CS 412 Intro. to Data Mining

Chapter 9. Classification: Advanced Methods

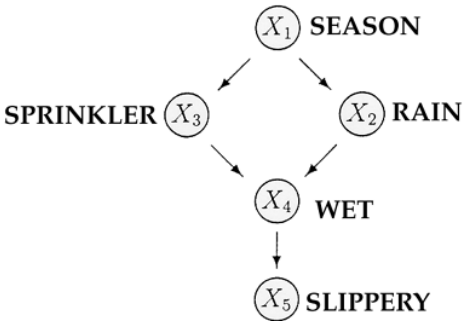
Jiawei Han, Computer Science, Univ. Illinois at Urbana-Champaign, 2017

Chapter 9. Classification: Advanced Methods

- 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

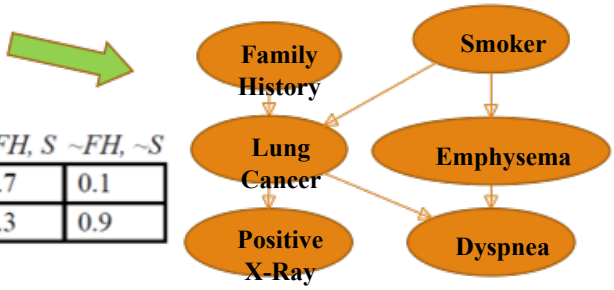
- Naïve Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable
  - This assumption is often too simple to model the real world well
- Bayesian network (or Bayes network, belief network, Bayesian model or probabilistic 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)
  - Ex. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases



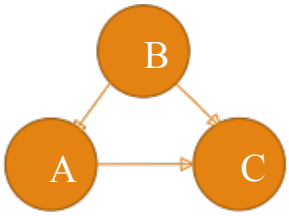
Bayesian Belief Networks

- **Bayesian belief network** (or **Bayesian network**, **probabilistic network**):
  - Allows *class conditional independencies* betwe
- Two components:
  - A *directed acyclic graph* (called a structure)
  - A set of *conditional probability tables* (CPTs)
- *Directed Acyclic Graph* (DAG):
  - Represents dependency among the variables (*causal influence* relationship)
  - Gives a specification of joint probability distribution

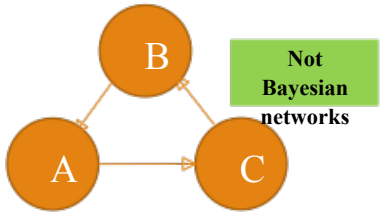
Nodes: random variables Links: dependency



	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

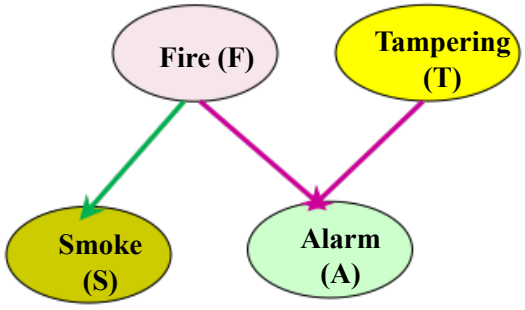


directed **acyclic** graphical



directed **cyclic** graphical model

A Bayesian Network and Its CPTs



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

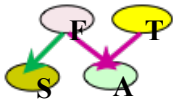
Fire	Smoke	Θs|f
True	True	.90
False	True	.01

Fire	Tampering	Alarm	Θa|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:

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



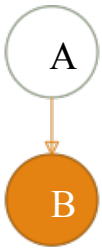
  - HMM (Hidden Markov Model): often used to model dynamic systems whose states are not observable, yet their outputs are
- **Synthesis from other specifications**
  - E.g., from a formal system design: block diagrams & info flow
- **Learning from data** (e.g., from medical records or student admission record)
  - Learn parameters give its structure or learn both structure and parms
  - Maximum likelihood principle: favors Bayesian networks that maximize the probability of observing the given data set

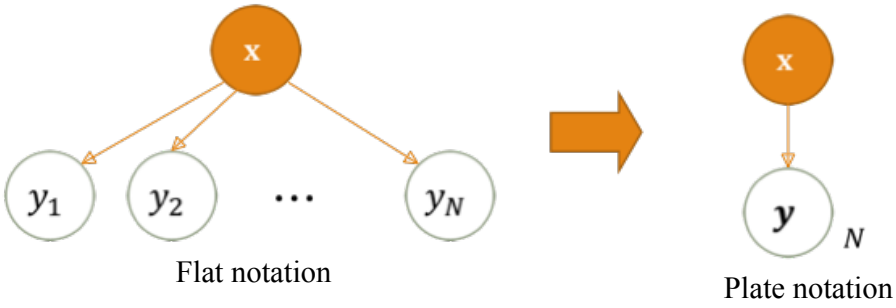
Training Bayesian Networks: Several Scenarios

- 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
- 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

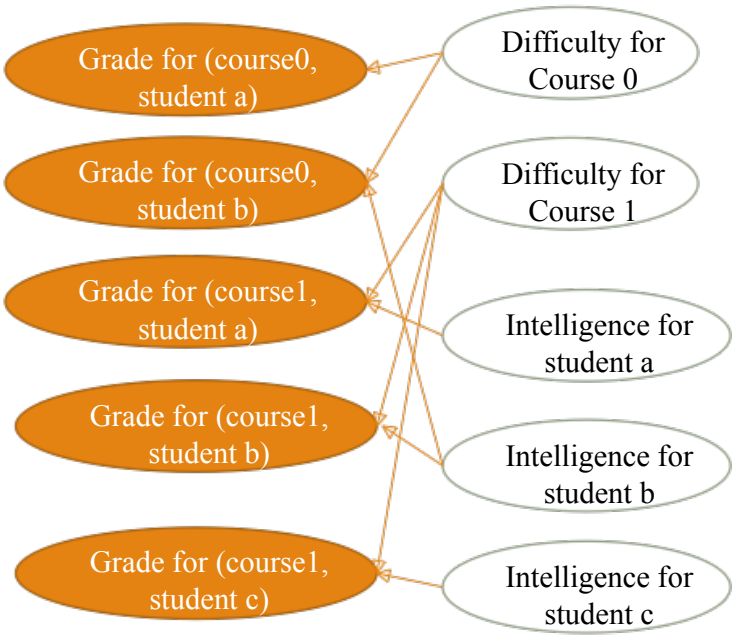
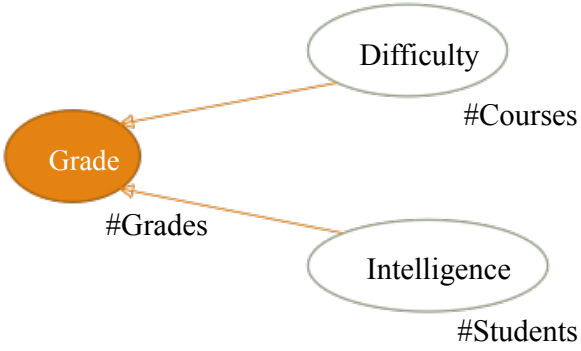


Plate notation

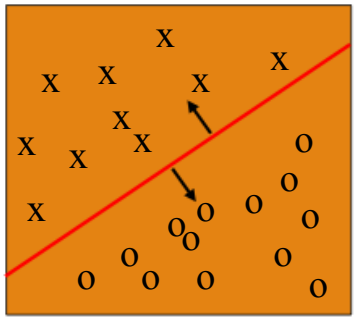


Chapter 9. Classification: Advanced Methods

- 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

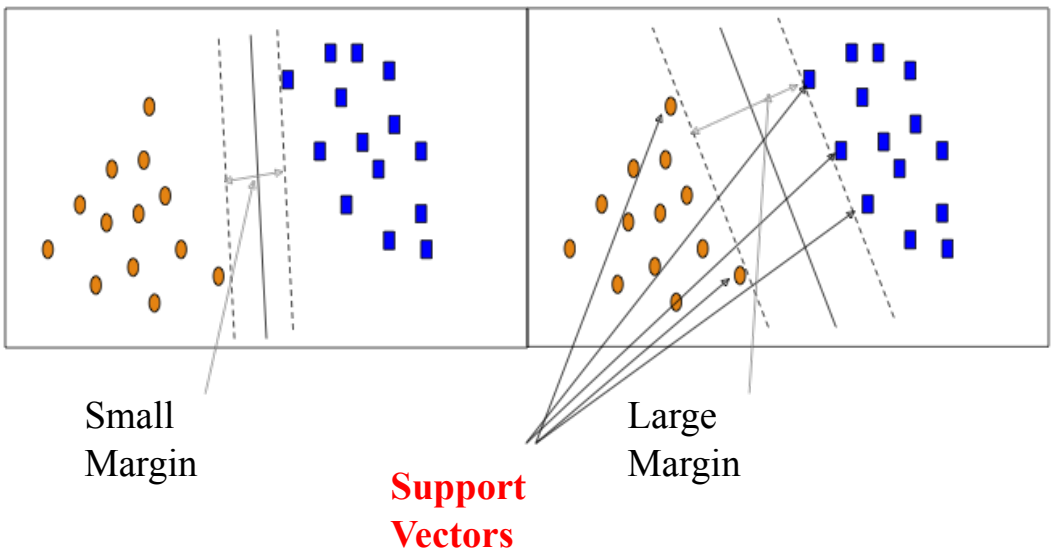
- **Classification:** Predicts categorical class labels
  - E.g., Personal homepage classification
    - xi = (x1, x2, x3, …), yi = +1 or –1
    - x1 : # of word “homepage”
    - x2 : # of word “welcome”
- Mathematically, x □ X = □n, y □ Y = {+1, –1},
  - We want to derive a function f: X □ Y
- Linear Classification
  - Binary Classification problem
  - Data above the red line belongs to class ‘x’
  - Data below red line belongs to class ‘o’
  - Examples: SVM, Perceptron, Probabilistic Classifiers



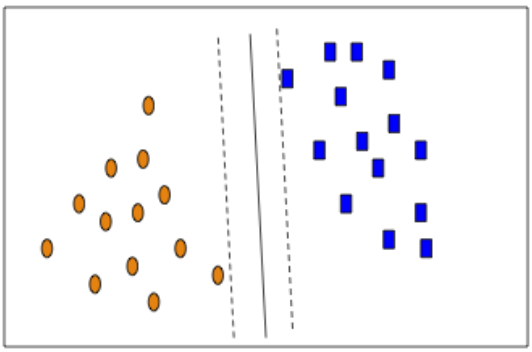
SVM—Support Vector Machines

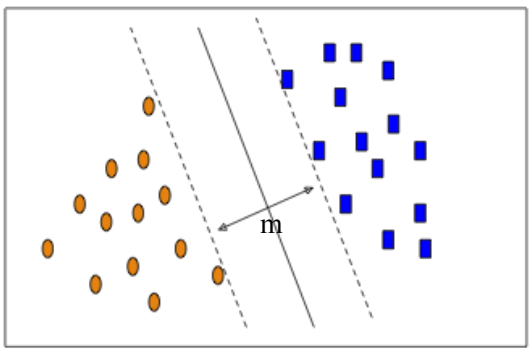
- A relatively new classification method for both linear and nonlinear data
  - Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, 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



SVM—When Data Is Linearly Separable





Let data D be (**X**1, y1), …, (**X**|D|, y|D|), where **X**i is the set of training tuples associated with the class labels yi

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

SVM—Linearly Separable

- A separating hyperplane can be written as

  W ● **X** + b = 0

  where **W**={w1, w2, …, wn} is a weight vector and b a scalar (bias)
- For 2-D, it can be written as: w0 + w1 x1 + w2 x2 = 0
- The hyperplane defining the sides of the margin:

  H1: w0 + w1 x1 + w2 x2 ≥ 1 for yi = +1, and

  H2: w0 + w1 x1 + w2 x2 ≤ – 1 for yi = –1
- Any training tuples that fall on hyperplanes H1 or H2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem:
  - Quadratic objective function and linear constraints □ *Quadratic Programming (QP)* □ Lagrangian multipliers

SVM—Linearly Inseparable

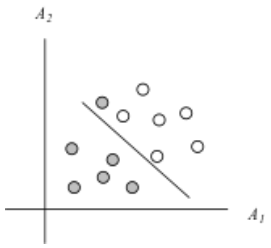
- Transform the original input data into a higher dimensional space

Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3, \phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new ($\mathbf{Z}$) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned}$$

■

- Search for a linear separating hyperplane in the new space



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(**Xi**, **Xj**) to the original data, i.e.,
  - K(**Xi**, **Xj**) = Φ(**Xi**) Φ(**Xj**)
- Typical Kernel Functions

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

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

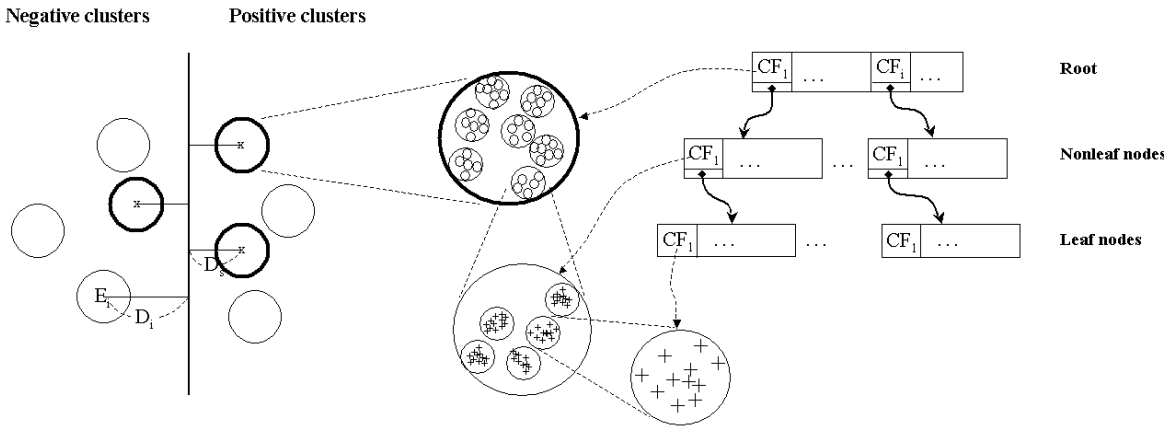
$$\textbf{Sigmoid kernel}: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \tanh(\kappa \boldsymbol{X_i} \cdot \boldsymbol{X_j} - \delta)$$

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

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

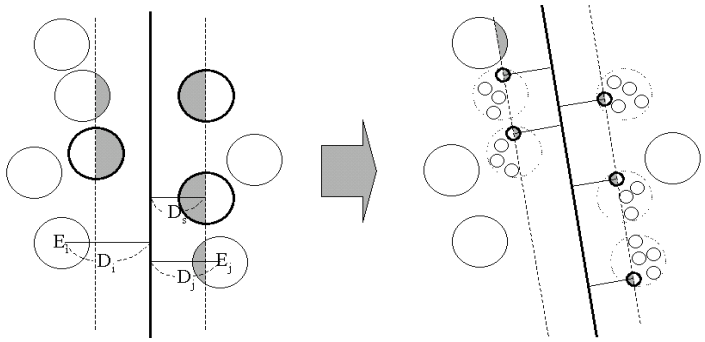
Scaling SVM by Hierarchical Micro-Clustering



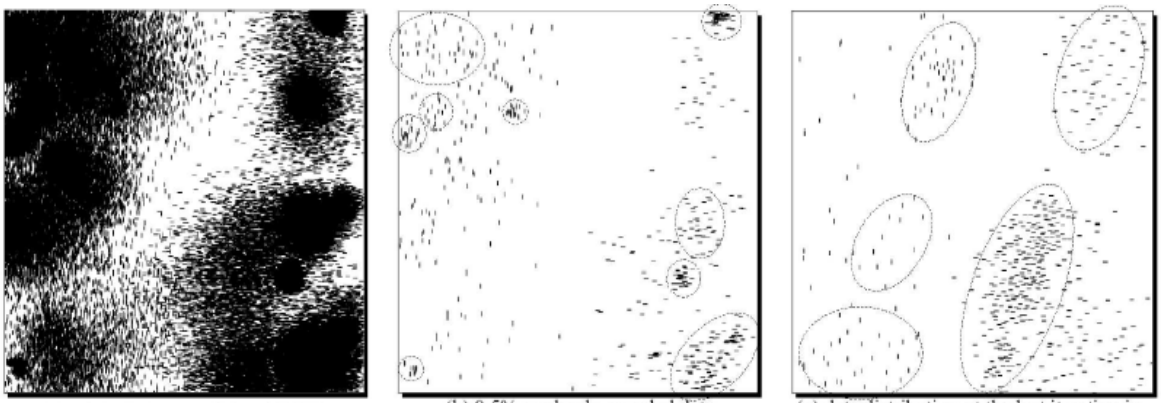
- Construct two CF-trees (i.e., statistical summary of the data) from positive and negative data sets independently (with one scan of the data set)
- Micro-clustering: Hierarchical indexing structure
  - Provide finer samples closer to the boundary and coarser samples farther from the boundary

Selective Declustering: Ensure High Accuracy

- De-cluster only the cluster Ei such that
  - Di – Ri < Ds, where Di is the distance from the boundary to the center point of Ei and Ri is the radius of Ei
  - 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

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 Related Links

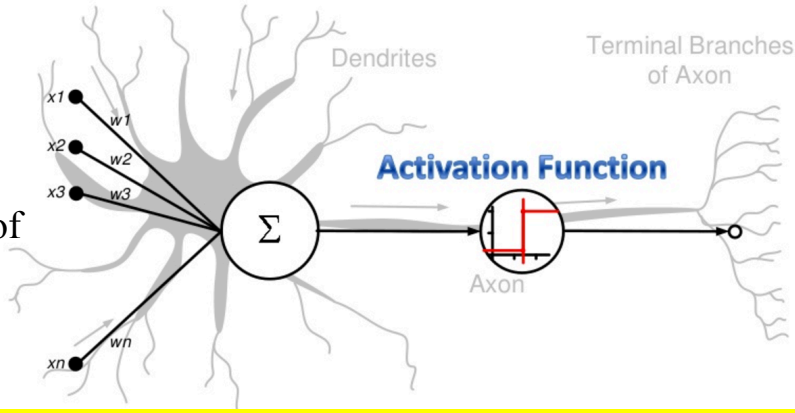
- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than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

Chapter 9. Classification: Advanced Methods

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

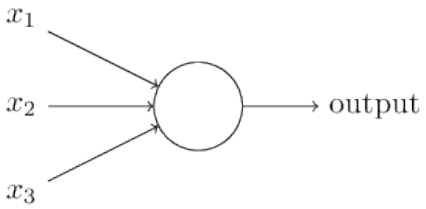
Neural Network for Classification

- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- 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

Perceptron: Predecessor of a Neural Network



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

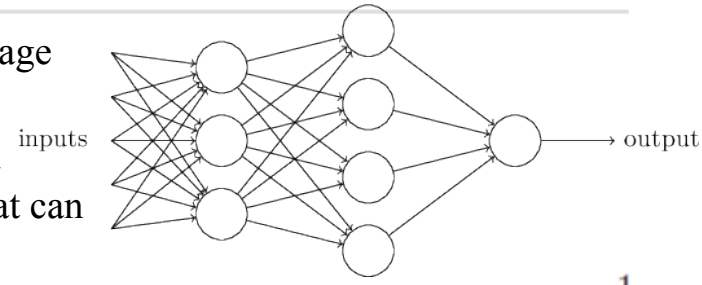
- The perceptron algorithm: invented in 1957 by Frank Rosenblatt
- Input: An *n*-dimensional input vector **x** (with n variables)
- Output: 1 or 0 depending on if the weighted sum passes a threshold
- Perceptron: A device that makes decisions by weighing up evidence
- Often written in the vector form, using bias (*b*) instead of threshold, as

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

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

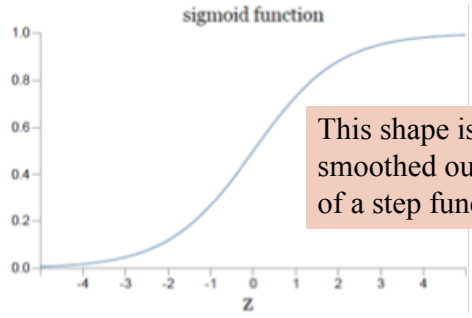
Sigmoid Neurons

- 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 simoid function: σ(*w*•*x* + b) , i.e.,
- The smoothness of σ means that small changes in the Δwj weights and in the Δb bias will produce a small change Δoutput in the output from the neuron



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

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



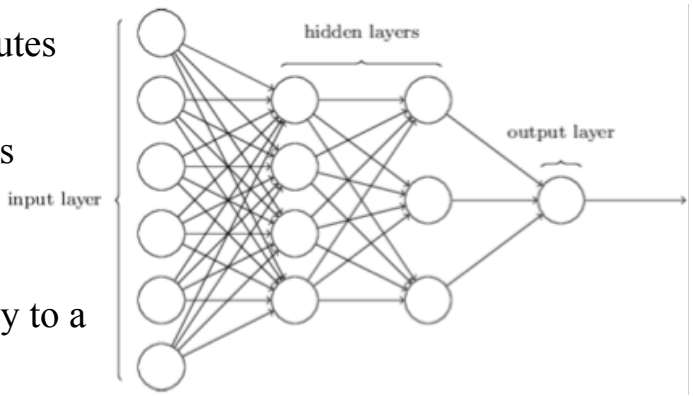
This shape is a smoothed out version of a step function

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

i.e., Δoutput is a *linear function* of the changes Δwj and Δb

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

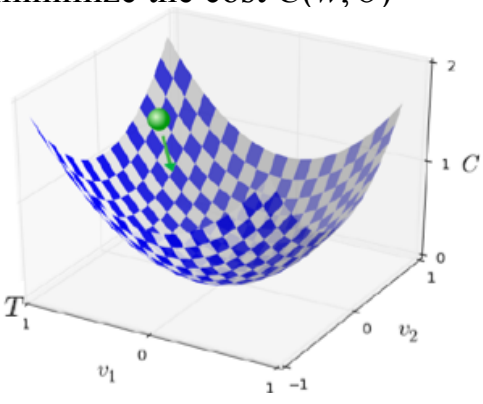
- A quadratic cost (objective) function C (or mean square error, MSE)

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

  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 Δ*v*1 and Δ*v*2 to make Δ*C* negative; i.e., the ball is rolling down into the vall

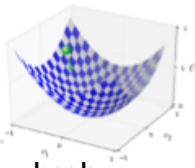
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$



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

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

  where ∇C is the gradient vector: $\nabla C \equiv \left(\frac{\partial C}{\partial v_1},\ldots,\frac{\partial C}{\partial v_m}\right)^T$



Stochastic Gradient Descent

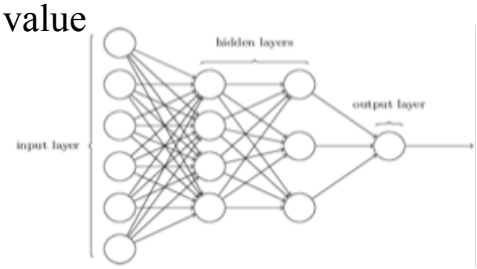
- Gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease *C*
- To compute gradient ∇C, we need to compute the gradients ∇Cx separately for each training input, x, and then average them: slow when the # of training inputs is large
- *Stochastic 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

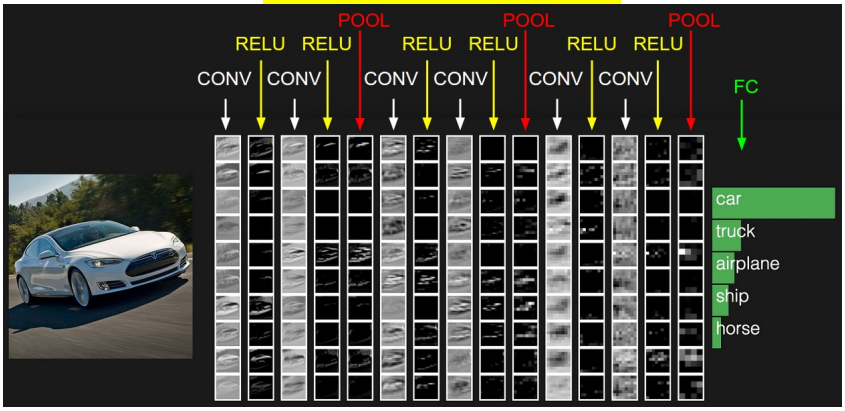
Deep (Feed Forward) Neural Networks

- 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

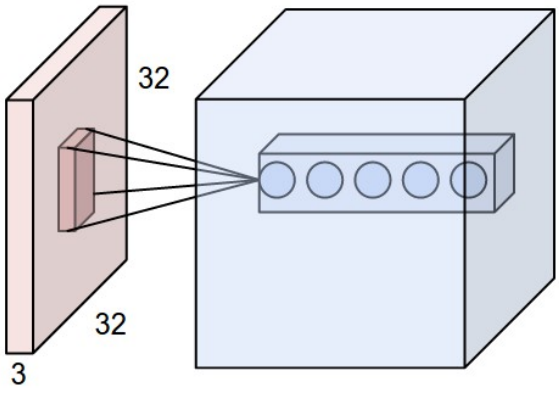
ReLU: Rectified Linear Unit



Convolutional Neural Networks: Local Connectivity

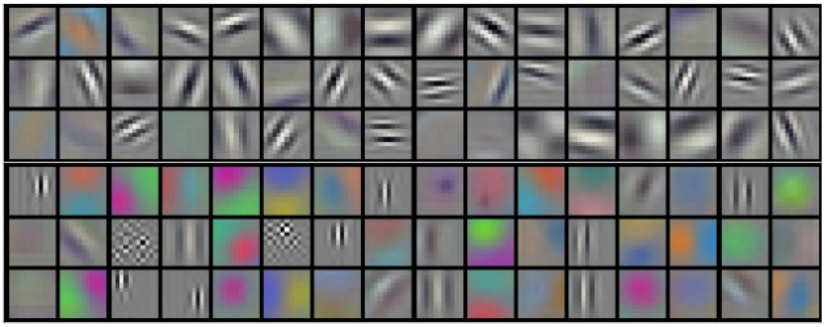
- Local Connectivity
  - Receptive fields: Each hidden unit is connected only to a sub-region of the image
    - Manageable number of parameters
    - Efficient computation of pre-activation
  - Spatial arrangements
    - Depth: Number of filters
    - Stride: how to slide the filter
    - Zero-padding: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

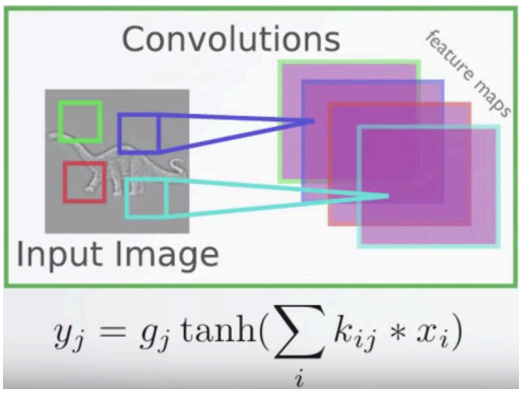
$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q}\, k_{r-p,r-q}$$



Convolutional Neural Networks: Parameter Sharing

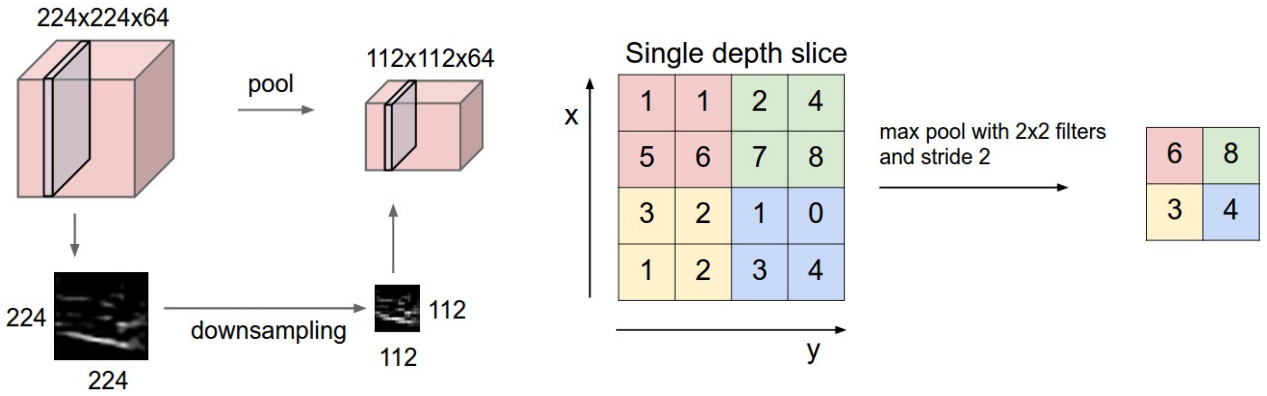
- Parameter sharing
  - Discrete convolution: share matrix of parameters across certain units
    - Reduces even more the number of parameters
    - Extract the same feature at every position





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

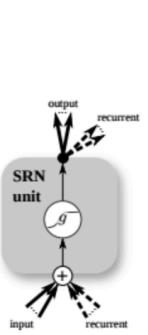
- Modeling the time dimension: by creating cycles in the network (thus "recurrent")
  - Adding feedback loops connected to past decisions
  - Long-term dependencies: Use hidden states to preserve sequential information
- RNNs are trained to generate sequences: Output at each timestamp is based on both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

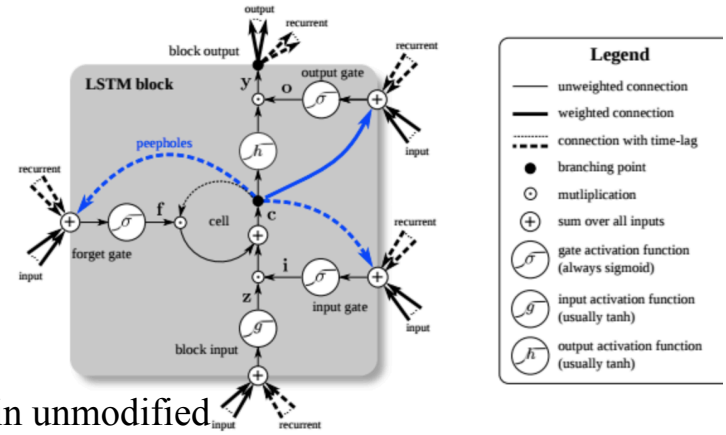
$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

  - 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

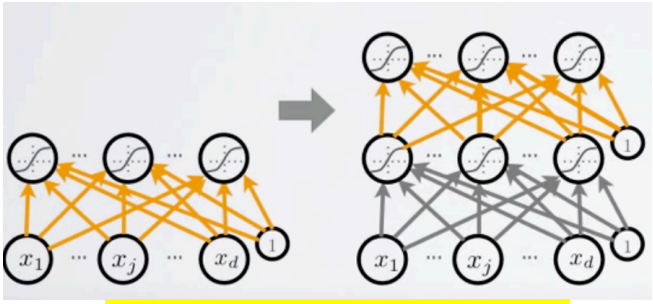




- The gating structure allows information to be retained across many timestamps
  - Also allows gradient to flow across many timestampls
- By back-propagating errors and adjusting weights, to learn what to store, and when to allow reads, writes and erasures
- Applications: Handling sequence and time 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

Chapter 9. Classification: Advanced Methods

- 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

R1: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes

  - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - ncovers = # of tuples covered by R1
  - ncorrect = # of tuples correctly classified by R1

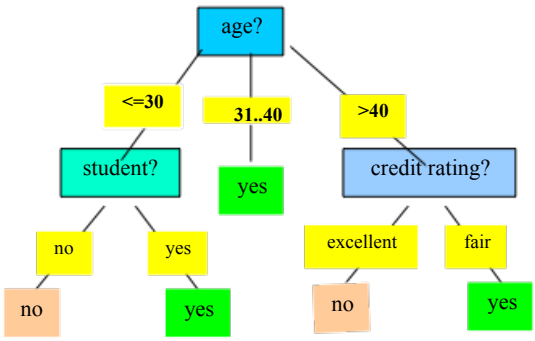
coverage(R1) = ncovers /|D| /* D: training data set */

accuracy(R1) = ncorrect / ncovers

- 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,

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*

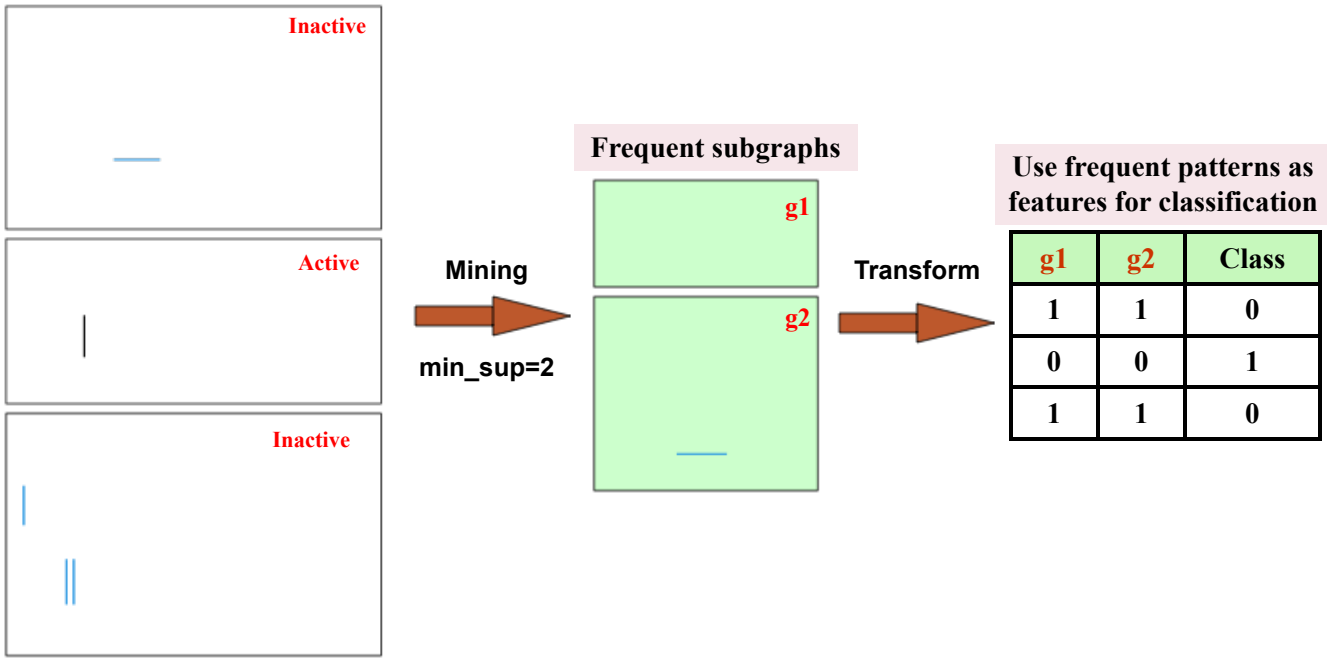
Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class Ci will cover many tuples of Ci but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - 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 user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

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

Pattern-Based Classification on Graphs



Associative or Pattern-Based Classification

- **Data:** Transactions, microarray data, … → **Patterns or association rules**
- **Classification Methods** (Some interesting work):
  - CBA [Liu, Hsu & Ma, KDD'98]: Use high-conf., high-support *class association rules* to build classifiers To be discussed here
  - Emerging patterns [Dong & Li, KDD'99]: Patterns whose support changes significantly between the two classes
  - CMAR [Li, Han & Pei, ICDM'01]: Multiple rules in predicti To be discussed here
  - CPAR [Yin & Han, SDM'03]: Beam search on multiple prediction rules
  - RCBT [Cong et al., SIGMOD'05]: Build classifier based on mining top-k covering rule groups with row enumeration (for high-dimensional data)
  - Lazy classifier [Veloso, Meira & Zaki, ICDM'06]: For a test t, project training data D on t, mine rules from Dt , predict on the best rule
  - Discriminative pattern-based classification [Cheng et al., ICDE'07] To be discussed here

CBA: Classification Based on Associations

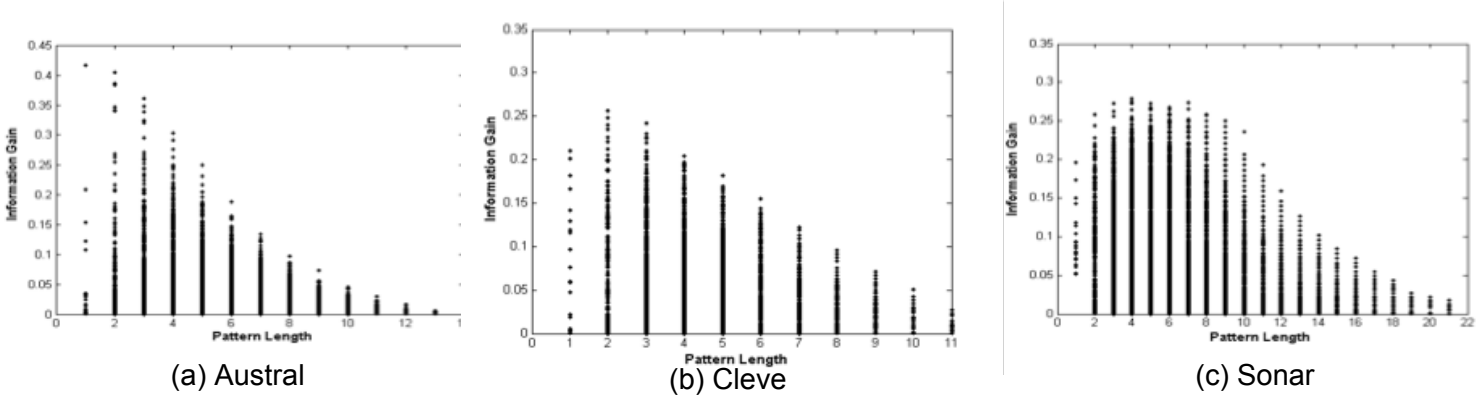
- CBA [Liu, Hsu and Ma, KDD'98]
- Method
  - Mine high-confidence, high-support class association rules
  - LHS: conjunctions of attribute-value pairs); RHS: class labels
    p1 ^ p2 … ^ pl □ "Aclass-label = C" (confidence, support)
  - Rank rules in descending order of confidence and support
  - Classification: 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

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

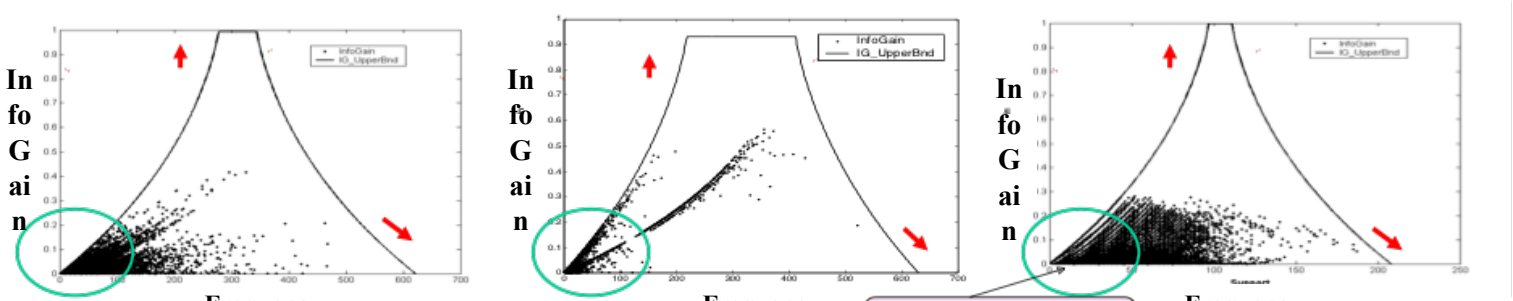


(a) Austral (b) Cleve (c) Sonar

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

Low support, low info gain

Information Gain Formula:

$$IG(C \mid X) = H(C) - H(C \mid X)$$

Entropy of given data

$$H(C) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Conditional entropy of study focus

$$H(C \mid X) = \sum_j P(X = x_j) H(Y \mid X = x_j)$$

Discriminative Pattern-Based Classification: Experimental Results

Table 1. Accuracy by SVM on Frequent Combined Features vs. Single Features

Data	Single Feature			Freq. Pattern	
	Item_All	*Item_FS*	*Item_RBF*	*Pat_All*	*Pat_FS*
anneal	**99.78**	**99.78**	99.11	99.33	99.67
austral	85.01	85.50	85.01	81.79	**91.14**
auto	83.25	84.21	78.80	74.97	**90.79**
breast	97.46	97.46	96.98	96.83	**97.78**
cleve	84.81	84.81	85.80	78.55	**95.04**
diabetes	74.41	74.41	74.55	77.73	**78.31**
glass	75.19	75.19	74.78	79.91	**81.32**
heart	84.81	84.81	84.07	82.22	**88.15**
hepatic	84.50	89.04	85.83	81.29	**96.83**
horse	83.70	84.79	82.36	82.35	**92.39**
iono	93.15	94.30	92.61	89.17	**95.44**
iris	94.00	**96.00**	94.00	95.33	**96.00**
labor	89.99	91.67	91.67	94.99	**95.00**
lymph	81.00	81.62	84.29	83.67	**96.67**
pima	74.56	74.56	76.15	76.43	**77.16**
sonar	82.71	86.55	82.71	84.60	**90.86**
vehicle	70.43	72.93	72.14	73.33	**76.34**
wine	98.33	99.44	98.33	98.30	**100**
zoo	97.09	97.09	95.09	94.18	**99.00**

Table 2. Accuracy by C4.5 on Frequent Combined Features vs. Single Features

Dataset	Single Features		Frequent Patterns	
	Item_All	*Item_FS*	*Pat_All*	*Pat_FS*
anneal	98.33	98.33	97.22	**98.44**
austral	84.53	84.53	84.21	**88.24**
auto	71.70	77.63	71.14	**78.77**
breast	95.56	95.56	95.40	**96.35**
cleve	80.87	80.87	80.84	**91.42**
diabetes	**77.02**	**77.02**	76.00	76.58
glass	75.24	75.24	76.62	**79.89**
heart	81.85	81.85	80.00	**86.30**
hepatic	78.79	85.21	80.71	**93.04**
horse	83.71	83.71	84.50	**87.77**
iono	92.30	92.30	92.89	**94.87**
iris	**94.00**	**94.00**	93.33	93.33
labor	86.67	86.67	**95.00**	91.67
lymph	76.95	77.62	74.90	**83.67**
pima	75.86	75.86	76.28	**76.72**
sonar	80.83	81.19	**83.67**	**83.67**
vehicle	70.70	71.49	**74.24**	73.06
wine	95.52	93.82	96.63	**99.44**
zoo	91.18	91.18	95.09	**97.09**

Discriminative Pattern-Based Classification: Scalability Tests

Table 3. Accuracy & Time on Chess Data

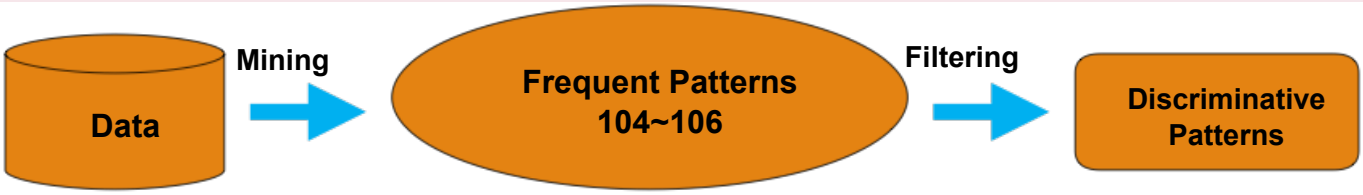
min_sup	#Patterns	Time (s)	SVM (%)	C4.5 (%)
1	N/A	N/A	N/A	N/A
2000	68,967	44.703	92.52	97.59
2200	28,358	19.938	91.68	97.84
2500	6,837	2.906	91.68	97.62
2800	1,031	0.469	91.84	97.37
3000	136	0.063	91.90	97.06

Table 4. Accuracy & Time on Waveform Data

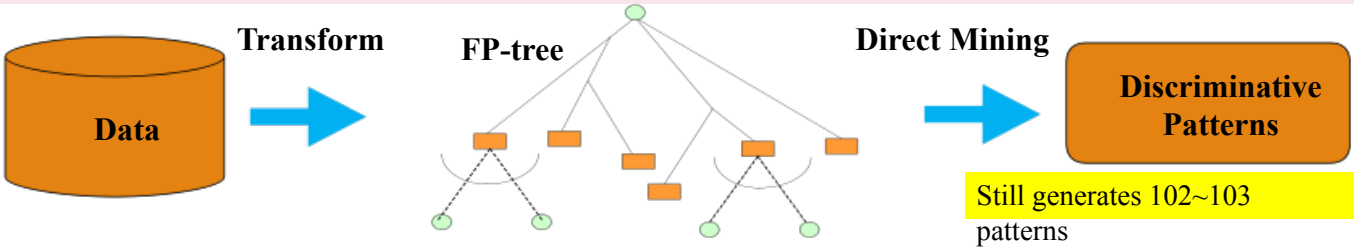
min_sup	#Patterns	Time (s)	SVM (%)	C4.5 (%)
1	9,468,109	N/A	N/A	N/A
80	26,576	176.485	92.40	88.35
100	15,316	90.406	92.19	87.29
150	5,408	23.610	91.53	88.80
200	2,481	8.234	91.22	87.32

Patterns

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



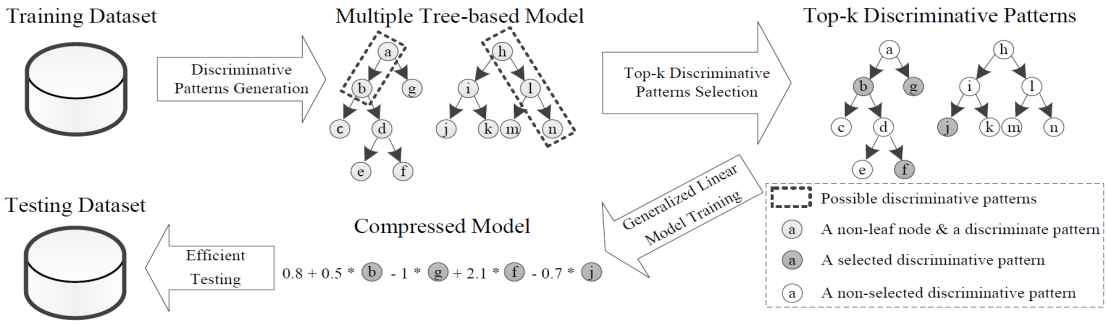
DDPMine [Cheng et al., ICDE'08]: Direct mining of discriminative patterns: Efficient



Still generates 102~103 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 s
  - Age (A): Positive Integers no more t
  - Gender (G): Male or Female.
  - Lab Test 1 (LT1): Categorical values
  - Lab Test 2 (LT2): Continuous values
- The positive label of the hypo-disease

Explanatory Discriminative Patterns: Evaluation

- Accuracy:
  - **DPClass 99.99% (perfect)**
  - DDPMine 95.64% (reasonable)
- Top-3 Discriminative Patterns:
  - **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
- Only top-20 patterns are used in DPClass
- 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%

- An extension of DPClass has been applied to health study
  - Cheng et al, "Mining Discriminative Patterns to Predict Health Status for Cardiopulmonary Patients", *ACM-BCB'16*

Chapter 9. Classification: Advanced Methods

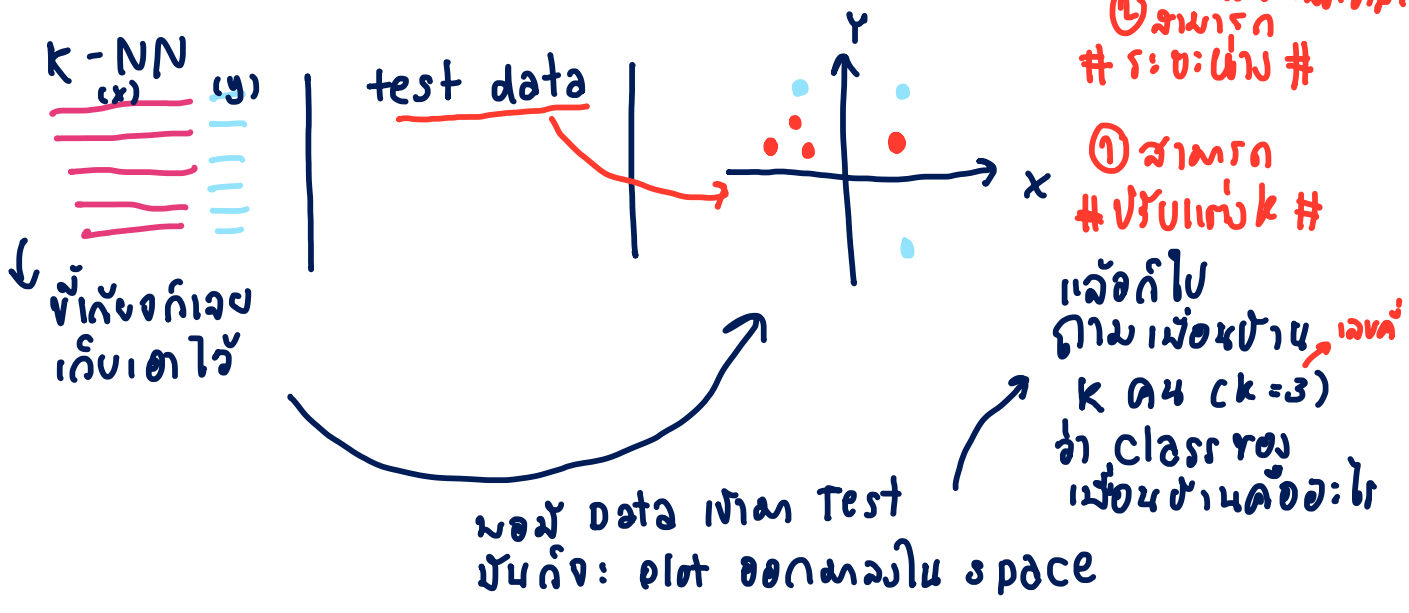
- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary

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

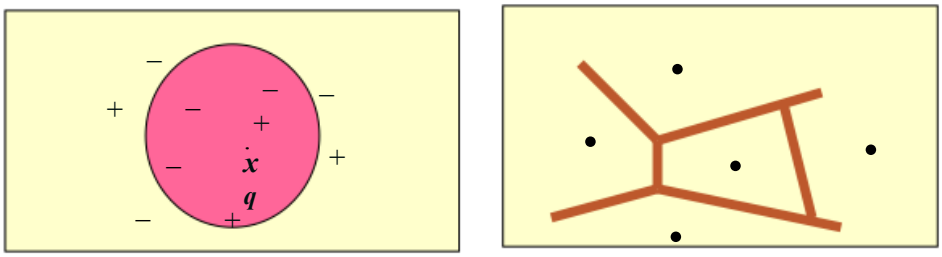
Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - 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(**X1**, **X2**)
- 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 *xq*
- 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 real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query *xq*
    - Give greater weight to closer neighbors

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

- Robust to noisy data by averaging *k*-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

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

- 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: based on an analogy to biological evolution
- An initial **population** is created consisting of randomly generated rules
  - Each rule is represented by a string of bits
  - E.g., if A1 and ¬A2 then C2 can be encoded as 100
  - If an attribute has k > 2 values, k bits can be used
- Based on the notion of survival of the **fittest**, a new population is formed to consist of the fittest rules and their offspring
- The *fitness of a rule* is represented by its classification accuracy on a set of training examples
- 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

Chapter 9. Classification: Advanced Methods

- 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)
  - genetic algorithms
  - rough set and fuzzy set

References (1)

- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- 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