CSE 5243 INTRO. TO DATA MINING

Classification (Basic Concepts & Advanced Methods)

Yu Su, CSE@The Ohio State University

Slides adapted from UIUC CS412 by Prof. Jiawei Han and OSU CSE5243 by Prof. Huan Sun

Classification: Advanced Methods

- Lazy Learners and K-Nearest Neighbors
- Neural Networks
- Support Vector Machines
- Additional Topics: Semi-Supervised Methods, Active Learning, etc.
- 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 previously 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.

  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

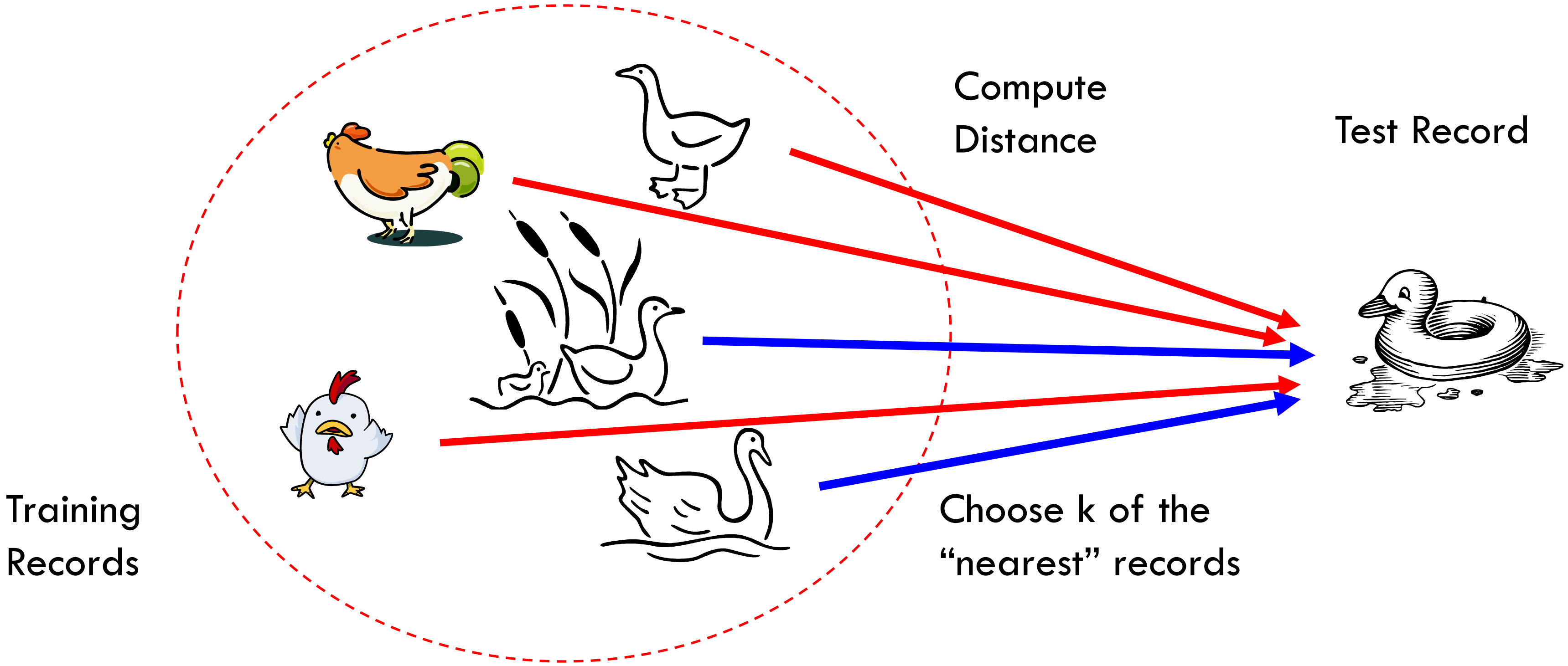
k-Nearest Neighbor (k-NN):

- Training method:
  - Save the training examples
- At prediction time:
  - Find the k training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example x
  - Classify x as the most frequent class among those y_i's.

- O(q) for each tuple to be classified. (Here q is the size of the training set.)

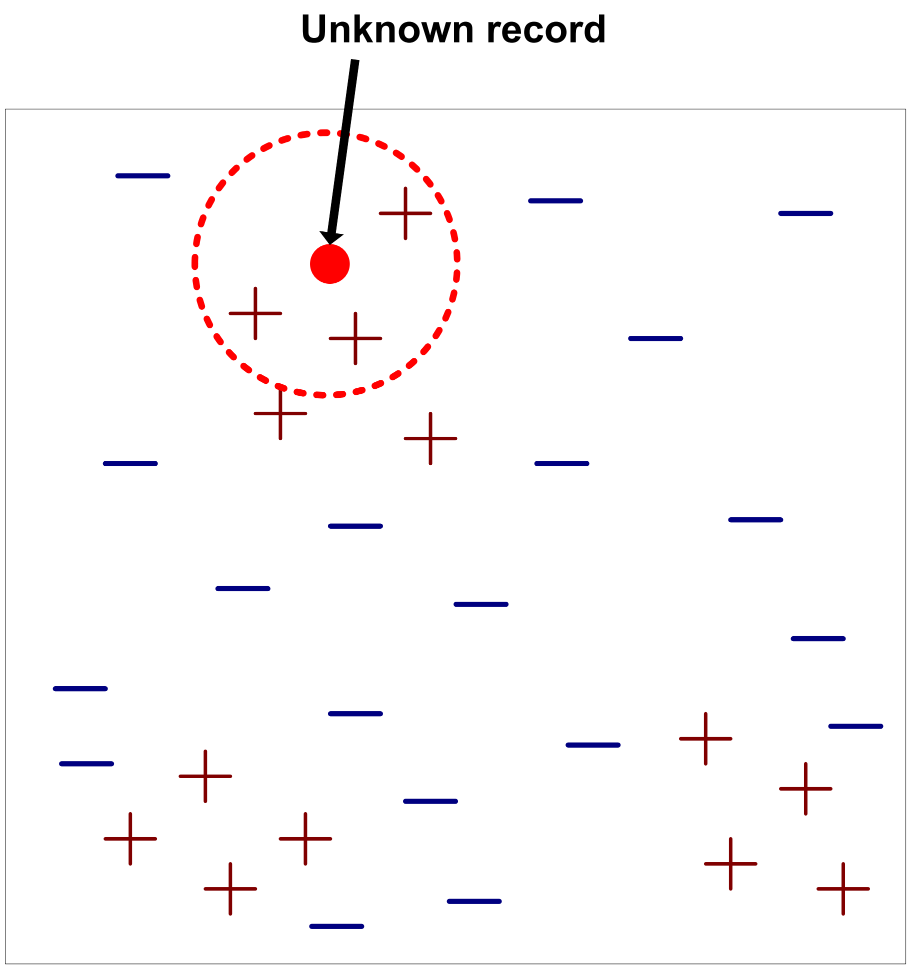
Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



Slides adapted from SlideShare

Nearest Neighbor Classifiers

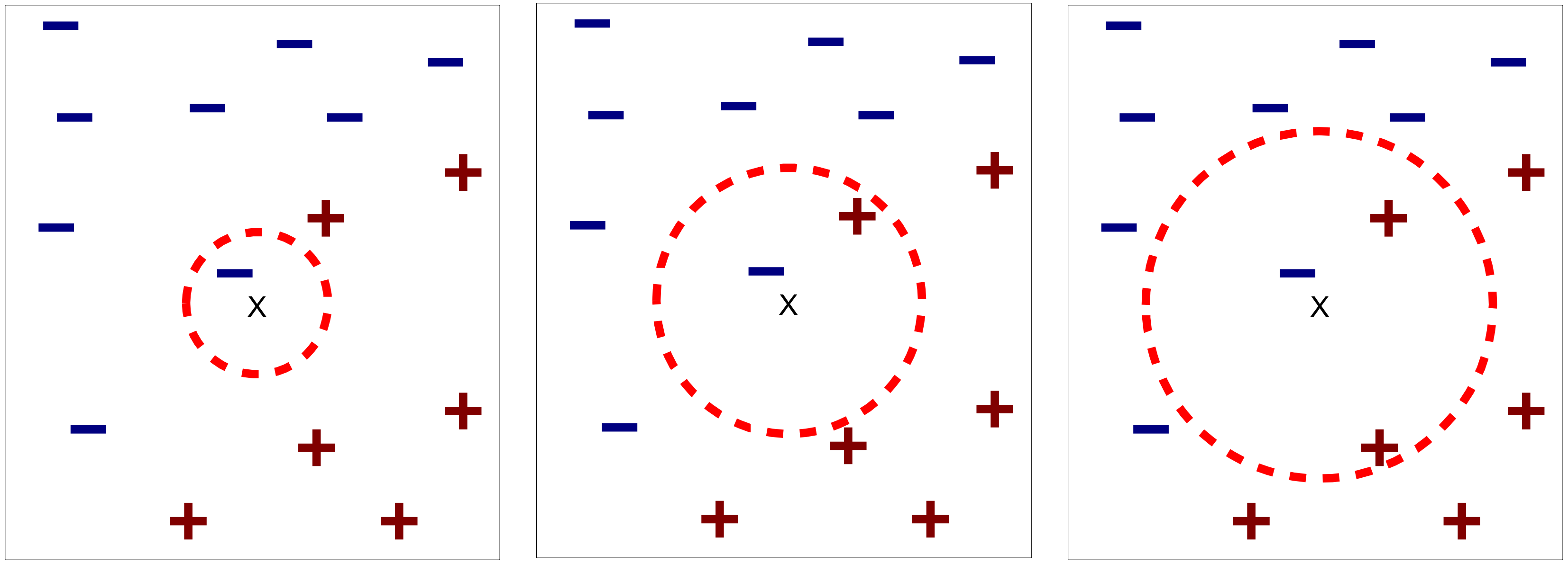


- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Slides adapted from SlideShare

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k shortest distance to x

Slides adapted from SlideShare

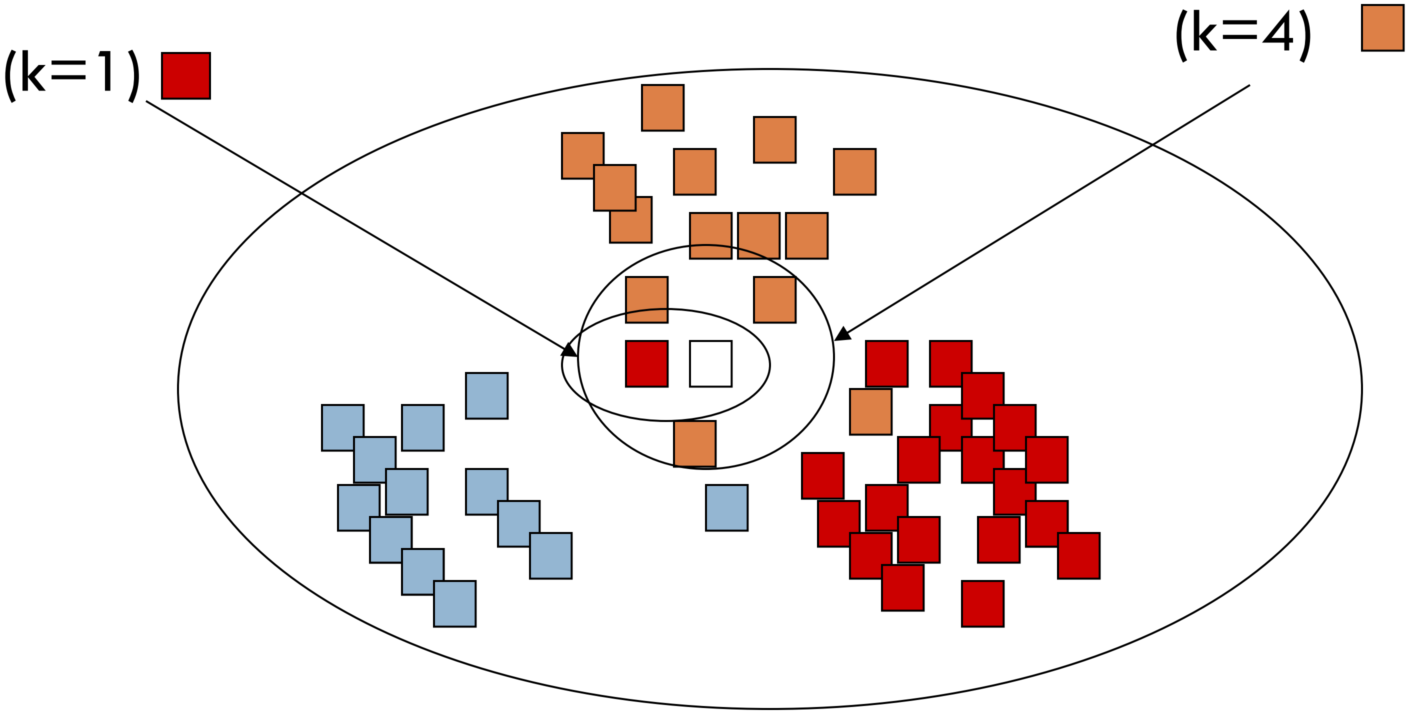
Example

- Data
  - Two attributes: acid durability and strength
  - Label: a special paper tissue is good or not
  - X1 = Acid Durability, X2= Strength, Y =classification
    D1=(7, 7, Bad), D2= (7, 4, Bad), D3= (3, 4, Good), D4=(1, 4, Good)
- Query: X1 = 3, and X2 = 7. Let us set K = 3.
- Distance metric: Euclidean distance
- Distance between query and all training examples.
  D1's Squared Distance to query (3, 7): (7-3)^2 + (7-7)^2 = 16
  D2's: 25, D3's: 9, D4's: 13
- Gather the category Y of the 3 nearest neighbors: Bad, Good, Good
- Majority voting for the predicted label: Good

http://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

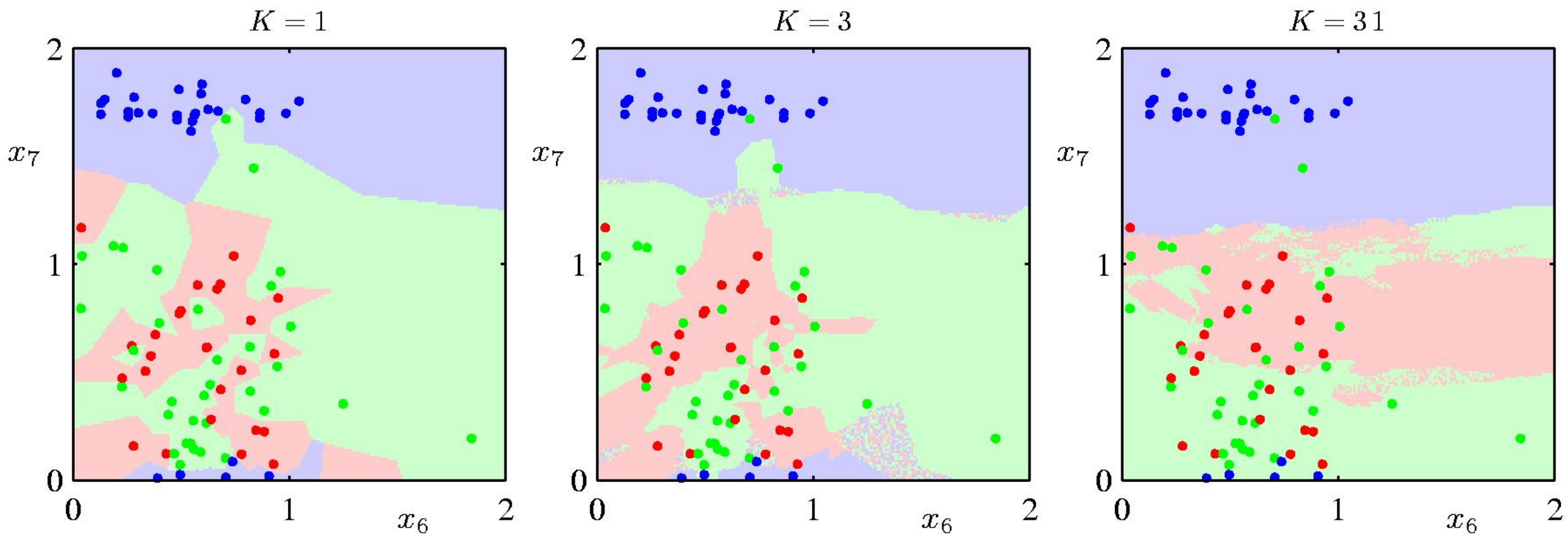
K-Nearest-Neighbour (*k*-NN) Classifier

How many neighbors should we count ?



Slides adapted from k-NN lectures given by Prof. Rong Jin at MSU

K-Nearest-Neighbour (*k*-NN) Classifier



- K acts as a smoother

Slides adapted from k-NN lectures given by Prof. Rong Jin at MSU

How to Choose K: Hold-out/Cross Validation

- Divide training examples into two sets
  - A training set (80%) and a validation set (20%)
- Predict the class labels for validation set by using the examples in training set
- Choose the number of neighbors *k* that maximizes the classification accuracy

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 x_q
    - 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

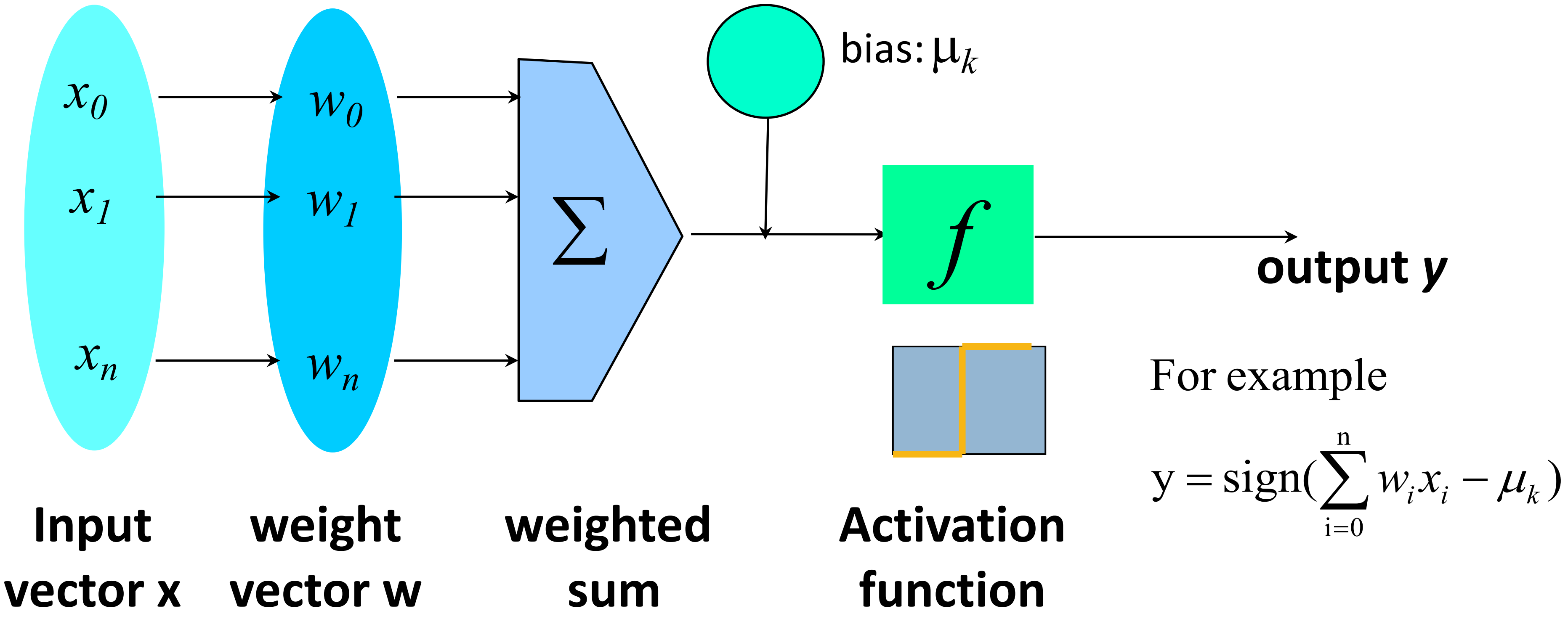
Classification: Advanced Methods

- Lazy Learners and K-Nearest Neighbors
- Neural Networks
- Support Vector Machines
- Bayesian Belief Networks
- Additional Topics: Semi-Supervised Methods, Active Learning, etc.
- Summary

Neural Network for Classification

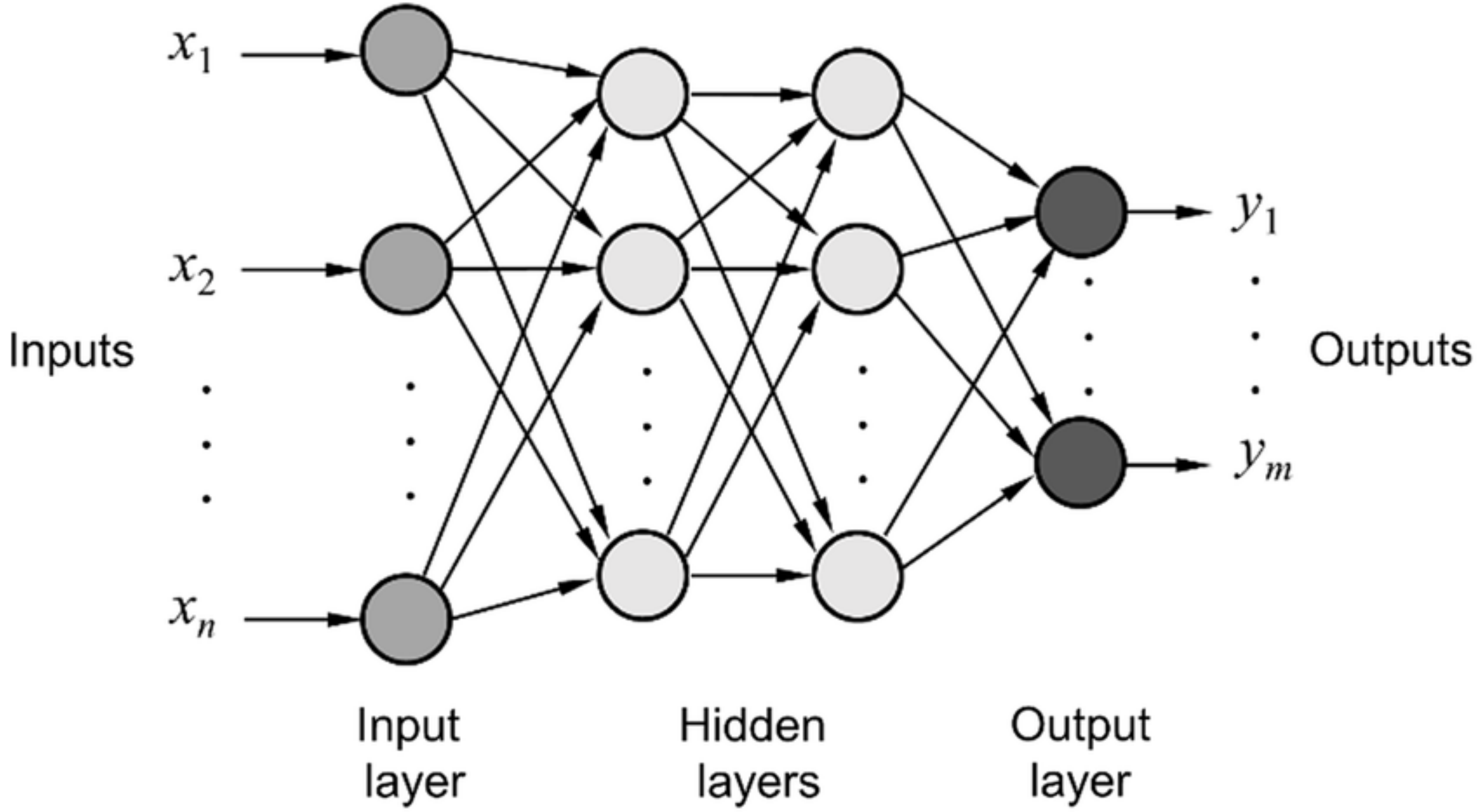
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
  - During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units
- Backpropagation: A **neural network** learning algorithm

Neuron: A Hidden/Output Layer Unit



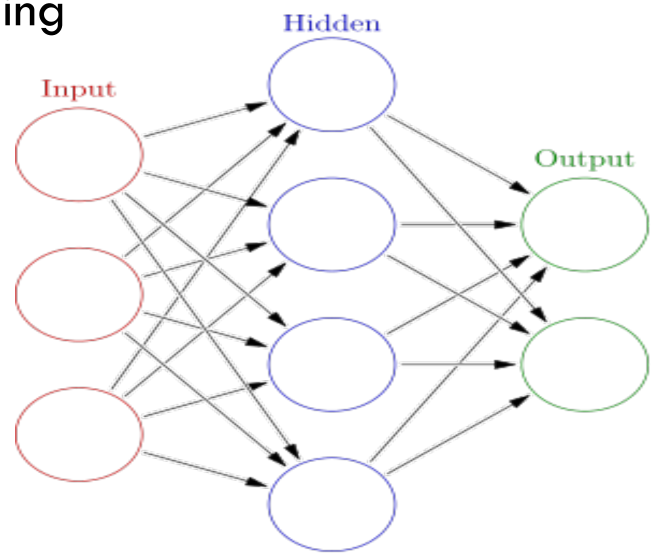
- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

A Multi-Layer Fully-Connected Feed-Forward Neural Network



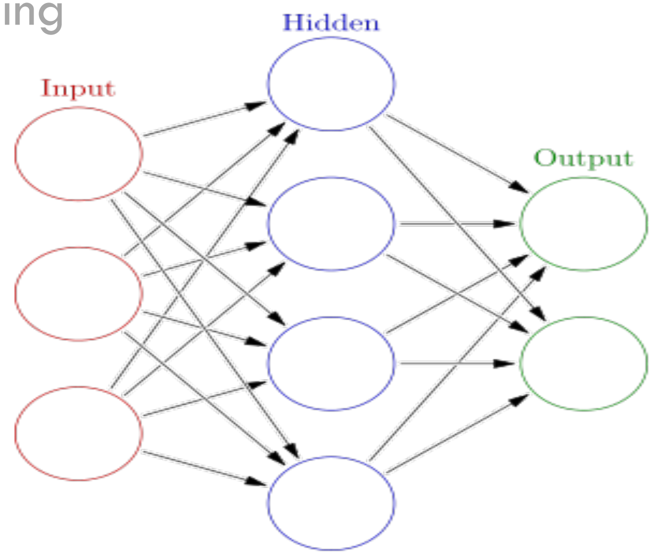
How a Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**



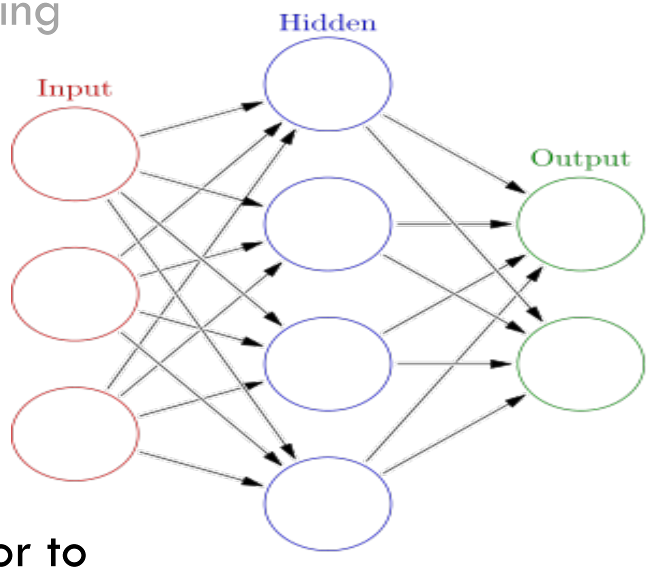
How a Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer,** which emits the network's prediction



How a Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function

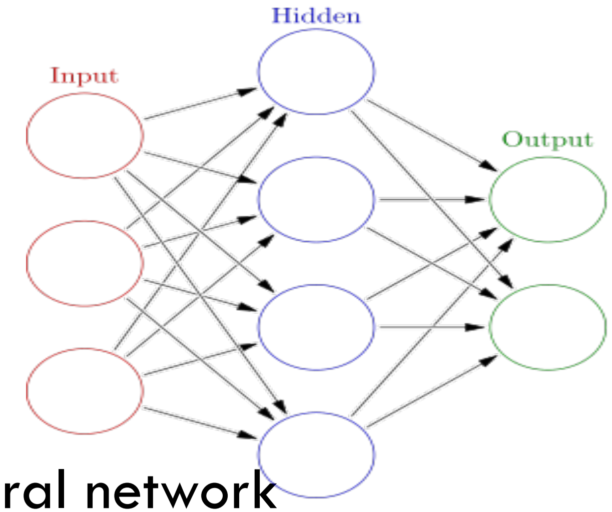


Defining a Network Topology

- Decide the **network topology**
  - Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- (Optional) Normalize the input values for each attribute measured in the training tuples, e.g., to [0.0, 1.0]
- **Output**, if for classification and more than two classes, one output unit per class is used
- Assign initial values to parameters (usually random sampling from normal distribution)
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- **Backpropagation:** The *de facto* supervised learning algorithm for neural networks
  - Short for "backward propagation of errors"
- 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 error** (squared error, cross entropy, et.c) 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 down 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.)

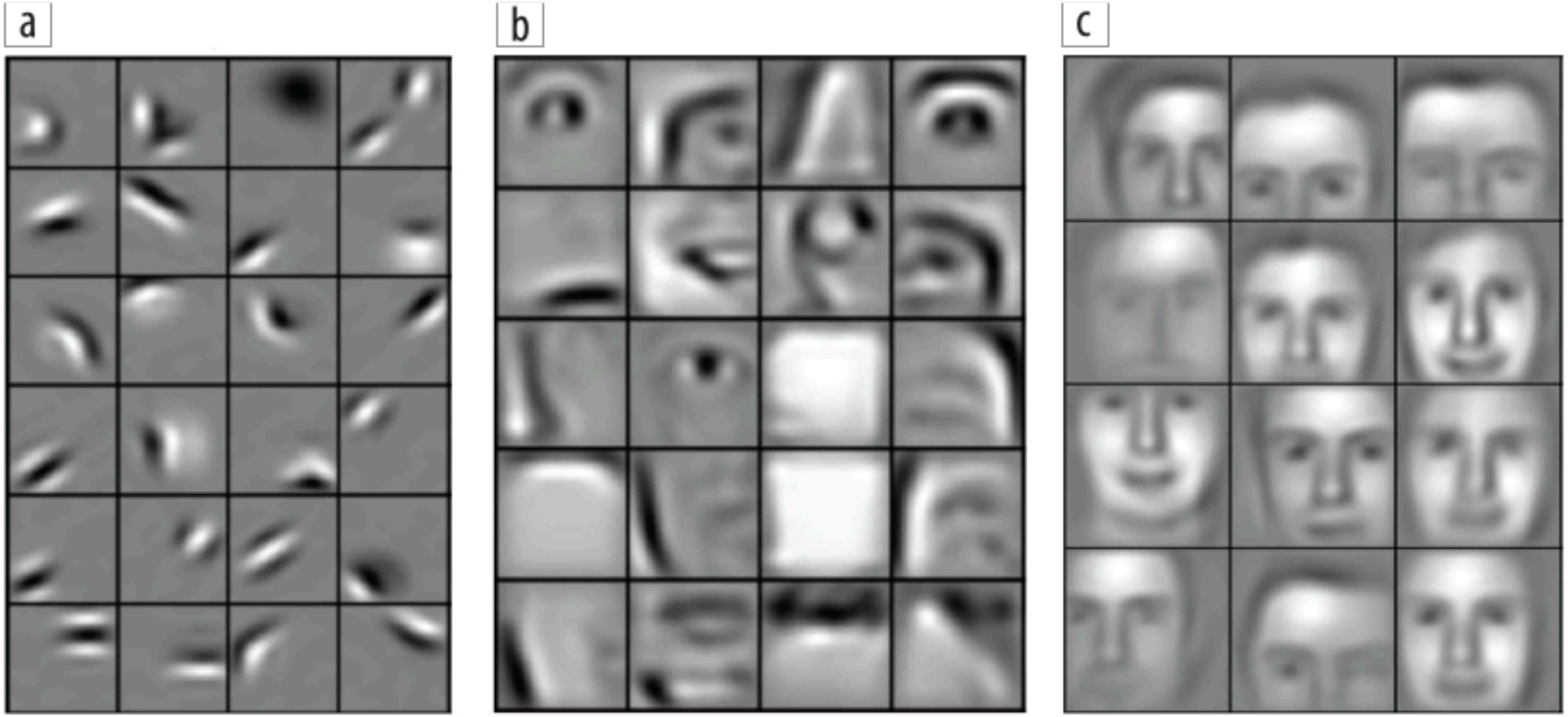


- Good news: backpropagation is readily supported in all neural network libraries like Tensorflow and PyTorch.

From Neural Networks to Deep Learning

- Train networks with many layers (vs. shallow nets with just 1 or 2 hidden layers)
- 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 current models, 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)

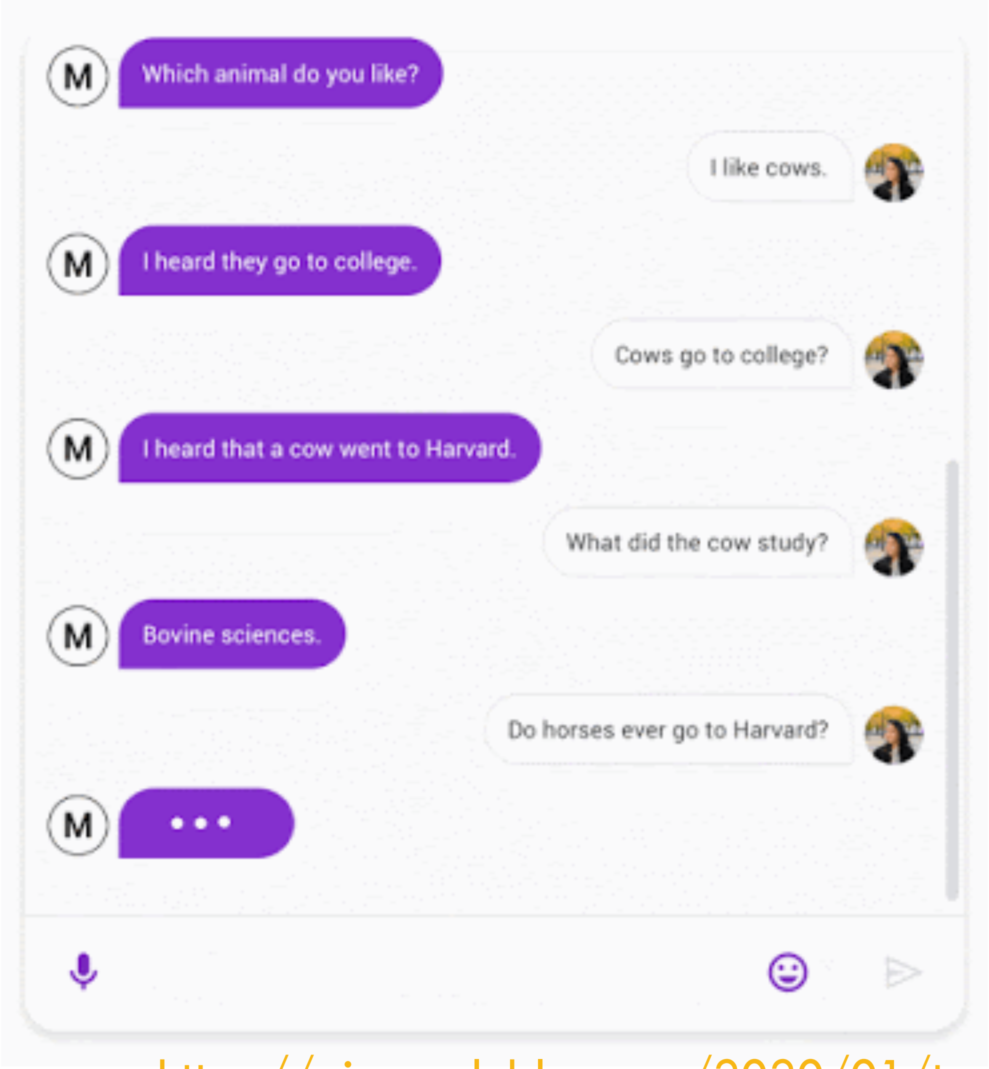
Deep Learning – Object Recognition



Layer 1 Layer 2 Layer 3

Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations – ICML 2019

Deep Learning – Conversational Agents



https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html

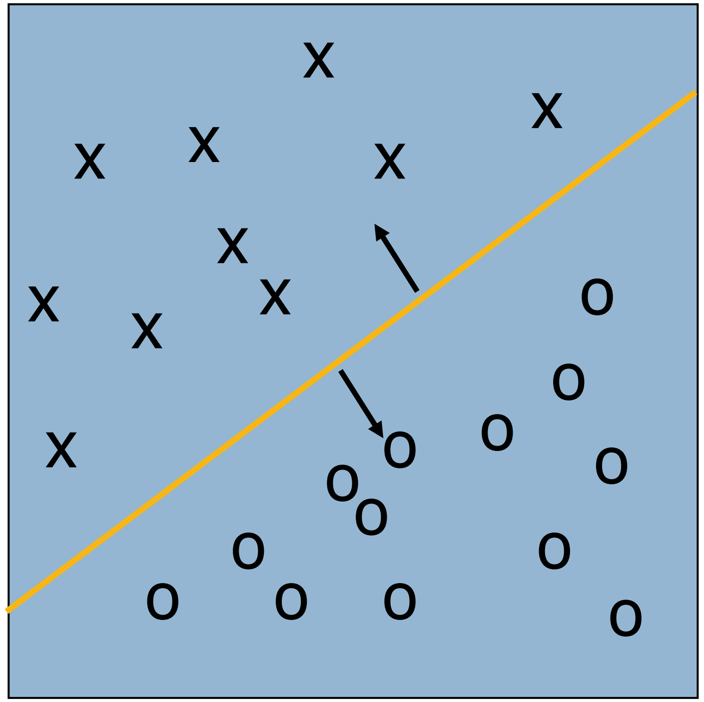
Deep Learning - DeepFake

Classification: Advanced Methods

- Lazy Learners and K-Nearest Neighbors
- Neural Networks
- Support Vector Machines
- Bayesian Belief Networks
- Additional Topics: Semi-Supervised Methods, Active Learning, etc.
- Summary

Classification: A Mathematical Mapping

- **Classification:** predicts categorical class labels
  - E.g., Personal homepage classification
    - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
    - x_1 : # of word "homepage"
    - x_2 : # of word "welcome"
- Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$,
  - We want to derive a function $f: X \rightarrow Y$
- 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



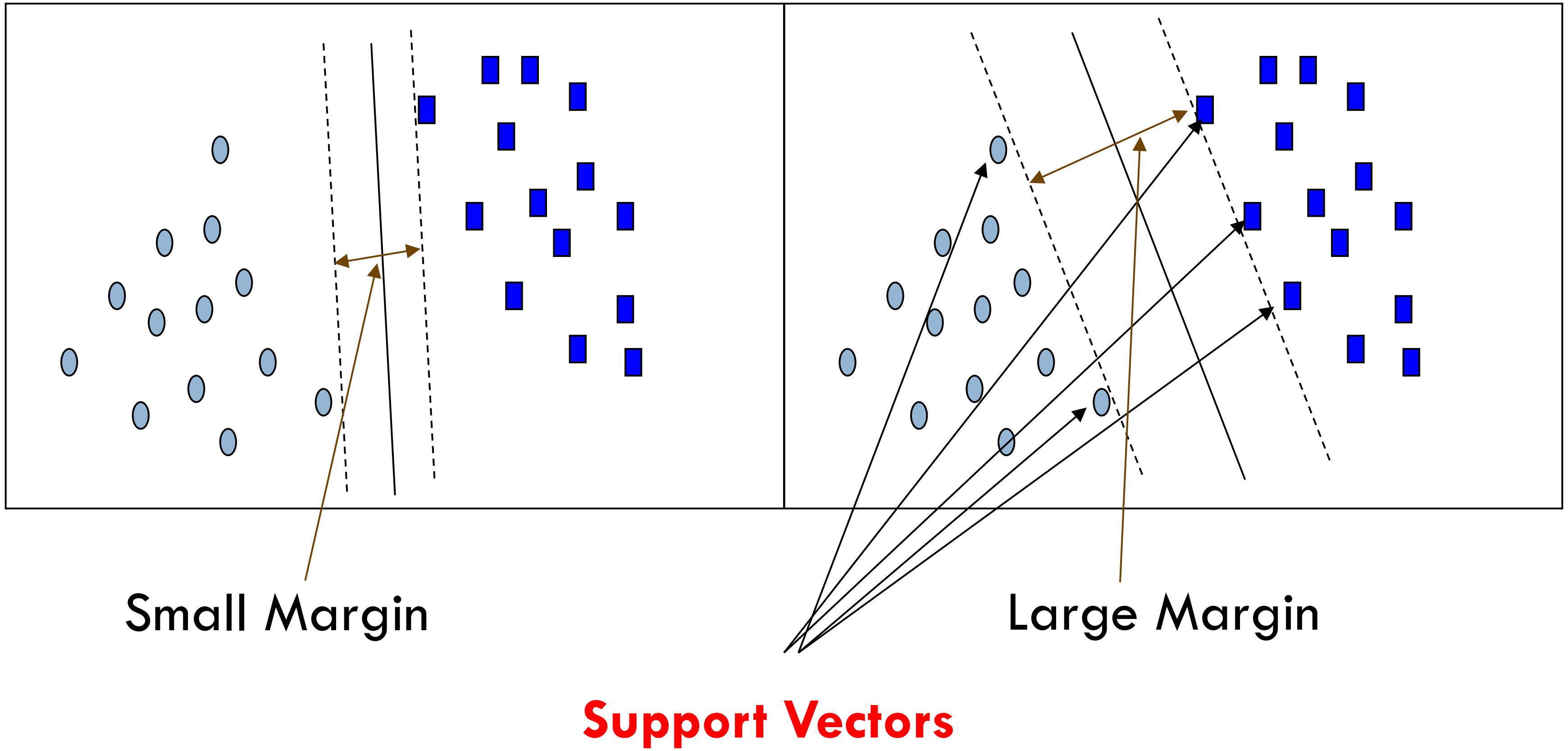
SVM—Support Vector Machines

- A relatively new (compared to decision tree or naïve bayes) classification method for both linear and nonlinear data
- 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)

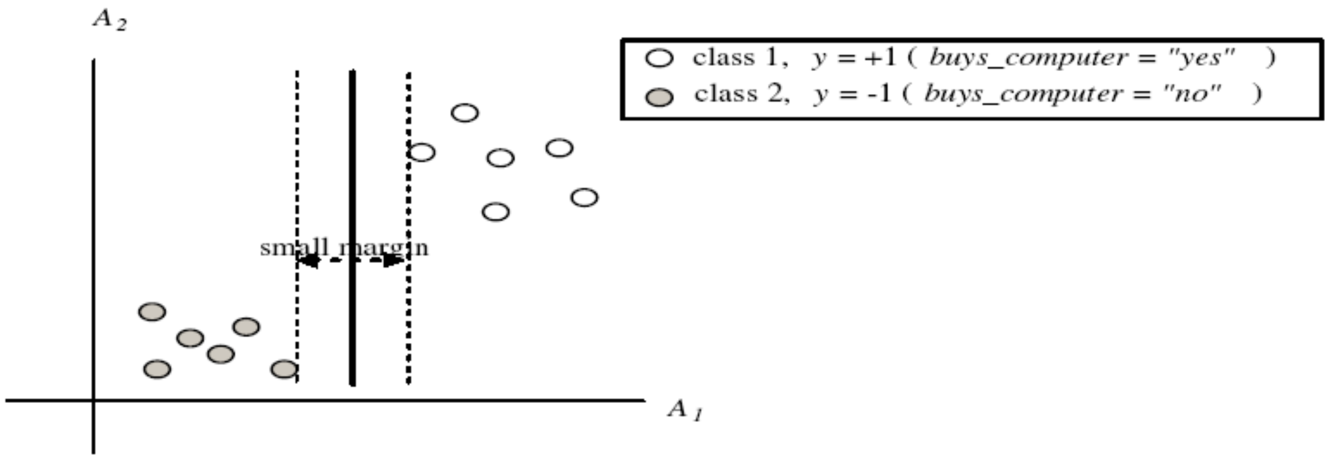
SVM—History and Applications

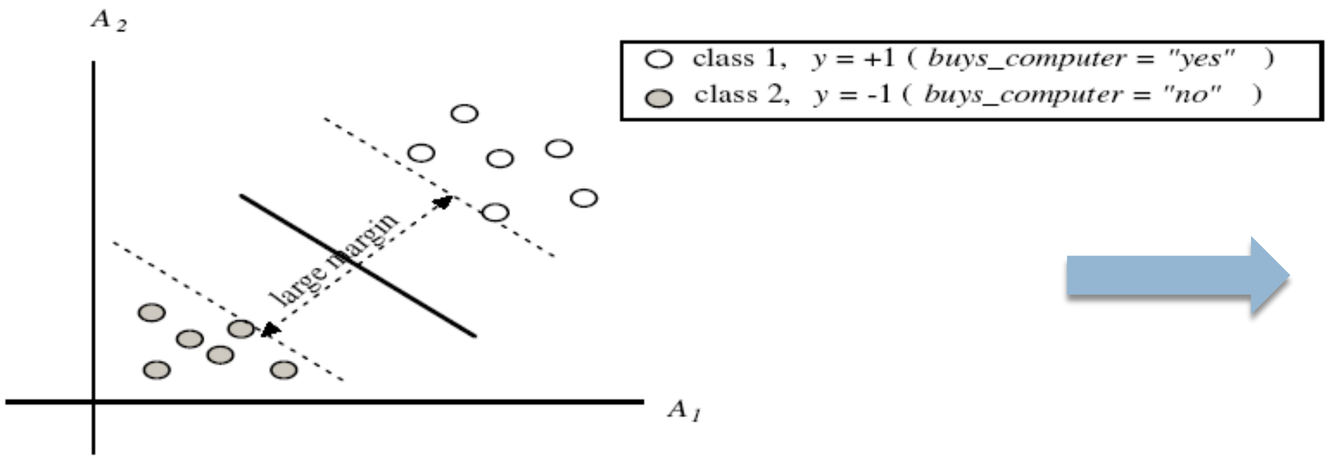
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

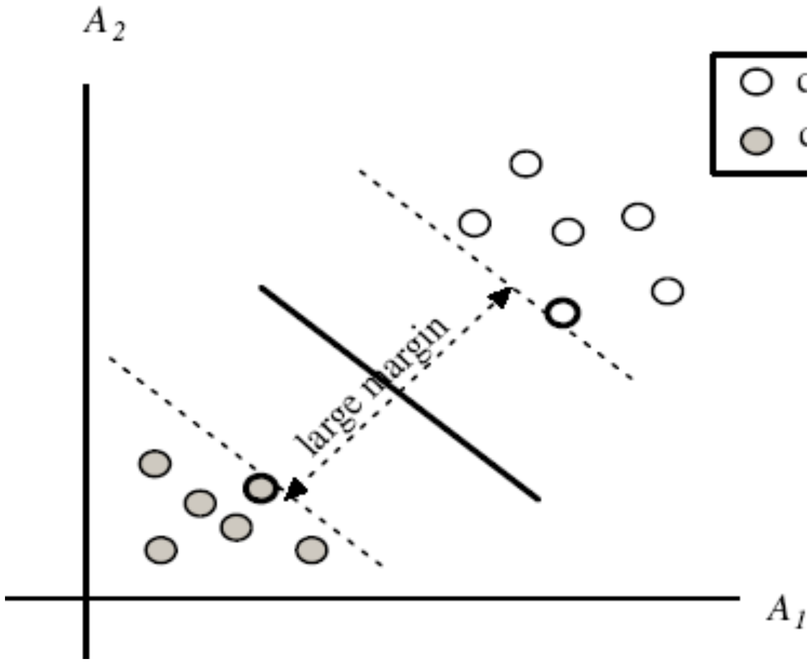
General Philosophy: Maximum Margin Principle



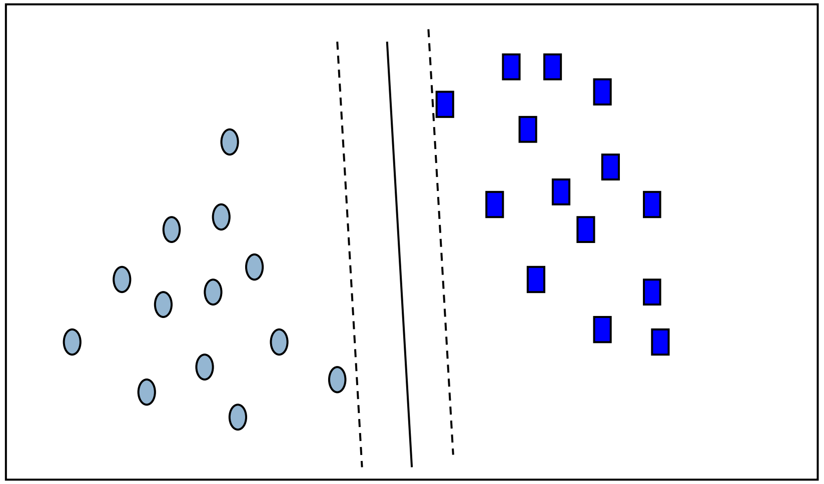
SVM—Margins and Support Vectors

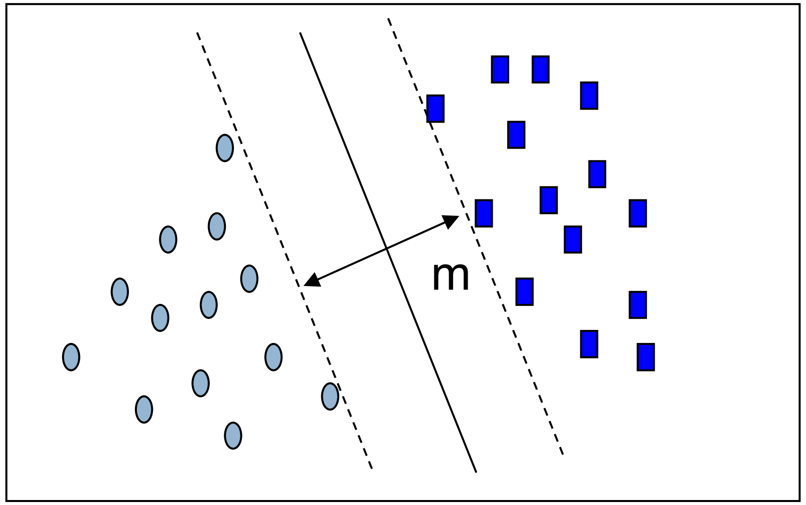






SVM—When Data Is Linearly Separable





Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), where $\mathbf{X}_i$ is the set of training tuples associated with the class labels y_i

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

  $$\mathbf{W} \bullet \mathbf{X} + b = 0$$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D it can be written as: $w_0 + w_1\, x_1 + w_2\, x_2 = 0$
- 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

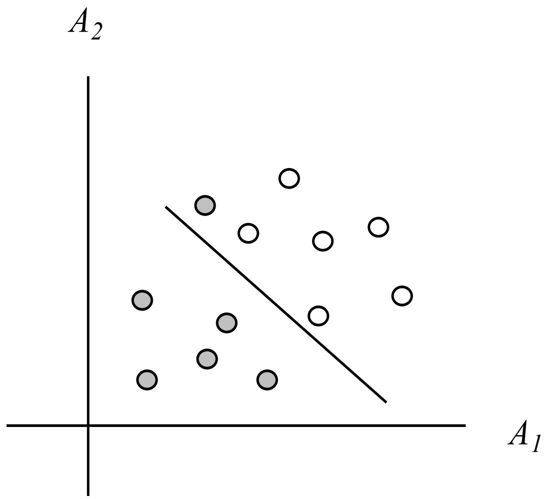
  H_2: $w_0 + w_1\, x_1 + w_2\, x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem:
  - Quadratic objective function and linear constraints → *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 $d(\mathbf{Z}) = \mathbf{W}\mathbf{Z} + b$, 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,

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



- Search for a linear separating hyperplane in the new space

Why Is SVM 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)
- If all other training examples were removed and the training was repeated, the same separating hyperplane would still be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

Kernel Functions for Nonlinear Classification

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

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

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

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

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

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

Summary: Classification

- Basic methods
  - Decision tree / Naïve Bayes classifier
- Advanced methods
  - K-nearest neighbors / Neural network / Support vector machine
- Ensemble methods
  - Bagging / boosting / random forest
- Practical issues
  - Evaluation: confusion matrix/accuracy/precision/recall/F-1/ROC, hold-out, cross-validation
  - Overfitting / underfitting