Non-Bayesian Composite Classifiers K-NN, SVM, ANN and DT

An Introduction

Dr Muhammad Sarim

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Non-Bayesian Classifiers

- We have been using Bayesian classifiers that make decisions according to the posterior probabilities.

Non-Bayesian Classifiers

- We have been using Bayesian classifiers that make decisions according to the posterior probabilities.
- We have discussed parametric and non-parametric methods that learn classifiers by estimating the probabilities using training data.

Non-Bayesian Classifiers

- We have been using Bayesian classifiers that make decisions according to the posterior probabilities.
- We have discussed parametric and non-parametric methods that learn classifiers by estimating the probabilities using training data.
- We will study new techniques that use training data to learn the classifiers directly without estimating any probabilistic structure.

Non-Bayesian Classifiers

- We have been using Bayesian classifiers that make decisions according to the posterior probabilities.
- We have discussed parametric and non-parametric methods that learn classifiers by estimating the probabilities using training data.
- We will study new techniques that use training data to learn the classifiers directly without estimating any probabilistic structure.
- In particular, we will study the k-nearest neighbor classifier, linear discriminant functions and support vector machines, neural networks, and decision trees.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

The Nearest Neighbor Classifier

- Given the training data $\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ as a set of n labeled examples, the *nearest neighbor classifier* assigns a test point $\mathbf{x}$ the label associated with its closest neighbor in $\mathcal{D}$.

The Nearest Neighbor Classifier

- Given the training data $\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ as a set of n labeled examples, the *nearest neighbor classifier* assigns a test point $\mathbf{x}$ the label associated with its closest neighbor in $\mathcal{D}$.
- Closeness is defined using a distance function.

The Nearest Neighbor Classifier

- Given the training data $\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ as a set of n labeled examples, the *nearest neighbor classifier* assigns a test point $\mathbf{x}$ the label associated with its closest neighbor in $\mathcal{D}$.
- Closeness is defined using a distance function.
- Given the distance function, the nearest neighbor classifier partitions the feature space into cells consisting of all points closer to a given training point than to any other training points.

The Nearest Neighbor Classifier

- All points in such a cell are labeled by the class of the training point, forming a *Voronoi tesselation* of the space.

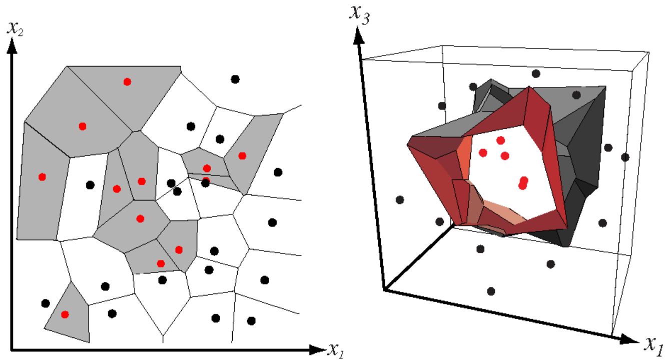


Figure: In two dimensions, the nearest neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the class of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

The *k*-Nearest Neighbor Classifier

- The *k-nearest neighbor classifier* classifies $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples.

The *k*-Nearest Neighbor Classifier

- The *k-nearest neighbor classifier* classifies $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples.
- In other words, a decision is made by examining the labels on the k-nearest neighbors and taking a vote.

The *k*-Nearest Neighbor Classifier

- The *k-nearest neighbor classifier* classifies $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples.
- In other words, a decision is made by examining the labels on the k-nearest neighbors and taking a vote.

The *k*-Nearest Neighbor Classifier

- The *k-nearest neighbor classifier* classifies **x** by assigning it the label most frequently represented among the k nearest samples.
- In other words, a decision is made by examining the labels on the k-nearest neighbors and taking a vote.

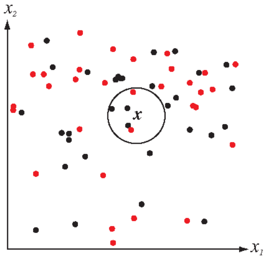


Figure: The k-nearest neighbor query forms a spherical region around the test point **x** until it encloses k training samples, and it labels the test point by a majority vote of these samples. In the case for $k = 5$, the test point will be labeled as black.

The k-Nearest Neighbor Classifier

- The computational complexity of the nearest neighbor algorithm — both in space (storage) and time (search) — has received a great deal of analysis.

The *k*-Nearest Neighbor Classifier

- The computational complexity of the nearest neighbor algorithm — both in space (storage) and time (search) — has received a great deal of analysis.
- In the most straightforward approach, we inspect each stored point one by one, calculate its distance to $\mathbf{x}$, and keep a list of the k closest ones.

The *k*-Nearest Neighbor Classifier

- The computational complexity of the nearest neighbor algorithm — both in space (storage) and time (search) — has received a great deal of analysis.
- In the most straightforward approach, we inspect each stored point one by one, calculate its distance to $\mathbf{x}$, and keep a list of the k closest ones.
- There are some parallel implementations and algorithmic techniques for reducing the computational load in nearest neighbor searches.

The *k*-Nearest Neighbor Classifier

- Examples of algorithmic techniques include

The *k*-Nearest Neighbor Classifier

- Examples of algorithmic techniques include
  - computing partial distances using a subset of dimensions, and eliminating the points with partial distances greater than the full distance of the current closest points,

The *k*-Nearest Neighbor Classifier

- Examples of algorithmic techniques include
  - computing partial distances using a subset of dimensions, and eliminating the points with partial distances greater than the full distance of the current closest points,
  - using search trees that are hierarchically structured so that only a subset of the training points are considered during search,

The *k*-Nearest Neighbor Classifier

- Examples of algorithmic techniques include
  - computing partial distances using a subset of dimensions, and eliminating the points with partial distances greater than the full distance of the current closest points,
  - using search trees that are hierarchically structured so that only a subset of the training points are considered during search,
  - editing the training set by eliminating the points that are surrounded by other training points with the same class label.

Contents

1 Introduction
2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines
5 Neural Networks
6 Decision Tree

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$
  - Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$
  - Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
  - Symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$
  - Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
  - Symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$
  - Triangle inequality: $D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}) \geq D(\mathbf{x}, \mathbf{z})$

Distance Functions

- The nearest neighbor classifier relies on a *metric* or a *distance function* between points.
- For all points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must have the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$
  - Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
  - Symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$
  - Triangle inequality: $D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}) \geq D(\mathbf{x}, \mathbf{z})$
- If the second property is not satisfied, $D(\cdot, \cdot)$ is called a pseudometric.

Distance Functions

- A general class of metrics for d-dimensional patterns is the *Minkowski metric*

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{1/p}$$

also referred to as the L_p *norm*.

Distance Functions

- A general class of metrics for d-dimensional patterns is the *Minkowski metric*

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{1/p}$$

  also referred to as the L_p *norm*.

- The *Euclidean distance* is the L_2 norm

$$L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^2 \right)^{1/2}$$

Distance Functions

- A general class of metrics for d-dimensional patterns is the *Minkowski metric*

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{1/p}$$

also referred to as the L_p *norm*.

- The *Euclidean distance* is the L_2 norm

$$L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^2 \right)^{1/2}$$

- The *Manhattan* or *city block distance* is the L_1 norm

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|$$

Distance Functions

- The L_∞ norm is the maximum of the distances along individual coordinate axes

$$L_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|$$

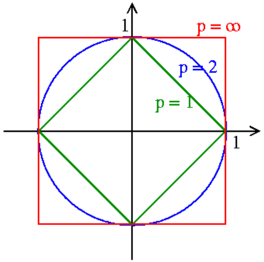


Figure: Each colored shape consists of points at a distance 1.0 from the origin, measured using different values of p in the Minkowski L_p metric.

Contents

1 Introduction
2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines
5 Neural Networks
6 Decision Tree

Feature Normalization

- We should be careful about scaling of the coordinate axes when we compute these metrics.

Feature Normalization

- We should be careful about scaling of the coordinate axes when we compute these metrics.
- When there is great difference in the range of the data along different axes in a multidimensional space, these metrics implicitly assign more weighting to features with large ranges than those with small ranges.

Feature Normalization

- We should be careful about scaling of the coordinate axes when we compute these metrics.
- When there is great difference in the range of the data along different axes in a multidimensional space, these metrics implicitly assign more weighting to features with large ranges than those with small ranges.
- *Feature normalization* can be used to approximately equalize ranges of the features and make them have approximately the same effect in the distance computation.

Feature Normalization

- The following methods can be used to independently normalize each feature.

Feature Normalization

- The following methods can be used to independently normalize each feature.
- *Linear scaling to unit range:*
  Given a lower bound l and an upper bound u for a feature $x \in \mathbb{R}$,
  $$\tilde{x} = \frac{x - l}{u - l}$$
  results in $\tilde{x}$ being in the $[0, 1]$ range.

Feature Normalization

- The following methods can be used to independently normalize each feature.
- *Linear scaling to unit range:*
  Given a lower bound l and an upper bound u for a feature $x \in \mathbb{R}$,
  $$\tilde{x} = \frac{x - l}{u - l}$$
  results in $\tilde{x}$ being in the $[0, 1]$ range.
- *Linear scaling to unit variance:*
  A feature $x \in \mathbb{R}$ can be transformed to a random variable with zero mean and unit variance as
  $$\tilde{x} = \frac{x - \mu}{\sigma}$$
  where μ and σ are the sample mean and the sample standard deviation of that feature, respectively.

Feature Normalization

- *Normalization using the cumulative distribution function:* Given a random variable $x \in \mathbb{R}$ with cumulative distribution function $F_x(x)$, the random variable $\tilde{x}$ resulting from the transformation $\tilde{x} = F_x(x)$ will be uniformly distributed in the $[0, 1]$ range.

Feature Normalization

- *Normalization using the cumulative distribution function:*
  Given a random variable $x \in \mathbb{R}$ with cumulative distribution function $F_x(x)$, the random variable $\tilde{x}$ resulting from the transformation $\tilde{x} = F_x(x)$ will be uniformly distributed in the $[0, 1]$ range.
- *Rank normalization:*
  Given the sample for a feature as $x_1, \ldots, x_n \in \mathbb{R}$, first we find the order statistics $x^{(1)}, \ldots, x^{(n)}$ and then replace each pattern's feature value by its corresponding normalized rank as

$$\tilde{x}_i = \frac{\underset{x_1,\ldots,x_n}{\text{rank}}(x_i) - 1}{n - 1}$$

  where x_i is the feature value for the i'th pattern. This procedure uniformly maps all feature values to the $[0, 1]$ range.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- k-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Linear Discriminant Functions

- A classifier that uses *discriminant functions* assigns a feature vector $\mathbf{x}$ to class w_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$$

where $g_i(\mathbf{x}), i = 1, \dots, c$, are the discriminant functions for c classes.

Linear Discriminant Functions

- A classifier that uses *discriminant functions* assigns a feature vector $\mathbf{x}$ to class w_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$$

  where $g_i(\mathbf{x}), i = 1, \ldots, c$, are the discriminant functions for c classes.
- A discriminant function that is a linear combination of the components of $\mathbf{x}$ is called a *linear discriminant function* and can be written as

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

  where $\mathbf{w}$ is the *weight vector* and w_0 is the *bias* (or threshold weight).

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

The Two-Category Case

- For the two-category case, the decision rule can be written as

$$\text{Decide} \quad \begin{cases} w_1 & \text{if } g(\mathbf{x}) > 0 \\ w_2 & \text{otherwise} \end{cases}$$

The Two-Category Case

- For the two-category case, the decision rule can be written as

$$\text{Decide} \quad \begin{cases} w_1 & \text{if } g(\mathbf{x}) > 0 \\ w_2 & \text{otherwise} \end{cases}$$

- The equation $g(\mathbf{x}) = 0$ defines the decision boundary that separates points assigned to w_1 from points assigned to w_2.

The Two-Category Case

- For the two-category case, the decision rule can be written as

$$\text{Decide} \quad \begin{cases} w_1 & \text{if } g(\mathbf{x}) > 0 \\ w_2 & \text{otherwise} \end{cases}$$

- The equation $g(\mathbf{x}) = 0$ defines the decision boundary that separates points assigned to w_1 from points assigned to w_2.
- When $g(\mathbf{x})$ is linear, the decision surface is a hyperplane whose orientation is determined by the normal vector $\mathbf{w}$ and location is determined by the bias w_0.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

The Multicategory Case

- There is more than one way to devise multicategory classifiers with linear discriminant functions.

The Multicategory Case

- There is more than one way to devise multicategory classifiers with linear discriminant functions.
- For example, we can pose the problem as c two-class problems, where the i'th problem is solved by a linear discriminant that separates points assigned to w_i from those not assigned to w_i.

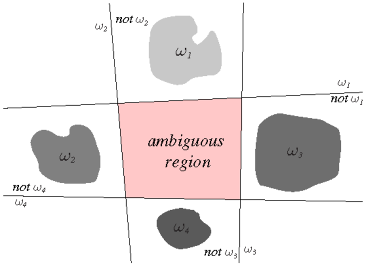
The Multicategory Case

- There is more than one way to devise multicategory classifiers with linear discriminant functions.
- For example, we can pose the problem as c two-class problems, where the i'th problem is solved by a linear discriminant that separates points assigned to w_i from those not assigned to w_i.
- Alternatively, we can use $c(c-1)/2$ linear discriminants, one for every pair of classes.

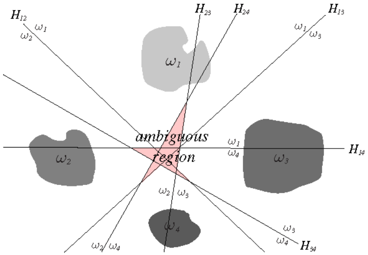
The Multicategory Case

- There is more than one way to devise multicategory classifiers with linear discriminant functions.
- For example, we can pose the problem as c two-class problems, where the i'th problem is solved by a linear discriminant that separates points assigned to w_i from those not assigned to w_i.
- Alternatively, we can use $c(c-1)/2$ linear discriminants, one for every pair of classes.
- Also, we can use c linear discriminants, one for each class, and assign $\mathbf{x}$ to w_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$.

The Multicategory Case



Boundaries separate w_i from $\neg w_i$.



Boundaries separate w_i from w_j.

Linear decision boundaries for a four-class problem devised as four two-class problems (left figure) and six pairwise problems (right figure). The pink regions have ambiguous category assignments.

The Multicategory Case

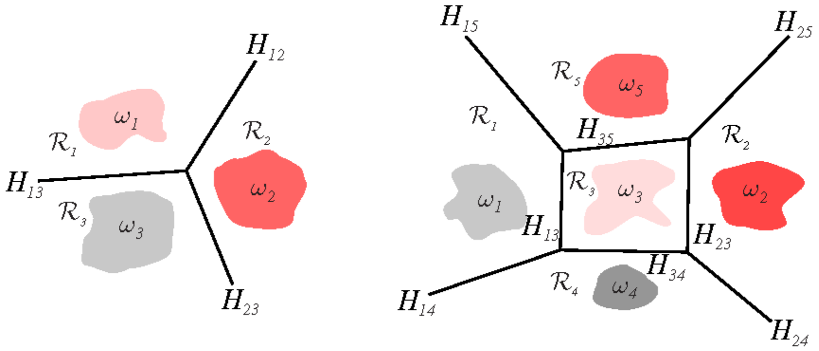


Figure: Linear decision boundaries produced by using one linear discriminant for each class. $\mathbf{w_i} - \mathbf{w_j}$ is the normal vector for the decision boundary that separates the decision region for class w_i from class w_j.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Generalized Linear Discriminant Functions

- The linear discriminant function $g(\mathbf{x})$ can be written as

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i$$

where $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_d)^T$.

Generalized Linear Discriminant Functions

- The linear discriminant function $g(\mathbf{x})$ can be written as

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i$$

  where $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_d)^T$.

- We can obtain the *quadratic discriminant function* by adding second-order terms as

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i + \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbf{w}_{ij} \mathbf{x}_i \mathbf{x}_j$$

  which result in more complicated decision boundaries (hyperquadrics).

Generalized Linear Discriminant Functions

- Adding higher-order terms gives the *generalized linear discriminant function*

$$g(\mathbf{x}) = \sum_{i=1}^{d'} \mathbf{a}_i \mathbf{y}_i(\mathbf{x}) = \mathbf{a}^T \mathbf{y}$$

where $\mathbf{a}$ is a d'-dimensional weight vector and d' functions $\mathbf{y}_i(\mathbf{x})$ are arbitrary functions of $\mathbf{x}$.

Generalized Linear Discriminant Functions

- Adding higher-order terms gives the *generalized linear discriminant function*

$$g(\mathbf{x}) = \sum_{i=1}^{d'} \mathbf{a}_i \mathbf{y}_i(\mathbf{x}) = \mathbf{a}^T \mathbf{y}$$

  where $\mathbf{a}$ is a d'-dimensional weight vector and d' functions $\mathbf{y}_i(\mathbf{x})$ are arbitrary functions of $\mathbf{x}$.
- The physical interpretation is that the functions $\mathbf{y}_i(\mathbf{x})$ map point $\mathbf{x}$ in d-dimensional space to point $\mathbf{y}$ in d'-dimensional space.

Generalized Linear Discriminant Functions

- Then, the discriminant $g(\mathbf{x}) = \mathbf{a}^T\mathbf{y}$ separates points in the transformed space using a hyperplane passing through the origin.

Generalized Linear Discriminant Functions

- Then, the discriminant $g(\mathbf{x}) = \mathbf{a}^T\mathbf{y}$ separates points in the transformed space using a hyperplane passing through the origin.
- This mapping to a higher dimensional space brings problems and additional requirements for computation and data.

Generalized Linear Discriminant Functions

- Then, the discriminant $g(\mathbf{x}) = \mathbf{a}^T\mathbf{y}$ separates points in the transformed space using a hyperplane passing through the origin.
- This mapping to a higher dimensional space brings problems and additional requirements for computation and data.
- However, certain assumptions can make the problem tractable.

Generalized Linear Discriminant Functions

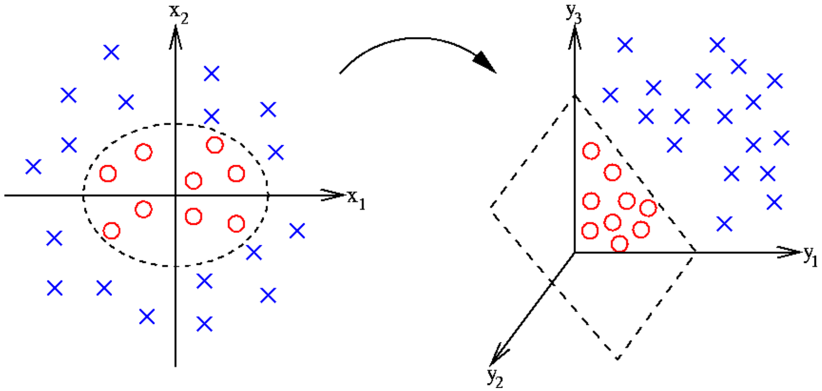


Figure: Mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ where points $(x_1, x_2)^T$ in the original space become $(y_1, y_2, y_3)^T = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$ in the new space. The planar decision boundary in the new space corresponds to a non-linear decision boundary in the original space.

Generalized Linear Discriminant Functions

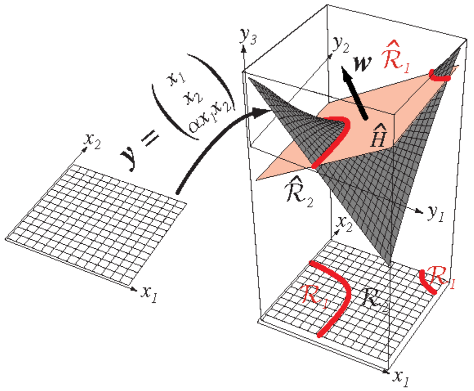


Figure: Mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ where points $(x_1, x_2)^T$ in the original space become $(y_1, y_2, y_3)^T = (x_1, x_2, \alpha x_1 x_2)^T$ in the new space. The decision regions $\hat{\mathcal{R}}_1$ and $\hat{\mathcal{R}}_2$ are separated by a plane in the new space where the corresponding regions $\mathcal{R}_1$ and $\mathcal{R}_2$ in the original space are separated by non-linear boundaries ($\mathcal{R}_1$ is also not connected).

Contents

1 Introduction

2 Nearest Neighbor Classifier
- *k*-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Support Vector Machines

- We have seen that linear discriminant functions are optimal if the underlying distributions are Gaussians having equal covariance for each class.

Support Vector Machines

- We have seen that linear discriminant functions are optimal if the underlying distributions are Gaussians having equal covariance for each class.
- In the general case, the problem of finding linear discriminant functions can be formulated as a problem of optimizing a criterion function.

Support Vector Machines

- We have seen that linear discriminant functions are optimal if the underlying distributions are Gaussians having equal covariance for each class.
- In the general case, the problem of finding linear discriminant functions can be formulated as a problem of optimizing a criterion function.
- Among all hyperplanes separating the data, there exists a unique one yielding the maximum margin of separation between the classes.

Support Vector Machines

- Given a set of training patterns and class labels as $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n) \in \mathbb{R}^d \times \{\pm 1\}$, the goal is to find a classifier function $f : \mathbb{R}^d \rightarrow \{\pm 1\}$ such that $f(\mathbf{x}) = y$ will correctly classify new patterns.

Support Vector Machines

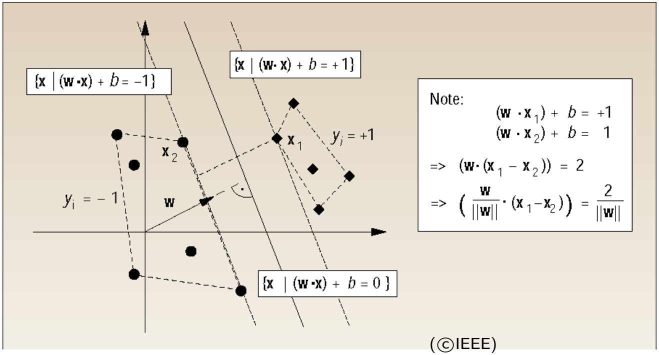
- Given a set of training patterns and class labels as $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n) \in \mathbb{R}^d \times \{\pm 1\}$, the goal is to find a classifier function $f : \mathbb{R}^d \to \{\pm 1\}$ such that $f(\mathbf{x}) = y$ will correctly classify new patterns.
- *Support vector machines* are based on the class of hyperplanes

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0, \quad \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

corresponding to decision functions

$$f(\mathbf{x}) = \mathrm{sign}((\mathbf{w} \cdot \mathbf{x}) + b)$$

Support Vector Machines



$\{x \mid (w \cdot x) + b = -1\}$
$\{x \mid (w \cdot x) + b = +1\}$
$\{x \mid (w \cdot x) + b = 0\}$
x_1
x_2
$y_i = +1$
$y_i = -1$
w
Note:
$(w \cdot x_1) + b = +1$
$(w \cdot x_2) + b = 1$
$\Rightarrow (w \cdot (x_1 - x_2)) = 2$
$\Rightarrow \left(\frac{w}{\|w\|} \cdot (x_1 - x_2)\right) = \frac{2}{\|w\|}$
(©IEEE)

Figure: A binary classification problem of separating balls from diamonds. The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half way between the two classes. There is a weight vector **w** and a threshold b such that the points closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x_i}) + b| = 1$ corresponding to $y_i((\mathbf{w} \cdot \mathbf{x_i}) + b) \geq 1$. The margin, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$.

Support Vector Machines

- To construct the optimal hyperplane, we can define the following optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to } y_i((\mathbf{w}\cdot\mathbf{x_i})+b) \geq 1, \quad i = 1,\ldots,n \end{aligned}$$

Support Vector Machines

- To construct the optimal hyperplane, we can define the following optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to } y_i((\mathbf{w}\cdot\mathbf{x_i})+b)\geq 1, \quad i=1,\dots,n \end{aligned}$$

- This constrained optimization problem is solved using Lagrange multipliers $\alpha_i \geq 0$ and the Lagrangian

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n}\alpha_i(y_i((\mathbf{w}\cdot\mathbf{x_i})+b)-1)$$

where L has to be minimized w.r.t the prime variables $\mathbf{w}$ and b, and maximized w.r.t. the dual variables α_i.

Support Vector Machines

- The solution can be obtained using quadratic programming techniques where the solution vector

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \, y_i \, \mathbf{x_i}$$

is the summation of a subset of the training patterns, called the *support vectors*, whose α_i are non-zero.

Support Vector Machines

- The solution can be obtained using quadratic programming techniques where the solution vector

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \, y_i \, \mathbf{x_i}$$

  is the summation of a subset of the training patterns, called the *support vectors*, whose α_i are non-zero.
- The support vectors lie on the margin and carry all relevant information about the classification problem (the remaining patterns are irrelevant).

Support Vector Machines

- Both the quadratic programming problem and the final decision function

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i \, y_i \, (\mathbf{x} \cdot \mathbf{x_i}) + b \right)$$

depend only on the dot products between patterns.

Support Vector Machines

- Both the quadratic programming problem and the final decision function

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i \, y_i \, (\mathbf{x} \cdot \mathbf{x_i}) + b\right)$$

  depend only on the dot products between patterns.
- We can generalize this result to the non-linear case by mapping the original input space into some other space $\mathcal{F}$ using a non-linear map $\Phi : \mathbb{R}^d \to \mathcal{F}$ and perform the linear algorithm in the $\mathcal{F}$ space which only requires the dot products

$$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})\Phi(\mathbf{y})$$

Support Vector Machines

- Even though $\mathcal{F}$ may be high-dimensional, a simple *kernel* $k(\mathbf{x}, \mathbf{y})$ such as the following can be computed efficiently.

Table: Common kernel functions.

Polynomial	$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$
Sigmoidal	$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \theta)$
Radial basis function	$k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/(2\sigma^2))$

Support Vector Machines

- Even though $\mathcal{F}$ may be high-dimensional, a simple *kernel* $k(\mathbf{x}, \mathbf{y})$ such as the following can be computed efficiently.

Table: Common kernel functions.

Polynomial	$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$
Sigmoidal	$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \theta)$
Radial basis function	$k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/(2\sigma^2))$

- Once a kernel function is chosen, we can substitute $\Phi(\mathbf{x_i})$ for each training example $\mathbf{x_i}$, and perform the optimal hyperplane algorithm in $\mathcal{F}$.

Support Vector Machines

- This results in the non-linear decision function of the form

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i \, y_i \, k(\mathbf{x}, \mathbf{x_i}) + b \right)$$

  where the parameters α_i are computed as the solution of the quadratic programming problem.

Support Vector Machines

- This results in the non-linear decision function of the form

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i \, y_i \, k(\mathbf{x}, \mathbf{x_i}) + b \right)$$

  where the parameters α_i are computed as the solution of the quadratic programming problem.
- In the original input space, the hyperplane corresponds to a non-linear decision function whose form is determined by the kernel.

Support Vector Machines

- SVMs are quite popular because of their intuitive formulation using computational learning theory and their high performances in practical applications.

Support Vector Machines

- SVMs are quite popular because of their intuitive formulation using computational learning theory and their high performances in practical applications.
- However, we must be careful about certain issues such as the following during implementation.

Support Vector Machines

- SVMs are quite popular because of their intuitive formulation using computational learning theory and their high performances in practical applications.
- However, we must be careful about certain issues such as the following during implementation.
- *Choice of kernel functions:* We can use training data to find the best performing kernel.

Support Vector Machines

- SVMs are quite popular because of their intuitive formulation using computational learning theory and their high performances in practical applications.
- However, we must be careful about certain issues such as the following during implementation.
- *Choice of kernel functions:* We can use training data to find the best performing kernel.
- *Computational requirements of the quadratic program:* Several algorithms exist for speeding up the optimization problem (see references).

Support Vector Machines

- *Extension to multiple classes:* We can train a separate SVM for each class, compute the output value using each SVM, and select the class that assigns the unknown pattern the furthest into the positive region.

Support Vector Machines

- *Extension to multiple classes:* We can train a separate SVM for each class, compute the output value using each SVM, and select the class that assigns the unknown pattern the furthest into the positive region.
- *Converting the output of an SVM to a posterior probability for post-processing:* We can fit a sigmoid model to the posterior probability $P(y = 1|f(\mathbf{x}))$ as

$$P(y = 1|f(\mathbf{x})) = \frac{1}{1 + \exp(a\,f(\mathbf{x}) + b)}$$

  where the parameters a and b are learned using maximum likelihood estimation from a training set.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- k-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Neural Networks

- Linear discriminants try to find hyperplane decision boundaries.

Neural Networks

- Linear discriminants try to find hyperplane decision boundaries.
- Kernel mappings can be used to obtain arbitrary decision regions.

Neural Networks

- Linear discriminants try to find hyperplane decision boundaries.
- Kernel mappings can be used to obtain arbitrary decision regions.
- *Neural networks* try to learn the parameters of the nonlinear mapping at the same time as the linear discriminant.

Neural Networks

- Linear discriminants try to find hyperplane decision boundaries.
- Kernel mappings can be used to obtain arbitrary decision regions.
- *Neural networks* try to learn the parameters of the nonlinear mapping at the same time as the linear discriminant.
- A neural network consists of an *input layer* , an *output layer* and usually one or more *hidden layers* that are interconnected by modifiable weights represented by links between layers.

Neural Networks

- The intermediate layers are called hidden because their activation are not directly seen from the input or output.

Neural Networks

- The intermediate layers are called hidden because their activation are not directly seen from the input or output.
- The function of units is loosely based on properties of biological neurons, and hence they are sometimes called *neurons* .

Neural Networks

- The intermediate layers are called hidden because their activation are not directly seen from the input or output.
- The function of units is loosely based on properties of biological neurons, and hence they are sometimes called *neurons* .
- In pattern recognition applications, the input units represent the components of a feature vector and the output units give the values of the discriminant functions used for classification.

Neural Networks

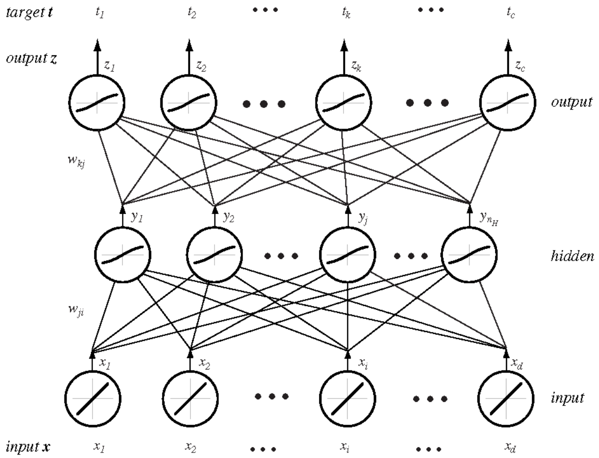


Figure: A $d - n_H - c$ fully connected three-layer network.

Feedforward Operation

- Each hidden unit computes the weighted sum of its inputs to form a net activation value, $net = \mathbf{w}^T\mathbf{x}$.

Feedforward Operation

- Each hidden unit computes the weighted sum of its inputs to form a net activation value, $net = \mathbf{w}^T\mathbf{x}$.
- Then, it emits an output that is a nonlinear function of its activation, $f(net)$.

Feedforward Operation

- Each hidden unit computes the weighted sum of its inputs to form a net activation value, $net = \mathbf{w}^T\mathbf{x}$.
- Then, it emits an output that is a nonlinear function of its activation, $f(net)$.
- Each output unit similarly computes its net activation based on the hidden unit signals in the previous layer and emits a value using a nonlinear function based on its activation.

Feedforward Operation

- Each hidden unit computes the weighted sum of its inputs to form a net activation value, $net = \mathbf{w}^T\mathbf{x}$.
- Then, it emits an output that is a nonlinear function of its activation, $f(net)$.
- Each output unit similarly computes its net activation based on the hidden unit signals in the previous layer and emits a value using a nonlinear function based on its activation.
- The output corresponds to a nonlinear function of the input feature vector.

Neural Networks

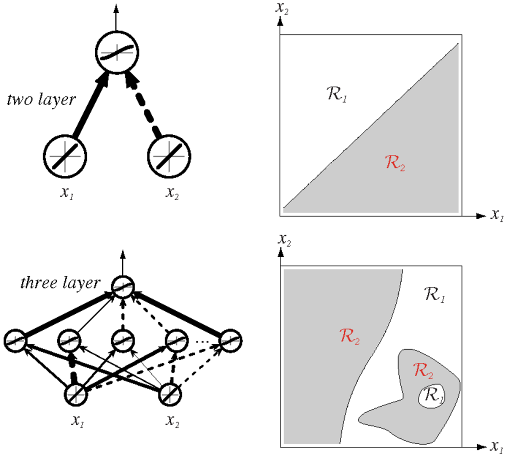


Figure: Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three or more layer networks can implement arbitrary decision boundaries.

Neural Networks

- We can think of the network as computing discriminant functions and can classify the input according to which discriminant function is the largest.

Neural Networks

- We can think of the network as computing discriminant functions and can classify the input according to which discriminant function is the largest.
- Theoretically, every decision (and every continuous function) can be implemented by a three-layer network given sufficient number of hidden units, proper nonlinearities and weights.

Neural Networks

- We can think of the network as computing discriminant functions and can classify the input according to which discriminant function is the largest.
- Theoretically, every decision (and every continuous function) can be implemented by a three-layer network given sufficient number of hidden units, proper nonlinearities and weights.
- However, this theorem does not tell us how we can learn the network structure, the nonlinear functions and the weights in practice.

Backpropagation Algorithm

- *Backpropagation* is one of the simplest and most general methods for supervised training of multilayer neural networks.

Backpropagation Algorithm

- *Backpropagation* is one of the simplest and most general methods for supervised training of multilayer neural networks.
- It is based on the least-mean-squared algorithm where the error is proportional to the square of the difference between the actual output and the desired output.

Backpropagation Algorithm

- *Backpropagation* is one of the simplest and most general methods for supervised training of multilayer neural networks.
- It is based on the least-mean-squared algorithm where the error is proportional to the square of the difference between the actual output and the desired output.
- The dependence of the error on the hidden-to-output layer weights is straightforward.

Backpropagation Algorithm

- *Backpropagation* is one of the simplest and most general methods for supervised training of multilayer neural networks.
- It is based on the least-mean-squared algorithm where the error is proportional to the square of the difference between the actual output and the desired output.
- The dependence of the error on the hidden-to-output layer weights is straightforward.
- The backpropagation algorithm allows us to calculate an effective error for each hidden unit and derive a learning rule for the input-to-hidden weights.

Backpropagation Algorithm

- Using the notation of Figure 1, define the training error on a pattern to be the sum over output units of the squared difference between the desired output t_k and the actual output z_k as

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2$$

Backpropagation Algorithm

- Using the notation of Figure 1, define the training error on a pattern to be the sum over output units of the squared difference between the desired output t_k and the actual output z_k as

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2$$

- The backpropagation learning rule changes the weights in a direction that reduces the error (gradient descent)

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

where η is the learning rate.

Backpropagation Algorithm

- For the hidden-to-output weights, the weight update rule becomes

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \eta(t_k - z_k) f'(net_k) y_j$$

  where net_k is the net activation value.

Backpropagation Algorithm

- For the hidden-to-output weights, the weight update rule becomes

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \eta(t_k - z_k) f'(net_k) y_j$$

where net_k is the net activation value.

- For the input-to-hidden weights, the weight update rule becomes

$$\begin{aligned} \Delta w_{ji} &= -\eta \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \\ &= \eta \left(\sum_{k=1}^{c} w_{kj}(t_k - z_k) f'(net_k) \right) f'(net_j) x_i \end{aligned}$$

Backpropagation Algorithm

- The algorithm is called “backpropagation” because it propagates the error from the output layer back to the hidden layers.

Backpropagation Algorithm

- The algorithm is called “backpropagation” because it propagates the error from the output layer back to the hidden layers.
- As with all gradient descent procedures, the exact behavior of the backpropagation algorithm depends on the starting point.

Backpropagation Algorithm

- The algorithm is called “backpropagation” because it propagates the error from the output layer back to the hidden layers.
- As with all gradient descent procedures, the exact behavior of the backpropagation algorithm depends on the starting point.
- The initial weights are randomly chosen from a uniform distribution symmetric around zero.

Backpropagation Algorithm

- There are three main training protocols for neural networks:

Backpropagation Algorithm

- There are three main training protocols for neural networks:
  - In *stochastic training* , patterns are chosen randomly from the training set and the weights are updated for each pattern presentation.

Backpropagation Algorithm

- There are three main training protocols for neural networks:
    - In *stochastic training* , patterns are chosen randomly from the training set and the weights are updated for each pattern presentation.
    - In *batch training* , all patterns are presented to the network before learning takes place.

Backpropagation Algorithm

- There are three main training protocols for neural networks:
  - In *stochastic training* , patterns are chosen randomly from the training set and the weights are updated for each pattern presentation.
  - In *batch training* , all patterns are presented to the network before learning takes place.
  - In *online training* , each pattern is presented only once.

Contents

1 Introduction

2 Nearest Neighbor Classifier
- k-Nearest Neighbor Classifier
- Distance Functions
- Feature Normalization

3 Linear Discriminant Functions
- The Two-Category Case
- The Multicategory Case
- Generalized Linear Discriminant Functions

4 Support Vector Machines

5 Neural Networks

6 Decision Tree

Decision Trees

- Most pattern recognition methods address problems where there is a natural measure of distance between feature vectors.

Decision Trees

- Most pattern recognition methods address problems where there is a natural measure of distance between feature vectors.
- What happens when the classification problem involves *nominal* data, e.g., descriptions that are discrete and without any natural notion of similarity or even ordering?

Decision Trees

- Most pattern recognition methods address problems where there is a natural measure of distance between feature vectors.
- What happens when the classification problem involves *nominal* data, e.g., descriptions that are discrete and without any natural notion of similarity or even ordering?
- A common representation for this kind of data is a *list of attributes* (instead of vector of real numbers).

Decision Trees

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.

Decision Trees

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.
- Such a sequence of questions can be represented as a *decision tree* .

Decision Trees

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.
- Such a sequence of questions can be represented as a *decision tree* .
- In a decision tree, the top node is called the *root node* , and is connected by directional links (*branches*) to other nodes.

Decision Trees

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.
- Such a sequence of questions can be represented as a *decision tree* .
- In a decision tree, the top node is called the *root node* , and is connected by directional links (*branches*) to other nodes.
- These nodes are similarly connected until terminal (*leaf*) nodes, which have no further links, are reached.

Decision Trees

- The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute.

Decision Trees

- The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute.
- Different branches from the root node correspond to different values of this attribute that are mutually distinct and exhaustive.

Decision Trees

- The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute.
- Different branches from the root node correspond to different values of this attribute that are mutually distinct and exhaustive.
- Based on the particular value of that attribute for that pattern, the appropriate branch is followed to a subsequent node.

Decision Trees

- The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute.
- Different branches from the root node correspond to different values of this attribute that are mutually distinct and exhaustive.
- Based on the particular value of that attribute for that pattern, the appropriate branch is followed to a subsequent node.
- Similar decisions are made in subsequent nodes until a leaf node is reached.

Decision Trees

- The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute.
- Different branches from the root node correspond to different values of this attribute that are mutually distinct and exhaustive.
- Based on the particular value of that attribute for that pattern, the appropriate branch is followed to a subsequent node.
- Similar decisions are made in subsequent nodes until a leaf node is reached.
- Each leaf node bears a class label and the pattern is assigned the label of the leaf node reached.

Decision Trees

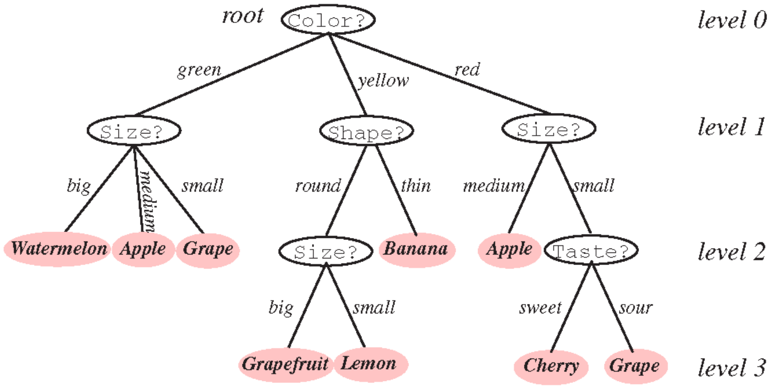


Figure: An example decision tree that uses the attributes {color, shape, taste, size}. Note that the same question can appear in different places in the tree and that different questions can have different numbers of branches. Moreover, different leaf nodes can be labeled by the same class.

Decision Trees

- Learning a decision tree is based on partitioning the set of training examples into smaller and smaller subsets where each subset is as *pure* as possible.

Decision Trees

- Learning a decision tree is based on partitioning the set of training examples into smaller and smaller subsets where each subset is as *pure* as possible.
- Purity for a particular subset is measured according to the number of training samples in that subset having the same class label.

Decision Trees

- Learning a decision tree is based on partitioning the set of training examples into smaller and smaller subsets where each subset is as *pure* as possible.
- Purity for a particular subset is measured according to the number of training samples in that subset having the same class label.
- Different criteria can grow trees differently.

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?
  - How many splits will there be at a node?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?
  - How many splits will there be at a node?
  - Which attribute should be tested at a node?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?
  - How many splits will there be at a node?
  - Which attribute should be tested at a node?
  - When should a node be declared as a leaf?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?
  - How many splits will there be at a node?
  - Which attribute should be tested at a node?
  - When should a node be declared as a leaf?
  - If a tree becomes too large, how can it be simplified?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
  - Are the attributes binary-valued or multi-valued?
  - How many splits will there be at a node?
  - Which attribute should be tested at a node?
  - When should a node be declared as a leaf?
  - If a tree becomes too large, how can it be simplified?
  - How can a class label be assigned to a leaf node?

Decision Trees

- Decision tree learning is a recursive process where the following questions arise:
    - Are the attributes binary-valued or multi-valued?
    - How many splits will there be at a node?
    - Which attribute should be tested at a node?
    - When should a node be declared as a leaf?
    - If a tree becomes too large, how can it be simplified?
    - How can a class label be assigned to a leaf node?
    - How should missing data be handled?

Decision Trees

- Each decision outcome at a node is called a *split*.

Decision Trees

- Each decision outcome at a node is called a *split* .
- Each tree can be represented using just binary splits (called a *binary tree*).

Decision Trees

- Each decision outcome at a node is called a *split* .
- Each tree can be represented using just binary splits (called a *binary tree*).
- For numerical attributes, splitting questions have the form “is $x \leq x_0$”?

Decision Trees

- Each decision outcome at a node is called a *split* .
- Each tree can be represented using just binary splits (called a *binary tree*).
- For numerical attributes, splitting questions have the form "is $x \leq x_0$"?
- For nonnumerical attributes, splitting questions have the form "is $x \in A$" where A is a subset of the possible values of x.

Decision Trees

- For a given node, the particular splitting attribute and the corresponding question can be chosen using a search according to the purity (or impurity) measures that are based on entropy, variance, Gini or misclassification criteria.

Decision Trees

- For a given node, the particular splitting attribute and the corresponding question can be chosen using a search according to the purity (or impurity) measures that are based on entropy, variance, Gini or misclassification criteria.
- Commonly used criteria about deciding when to stop splitting include thresholds on impurity or number of examples remaining at a node, or statistical tests on the significance of reduction in impurity.

Decision Trees

- For a given node, the particular splitting attribute and the corresponding question can be chosen using a search according to the purity (or impurity) measures that are based on entropy, variance, Gini or misclassification criteria.
- Commonly used criteria about deciding when to stop splitting include thresholds on impurity or number of examples remaining at a node, or statistical tests on the significance of reduction in impurity.
- Alternatively, a tree can be grown fully, and then can be *pruned* by considering the leaf nodes or even subtrees for elimination or merging.

Decision Trees

- For a given node, the particular splitting attribute and the corresponding question can be chosen using a search according to the purity (or impurity) measures that are based on entropy, variance, Gini or misclassification criteria.
- Commonly used criteria about deciding when to stop splitting include thresholds on impurity or number of examples remaining at a node, or statistical tests on the significance of reduction in impurity.
- Alternatively, a tree can be grown fully, and then can be *pruned* by considering the leaf nodes or even subtrees for elimination or merging.
- Once the leaf nodes are finalized, they can be labeled by the class that has the most patterns represented in the corresponding nodes.

Decision Trees

- An interesting problem that can arise during training, classification or both is *missing data* .

Decision Trees

- An interesting problem that can arise during training, classification or both is *missing data* .
- There are several possible reasons for a value to be missing, such as: it was not measured; there was an instrument malfunction; the attribute does not apply, or the attribute's value cannot be known.

Decision Trees

- An interesting problem that can arise during training, classification or both is *missing data* .
- There are several possible reasons for a value to be missing, such as: it was not measured; there was an instrument malfunction; the attribute does not apply, or the attribute's value cannot be known.
- Decision trees can handle missing data by using the primary decision attribute at a node whenever possible, and use alternative attributes when a pattern is missing the primary attribute.

Decision Trees

- An interesting problem that can arise during training, classification or both is *missing data* .
- There are several possible reasons for a value to be missing, such as: it was not measured; there was an instrument malfunction; the attribute does not apply, or the attribute's value cannot be known.
- Decision trees can handle missing data by using the primary decision attribute at a node whenever possible, and use alternative attributes when a pattern is missing the primary attribute.
- These alternative attributes are called *surrogate splits* and are found by maximizing the probability of making the same decision as the primary split.

Decision Trees

- In summary, tree-based tools offer the following advantages:

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.
  - They do not require any assumptions about neither the distributions nor the independence of attribute values.

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.
  - They do not require any assumptions about neither the distributions nor the independence of attribute values.
  - They are often easy to interpret by creating subgroups of data which the user may graphically analyze.

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.
  - They do not require any assumptions about neither the distributions nor the independence of attribute values.
  - They are often easy to interpret by creating subgroups of data which the user may graphically analyze.
  - They can also be easily converted to decision rules by tracing the tree from the root node to each leaf node and forming logical expressions.

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.
  - They do not require any assumptions about neither the distributions nor the independence of attribute values.
  - They are often easy to interpret by creating subgroups of data which the user may graphically analyze.
  - They can also be easily converted to decision rules by tracing the tree from the root node to each leaf node and forming logical expressions.
  - They automatically perform feature selection by using only the attributes that can partition the measurement space the most effectively.

Decision Trees

- In summary, tree-based tools offer the following advantages:
  - They can operate on both numerical (continuous) and nominal (discrete) measurements.
  - They do not require any assumptions about neither the distributions nor the independence of attribute values.
  - They are often easy to interpret by creating subgroups of data which the user may graphically analyze.
  - They can also be easily converted to decision rules by tracing the tree from the root node to each leaf node and forming logical expressions.
  - They automatically perform feature selection by using only the attributes that can partition the measurement space the most effectively.
  - They are capable of dealing with missing data.