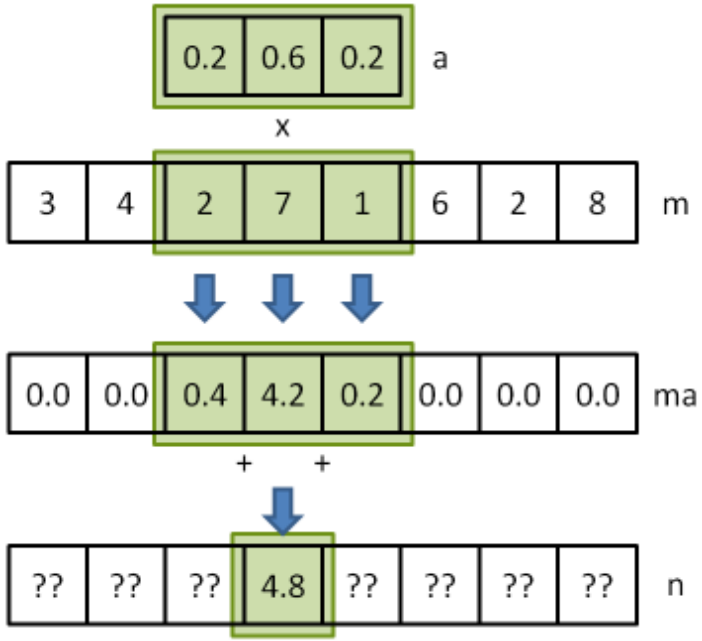
- The 'error' terms $\delta_j^{(l)}$ is a function of
  - All $\delta_k^{(l+1)}$ in the layer l+1, if layer l is a hidden layer
  - The overall loss value, if layer l is the output layer
- We can compute the error at the output, and distributed backwards throughout the network's layers (backpropagation)

From Neural Networks to Deep Learning

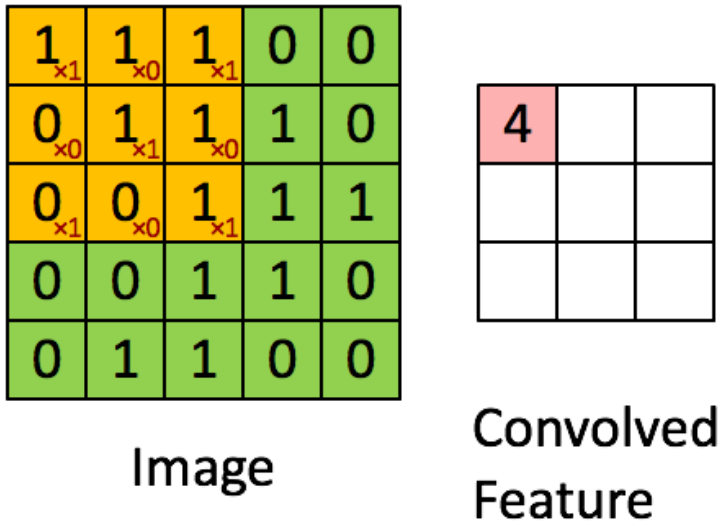
- **Deep Learning –** Training (deep) neural networks with
  - More neurons, more layers
  - More complex ways to connect layers
- Advantages
  - **Tremendous improvement of performance** in
    - Image recognition, natural language processing, AI game playing…
  - **Requires no (or less) feature engineering**, making end-to-end models possible
- Several factors lead to deep learning's success
  - Very large data sets
  - Massive amounts of computation power (GPU acceleration)
  - Advanced neural network structures and tricks
    - Convolutional neural networks, recurrent neural networks, …
    - Dropout, ReLU, residual connection, … (not covered)

Convolutional Neural Networks (CNN)

- What is convolution?



1D Convolution



2D Convolution

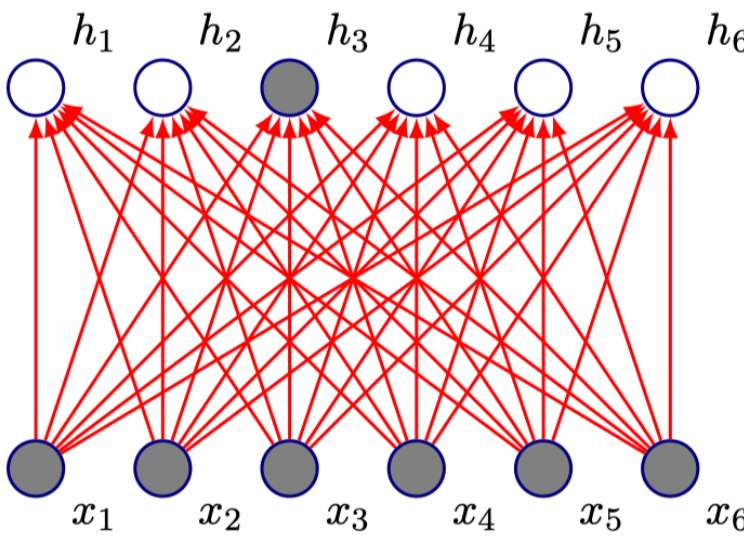
- The outputs are computed by sliding a **kernel** (of weights) on the inputs, and computing weighted sum locally

CNN Motivation

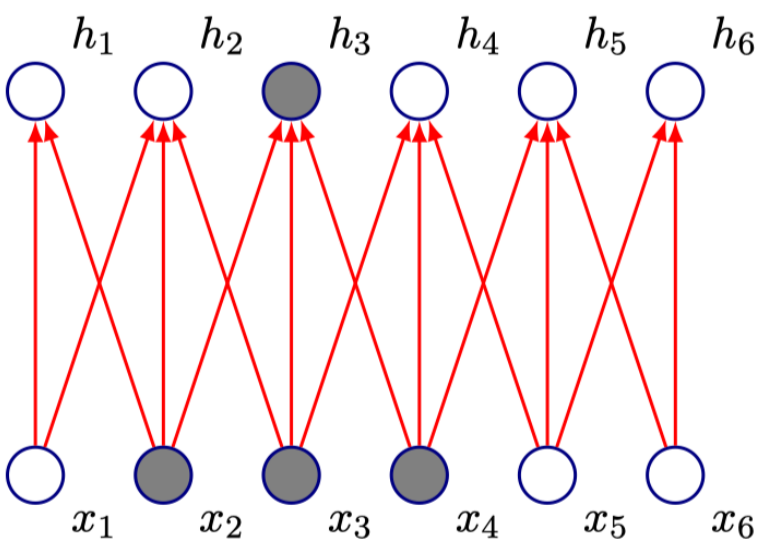
- Why not deep MLP?
  - **Computationally expensive** (Long training time)
  - **Hard to train** (slow convergence, local minima).
- Motivations of convolution
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
- The properties of CNNs are well aligned with properties of many forms of data (e.g. images, text), making them very successful

CNN Motivation

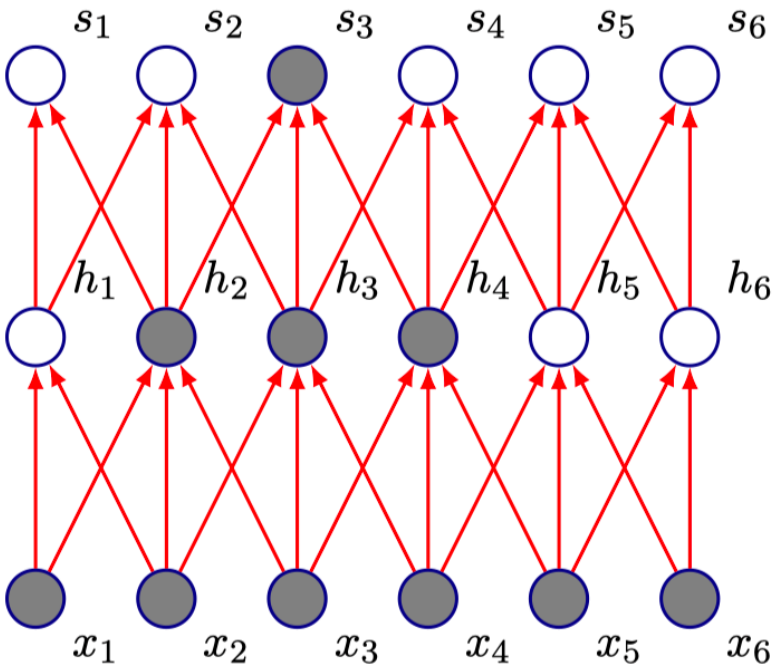
- Motivations of convolution
  - Sparse interactions
    - E.g. 1D convolution with kernel size 3
    - Units in deeper layers still connect to a wide range of inputs



MLP



CNN

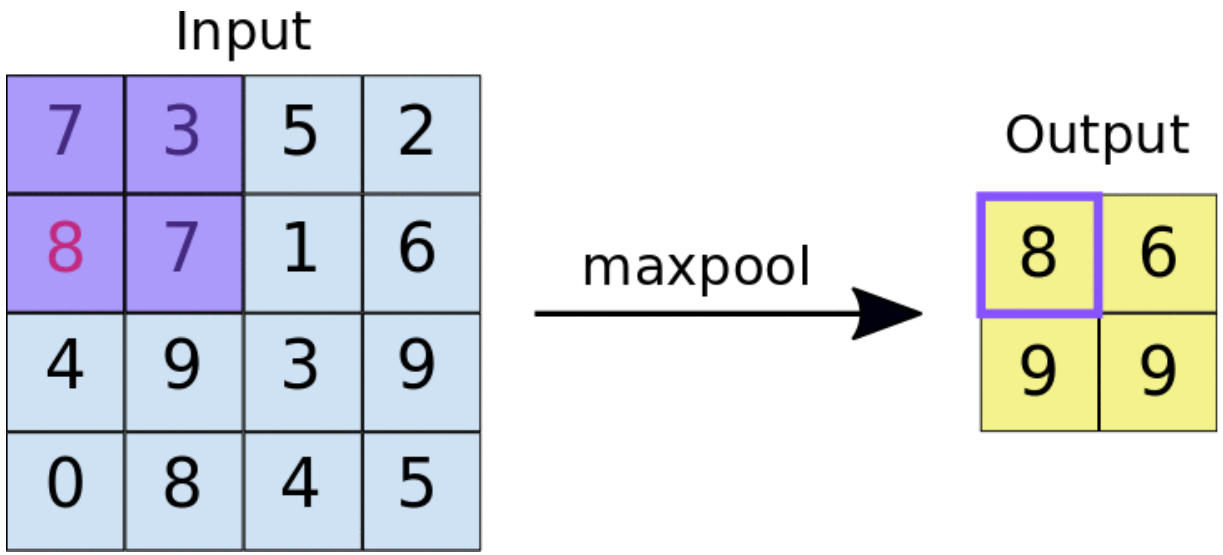


CNN Motivation

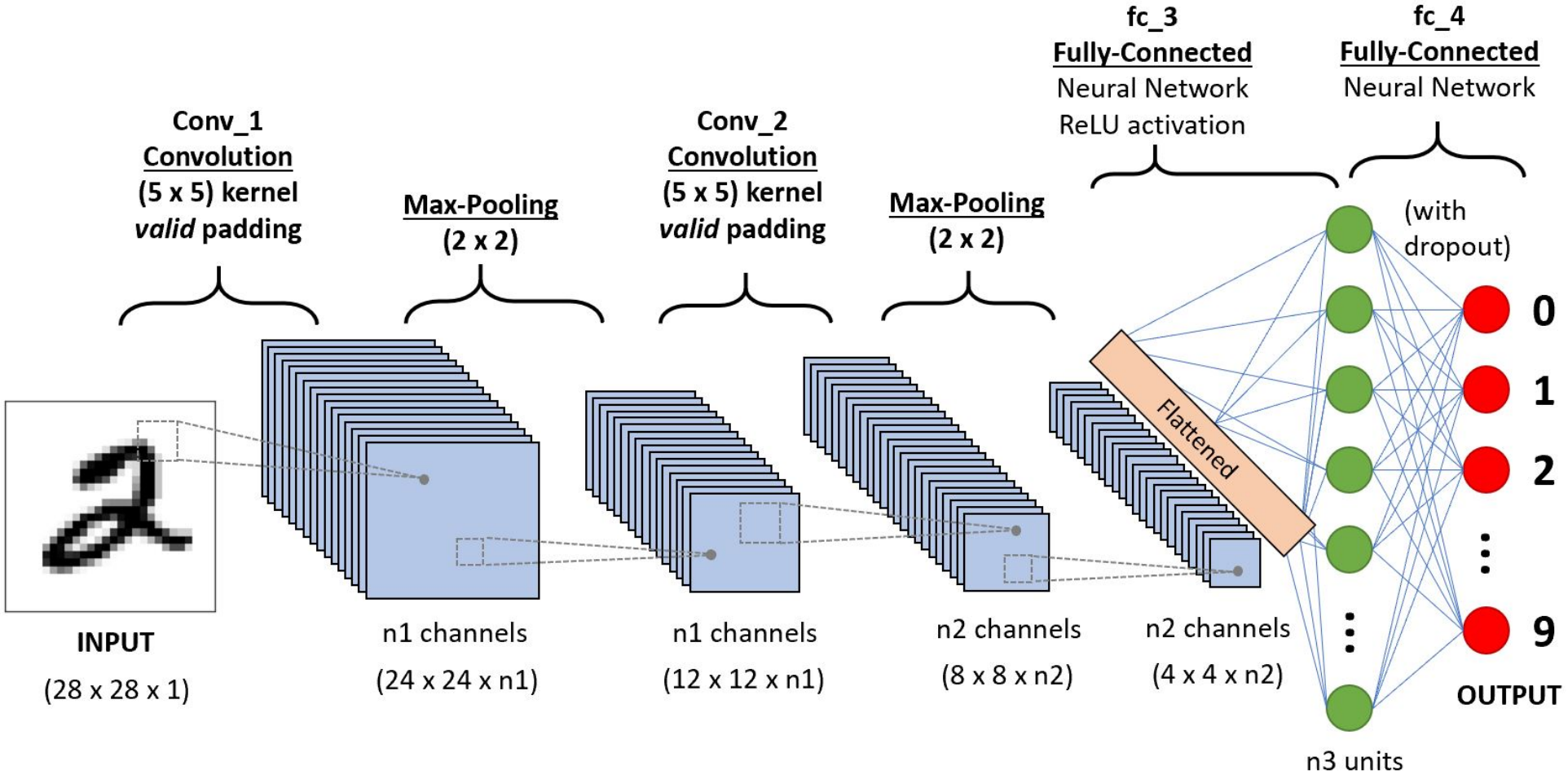
- Motivations of convolution
  - **Parameter sharing**
    - Each kernel is used on all locations of input
    - Reduce # of parameters
  - **Equivariance**
    - Same input at different location gives same output
    - E.g. a cat at the upper right corner and at the lower left corner of an image, will produce the same outputs
    - E.g. “University of Illinois” at the start of the sentence and at the end of the sentence produce the same outputs

CNN: Pooling Layer

- Pooling (Subsampling)
  - Pool hidden units in the same neighborhood
    - Introduces invariance to local translations
    - Reduces the number of hidden units in hidden layer



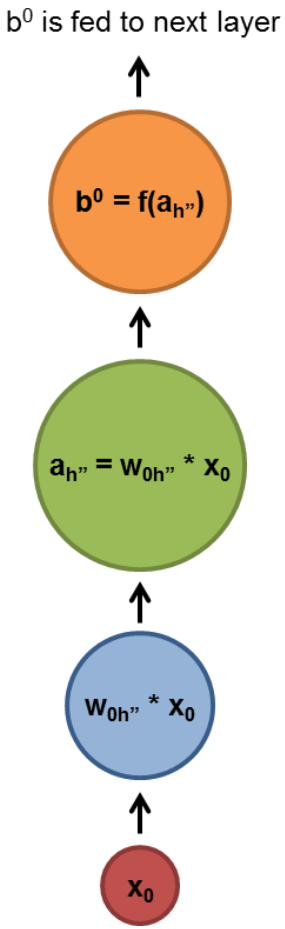
CNN for Image Recognition: Example



An example CNN for hand written digit recognition

Recurrent Neural Networks

- Handling sequences with **Recurrent Neural Networks (RNN)**
  - At each time step, the input and the previous hidden state are fed into the network



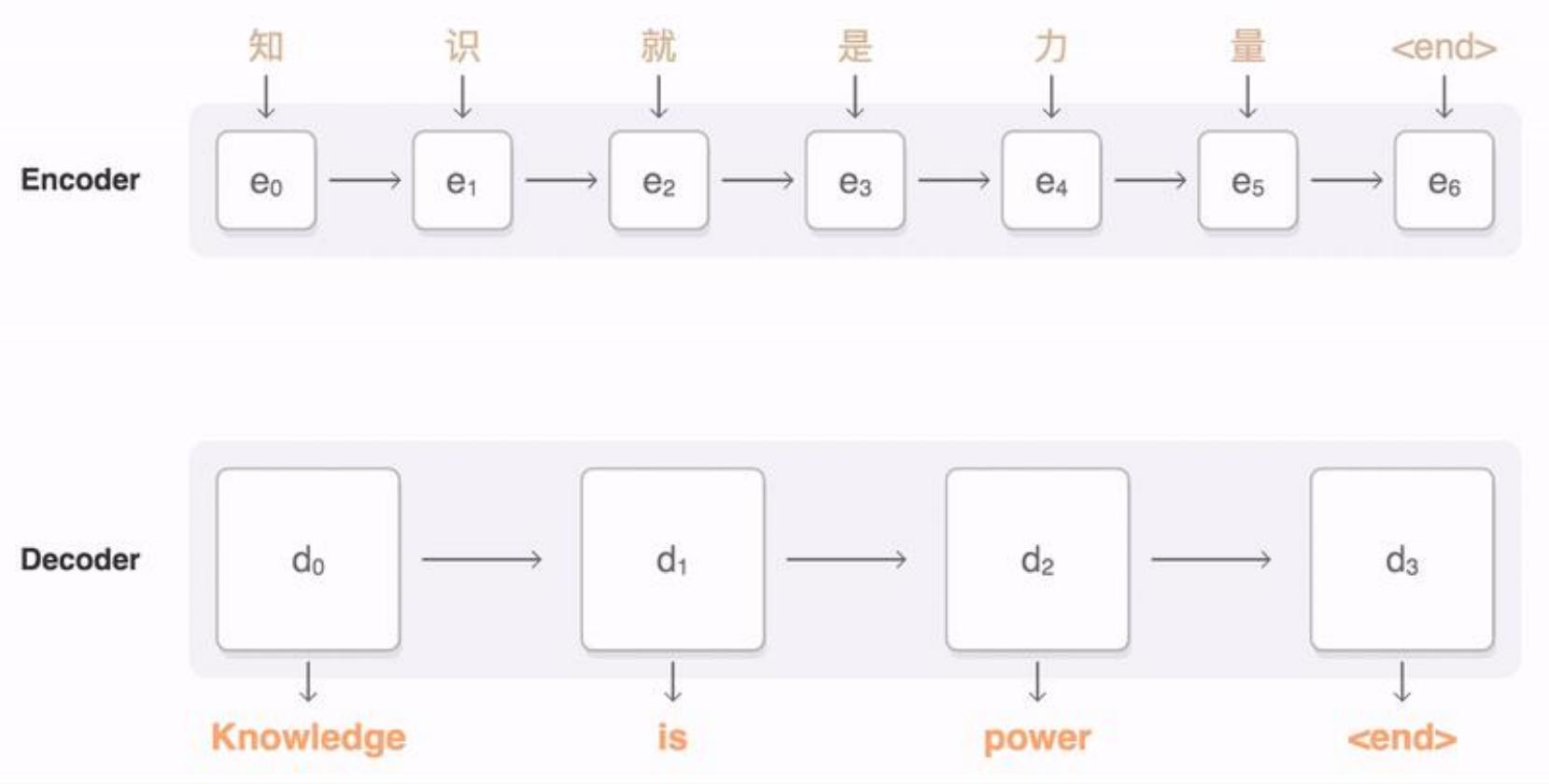
Recurrent Neural Networks: General Concepts

- Modeling the time dimension:
  - **Feedback loops** connected to past decisions
  - **Long-term dependencies**: Use hidden states to preserve sequential information
- RNNs are trained to generate sequences: Output at each timestamp is based on ALL inputs (current and previous)

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

  - Compute a gradient with the algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and Exploding Gradients
  - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
  - Solution: Use a variant of RNN: LSTM (1997, by Hochreiter and Schmidthuber)

RNN for Machine Translation: Example



Deep Learning Recap

- Pros
  - Very good performance on certain tasks, for certain types of data
    - Images: image recognition, segmentation, …
    - Text (sometimes): machine translation, language modeling,…
    - …
  - Requires very little feature engineering
  - Good generalization
    - E.g. models trained on ImageNet dataset for classification can help tasks such as segmentation
- Cons
  - Requires huge amounts of computation power
  - Black box model
  - Hard to tune the architecture and hyperparameters for new tasks

Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

Using IF-THEN Rules for Classification

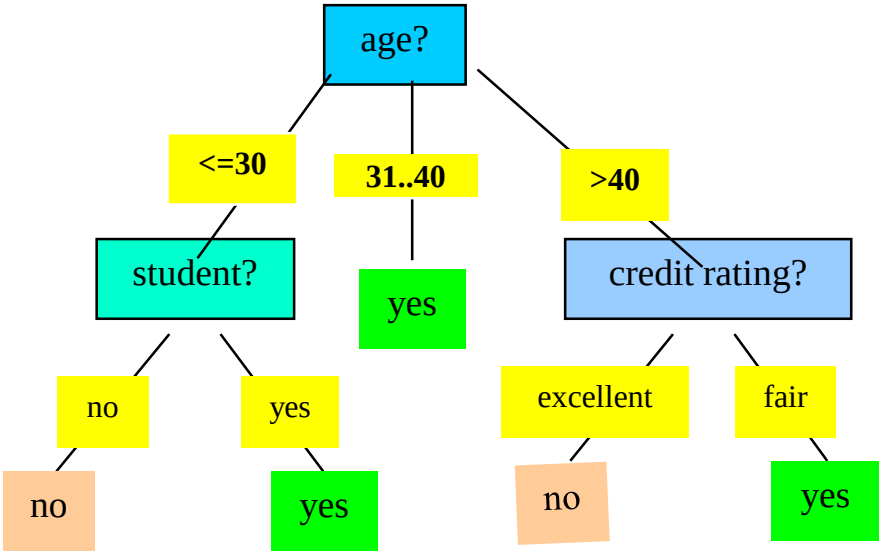
- Represent the knowledge in the form of **IF-THEN** rules

  R_1: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes
- Assessment of a rule: *coverage* and *accuracy*
  - coverage(R_1) = ratio of tuples covered by **the condition of** R_1 (THEN-part is not important for this)
  - accuracy(R_1) = ratio of tuples correctly classified by R_1 in the covered ones (both IF-part and THEN-part counts)

- If more than one rule are triggered, need **conflict resolution**
  - **Size ordering**: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute tests*)
  - **Class-based ordering**: decreasing order of *prevalence or misclassification cost per class*
  - **Rule-based ordering** (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule Extraction from a Decision Tree

- Rules are *easier to understand* than large trees
- One rule is created *for each path* from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive



- Example: Rule extraction from our *buys_computer* decision-tree

IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*

IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*

IF *age* = mid-age THEN *buys_computer* = *yes*

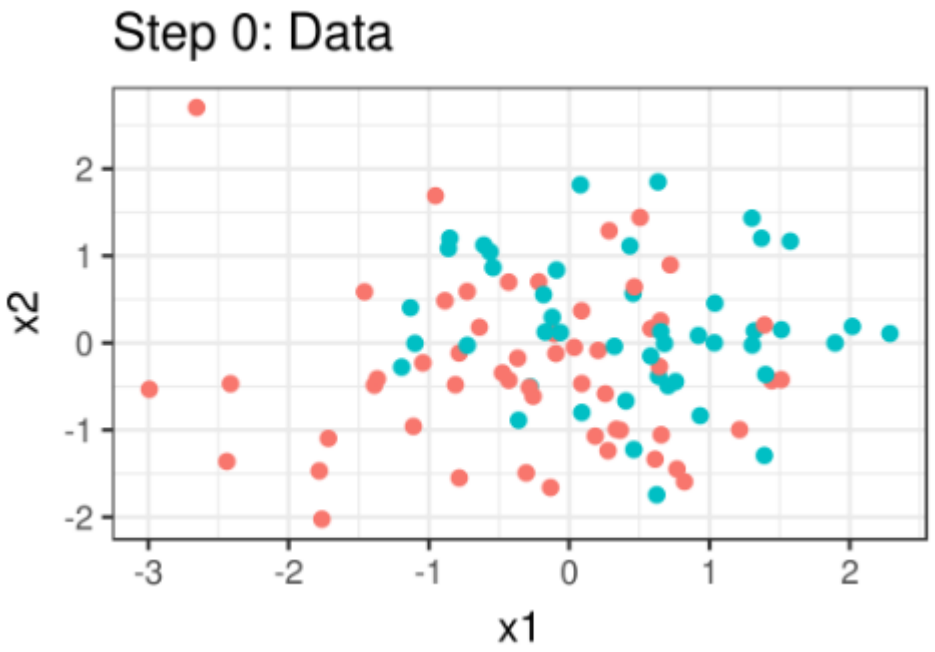
IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *no*

IF *age* = old AND *credit_rating* = *fair* THEN *buys_computer* = *yes*

Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

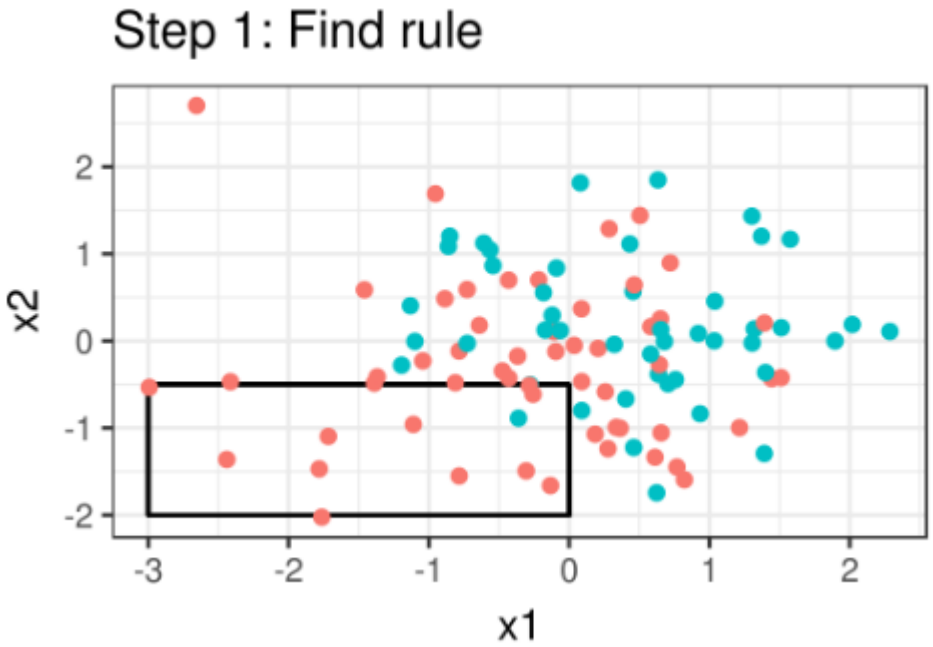
- *Step 0: Start with an empty list of rules.*



Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

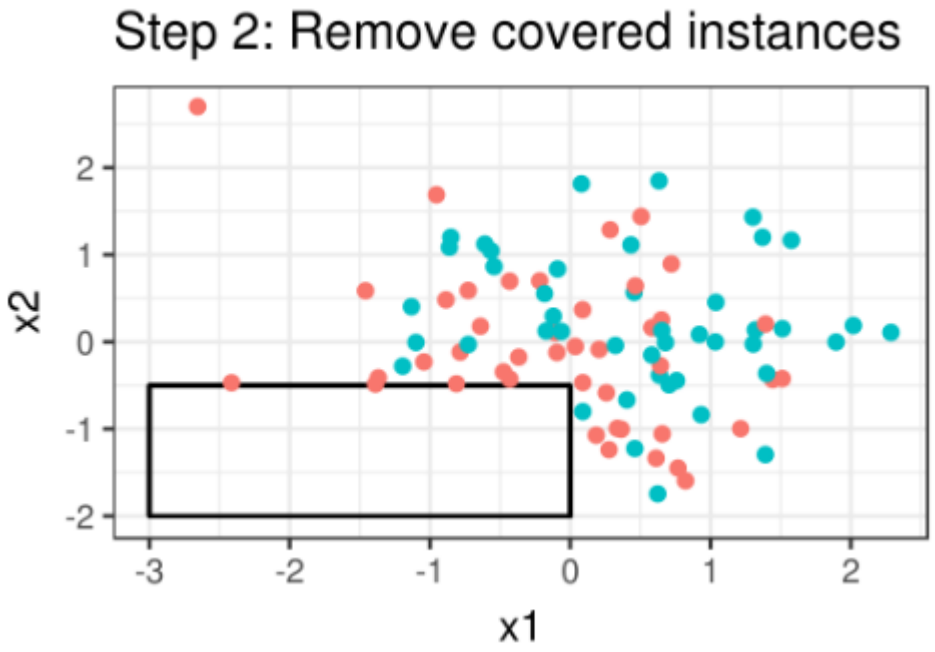
- *Step 1: Learn a rule r.*



Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

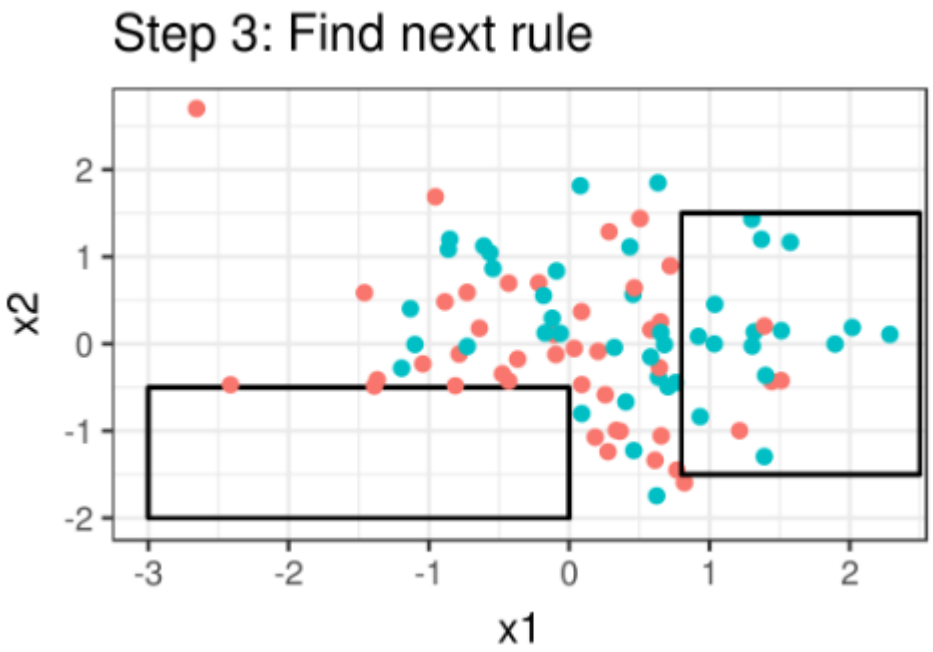
- *Step 2: The tuples covered by the rules are removed.*



Rule Induction: Sequential Covering Method

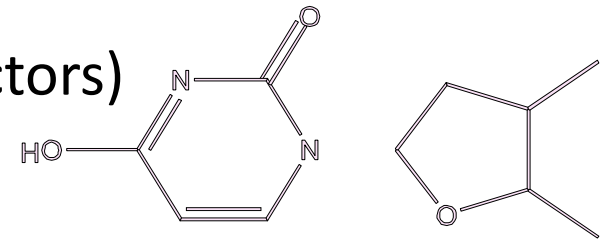
- Sequential covering algorithm: Extracts rules directly from training data
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

- *Step 3: Repeat the process* on the remaining tuples until *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a threshold.



Pattern-Based Classification, Why?

- **Pattern-based classification:** An integration of both themes
- **Why pattern-based classification?**
  - **Feature construction**
    - Higher order; compact; discriminative
    - E.g., single word → phrase (Apple pie, Apple i-pad)
  - **Complex data modeling**
    - Graphs (no predefined feature vectors)
    - Sequences
    - Semi-structured/unstructured Data



CBA: Classification Based on Associations

- CBA [Liu, Hsu and Ma, KDD'98]
- Method
  - Mine high-confidence, high-support class association rules
  - LHS: conjunctions of attribute-value pairs); RHS: class labels
    $p_1 \wedge p_2 \ldots \wedge p_l \rightarrow$ "$A_{\text{class-label}} = C$" (confidence, support)
  - Rank rules in descending order of confidence and support
  - Classification: Apply the first rule that matches a test case; o.w. apply the default rule
  - Effectiveness: Often found more accurate than some traditional classification methods, such as C4.5
  - Why? — Exploring high confident associations among multiple attributes may overcome some constraints introduced by some classifiers that consider only one attribute at a time

Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

Lazy vs. Eager Learning

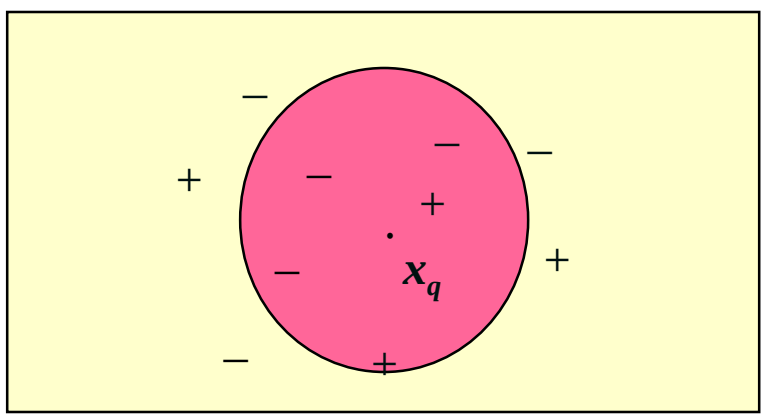
- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

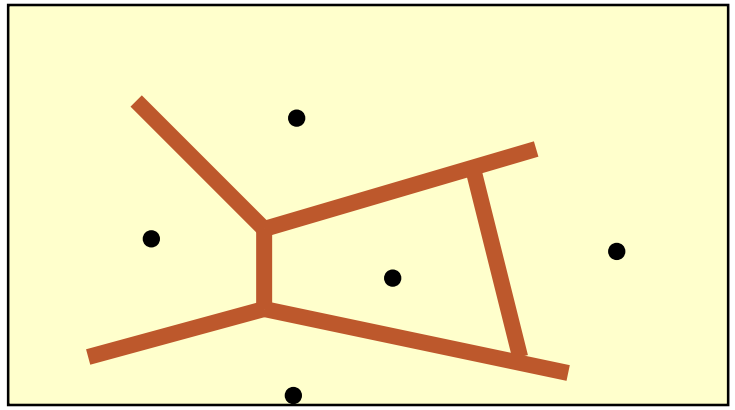
Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - <u>*k*-nearest neighbor approach</u>
    - Instances represented as points in a Euclidean space.
  - <u>Locally weighted regression</u>
    - Constructs local approximation
  - <u>Case-based reasoning</u>
    - Uses symbolic representations and knowledge-based inference

The *k*-Nearest Neighbor Algorithm

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X}_1$, $\mathbf{X}_2$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





Discussion on the *k*-NN Algorithm

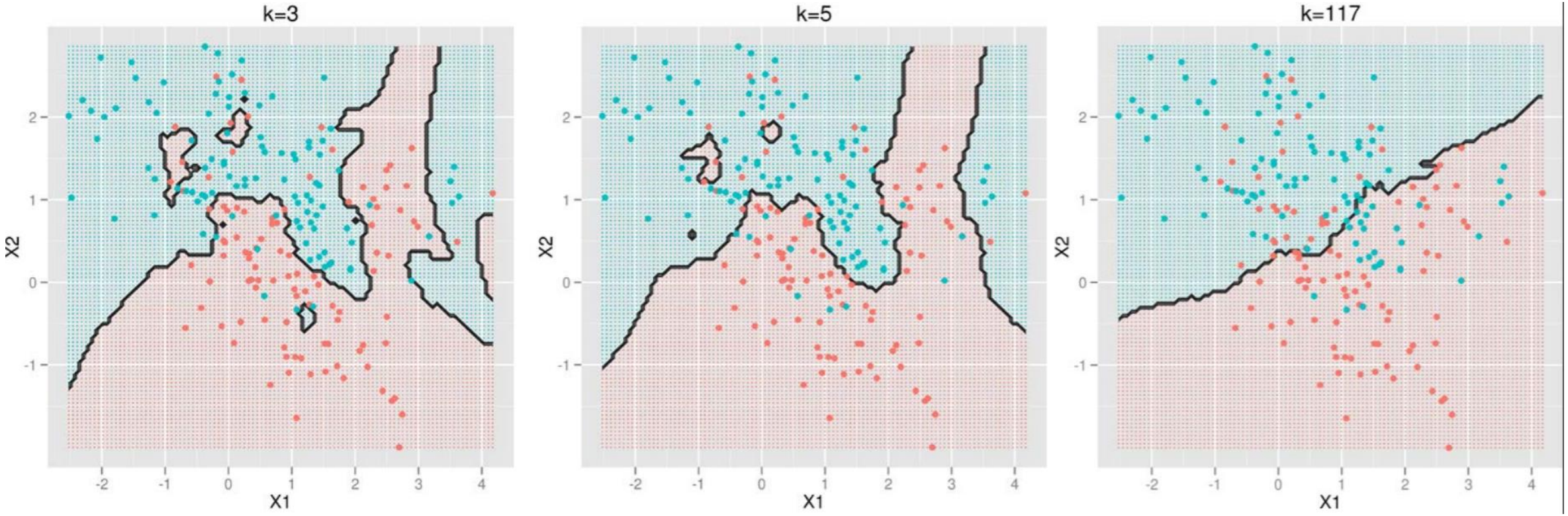
- *k*-NN for <u>real-valued prediction</u> for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- <u>Distance-weighted</u> nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors

$$w = \frac{1}{d(x_q, x_i)^2}$$

- <u>Pro:</u> Robust to noisy data by averaging *k*-nearest neighbors
- <u>Cons:</u>
  - Curse of dimensionality- distance between neighbors could be dominated by irrelevant attributes
    - To overcome it, axes stretch or elimination of the least relevant attributes
  - How to measure similarity?

Selection of k for kNN

- The number of neighbors k
  - Small k: overfitting (high var., low bias)
  - Big k: bringing too many irrelevant points (high bias, low var.)



http://scott.fortmann-roe.com/docs/BiasVariance.html

Case-Based Reasoning (CBR)

- **CBR**: Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
  - Instances represented by rich symbolic descriptions (e.g., function graphs)
  - Search for similar cases, multiple retrieved cases may be combined
  - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
- Challenges
  - Find a good similarity metric
  - Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

Chapter 9. Classification: Advanced Methods

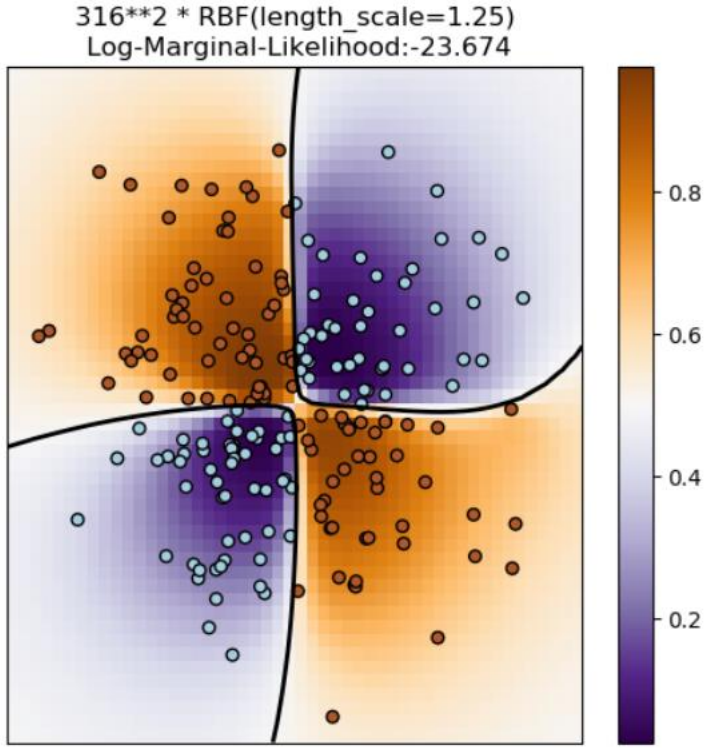
- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

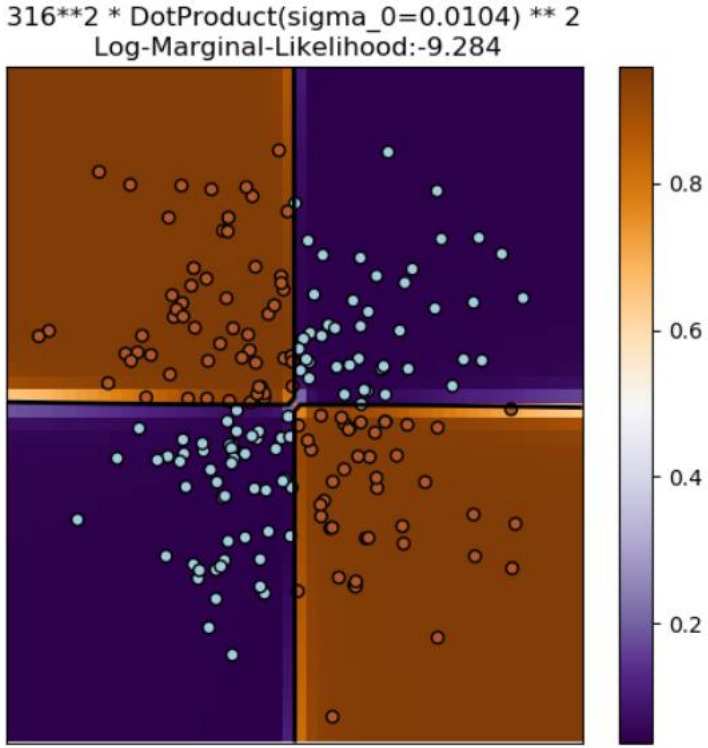
Genetic Algorithms (GA)

- Genetic Algorithm: (biological evolution)
- An initial **population** is created consisting of randomly generated rules
  - Each rule is represented by a string of bits
  - E.g., if A_1 and $\neg A_2$ then C_2 can be encoded as 100
  - If an attribute has k > 2 values, k bits can be used
- Fitness: classification accuracy on a set of training examples
- Survival of the **fittest** ->a new population (the fittest rules and their offspring)
- Offspring are generated by *crossover* and *mutation*
- The process continues until a population P evolves *when each rule in P satisfies a pre-specified threshold*
- Slow but easily parallelizable

Gaussian Process

- Lazy learning
- Probabilistic prediction





Chapter 9. Classification: Advanced Methods

- Ensemble Methods: Increasing the Accuracy
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

Summary

- Bayesian belief network (probabilistic networks)
- Support Vector Machine (SVM)
- Neural networks and Deep Learning
- Pattern-Based classification
- Other classification methods
  - lazy learners (KNN, case-based reasoning)

References (1)

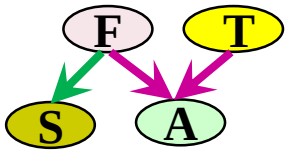
- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth International Group, 1984
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- N. Cristianini and J. Shawe-Taylor, Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000
- H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. KDD'03
- A. J. Dobson. An Introduction to Generalized Linear Models. Chapman & Hall, 1990
- R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification, 2ed. John Wiley, 2001
- T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, 2001
- S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008
- D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning, 1995

References (2): Rule and Pattern-Based Classification

- H. Cheng, X. Yan, J. Han & C.-W. Hsu, Discriminative Frequent Pattern Analysis for Effective Classification, ICDE'07
- H. Cheng, X. Yan, J. Han & P. S. Yu, Direct Discriminative Pattern Mining for Effective Classification, ICDE'08
- W. Cohen. Fast effective rule induction. ICML'95
- G. Cong, K. Tan, A. Tung & X. Xu. Mining Top-k Covering Rule Groups for Gene Expression Data, SIGMOD'05
- M. Deshpande, M. Kuramochi, N. Wale & G. Karypis. Frequent Substructure-based Approaches for Classifying Chemical Compounds, TKDE'05
- G. Dong & J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences, KDD'99
- W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu & O. Verscheure. Direct Mining of Discriminative and Essential Graphical and Itemset Features via Model-based Search Tree, KDD'08
- W. Li, J. Han & J. Pei. CMAR: Accurate and Efficient Classification based on Multiple Class-association Rules, ICDM'01
- B. Liu, W. Hsu & Y. Ma. Integrating Classification and Association Rule Mining, KDD'98
- J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. ECML'93
- Jingbo Shang, Wenzhu Tong, Jian Peng, and Jiawei Han, "DPClass: An Effective but Concise Discriminative Patterns-Based Classification Framework", SDM'16
- J. Wang and G. Karypis. HARMONY: Efficiently Mining the Best Rules for Classification, SDM'05
- X. Yin & J. Han. CPAR: Classification Based on Predictive Association Rules, SDM'03

How Are Bayesian Networks Constructed?

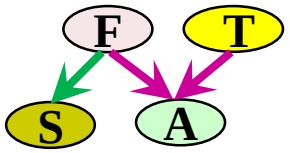
- **Subjective construction**: Identification of (direct) causal structure
  - People are quite good at identifying direct causes from a given set of variables & whether the set contains all relevant direct causes
  - Markovian assumption: Each variable becomes independent of its non-effects once its direct causes are known
    - E.g., S ← F → A ← T, path S → A is blocked once we know F → A



  - HMM (Hidden Markov Model): often used to model dynamic systems whose states are not observable, yet their outputs are

How Are Bayesian Networks Constructed?

- **Synthesis from other specifications**
  - E.g., from a formal system design: block diagrams & info flow
- **Learning from data** (e.g., from medical records or student admission record)
  - Learn parameters give its structure or learn both structure and params
  - Maximum likelihood principle: favors Bayesian networks that maximize the probability of observing the given data set



Linear SVM for Linearly Separable Data

- A separating hyperplane can be written as

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0$$

Model parameters to learn

where $w = (w_1, w_2, \dots, w_n)^T$ is a weight vector and b a scalar (bias)

  - For 2-D, it can be written as: $w_1\, x_1 \; + \; w_2\, x_2 + b = \; 0$

- The distance from any data point $\boldsymbol{x}$ to the separating hyperplane is

$$r = \frac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}$$

- Our objective is to maximize the distance of the closest data point to the hyperplane

$$\arg\max_{w,b}\left\{\frac{1}{\|\boldsymbol{w}\|}\min[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)]\right\}$$

- This is hard to solve, we shall convert it to an easier problem

Linear SVM for Linearly Separable Data

- If we rescale the model parameters $\boldsymbol{w} \to \kappa \boldsymbol{w}, b \to \kappa b$, the distance from any data point to the hyperplane is not going to change
- We can set $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$ for the closest data point to the hyperplane, then all the data points will satisfy the constraint

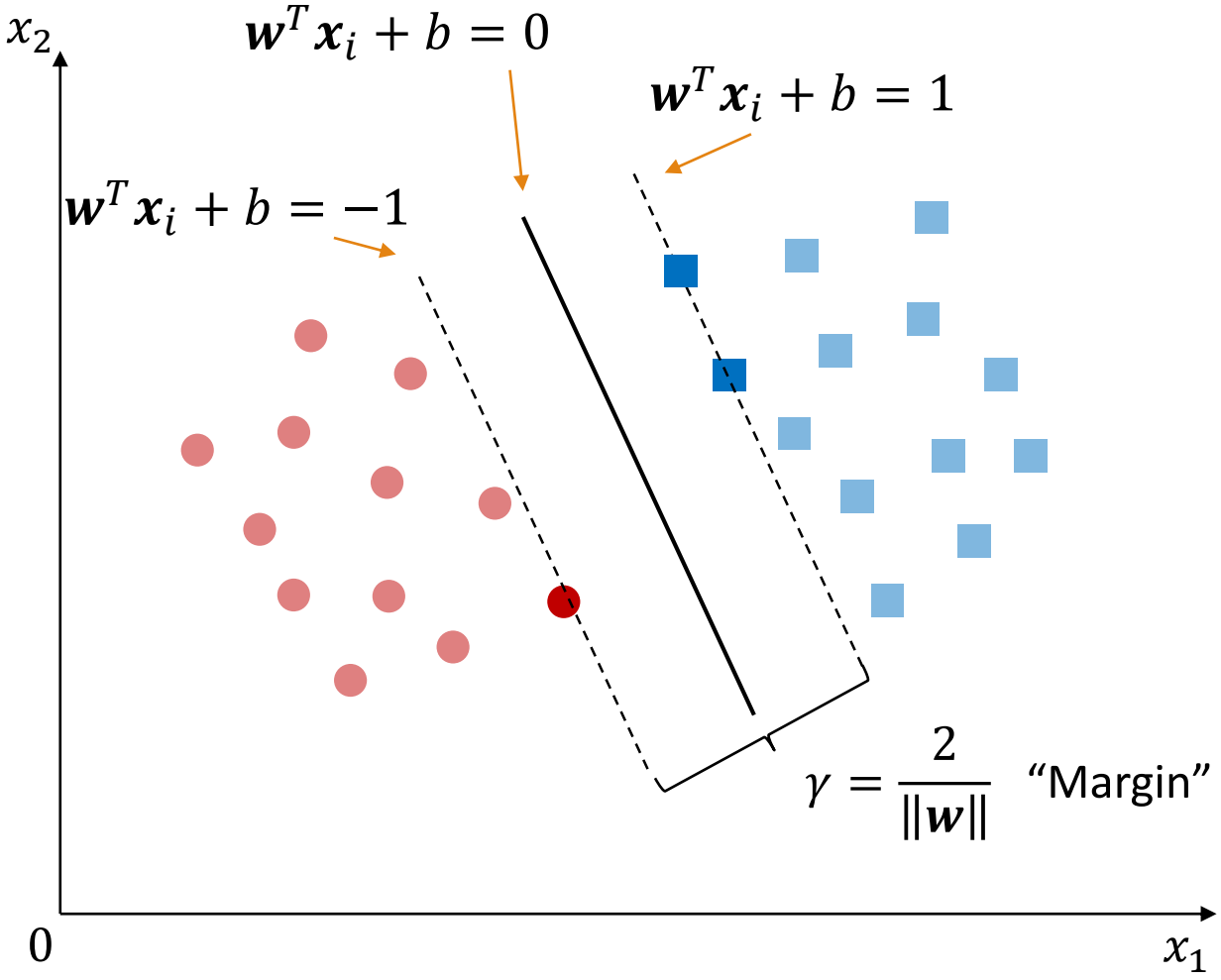
$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$$

- We will then maximize $\|\boldsymbol{w}\|^{-1}$ subject to this constraint. This is equivalent to minimizing $\|\boldsymbol{w}\|^2$

$$\boxed{\begin{aligned} \arg\min_{w,b} \quad & \|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1, \qquad i = 1,2,\ldots,n \end{aligned}}$$

- This is the basic form of SVM, and it can be solved by using *quadratic programming*

Linear SVM for Linearly Separable Data

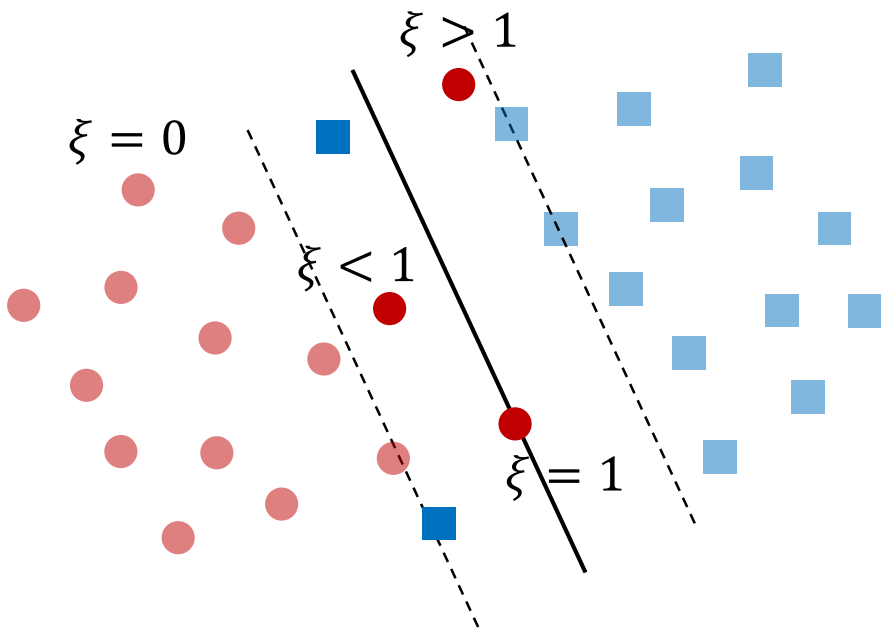


- The data points closest to the separating hyperplane are called **support vectors**

SVM for Linearly Inseparable Data

- We allow data points to be on the "wrong side" of the **margin boundary**
- Penalize points on the wrong side according to its distance to the margin boundary
- We define **slack variables**

$$\xi_i = \begin{cases} 0, & \text{correct side} \\ |y_i - f(\boldsymbol{x}_i)|, & \text{wrong side} \end{cases}$$
$$= \max\big(0, 1 - y_i(\boldsymbol{w}^T x_i + b)\big)$$



- Original constraint: $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$
- Updated constraint: $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i$

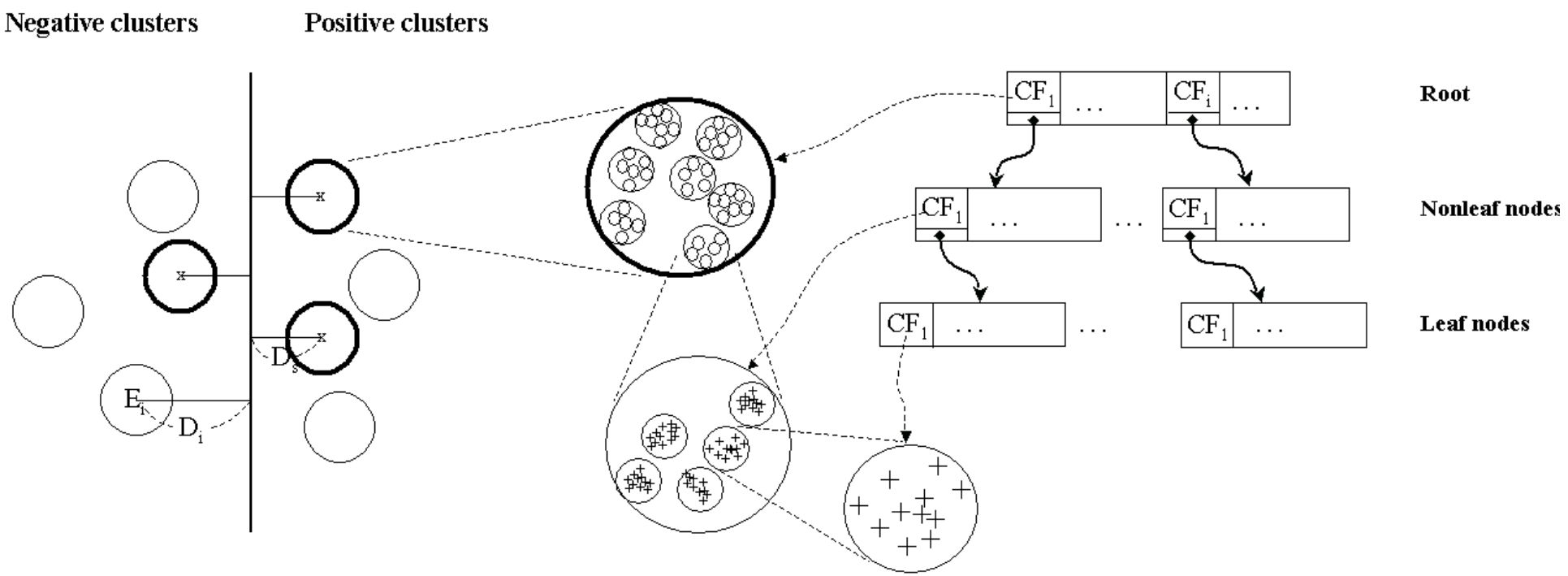
SVM for Linearly Inseparable Data

- Using the updated constraint, our objective becomes

$$\begin{aligned} \arg\min_{w,b} \quad & \|\boldsymbol{w}\|^2 + C\sum \xi_i \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \qquad\qquad i = 1,2,\ldots,n \end{aligned}$$

- C > 0 controls the trade-off between the slack variable penalty and the margin
- Limit C → ∞, we will recover the earlier support vector machine for separable data.
- This is the widely used *soft-margin SVM*

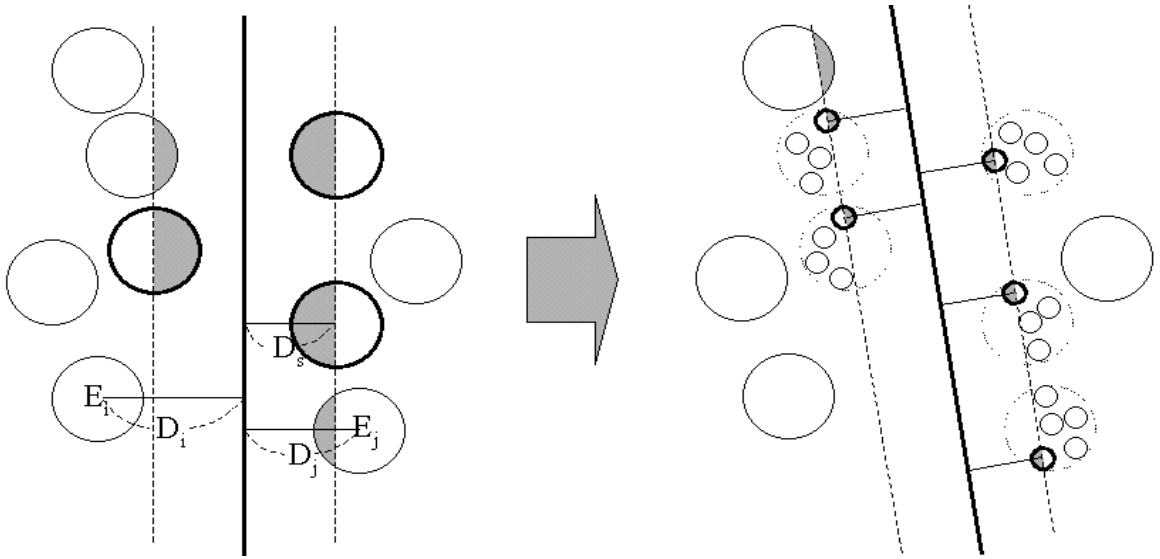
Scaling SVM by Hierarchical Micro-Clustering



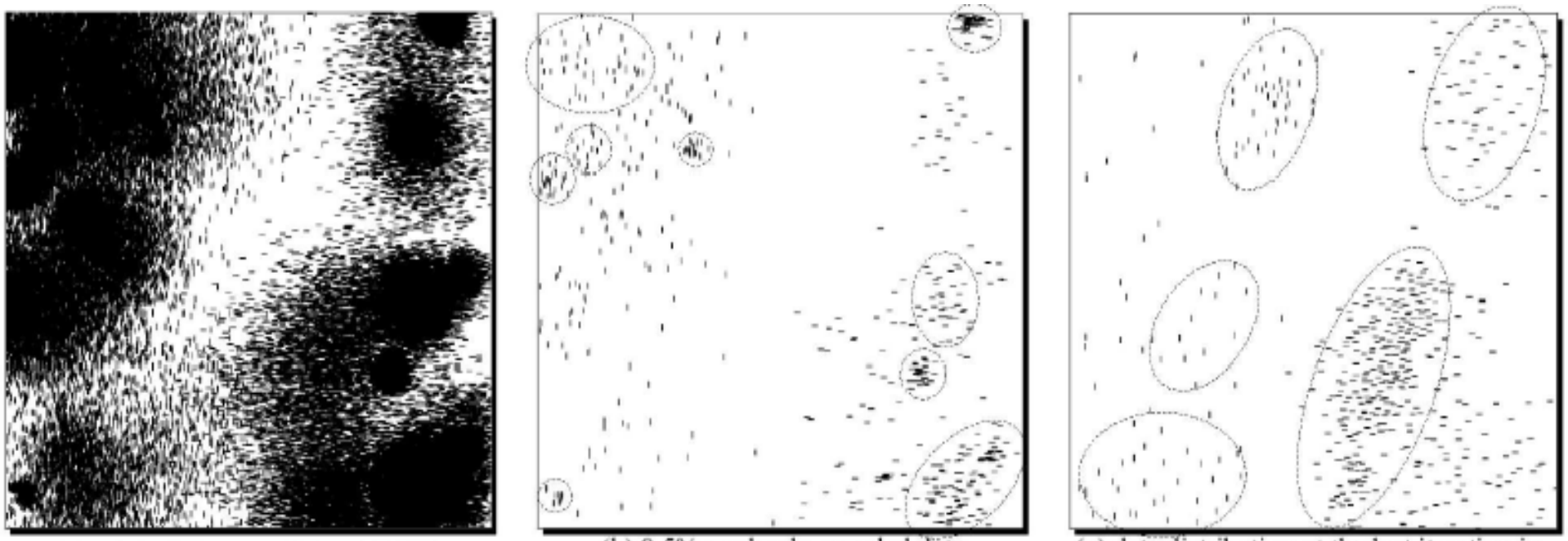
- Construct two CF-trees (i.e., statistical summary of the data) from positive and negative data sets independently (with one scan of the data set)
- Micro-clustering: Hierarchical indexing structure
  - Provide finer samples closer to the boundary and coarser samples farther from the boundary

Selective Declustering: Ensure High Accuracy

- De-cluster only the cluster E_i such that
  - $D_i - R_i < D_s$, where D_i is the distance from the boundary to the center point of E_i and R_i is the radius of E_i
  - Decluster only the cluster whose subclusters have possibilities to be the support cluster of the boundary
    - "Support cluster": The cluster whose centroid is a support vector



Accuracy and Scalability on Synthetic Dataset



(a) original data set ($N = 113601$)

(b) 0.5% randomly sampled data ($N = 603$)

(c) data distribution at the last iteration in CB-SVM ($N = 597$)

Figure 6: Synthetic data set in a two-dimensional space. '|': positive data; '–': negative data

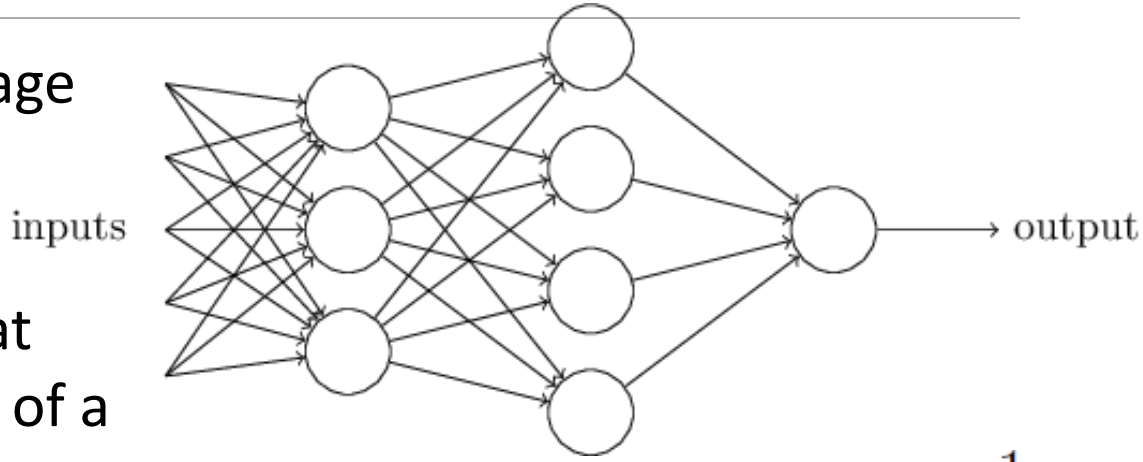
- Experiments on large synthetic data sets shows better accuracy than random sampling approaches and far more scalable than the original SVM algorithm

Sigmoid Neurons

- A many-layer network of perceptrons can engage in sophisticated decision making
- Instead of assigning weights of the edges by a person, we can devise *learning algorithms* that can automatically tune the weights and biases of a network of artificial neurons
- Use sigmoid neuron instead of perceptron: Output is not 0/1 but a sigmoid function: σ(*w*●*x* + b) , i.e.,
- The smoothness of σ means that small changes in the Δw_j weights and in the Δb bias will produce a small change Δ_{output} in the output from the neuron

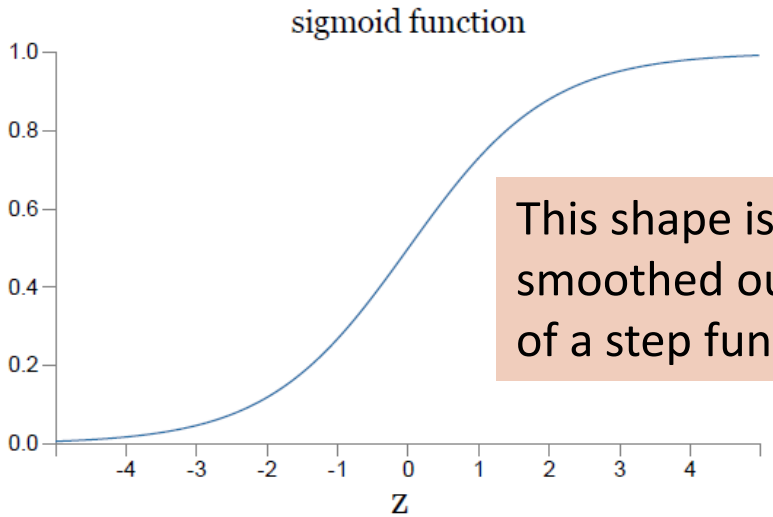
$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b$$

i.e., Δ_{output} is a *linear function* of the changes Δw_j and Δb



Sigmoid function: $\sigma(z) \equiv \frac{1}{1+e^{-z}}$

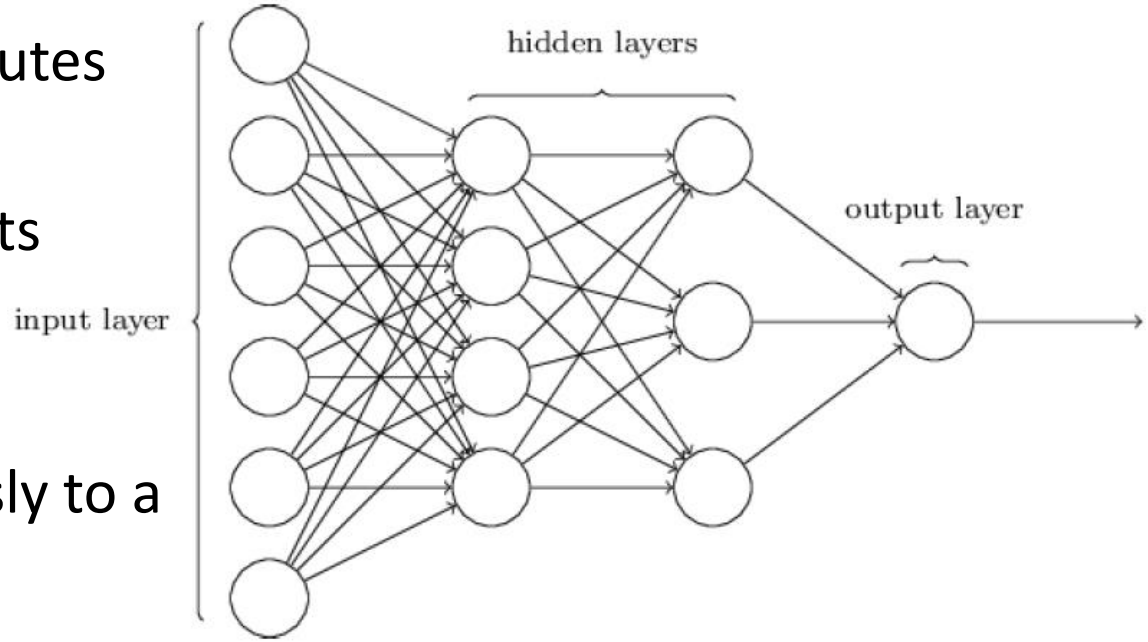
$$\frac{1}{1+exp(-\sum_j w_j x_j - b)}$$



This shape is a smoothed out version of a step function

Architecture of a (Feed-Forward) Neural Network (NN)

- **Input layer**
  - The **inputs** to NN correspond to the attributes measured for each training tuple
  - Inputs are fed simultaneously into the units making up the **input layer**
- **Hidden layer(s)**
  - Inputs are weighted and fed simultaneously to a hidden layer
  - The number of hidden layers is arbitrary



- **Output layer**
  - The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction

Neural Network Architecture: Feed-Forward vs. Recurrent

- **Feed-Forward Neural Network:** Typical neural network architecture
  - The output from one layer is used as input to the next layer (no loops)
  - Information is always fed forward, never fed back
  - From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function
- **Recurrent neural network**: Feedback loops are possible (cascade of neurons firing)
  - Some neurons fire for some limited duration of time, before becoming quiescent
  - That firing can stimulate other neurons, which may fire a little while later, also for a limited duration, which causes still more neurons to fire, and so on
  - Loops do not cause problems since a neuron's output only affects its input at some later time, not instantaneously

Learning with Gradient Descent

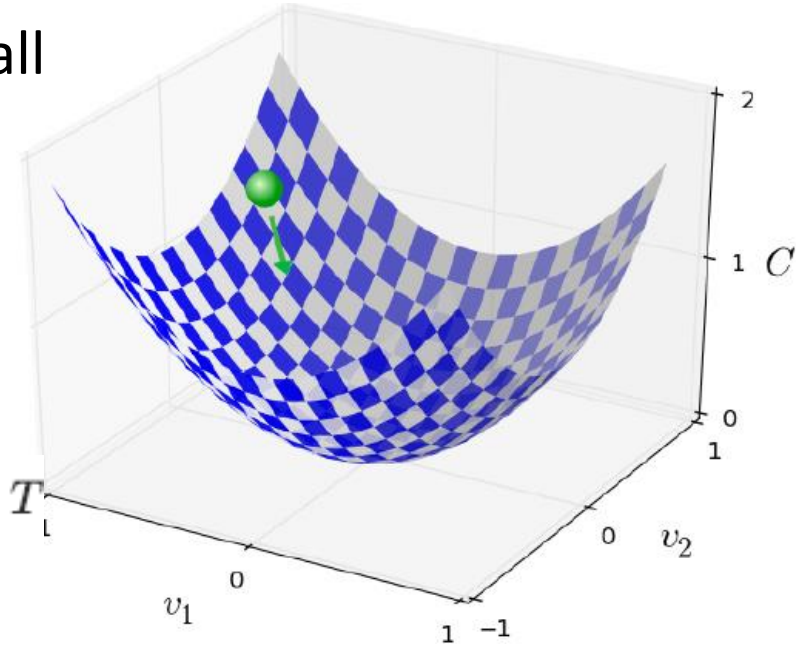
- A quadratic cost (objective) function C (or mean square error, MSE)

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x)-a\|^2$$

where w: the collection of all weights in the network, b: all the biases, n: the total # of training inputs, a: the vector of outputs from the network when *x* is input

- Goal of training a network: Find weights and biases which minimize the cost $C(w, b)$
- That is, choose Δv_1 and Δv_2 to make ΔC negative; i.e., the ball is rolling down into the valley:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$



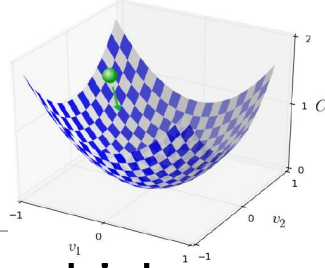
- The change ΔC *in C by a small change in v,* Δv*:*

$$\Delta C \approx \nabla C \cdot \Delta v$$

where ∇C is the gradient vector:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)$$

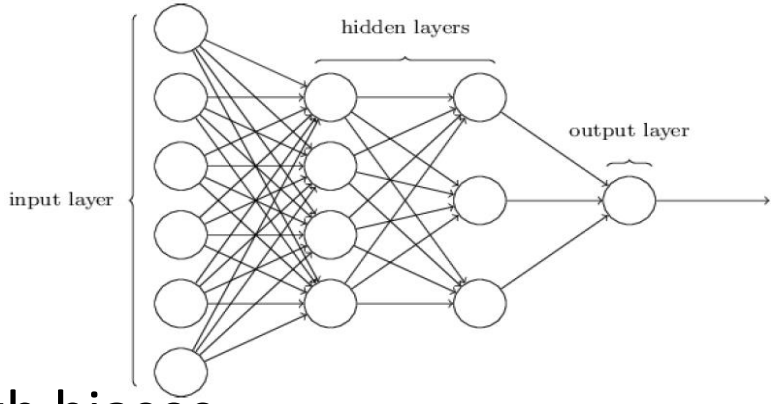
Stochastic Gradient Descent



- Gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease *C*
- To compute gradient ∇C, we need to compute the gradients ∇C_x separately for each training input, x, and then average them: slow when the # of training inputs is large
- *Stochastic gradient descent* (SGD): Speed up learning
  - Computing for a small sample of randomly chosen training inputs and *averaging over them*, we can quickly get a good estimate of the true gradient
  - Method: Randomly pick out a small number (***mini-batch***) *m* of randomly chosen training inputs. Provided the sample size is large enough, we expect that the average value will be roughly equal to the average over all, that is, $\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$
- Stochastic gradient descent in neural networks:
  - Pick out a randomly chosen minibatch of training inputs and train with them; then pick out another minibatch, until inputs exhausted—complete an *epoch* of training
  - Then we start over with a new training epoch

Backpropagation for Fast Gradient Computation

- **Backpropagation**: Reset weights on the "front" neural units and this is sometimes done in combination with training where the correct result is known
- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction
  - From the output layer, through each hidden layer back to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
    - Terminating condition (when error is very small, etc.)



More on Backpropagation

- With backpropagation, we distribute the "blame" backward through the network
  - Each hidden node sending input to the current node is somewhat "responsible" for some portion of the error in each neuron to which it has forward connection
- Local minima and backpropagation
  - Backpropagation can be stuck at local minima
  - But in practice it generally performs well
- Is backpropagation too slow?
  - Historically, backpropagation has been considered slow
  - Recent advances in computer power through parallelism and GPUs (graphics processing units) have reduced time substantially for training neural networks

From Neural Networks to Deep Learning

- Train networks with many layers (vs. shallow nets with just a couple of layers)
  - More neurons than previous networks
  - More complex ways to connect layers
  - Tremendous computing power to train networks
  - Automatic feature extraction
- Multiple layers work to build an improved feature space
  - Analogy: Signals passing through regions of the visual cortex
    - Example: For face recognition: edge → nose → face, layer-by-layer
- Popular Deep Learning Frameworks for Classification
  - Deep Feedforward Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks

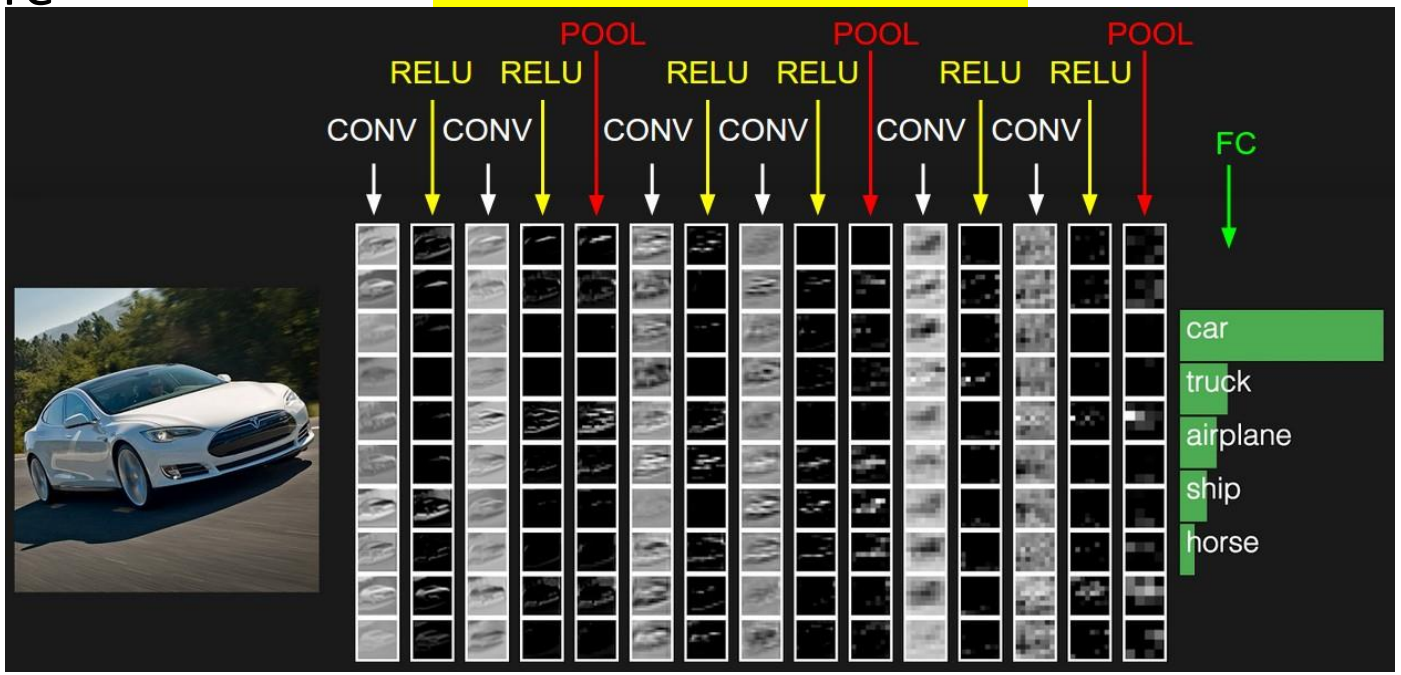
Deep (Feed Forward) Neural Networks

- How multiple layers work to build an improved feature space?
  - First layer learns 1st order features (e.g., edges, …)
  - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
  - In Deep Belief Networks (DBNs), layers often learn in an unsupervised mode and discover general features of the input space—serving multiple tasks related to the unsupervised instances (image recognition, etc.)
  - Then final layer features are fed into supervised layer(s)
    - And entire network is often subsequently tuned using supervised training of the entire net, using the initial weightings learned in the unsupervised phase
  - Could also do fully supervised versions (back-propagation)

Convolutional Neural Networks: General Architecture

- Learn high-order features in the data via convolutions
  - Well suited to object recognition with images (e.g., computer vision)
  - Build position- and (somewhat) rotation-invariant features from raw image data
- CNN leverages learnable visual filters and globally shared local features
  - Specifics: high dimensional, 2D topology of pixels, invariance to translations, etc.
- High-level general CNN architecture
  - Input layer
  - Feature-extraction layers (Convolution—ReLU—Pool)
  - Classification layers
- CNN properties
  - Local connectivity
  - Parameter sharing
  - Subsampling

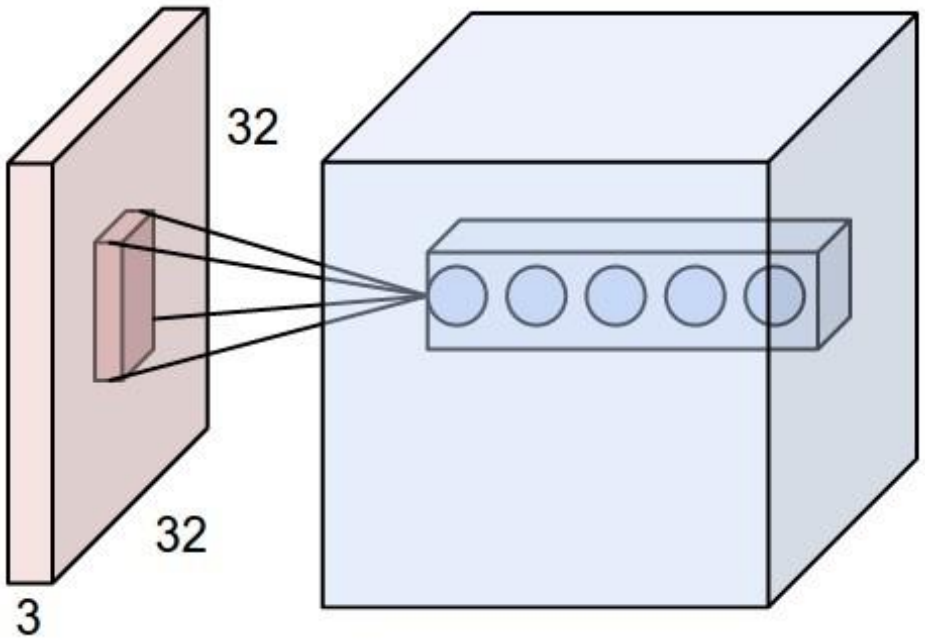
ReLU: Rectified Linear Unit



Convolutional Neural Networks: Local Connectivity

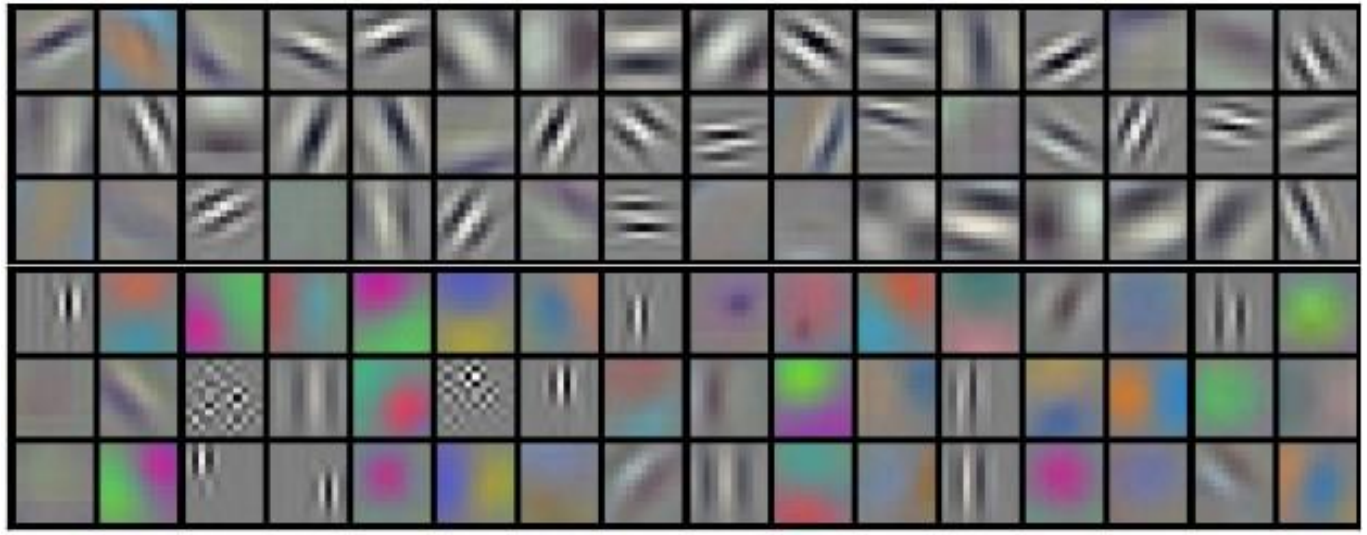
- Local Connectivity
  - Receptive fields: Each hidden unit is connected only to a sub-region of the image
    - Manageable number of parameters
    - Efficient computation of pre-activation
  - Spatial arrangements
    - Depth: Number of filters
    - Stride: how to slide the filter
    - Zero-padding: deal with the border

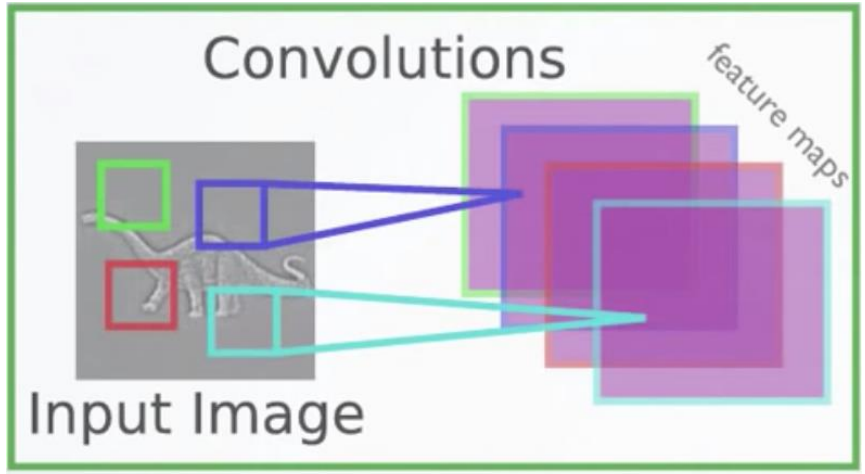
$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q}\, k_{r-p,r-q}$$



Convolutional Neural Networks: Parameter Sharing

- Parameter sharing
  - Discrete convolution: share matrix of parameters across certain units
    - Reduces even more the number of parameters
    - Extract the same feature at every position

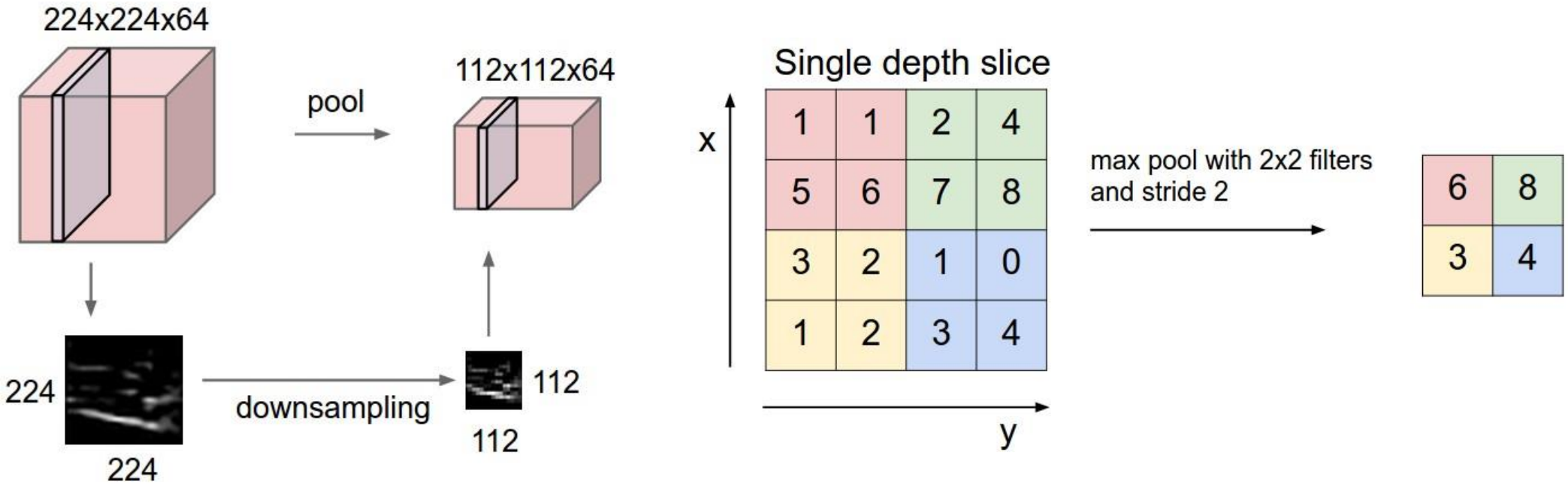




$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

Convolutional Neural Networks: Subsampling

- Subsampling
  - Pooling: pool hidden units in the same neighborhood
    - Introduces invariance to local translations
    - Reduces the number of hidden units in hidden layer



Recurrent Neural Networks: General Concepts

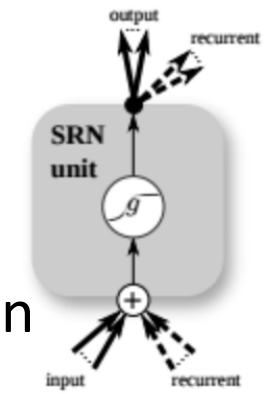
- Modeling the time dimension: by creating cycles in the network (thus "recurrent")
  - Adding feedback loops connected to past decisions
  - Long-term dependencies: Use hidden states to preserve sequential information
- RNNs are trained to generate sequences: Output at each timestamp is based on both the current input and the inputs at all previous timestamps

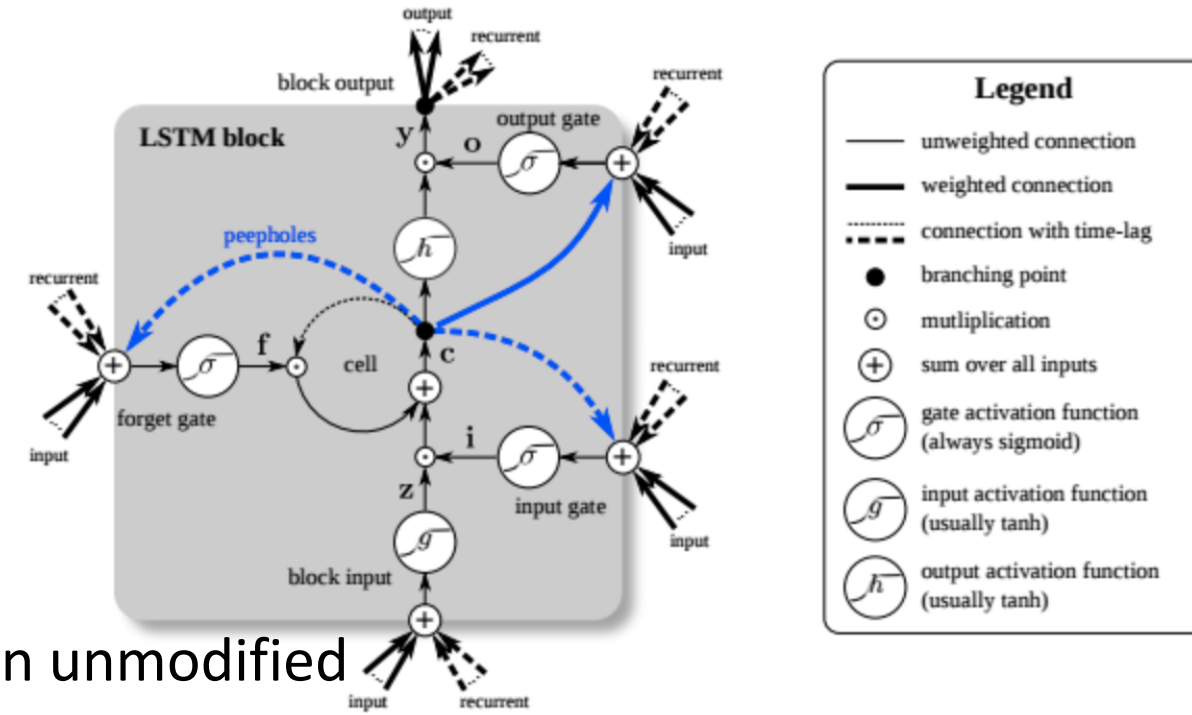
$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

  - Compute a gradient with the algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and Exploding Gradients
  - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
  - Solution: Use a variant of RNN: LSTM (1997, by Hochreiter and Schmidthuber)

LSTM: One Variant of Recurrent Neural Network

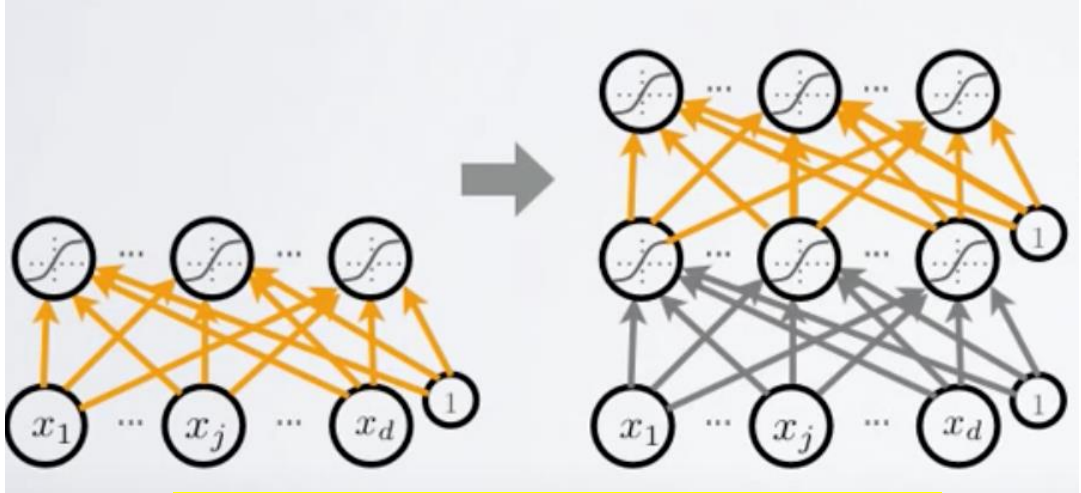
- Critical components of LSTM
  - Memory cells
  - 3 Gates (input, forget, output)
- Use gated cells to
  - Write, store, forget information
- When both gates are closed
  - The contents of the memory cell will remain unmodified
- The gating structure allows information to be retained across many timestamps
  - Also allows gradient to flow across many timestampls
- By back-propagating errors and adjusting weights, to learn what to store, and when to allow reads, writes and erasures
- Applications: Handling sequence and time series data
  - E.g., NLP, video analysis, image captioning, robotics control





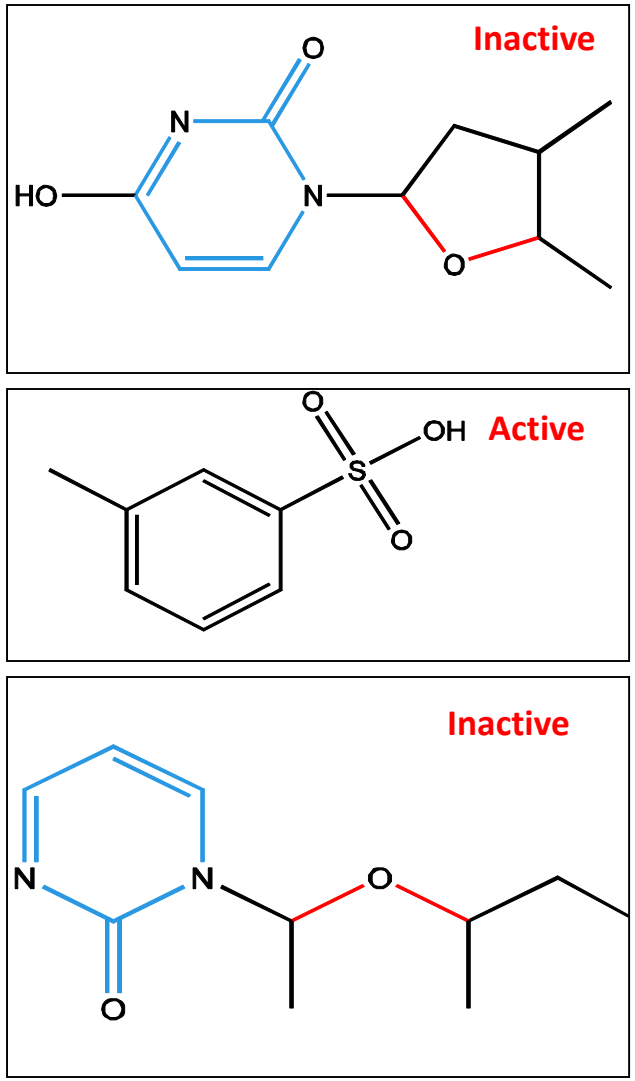
Difficulties of Training and Improvements

- Vanishing gradient problem: Saturated units block gradient propagation
  - Need better optimization (than SGD)
- Overfitting: high variance/low bias situation
  - Better regularization (than L1, L2 norm)
    - Unsupervised pre-training
    - Statistical dropout
    - Other popular approaches
      - Batch normalization, residual networks, highway networks, attention, etc.



Pre-training of stacked autoencoders

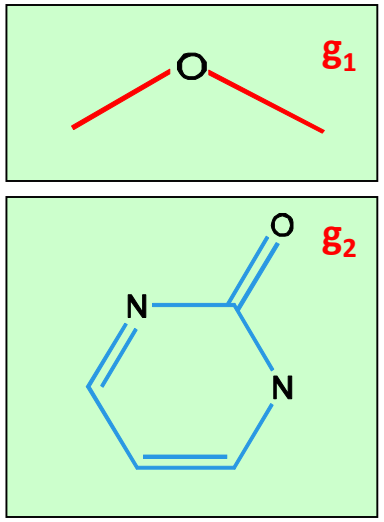
Pattern-Based Classification on Graphs



Mining

min_sup=2

Frequent subgraphs



Transform

Use frequent patterns as features for classification

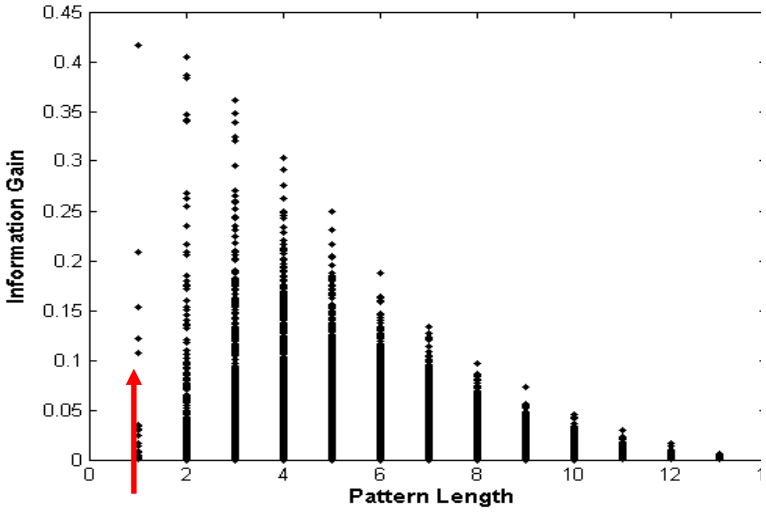
g_1	g_2	Class
1	1	0
0	0	1
1	1	0

Discriminative Pattern-Based Classification

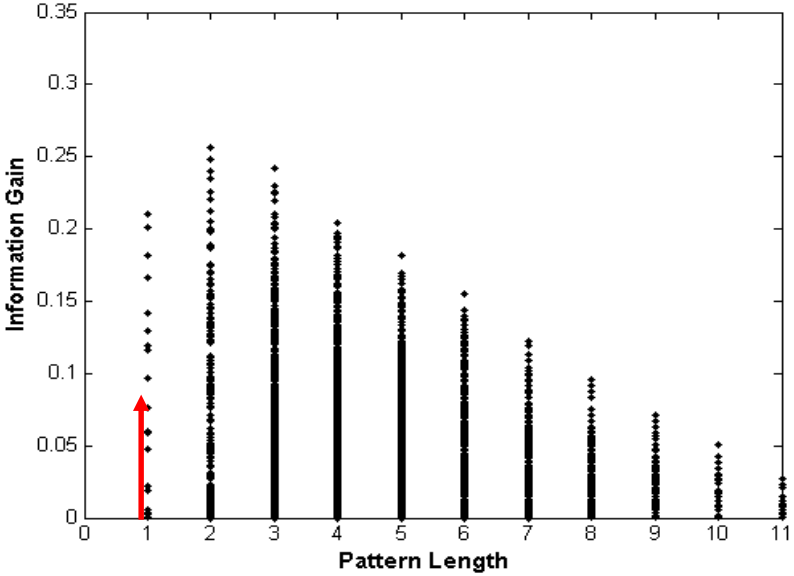
- Discriminative patterns as features for classification [Cheng et al., ICDE'07]
- **Principle:** Mining discriminative frequent patterns as high-quality features and then apply any classifier
- **Framework (PatClass)**
  - Feature construction by *frequent itemset mining*
  - Feature selection (e.g., using Maximal Marginal Relevance (MMR))
    - Select discriminative features (i.e., that are relevant but minimally similar to the previously selected ones)
    - Remove redundant or closely correlated features
  - Model learning
    - Apply a general classifier, such as SVM or C4.5, to build a classification model

On the Power of Discriminative Patterns

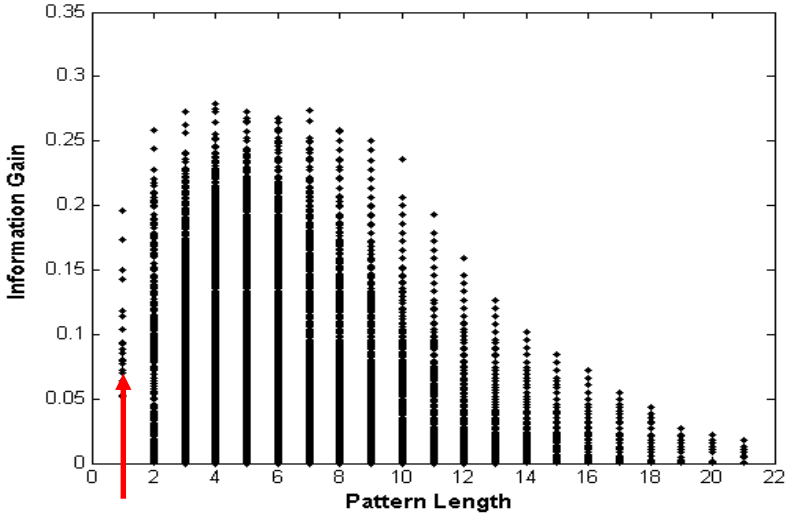
- K-itemsets are often more informative than single features (1-itemsets) in classification
- Computation on real datasets shows: The discriminative power of k-itemsets (for k > 1 but often ≤ 10) is higher than that of single features



(a) Austral



(b) Cleve

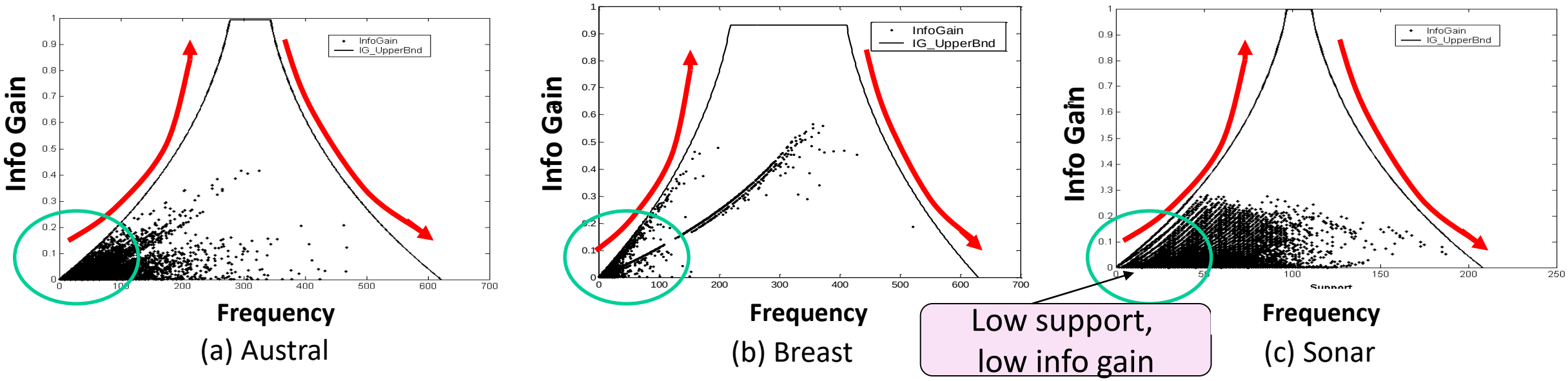


(c) Sonar

Information Gain vs. Pattern Length

Information Gain vs. Pattern Frequency

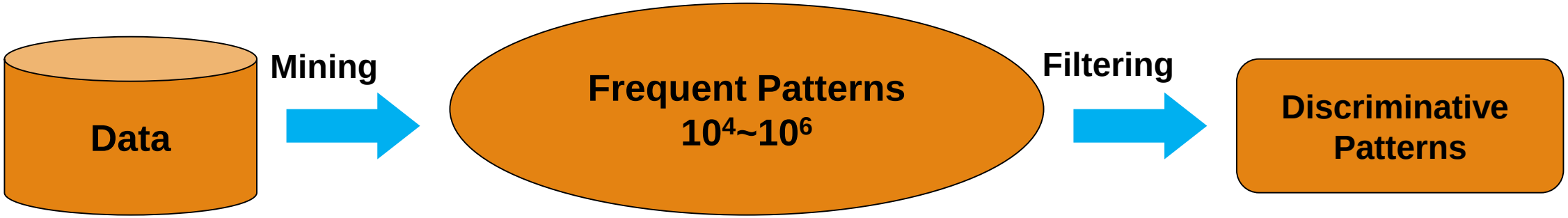
- Computation on real datasets shows: Pattern frequency (but not too frequent) is strongly tied with the discriminative power (information gain)
- Information gain upper bound monotonically increases with pattern frequency



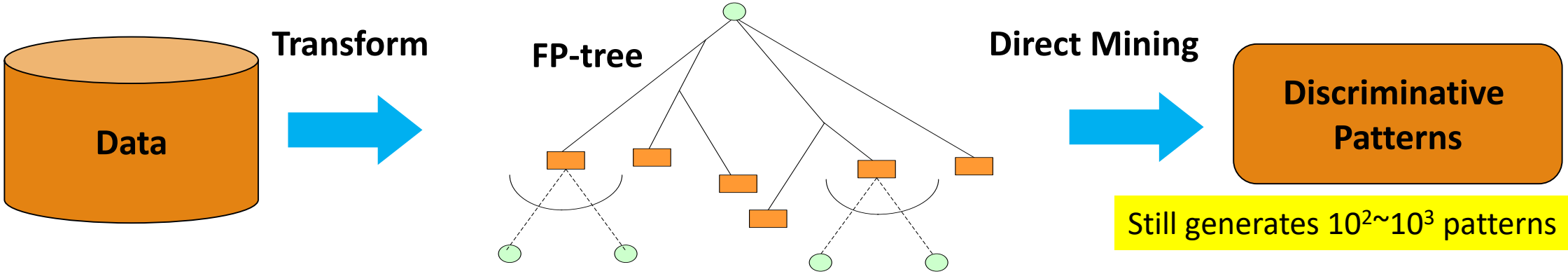
(a) Austral (b) Breast (c) Sonar

Mining Concise Set of Discriminative Patterns

Frequent pattern mining, then getting discriminative patterns: Expensive, large model



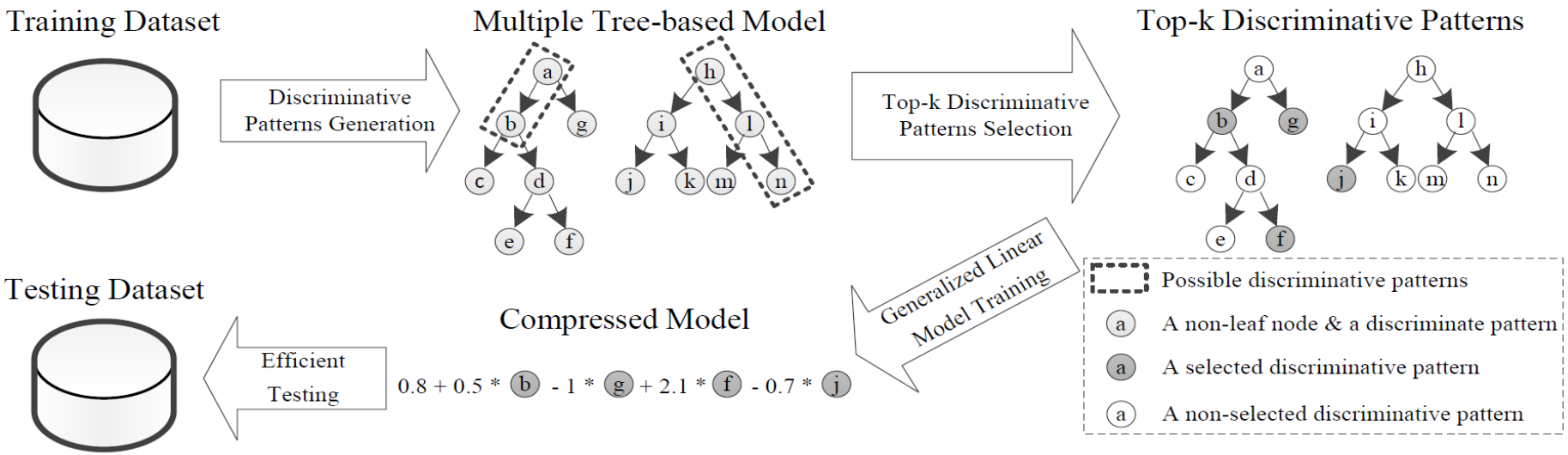
DDPMine [Cheng et al., ICDE'08]: Direct mining of discriminative patterns: Efficient



Still generates 10^2~10^3 patterns

DPClass [Shang et al, SDM'16]: A better solution—Efficient, effective, and generating a very limited number of (such as only 20 or so) patterns

DPClass: Discriminative Pattern-based Classification



Input: A feature table for training data

- Adopt every prefix path in an (extremely) random forest as a candidate pattern
  - The split points of continuous variables are automatically chosen by random forest → No discretization!
- Run top-k (e.g., top-20) pattern selection based on training data
- Train a generalized linear model (e.g., logistic regression) based on "bag-of-patterns" representations of training data

Explanatory Discriminative Patterns: Generation

- Example: For each patient, we have several uniformly sampled features as follows

Features	Age	Gender	Lab Test 1 (LT1)	Lab Test 2(LT2)
Values	Positive Integers	Male or Female	A, B, O, AB	Real value in [0, 1]

- The positive label of the hypo-disease will be given when at least one of the following rules holds

Features	Age	Gender	Lab Test 1 (LT1)	Lab Test 2(LT2)
Rule 1	> 18	Male	AB	>= 0.6
Rule 2	> 18	Female	O	>= 0.5
Rule 3	<= 18			>= 0.9

  - Training: 10^5 random patients + 0.1% noise
  - Flip the binary labels with 0.1% probability
- Testing: 5×10^4 random patients in test

Explanatory Discriminative Patterns: Evaluation

- Ground Truth:

Features	Age	Gender	Lab Test 1 (LT1)	Lab Test 2(LT2)
Rule 1	> 18	Male	AB	>= 0.6
Rule 2	> 18	Female	O	>= 0.5
Rule 3	<= 18			>= 0.9

- Top-3 Discriminative Patterns for each model:
  - **DPClass (perfect):**
    - (age > 18) and (gender = Female) and (LT1 = O) and (LT2 ≥ 0.496)
    - (age ≤ 18) and (LT2 ≥ 0.900)
    - (age > 18) and (gender = Male) and (LT1 = AB) and (LT2 ≥ 0.601)
  - DDPMine (poor):
    - (LT2 > 0.8)
    - (gender = Male) and (LT1 = AB) and (LT2 ≥ 0.6) and (LT2 < 0.8)
    - (gender = Female) and (LT1 = O) and (LT2 ≥ 0.6) and (LT2 < 0.8)

A Comparison on Classification Accuracy

- DPClass: Discriminative & frequent at the same time, then select top-k
- Two methods on pattern selection
  - Forward vs. LASSO
- In comparison with DDPMine and Random Forest, DPClass maintains high accuracy

	Dataset	DPClass (Forward)	DPClass (LASSO)	DDPMine	Random Forest
low-dimensional data	adult	**85.66%**	84.33%	83.42%	85.45%
	hypo	**99.58%**	99.28%	92.69%	97.22%
	sick	98.35%	**98.87%**	93.82%	94.03%
	crx	**89.35%**	87.96%	87.96%	**89.35%**
	sonar	**85.29%**	83.82%	73.53%	83.82%
	chess	92.25%	92.05%	90.04%	**94.22%**
high-dimensional data	namao	97.17%	96.94%	96.83%	**97.86%**
	musk	95.92%	95.71%	93.29%	**96.60%**
	madelon	74.50%	**76.00%**	59.84%	56.50%