Classification

Outline

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Neural networks
- Ensemble Methods
- Summary

Prediction Problems: Classification vs. Numeric Prediction

- **Classification**
  - Predicts categorical class labels (discrete or nominal)
  - Constructs a model based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- **Numeric Prediction**
  - models continuous-valued functions, i.e., predicts unknown or missing values
- **Typical applications**
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

Classification examples in cyber security

- Examples
  - For every file sent through the network, does it contain malware?
  - For every login attempt, has someone's password been compromised?
  - For every email received, is it a phishing attempt?
  - For every request to your servers, is it a denial-of-service (DoS) attack?
  - For every outbound request from your network, is it a bot calling its command and-control server?
- Classify all events in your network as malicious or legitimate.

Decision boundary

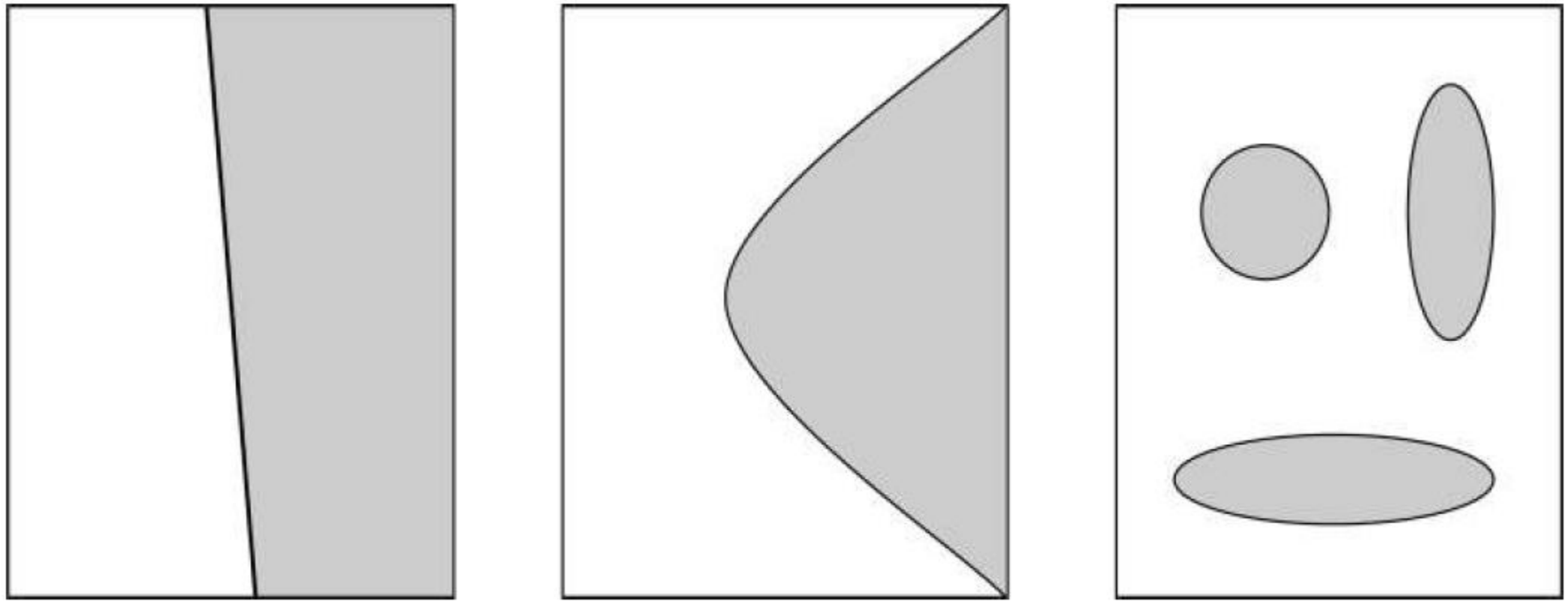
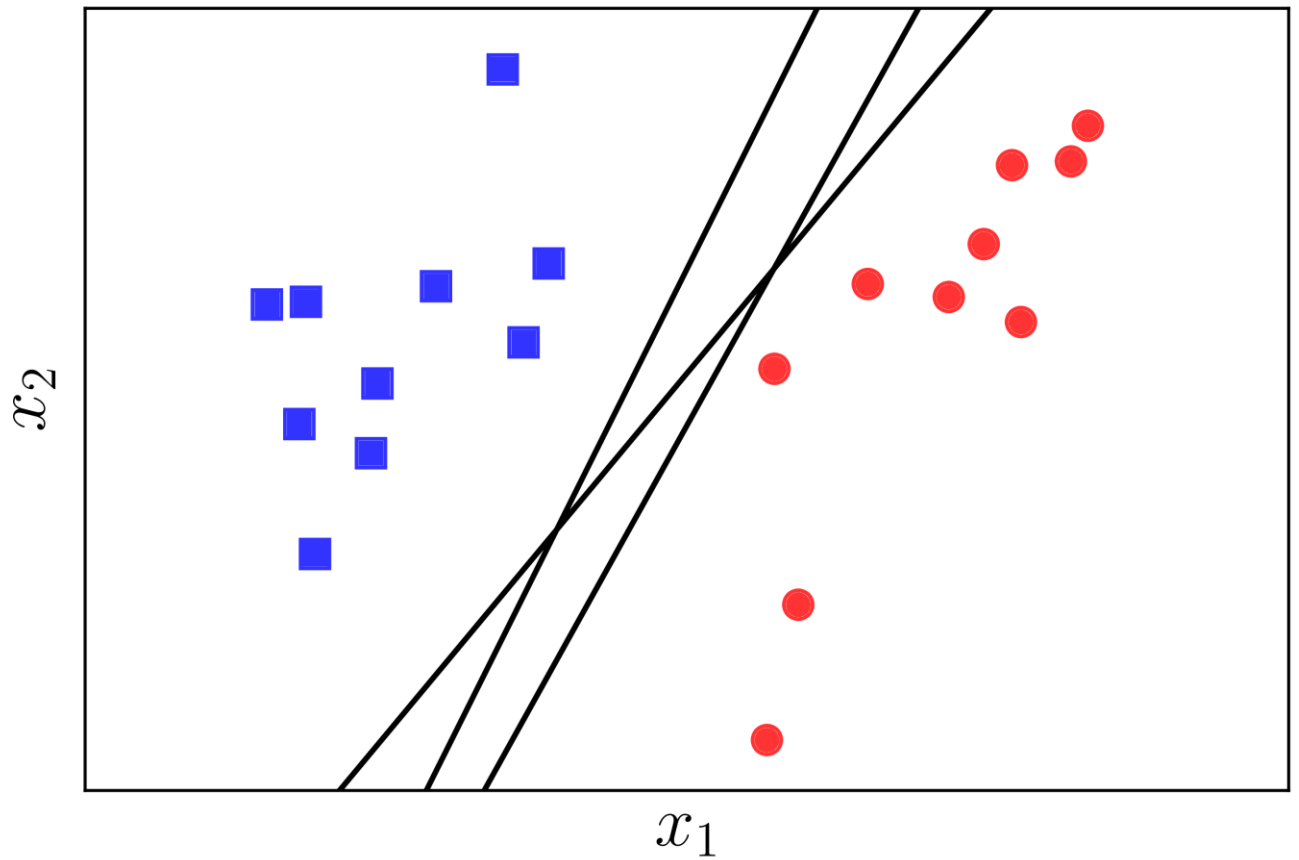


Figure 2-1. Examples of two-dimensional spaces divided by a decision boundary



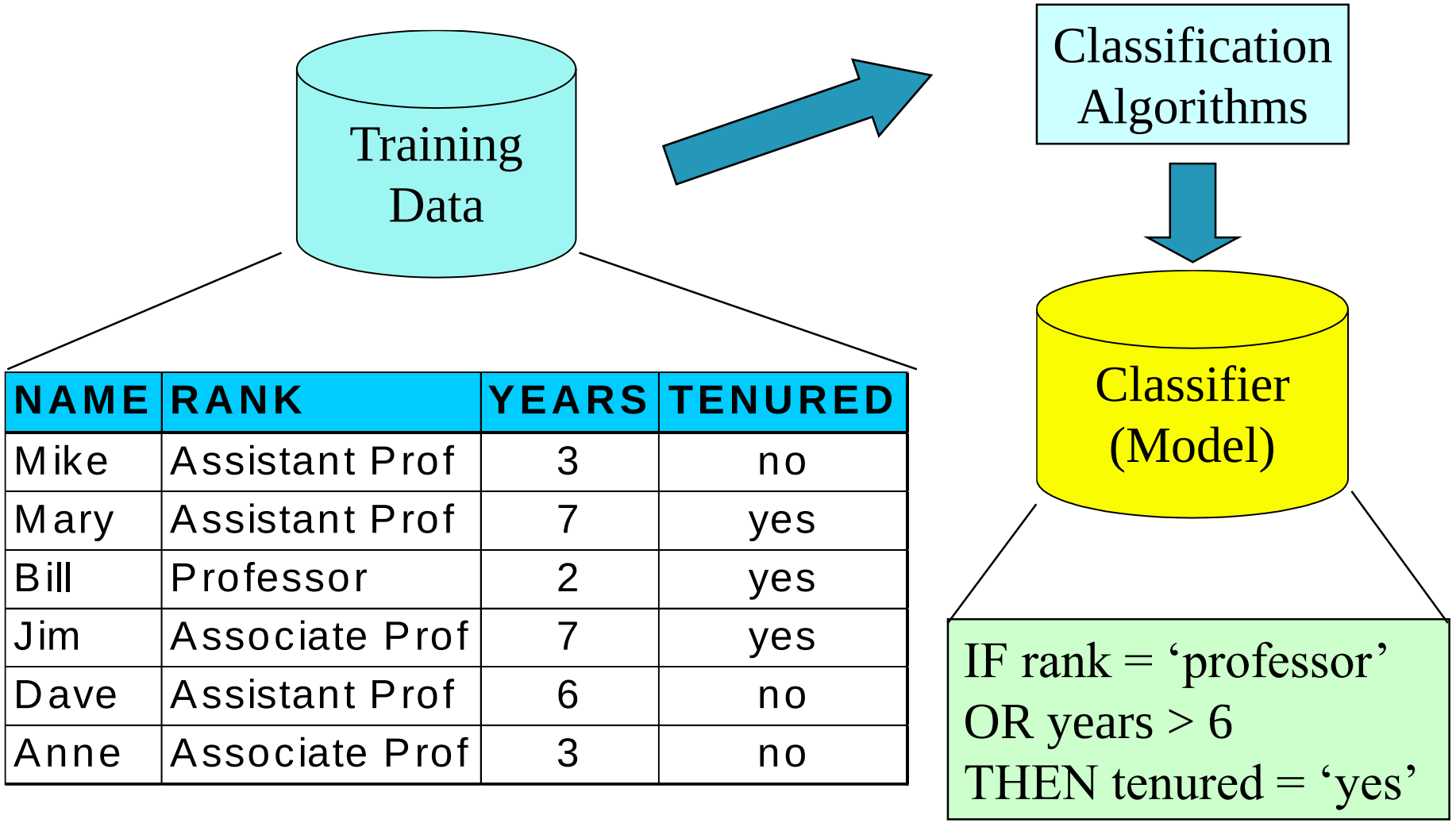
https://machinelearningcoban.com/2017/04/09/smv/

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data

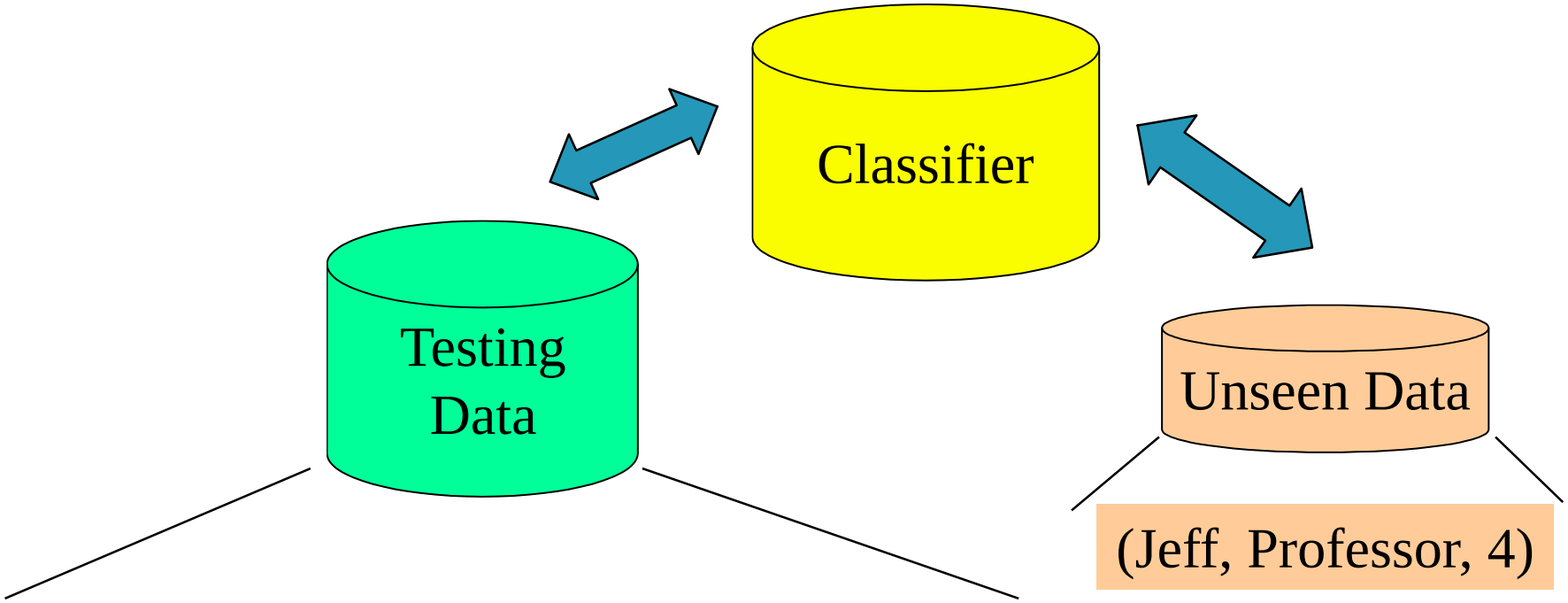
Note: If *the test set* is used to select models, it is called validation (test) set

Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Model Families

- Any given machine learning algorithm restricts itself to finding a certain type of *decision boundary* or *probability function* that can be described by a finite number of model parameters.
- Probability function:
  - instead of mapping each point in the vector space to a label (*decision boundary)*, we can map each point to a probability of a label
- The simplest decision boundary is a linear decision boundary—that is, a hyperplane in the vector space.
  - Notes: Linear classifier based on the linear combination of characteristics

Model Families

- An oriented hyperplane *H* in an n-dimensional vector space: Described by an n-dimensional vector θ orthogonal to the hyperplane, plus another vector β indicating how far the hyperplane is from the origin:

$$\mathbf{H: \theta \,.\, (x - \beta) = 0}$$

- =>Dividing the vector space in two; to assign probabilities we want to look at the **distance** of the point **x** from the hyperplane H. We can thus compute a real-valued "score":

$$s(\mathbf{x}) = \theta \,.\, (\mathbf{x} - \beta) = \theta \,.\, \mathbf{x} + b \qquad \text{where we have let } b = -\theta \,.\, \beta.$$

- Notes: *n + 1* model parameters *(n parameters to describe the vector θ, and one "offset" parameter b)*.
- To turn the score into a classification,
  - We simply choose a threshold *t* above which all scores indicate "true" , and below which all scores indicate "false".
- If we want to map the real-valued score s(**x**) to a probability,
  - we must apply a function that maps the real numbers to the interval [0,1].
  - The standard function to apply is known as the logistic function or sigmoid function

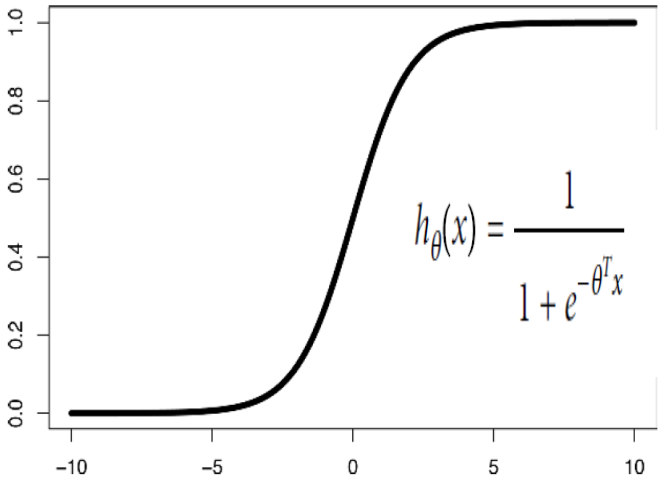


Figure 2-2. The sigmoid function

Loss functions

- *A loss function* is a function that maps a set of pairs of (predicted label, truth label) to a real number. The goal of a machine learning algorithm is to find the model parameters that produce predicted labels for the training set that minimize *the loss function*.
- In regression problems (*outputs a real number instead of a label*),
  - Model: $\boldsymbol{y} = f(\boldsymbol{w}^T\boldsymbol{x})$ where $f(s)=s$
  - the standard loss function is the sum of squared errors.

$$C(Y) = \sum_i (\hat{y}_i - y_i)^2$$

Loss functions

- For logistic regression:
  - The goal of logistic regression is to find parameters that produce probabilities p_i that maximize the likelihood
  - Model: $\boldsymbol{y} = f(\boldsymbol{w}^T\boldsymbol{x})$ where *f is a logistics function*
  - Model can be used for classification (linear)
  - We use *negative log likelihood* as the loss function (minimize instead of maximize)

Loss functions

- The *likelihood* of a set of probability predictions $\{p_i\}$ for *a given set of ground truth labels* $\{y_i\}$ *(given data)* is defined to be the probability that these truth labels would have arisen if sampled from a set of *binomial distributions* according to the probabilities $\{p_i\}$
- The likelihood of the entire set of predictions is the product of the individual likelihoods

$$\mathscr{L}(\{p_i\},\{y_i\}) = \prod_{y_i=0}(1-p_i)\cdot\prod_{y_i=1}p_i$$

$$\ell(\{p_i\},\{y_i\}) = -\sum_i\left((1-y_i)\log(1-p_i)+y_i\log p_i\right)$$

Alg1: Logistic regression for classification

- We use p and 1-p for comparison
- p= P(y=1|x;w); 1-p = P(y=0|x;w)
- P(y=1|x;w)>0.5 for class 1
- Logistic regression takes as input numerical feature vectors and attempts to predict *the log odds* of each data point occurring (ln p/(1-p)); we can convert *the log odds* to probabilities by using the sigmoid function discussed earlier
- Linear classifier

  $P(y=1|x;w) > 0.5$

  $\Leftrightarrow 1/(1+e^{-wTx}) > 0.5$

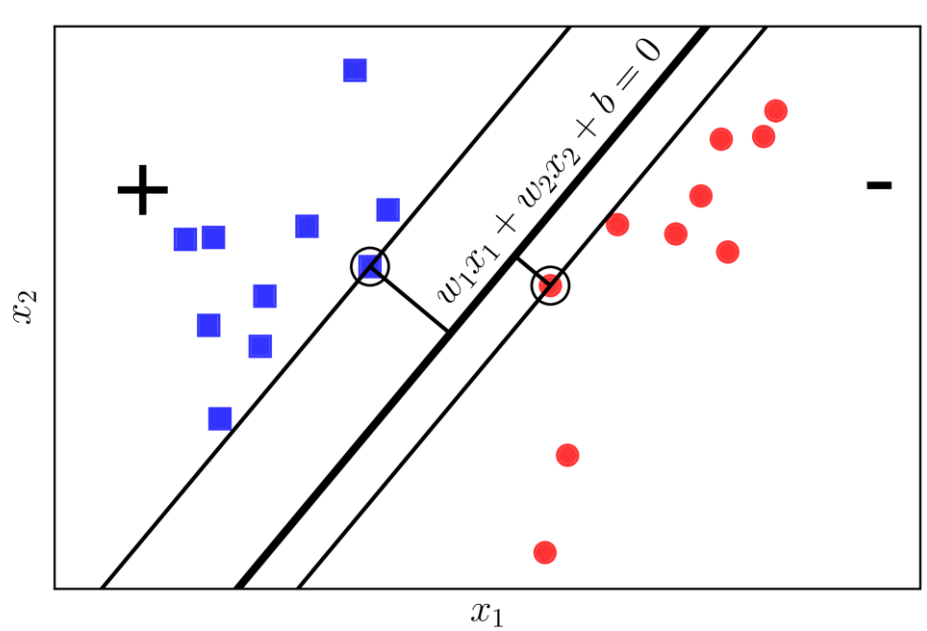
  $\Leftrightarrow e^{-wTx} < 1$

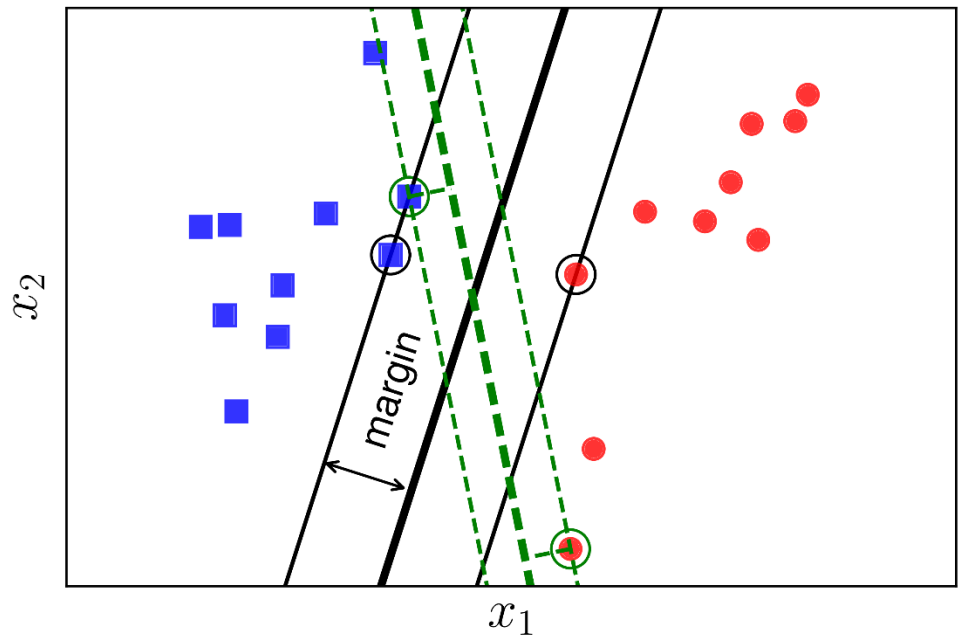
  $\Leftrightarrow w^{T}x>0$ (linear decision boundary)

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Alg2: SVM

- **A support vector machine (SVM) is (in its simplest form) a linear classifier,**
  - which means that it produces a hyperplane in a vector space that attempts to separate the two classes in the dataset.
- **The SVM classifier attempts to find the maximum-margin hyperplane separating the two classes,**
  - where "margin" indicates the distance from the separating plane to the closest data points on each side
- **For the case in which the data is not linearly separable, points within the margin are penalized proportionately to their distance from the margin**





https://machinelearningcoban.com/2017/04/09/smv/

SVM

- The difference between logistic regression and SVMs is the loss function.
  - Logistic regression uses a log-likelihood function that penalizes all points proportionally to the error in the probability estimate, even those on the correct side of the hyperplane.
  - An SVM, on the other hand, uses a hinge loss, which penalizes only those points on the wrong side of the hyperplane or very near it on the correct side.

SVM

$$\beta + C \sum_{i=1}^{N} \xi_i$$

where β is the margin, ξ_i is the distance from the ith support vector to the margin, and C is a model hyperparameter that determines the relative contribution of the two terms.

To classify a new data point x, we simply determine which side of the plane x falls on.

If we want to get a real-valued score we can compute the distance from x to the separating plane and then apply a sigmoid to map to [0,1].

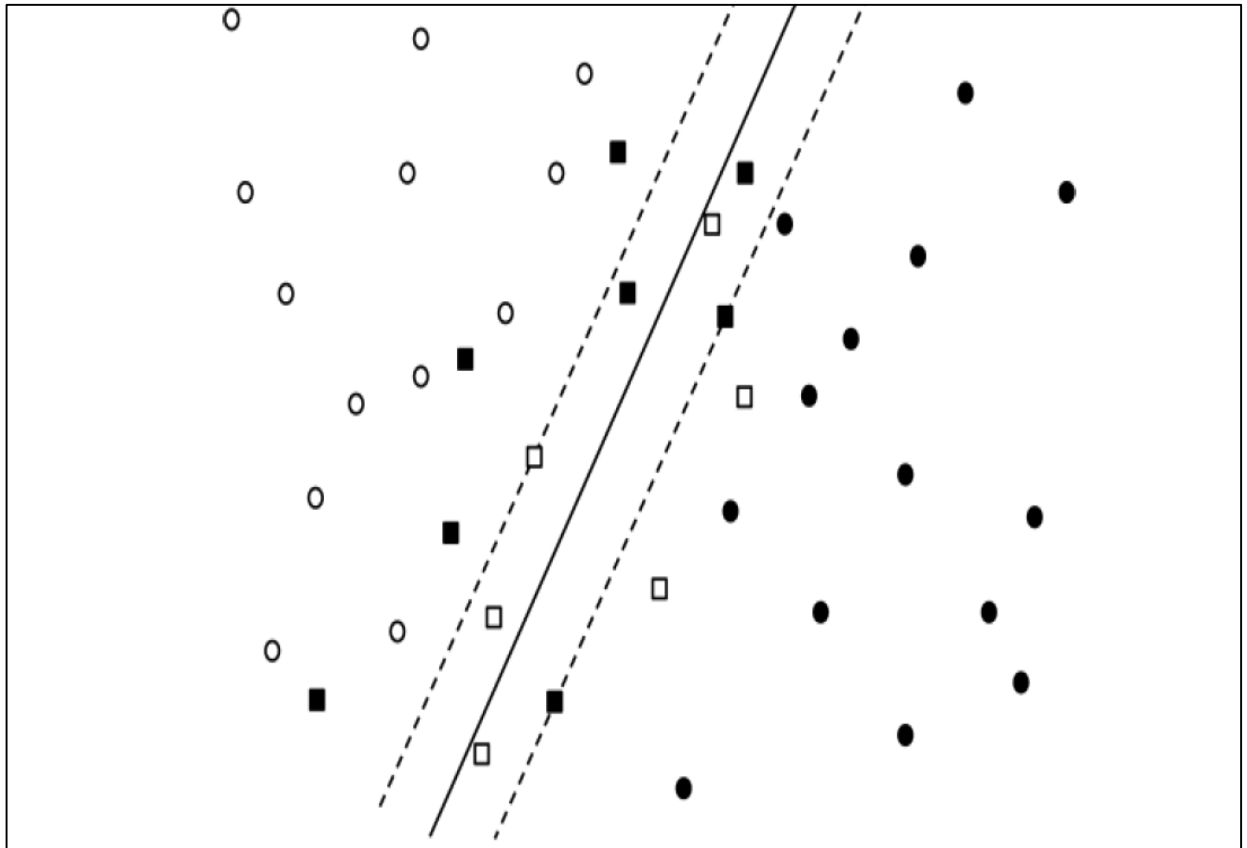
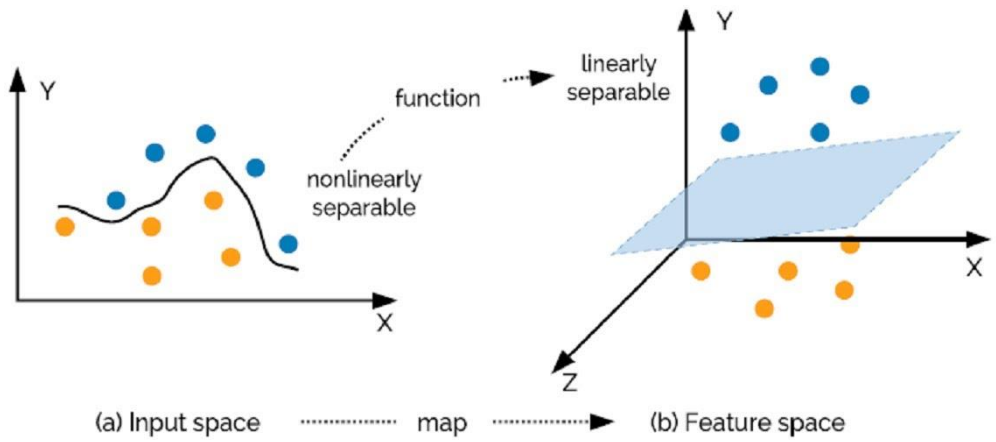


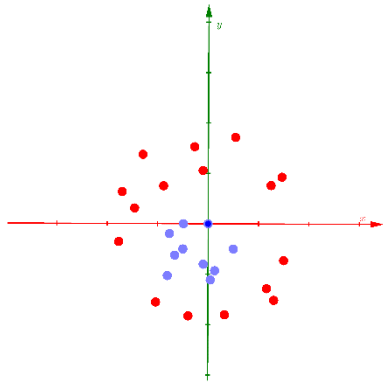
Figure 2-7. Classification boundary (dark line) and margins (dashed lines) for linear SVM separating two classes (black and white points); squares represent support vectors

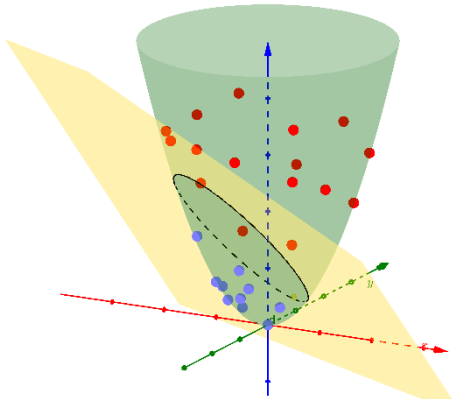
SVM

- The real power of SVMs comes from the kernel trick, which is a mathematical transformation that takes a linear decision boundary and produces a nonlinear boundary
- The most popular choice is the radial basis function $K(x,y) = e^{-\gamma|x-y|}$

Kernel Trick (SVM)…







https://machinelearningcoban.com/2017/04/22/kernelsmv/

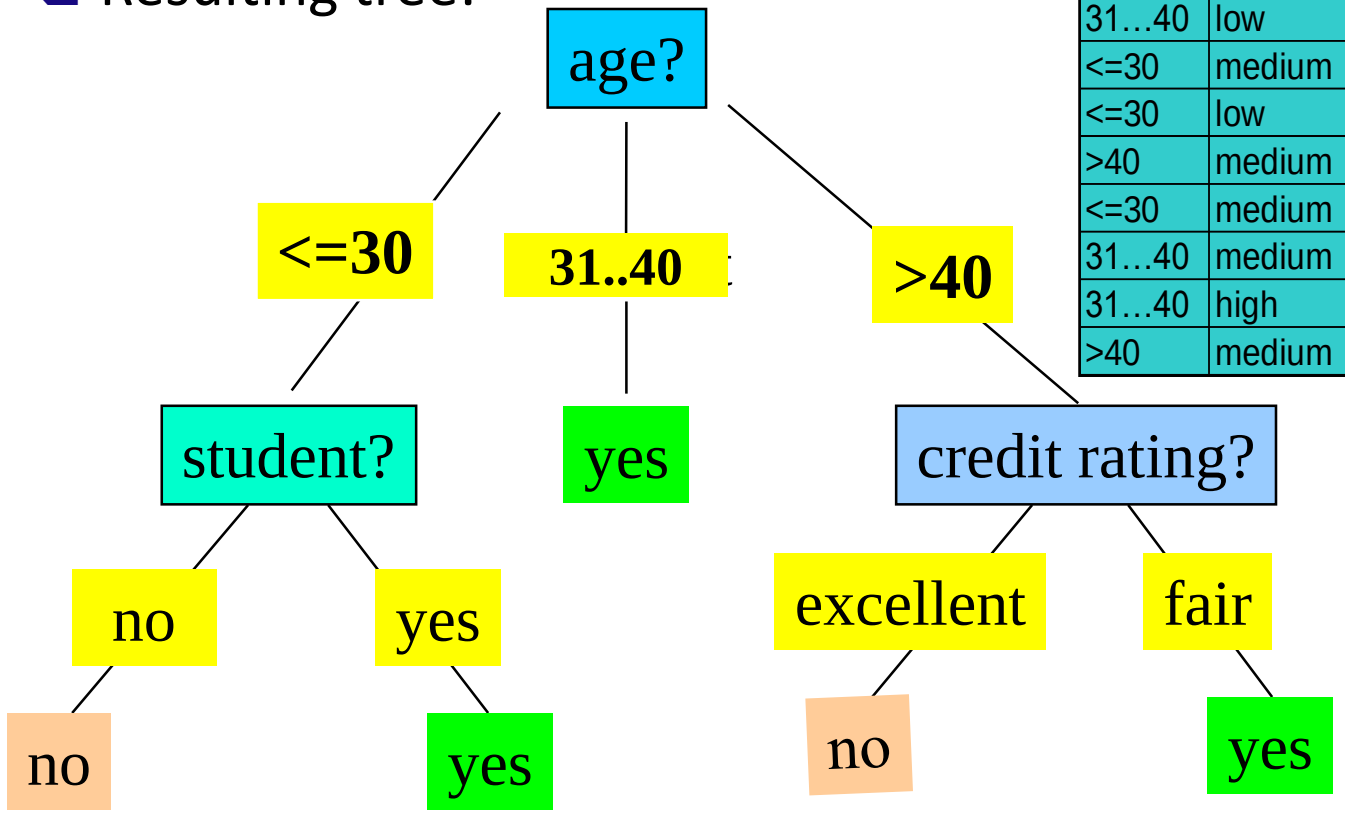
Decision trees

- A decision tree is, as its name suggests, a binary tree data structure that is used to make a decision.
- Trees are a very intuitive way of displaying and analyzing data and are popularly used even outside of the machine learning field.
- An important quality of decision trees is the relative ease of explaining classification or regression results, since every prediction can be expressed in a series of Boolean conditions that trace a path from the root of the tree to a leaf node.
- For example, if a decision tree model predicted that a malware sample belongs to malware family ***A***, we know it is because
    - (1) the binary was signed before 2015,
    - (2) does not hook into the window manager framework,
    - (3) does make multiple network calls out to particular country IP addresses

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

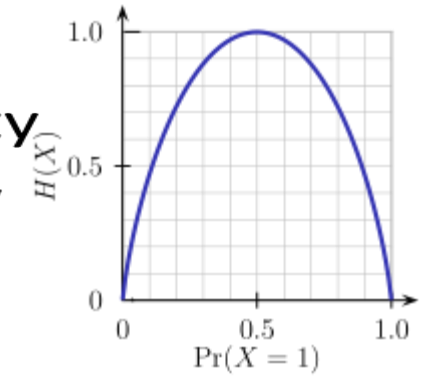


Algorithm for Decision Tree Induction

- **Basic algorithm (a greedy algorithm)**
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \ldots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = “yes”
- Class N: buys_computer = “no”

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem: $$P(B) = \sum_{i=1}^{M} P(B|A_i)P(A_i)$$
- Bayes' Theorem: $$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posteriori probability):* the probability that the hypothesis holds given the observed data sample **X**
  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds

Prediction Based on Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Naïve Bayes Classifier

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

  needs to be maximized

Naïve Bayes Classifier

- A *naïve* simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes) This assumption never actually holds in real life.
  - For example, consider a spam classifier for which the features are the words in the message.
- The Naive Bayes assumption posits that a spam message is composed by sampling words independently, where each word ***w*** has a probability $\mathbf{p}_{\mathbf{w,\ spam}}$ of being sampled, and similarly for good messages.
- This assumption is clearly unreasonable; for one thing, it completely ignores word ordering.

Naïve Bayes Classifier

- A *naïve* simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times ... \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k|C_i)$ is

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes Classifier: Comments

- **Advantages**
  - Easy to implement
  - Good results obtained in most of the cases
- **Disadvantages**
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.

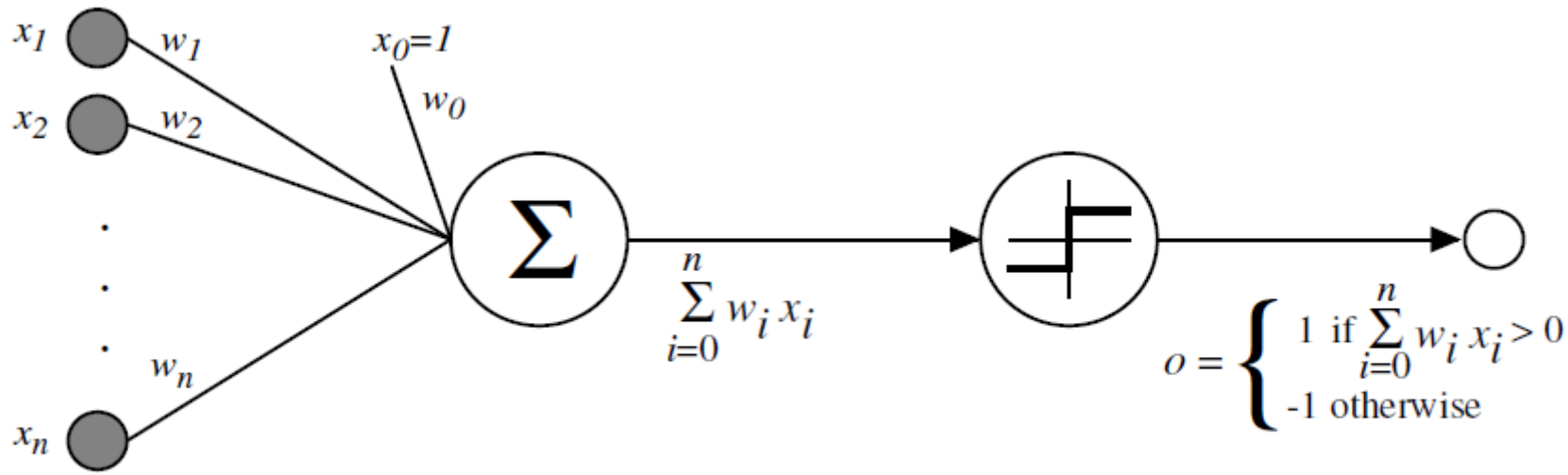
      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier
- How to deal with these dependencies? Bayesian Belief Networks

Neural networks

- Artificial neural networks (ANNs) are a class of machine learning techniques that have seen a resurgence in popularity recently.
- The human brain is composed of a humongous number of neurons (on the order of 10 billion), each with connections to tens of thousands of other neurons.
  - Each neuron receives electrochemical inputs from other neurons, and if the sum of these electrical inputs exceeds a certain level, the neuron then triggers an output transmission of another electrochemical signal to its attached neurons.
  - If the input does not exceed this level, the neuron does not trigger any output.

Neural networks

- ANNs were originally attempts at modeling neurons in the brain to achieve humanlike learning. Individual neurons were modeled with simple mathematical step functions (called activation functions), taking in weighted input from some neurons and emitting output to some other neurons if triggered.
- This mathematical model of a biological neuron is also called a perceptron.
- Armed with perceptrons, we then can form a plethora of different neural networks by varying the topology, activation functions, learning objectives, or training methods of the model

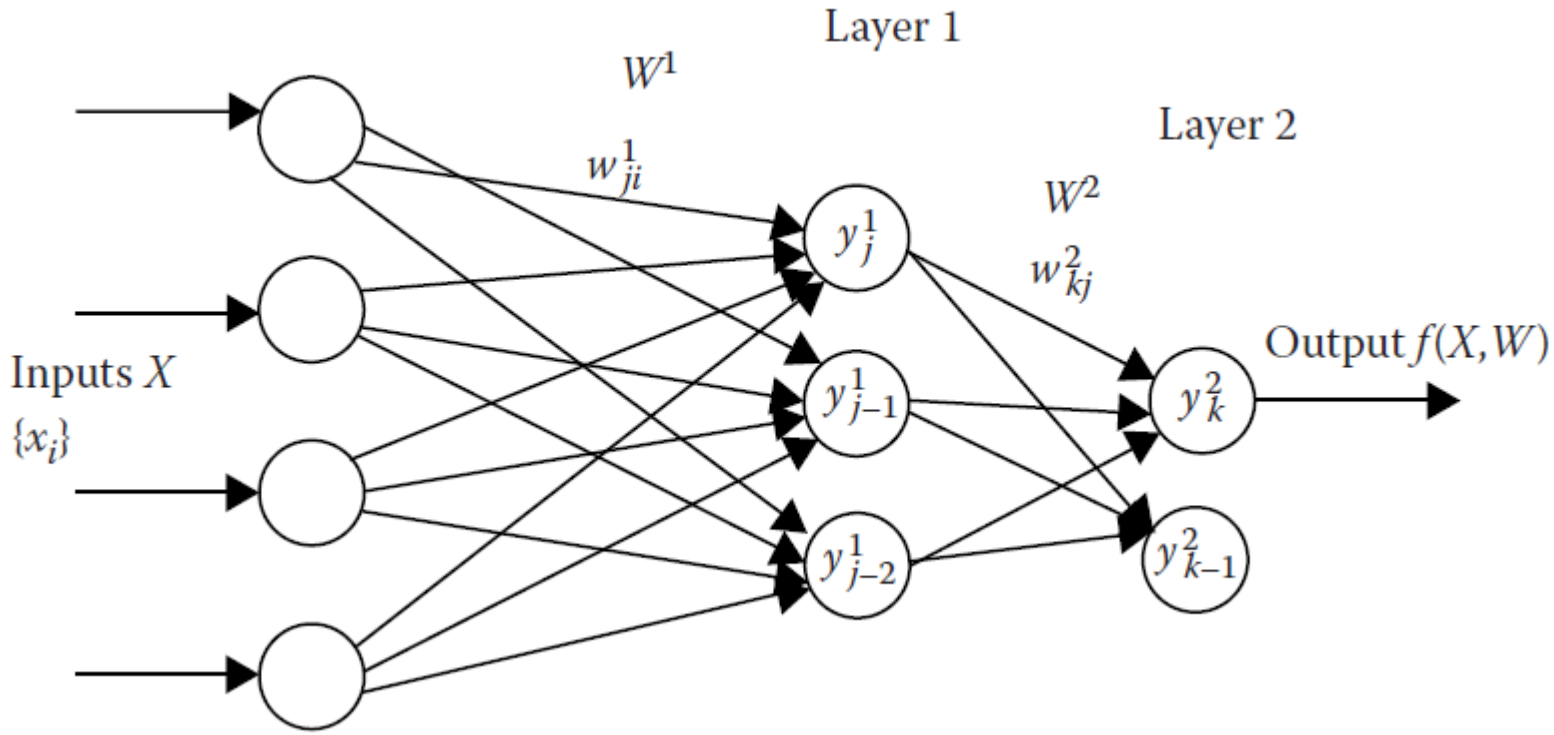


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron is a linear classifier
$f(w^T x)=o(w^T x)$

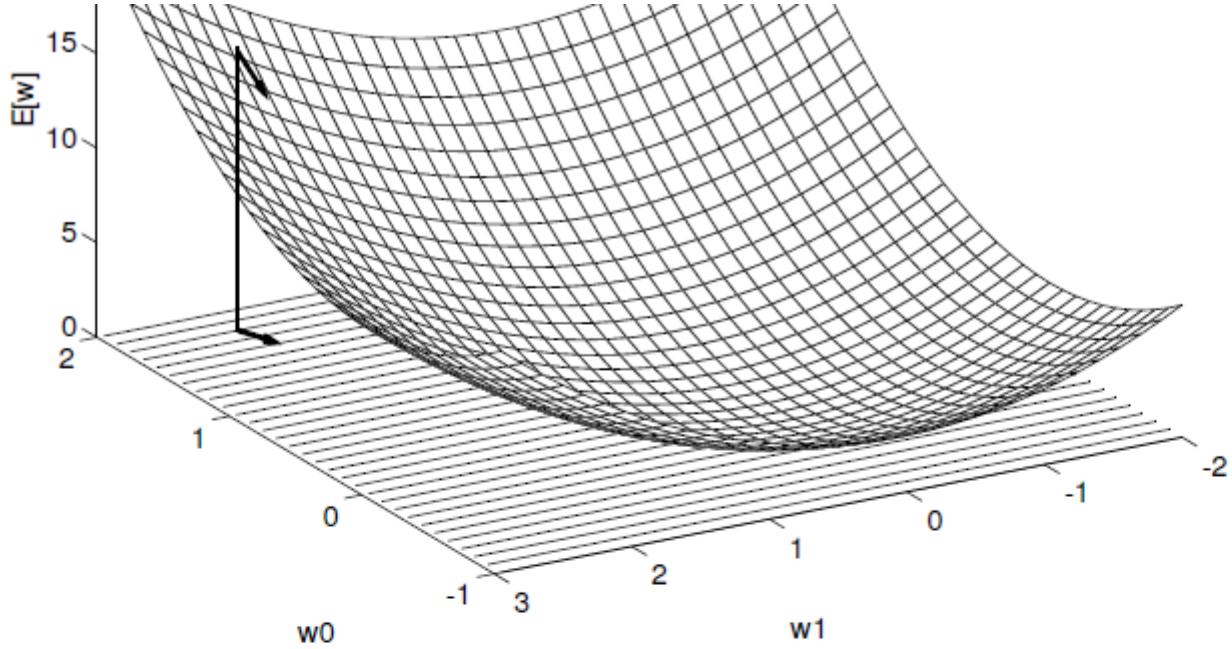


$$y_j^1 = T_f\left(\sum_i x_i \cdot w_{ji}^1\right).$$

$$y_k^2 = T_f\left(\sum_j y_j^1 \cdot w_{kj}^2\right)$$

Neural networks

- Typically, ANNs are made up of neurons arranged in *layers*. Each neuron in a layer receives input from the previous layer and, if activated, emits output to one or more neurons in the next layer.
- Each connection of two neurons is associated with a *weight*, and each neuron or layer might also have an associated *bias*.
- These are the parameters to be trained by the process of *backpropagation*, which we describe simply and briefly.
- Before starting, all of the weights and biases are randomly initialized. For each sample in the training set, we perform two steps:
  - 1. *Forward pass*. Feed the input through the ANN and get the current prediction.
  - 2. *Backward pass*. If the prediction is correct, reward the connections that produced this result by increasing their weights in proportion to the confidence of the prediction. If the prediction is wrong, penalize the connections that contributed to this wrong result.



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

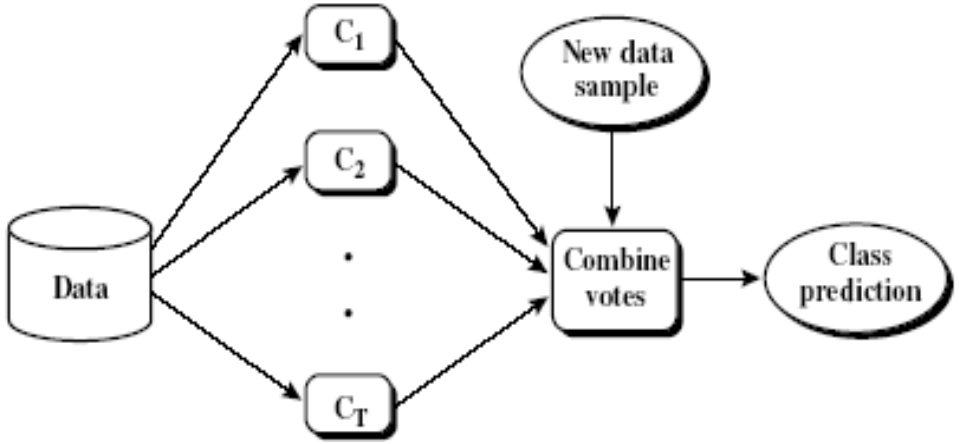
Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

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*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

Boosting

- **Analogy**: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow 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
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Summary (I)

- Classification is a form of data analysis that extracts models describing important data classes.
- Effective and scalable methods have been developed for decision tree induction, Naive Bayesian classification, rule-based classification, and many other classification methods.
- Evaluation metrics include: accuracy, sensitivity, specificity, precision, recall, *F* measure, and F_{β} measure.
- Stratified k-fold cross-validation is recommended for accuracy estimation. Bagging and boosting can be used to increase overall accuracy by learning and combining a series of individual models.

Summary (II)

- Significance tests and ROC curves are useful for model selection.
- There have been numerous comparisons of the different classification methods; the matter remains a research topic
- No single method has been found to be superior over all others for all data sets
- Issues such as accuracy, training time, robustness, scalability, and interpretability must be considered and can involve trade-offs, further complicating the quest for an overall superior method