Supervised Learning (Part 2) GE 461, Spring 202)

Slide credits: Çiğdem Gündüz Demir

Adding Nonlinearity to Linear Discriminants

(revisited)

- Linear discriminants yield hyperplane decision boundaries
- If they are not sufficient to construct a "good" model
 - Transform the space into a new one using nonlinear mappings and construct linear discriminants on the transformed space → Support vector machines
 - Learn nonlinearity at the same time as you learn the linear discriminants → Neural networks

XOR Problem



Support vector machines use the idea of nonlinear mapping to find a linearly separable space

XOR Problem



Neural networks learn the nonlinearity at the same time as they learn the linear discriminants (learn all the weights at the same time)

(Slide credit: Selim Aksoy, CS Bilkent)

• 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 $\mathbf{w} = (\mathbf{w}_1, \dots, \mathbf{w}_d)^T$.

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).



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 a is a d'-dimensional weight vector and d' functions $y_i(x)$ are arbitrary functions of x.

The physical interpretation is that the functions y_i(x) map point x in d-dimensional space to point y in d'-dimensional space.





Figure 3: 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.





Figure 4: 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).

CS 551, Fall 2019

©2019, Selim Aksoy (Bilkent University)



- 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.



Figure 5: The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points (left). Maximizing the margin leads to a particular choice of decision boundary (right). The location of this boundary is determined by a subset of the data points, known as the support vectors, which are indicated by the circles.



CS 551, Fall 2019

©2019, Selim Aksoy (Bilkent University)

- ► Given a set of training patterns and class labels as (x₁, y₁),..., (x_n, y_n) ∈ ℝ^d × {±1}, the goal is to find a classifier function f : ℝ^d → {±1} such that f(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

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





Figure 6: 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 \mathbf{w} and a threshold b such that the points closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$ corresponding to $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \ge 1$. The margin, measured perpendicularly to the hyperplane, equals $2/||\mathbf{w}||$.



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

minimize
$$\frac{1}{2} \|\mathbf{w}\|^2$$

subject to $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \ge 1, \quad i = 1, \dots, n.$

• This constrained optimization problem is solved using Lagrange multipliers $\alpha_i \ge 0$ 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 w and *b*, and maximized w.r.t. the dual variables α_i .

polynomial kernel $K(x, x_i) = (x \cdot x_i + 1)^p$

radial basis function kernel $K(x, x_i) = exp(-\gamma ||x - x_i||^2)$

sigmoidal kernel $K(x, x_i) = tanh(\eta x \cdot x_i + \nu)$

It gives a solution where the vector w is a linear combination of support vectors

- The maximum margin can be written as dot products between the support vectors and all samples
- When the input space is transformed using a mapping function Φ, the dot product between a support vector and a sample is Φ(x). Φ(x_i)
- You do not need to know the mapping function itself but the dot product in the transformed space
- This is known as the **KERNEL TRICK** in support vector machines



For non-separable data, we relax the constraints



where ξ_i , i = 1, ..., n, are called the slack variables and *C* is a regularization parameter.

• The term $C \sum_{i=1}^{n} \xi_i$ can be thought of as measuring some amount of misclassification where lowering the value of *C* corresponds to a smaller penalty for misclassification (see references).



- A decision tree provides a classification or regression model built in the form of a tree structure
- It corresponds to partitioning the input space into localized regions, each of which can make different decision
- Decision tree learning aims to find these partitions



- It is composed of internal decision nodes and leaves
 - An internal node corresponds to a test function whose discrete outcomes label the branches
 - A leaf defines a localized region (and a class for classification and a numerical value for regression)



- In training, the goal is to construct a tree yielding the minimum error
 - At each step, the "best" split is selected among all possible ones
 - Tree construction iteratively continues until all leaves are pure
 - This is the basis of CART, ID3, and C4.5 algorithms
- For an unseen instance, start at the root, take branches according to the test outcomes until a leaf is reached
 - The value in the leaf is the output





- Univariate trees
 - Test functions use one feature at a time
 - Define splits orthogonal to the coordinate axes



Discrete features



Continuous features

- Multivariate trees
 - Test functions use more than one feature at a time



- For tree construction, iteratively select the "best" split until all leaves are pure
- What is the "best" split?
 - The goodness of a split is quantified by an impurity measure
 - Entropy is one of the most commonly used measures



Construct a tree for the training instances below

	X ₁	X 2	class
S_1	red	0.5	1
S ₂	red	0.2	2
S ₃	green	0.5	2
S ₄	blue	0.1	1
S ₅	red	-0.5	2
S ₆	green	0.1	1
S ₇	green	0.4	2
S ₈	blue	0.0	2

- At every step
 - 1. List all possible splits
 - 2. Calculate the entropy for every split
 - 3. Select the one with the minimum entropy

Construct a tree for the training instances below

	x 1	x ₂	class
S_1	red	0.5	1
S ₂	red	0.2	2
S ₃	green	0.5	2
S ₄	blue	0.1	1
S ₅	red	-0.5	2
S ₆	green	0.1	1
S ₇	green	0.4	2
S ₈	blue	0.0	2

- For classification
 - Each different value of a discrete
 feature will define a split
 - Halfway between continuous feature values of the samples belonging to different classes will be split points
 - Possible splits:

$x_1 = red$	x ₁ = green	$x_1 = blue$
$x_2 \le 0.05$	x ₂ ≤ 0.15	x ₂ ≤ 0.45

Construct a tree for the training instances below

	X ₁	X 2	class
S_1	red	0.5	1
S ₂	red	0.2	2
S ₃	green	0.5	2
S ₄	blue	0.1	1
S_5	red	-0.5	2
S ₆	green	0.1	1
S ₇	green	0.4	2
S ₈	blue	0.0	2

 Calculate the entropy for all possible splits 		
$I(x_{1} = red) = P_{Yes} I(Yes) + P_{No} I(No)$ = $\frac{3}{8} \left(-\frac{1}{3} \log \frac{1}{3} - \frac{2}{3} \log \frac{2}{3} \right) + \frac{5}{8} \left(-\frac{2}{5} \log \frac{2}{5} - \frac{3}{5} \log \frac{3}{5} \right)$	$x_1 = red$ $x_1 = red$ $S_1 - Class 1 \qquad S_3 - Class 2$ $S_2 - Class 2 \qquad S_4 - Class$ $S_5 - Class 2 \qquad S_6 - Class$ $S_7 - Class 3 \qquad S_7 - Clas 3 \qquad S_7 - Cla$	2 1 1 2
= 0.9512 $I(x_2 \le 0.05) = P_{Y_{es}} I(Y_{es}) + P_{N_0} I(N_0)$	$S_7 - Class $ $S_8 - Class $	2

$$\begin{aligned} (x_2 \le 0.05) &= P_{Yes} I(Yes) + P_{No} I(No) \\ &= \frac{2}{8} \left(-0 \log 0 - 1 \log 1 \right) + \\ &= \frac{6}{8} \left(-\frac{3}{6} \log \frac{3}{6} - \frac{3}{6} \log \frac{3}{6} \right) \\ &= 0.7500 \end{aligned}$$

Construct a tree for the training instances below

	X 1	X 2	class
S ₁	red	0.5	1
S ₂	red	0.2	2
S ₃	green	0.5	2
S ₄	blue	0.1	1
S ₅	red	-0.5	2
S ₆	green	0.1	1
S ₇	green	0.4	2
S ₈	blue	0.0	2

 Select the split with the minimum entropy and continue



Continue for this branch

List all possible splits for this branch, calculate the entropy for each, and select the one with the min entropy

Alternative Splitting Criteria

Entropy at node m

$$I(m) = -\sum_{i=1}^{C} P_m(C_i) \log P_m(C_i)$$

Probability of having *i-th class at node m*

Gini impurity at node m $I(m) = \sum_{i \neq j} P_m(C_i) \quad P_m(C_j) = \frac{1}{2} \left[1 - \sum_i \left(P_m(C_i) \right)^2 \right]$

Misclassification impurity at node m

 $I(m) = 1 - \max_{i} P_m(C_i)$

When to Stop Splitting

- Until all leaves are pure → Overfitting
- To prevent overfitting
 - Set a small threshold value in the reduction in impurity
 - Use cross validation techniques (e.g., continue splitting if the cross validation error is decreasing)
 - Use an explicit measure of the complexity to encode the training samples and the tree, stop growing when the encoding size is minimized (*minimum description length principle*)
 - Use statistical tests (e.g., use chi-squared statistic to understand if a split differs significantly from a random one)
- Then, it might be useful to keep the classes existing in a leaf together with their class probabilities

Pruning

- <u>Prepruning</u>: Stop growing the tree earlier before it overfits the training samples
- <u>Postpruning</u>: Grow the tree until it overfits the training samples (all leaves are pure) then prune the grown tree
 - Reduced error pruning: Remove nodes (or subtrees) only if the pruned tree performs no worse than the unpruned one over the validation set
 - Rule post pruning: Convert a tree into a set of rules and simplify (prune) each rule by removing any preconditions that result in no-worse-than validation performance

if
$$\underbrace{(x_1 \ge \theta_1)}_{\mathbf{A}}$$
 and $\underbrace{(x_2 \ge \theta_2)}_{\mathbf{B}}$ then Class 2

Try removing A or B and see what happens on the validation set



Rule Extraction

 One advantage of using a decision tree classifier is its ability to extract human interpretable rules



- *Rule 2:* if (contact = no) and (feeding = yes) then medium-risk
- *Rule 3:* if (contact = no) and (feeding = no) and (close-living = yes) then medium-risk
- *Rule 4:* if (contact = no) and (feeding = no) and (close-living = no) then low-risk

Rule support is the percentage of the training samples covered by the rule

Regression Trees

- Continuous outputs at leaves (instead of class labels)
- Error measure is used for the goodness of a split (instead of an impurity measure)
- Iteratively grow the tree until the error measure falls below a certain threshold



Mean square error at node m



output at node m

$$E(S) = P_{left} E(left) + P_{right} E(right)$$

Mean square error of a binary split S

To estimate f_m

The mean (median) over the outputs of the training samples at node m could be used (piecewise constant approx.)

A linear function is fit over the outputs of the training samples at node m and its output value could be used (piecewise linear approx.)