COMP 448/548: Medical Image Analysis

Basic of classifiers

Çiğdem Gündüz Demir
cgunduz@ku.edu.tr

1

Learning

- In our lives, we take actions based on
  - What we observe in our environments
  - What we have previously learned
- In order to achieve a task, we should
  - Have relevant information representing the environment
  - Know the possible set of actions
  - Know the process to take an action based on the information
    - This process relies on our past experience

Face recognition
Chess playing
Car driving
Stock price prediction
Cancer diagnosis
Treatment selection
Screening
Cell counting

2

Machine learning

- The goal of machine learning is to design systems that
  - Automatically achieves tasks (output) similar to us
  - Depending on the environment (input)
  - Based on the past experience (training samples)
  - With respect to some performance measures (e.g., accuracy)

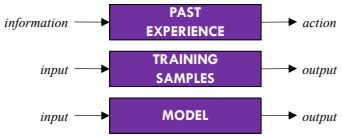
Supervised learning:
- **There is a teacher providing a label (output) for each training sample**
- **The task is to map an input space to an output space**

Unsupervised learning:
- **There is no explicit teacher that provides sample labels (outputs)**
- **The task is to find regularities (clusters) in the input space**

3

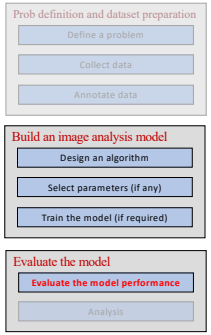
Supervised learning

- We believe that there is a process underlying training samples (their inputs and outputs)
  - We may not identify this process completely
  - But we can construct a model approximating the process
    - A function that distinguishes discrete outputs (**classification**)
    - A functional description of output in terms of inputs (**regression**)
  - **Supervised learning focuses on constructing such models**



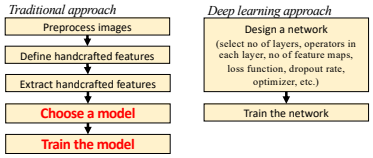
4

Supervised learning



The goodness of a model depends on

- How good its approximation is
  - No model fits all problems
  - Different models have different assumptions
- How well training samples represent the real-world
  - There may exist noise and exceptions in the samples
  - Some parts may not be covered by the samples



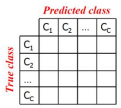
5

Model evaluation

Accuracy

- Percentage of correctly classified samples
- We may also want to consider class-based accuracies, especially when there is an unbalanced distribution among classes

Confusion matrix



Do not use the same set of (training) samples both for learning a model and its evaluation!!!

- If available, use an independent test set
- If not, create multiple independent training and test sets by partitioning samples many times
  - Bootstrapping, k-fold cross-validation, leave-one-out

6

Bayesian decision theory

7

Bayesian decision theory

- It is the fundamental statistical approach in classification
- Here it is assumed that
  1. The decision problem is posed in probabilistic terms and
  2. All relevant probability values are known

8

Bayesian decision theory


- A simple decision problem: **Fish classification**
- Let's assume that a fish emerges nature in one of the following states

State of nature: $C = \begin{cases} C_1 & \text{for hamsi} \\ C_2 & \text{for barbun} \end{cases}$

- To predict what type will emerge next, we consider C as a random variable, which is described probabilistically

Prior probabilities (a priori probabilities): $P(C_1)$ and $P(C_2)$ reflect our previous knowledge before the fish appears

$$P(C_1) + P(C_2) = 1 \quad \text{(if no other species exist)}$$

9

Bayesian decision theory


- Decide whether a fish is hamsi or barbun when
  1. We are not allowed to see the fish
  2. We know the prior probabilities
  3. The cost is the same for all incorrect decisions

Decision rule: $\text{Select} \begin{cases} \text{hamsi} & \text{if } P(C_1) > P(C_2) \\ \text{barbun} & \text{otherwise} \end{cases}$

In this case, we always make the same decision!!!

10

Bayesian decision theory


- Fortunately, we usually have more information for making our decisions
  - E.g., we can see the fish, measure its color intensity
  - We make this measurement relying on the fact that hamsi and barbun emerge nature in different colors
- This difference can be expressed in probabilistic terms, considering color intensity x as a continuous random variable whose distribution depends on the state of nature

Class-conditional probability density functions (likelihoods): $P(x|C_1)$ and $P(x|C_2)$ are the probability of observing color intensity x when the state of nature is C_1 and C_2, respectively

11

Bayesian decision theory


- Now let's combine this measurement with our previous knowledge

Joint probability

$$P(C_k, x) = P(C_k|x) \cdot P(x) = P(x|C_k) \cdot P(C_k)$$

BAYES FORMULA

$$\underbrace{P(C_k|x)}_{\text{Posterior}} = \frac{\overbrace{P(x|C_k)}^{\text{Likelihood}} \cdot \overbrace{P(C_k)}^{\text{Prior}}}{\underbrace{P(x)}_{\text{Evidence}}}$$

$$P(x) = \sum_{k=1}^{N} P(x|C_k) \cdot P(C_k)$$

$$\sum_{k=1}^{N} P(C_k|x) = 1$$

Posterior probabilities (a posteriori probabilities): $P(C_1|x)$ and $P(C_2|x)$ reflect our beliefs of having a particular fish species when the color intensity of the fish is measured as x

12

Bayesian decision theory

- Now decide whether a fish is hamsi or barbun when
  1. We can see the fish and measure its color x
  2. We know the prior probabilities and likelihoods
  3. The cost is the same for all incorrect decisions

Decision rule: Select $\begin{cases}\text{hamsi} & \text{if } P(C_1|x) > P(C_2|x) \\ \text{barbun} & \text{otherwise}\end{cases}$

We use the Bayes' decision rule to minimize the probability of error $P(error) = \int_{-\infty}^{\infty} P(error, x)dx = \int_{-\infty}^{\infty} P(error|x)P(x)dx$

For every x, keep P(error|x) as small as possible, by selecting the state of nature (class) with the highest posterior probability

13

Bayesian decision theory

- Now decide whether a fish is hamsi or barbun when
  1. We can see the fish and measure its color x
  2. We know the prior probabilities and likelihoods
  3. The cost is the same for all incorrect decisions

Decision rule: Select $\begin{cases}\text{hamsi} & \text{if } P(C_1|x) > P(C_2|x) \\ \text{barbun} & \text{otherwise}\end{cases}$

Select $\begin{cases}\text{hamsi} & \text{if } P(x|C_1) \cdot P(C_1) > P(x|C_2) \cdot P(C_2) \\ \text{barbun} & \text{otherwise}\end{cases}$

- Evidence is unimportant since it is the same for all states of nature
- *Equal priors* → Observing each state of nature is equally likely
- *Equal likelihoods* → Measurement x gives no information

14

Bayesian decision theory

- Now let's generalize the decision problem

States of nature $\{C_1, C_2, \cdots, C_N\}$
Possible actions $\{\alpha_1, \alpha_2, \cdots, \alpha_A\}$
Loss function $\lambda(\alpha_i|C_k)$

Let $x \in R^d$ be a feature vector in a d-dimensional space.
How would you take an action for x?
For this x, we would take the action α_i that minimized the loss $\lambda(\alpha_i|C_k)$ if we knew C_k is its true state of nature.

However, we do not know the true state of nature
Thus, we will take the action based on expectation

15

Bayesian decision theory

- The expected loss associated with taking action α_i

$\underbrace{R(\alpha_i|x)}_{\text{Conditional risk}} = \sum_{k=1}^{N} P(C_k|x) \cdot \lambda(\alpha_i|C_k)$

$P(C_k|x) = \frac{P(x|C_k) \cdot P(C_k)}{P(x)}$

- We take the action that minimizes the conditional risk

$\underbrace{\alpha^*}_{\text{Optimal action}} = \underset{i}{\operatorname{argmin}}\, R(\alpha_i|x)$

The resulting minimum risk R* is called *Bayes risk*

16

Minimum error-rate classification

- In multi-class classification
  - Each state of nature is usually associated with a class
  - Each action is usually interpreted as deciding on a class (sometimes other actions — e.g., reject action, are defined)
  - Zero-one loss function is commonly used

$$\lambda(\alpha_i|C_k) = \begin{cases} 0 & \text{if } i = k \quad \text{(correct classification)} \\ 1 & \text{if } i \neq k \quad \text{(all incorrect classifications)} \end{cases}$$

The optimal action is

$$\alpha^* = \underset{k}{\operatorname{argmax}}\, P(C_k|x)$$

When zero-one loss function is used, selecting the action that minimizes the conditional risk is equivalent to selecting the action that maximizes the posterior probability

17

Classifiers and discriminant functions

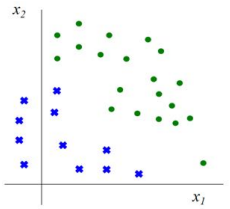
- A classifier is represented with a set of discriminant functions $g_k(x)$ for $k = 1, 2, \cdots, N$
- A given instance x is then classified with the class C_k for which the discriminant function $g_k(x)$ is the maximum

1. **Likelihood-based approaches**
2. **Discriminant-based approaches**

18

Likelihood-based approaches

- They estimate class probabilities on training samples and then use them to define the discriminant functions
- **Bayes classifier**
  - Defines a discriminant function using the conditional risk



$$g_k(x) = -R(\alpha_k|x)$$

$$g_k(x) = \underbrace{P(C_k|x)}_{\text{when 0-1 loss function is used}}$$

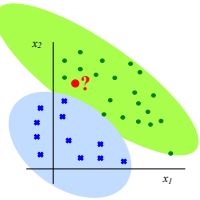
$$g_k(x) = \hat{P}(C_k|x) = \frac{\hat{P}(x|C_k) \cdot \hat{P}(C_k)}{\hat{P}(x)} \equiv \underbrace{\hat{P}(x|C_k) \cdot \hat{P}(C_k)}$$

For each class, estimate the prior and the likelihood on training samples that belong to this class

19

Likelihood-based approaches

- **Parametric approach**
  - Assumes a parametric form on the probability distributions and estimate their parameters on training samples
  - For a given instance x, it estimates its class probabilities using these distributions
  - Maximum likelihood estimation and Bayesian estimation

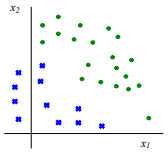


- **Nonparametric approach**
  - Does not have such assumption
  - It estimates the class probabilities of the instance x using the nearby points of this instance
  - Parzen windows, k-nearest neighbors

20

Discriminant-based approaches

- They learn discriminant functions directly on training samples
- They make an assumption on the form of discriminant functions and learn their parameters on training samples without estimating class probabilities
- **Linear discriminants assume that each discriminant function is a linear combination of the input features**



Linear discriminants

Linear discriminants

- They define $g_k(x)$ as a linear combination of the input features

$$g_k(x|W_k) = \sum_{i=1}^{d} W_{ki}\ x_i + W_{k0}$$

let's define $x_0 = 1$

$$g_k(x|W_k) = \sum_{i=0}^{d} W_{ki}\ x_i$$

d : number of input dimensions
W_k : weight vector for the k-th class

- Learning involves learning parameters (weights) W_k for each class C_k from training samples
- For that, we will define a criterion function and learn the weights that minimize/maximize this function

Linear discriminants

- They yield hyperplane decision boundaries

Consider two-class classification

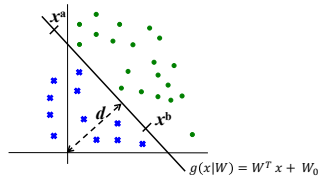
$$g_1(x|W_1) = \sum_{i=1}^{d} W_{1i}\ x_i + W_{10}$$

$$g_1(x|W_1) = W_1^T x + W_{10}$$
$$g_2(x|W_2) = W_2^T x + W_{20}$$

$$g(x|W_1, W_2) = g_1(x) - g_2(x) \quad \text{Choose} \begin{cases} C_1 & \text{if } g(x) \geq 0 \\ C_2 & \text{otherwise} \end{cases}$$

$$g(x|W_1, W_2) = (W_1 - W_2)^T x + (W_{10} - W_{20})$$
$$g(x|W) = W^T x + W_0$$

This is another linear function



Let's take two points on the decision boundary

$g(x^a|W) = g(x^b|W)$

$W^T x^a + W_0 = W^T x^b + W_0$

$W^T (x^a - x^b) = 0$

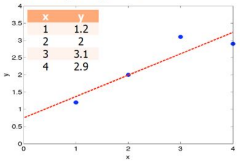
W determines the hyperplane's orientation (W is normal to any vector on the hyperplane)

W0 determines the hyperplane's location with respect to the origin

How to learn?

Although we will use linear discriminants for classification, let's first consider a linear regression problem

Construct a linear model on the following data points



x	y
1	1.2
2	2
3	3.1
4	2.9

construct a linear model

$f(x) = W\,x + W_0$

define a criterion function

$$loss(W, W_0) = \frac{1}{2}\sum_t (f(x^t) - y^t)^2$$

Sum of squared errors

Select W and W_0 that minimize this error on the training samples $\{x^t, y^t\}_{t=1}^M$

$$\frac{\partial loss}{\partial W} = 0 \qquad \frac{\partial loss}{\partial W_0} = 0$$

Analytical solution

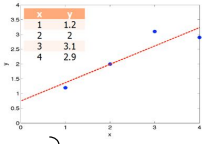
$loss(W, W_0) = \frac{1}{2}\sum_t (f(x^t) - y^t)^2 \qquad f(x) = W\,x + W_0$

$$\frac{\partial loss}{\partial W} = \frac{1}{2}\,2\sum_t (W x^t + W_0 - y^t)x^t = W\sum_t (x^t)^2 + W_0 \sum_t x^t - \sum_t x^t\,y^t = 0$$

$$\frac{\partial loss}{\partial W_0} = \frac{1}{2}\,2\sum_t (W x^t + W_0 - y^t) = W\sum_t x^t + W_0 \sum_t 1 - \sum_t y^t = 0$$

In our example

$\sum_t x^t = 10$
$\sum_t y^t = 9.2$
$\sum_t (x^t)^2 = 30$
$\sum_t x^t\,y^t = 26.1$
$M = 4$



x	y
1	1.2
2	2
3	3.1
4	2.9

$30\,W + 10\,W_0 - 26.1 = 0$
$10\,W + 4\,W_0 - 9.2 = 0$

2 equations
2 unknowns

$\Rightarrow \quad W = 0.62 \qquad W_0 = 0.75$

$f(x) = 0.62\,x + 0.75$

It might be very difficult to analytically solve this, also depending on the number of the weights

OR

There may be no analytical solution at all (if the linear system has a singular matrix, no solution or multiple solutions exist)

ITERATIVE OPTIMIZATION METHODS

Gradient descent algorithm

- One commonly used iterative optimization method
- Goal is to find the parameters that minimize the loss
  - Starting with random parameters, it iteratively updates them in the direction of the steepest descent (in the opposite direction of the gradient) until the gradient is zero (or small enough)

start with random weights W_i
do
  for all i
    $\Delta W_i = -\alpha \frac{\partial loss}{\partial W_i}$
  for all i
    $W_i = W_i + \Delta W_i$
until convergence

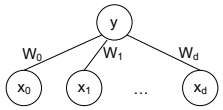
α is the learning rate, which determines how much to move in the direction of the steepest descent

→ if it is too small, convergence is slow
→ if it is too large, we may overshoot the minimum (divergence might occur)

This method finds the nearest minimum, which could be local. It does not guarantee to find the global minimum

Regression

Derive update rules for regression



$loss(W) = \sum_t loss^t(W)$

$loss^t(W) = \frac{1}{2}(f(x^t) - y^t)^2$

$f(x^t) = net^t$

$net^t = \sum_i x_i^t\,W_i$

$$\Delta W_i = -\alpha \frac{\partial loss(W)}{\partial W_i} = -\alpha \sum_t \frac{\partial loss^t(W)}{\partial W_i}$$

$$\frac{\partial loss^t(W)}{\partial W_i} = \frac{\partial loss^t(W)}{\partial net^t} \cdot \frac{\partial net^t}{\partial W_i}$$

$$\frac{\partial loss^t(W)}{\partial W_i} = \delta^t \cdot x_i^t$$

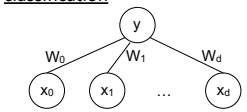
$$\Delta W_i = -\alpha \sum_t \delta^t \cdot x_i^t$$

$\delta^t = (net^t - y^t)$

$\boldsymbol{\Delta W_i = \alpha \sum_t (y^t - net^t) \cdot x_i^t}$

Classification (logistic regression)

Derive update rules for 2-class classification

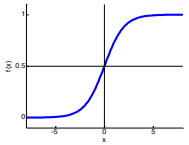


$$y^t = \begin{cases} 1 & \text{if } x^t \in C_1 \\ 0 & \text{if } x^t \in C_2 \end{cases}$$

$$f(x^t) = \sigma(net^t)$$

$$net^t = \sum_i x_i^t W_i$$

Logarithmic sigmoid function



$$\sigma(net^t) = \frac{1}{1+\exp(-net^t)}$$

$$\sigma'(net^t) = \sigma(net^t)(1-\sigma(net^t))$$

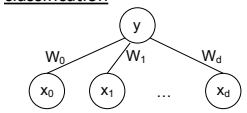
$$loss^t(W) = \underbrace{\frac{1}{2}(f(x^t)-y^t)^2}_{\text{Squared error}}$$

$$loss^t(W) = \underbrace{-y^t \cdot \log(f(x^t)) - (1-y^t)\cdot\log(1-f(x^t))}_{\text{Binary cross entropy}}$$

29

Classification (logistic regression)

Derive update rules for 2-class classification



$$loss(W) = \sum_t loss^t(W)$$

$$loss^t(W) = \frac{1}{2}(f(x^t)-y^t)^2$$

$$f(x^t) = \sigma(net^t)$$

$$net^t = \sum_i x_i^t W_i$$

$$\Delta W_i = -\alpha \frac{\partial loss(W)}{\partial W_i} = -\alpha \sum_t \frac{\partial loss^t(W)}{\partial W_i}$$

$$\frac{\partial loss^t(W)}{\partial W_i} = \frac{\partial loss^t(W)}{\partial net^t} \cdot \frac{\partial net^t}{\partial W_i}$$

$$\frac{\partial loss^t(W)}{\partial W_i} = \delta^t \cdot x_i^t$$

$$\Delta W_i = -\alpha \sum_t \delta^t \cdot x_i^t$$

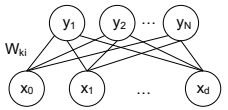
$\delta^t = (\sigma(net^t) - y^t)\cdot\sigma'(net^t)$ *When squared error is used as the loss function*

$$\boldsymbol{\Delta W_i = \alpha \sum_t (y^t - \sigma(net^t))\cdot\sigma(net^t)\cdot(1-\sigma(net^t))\cdot x_i^t}$$

30

Classification

Derive update rules for multi-class classification



Define output as a N-dimensional vector

$$y_k^t = \begin{cases} 1 & \text{if } x^t \in C_j \\ 0 & \text{if } x^t \notin C_j \end{cases}$$

$$f_k(x^t) = softmax(net_k^t)$$

$$net_k^t = \sum_i x_i^t W_{ki}$$

$$softmax(net_k^t) = \frac{\exp(net_k^t)}{\sum_{n=1}^N \exp(net_n^t)}$$

$$\frac{\partial softmax(net_n^t)}{\partial net_k^t} = softmax(net_k^t)(\delta_{kn} - softmax(net_n^t))$$

$$\delta_{kn} = \begin{cases} 1 & \text{if } k = n \\ 0 & \text{if } k \neq n \end{cases}$$ *Kronecker delta*

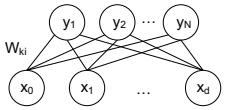
$$loss^t(W) = \underbrace{\frac{1}{2}\sum_{n=1}^N (f_n(x^t) - y_n^t)^2}_{\text{Squared error}}$$

$$loss^t(W) = \underbrace{-\sum_{n=1}^N y_n^t \cdot \log(f_n(x^t))}_{\text{Categorical cross entropy}}$$

31

Classification

Derive update rules for multi-class classification



$$loss(W) = \sum_t loss^t(W)$$

$$loss^t(W) = \frac{1}{2}\sum_{n=1}^N (f_n(x^t) - y_n^t)^2$$

$$f_k(x^t) = softmax(net_k^t)$$

$$net_k^t = \sum_i x_i^t W_{ki}$$

$$\Delta W_{ki} = -\alpha \frac{\partial loss(W)}{\partial W_{ki}} = -\alpha \sum_t \frac{\partial loss^t(W)}{\partial W_{ki}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ki}} = \frac{\partial loss^t(W)}{\partial net_k^t} \cdot \frac{\partial net_k^t}{\partial W_{ki}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ki}} = \delta_k^t \cdot x_i^t$$

$$\Delta W_{ki} = -\alpha \sum_t \delta_k^t \cdot x_i^t$$

$\delta_k^t = \sum_{n=1}^N (softmax(net_n^t) - y_n^t)\cdot\frac{\partial softmax(net_n^t)}{\partial net_k^t}$ *When squared error is used as the loss function*

$$\boldsymbol{\Delta W_{ki} = \alpha \sum_t \sum_{n=1}^N (y_n^t - softmax(net_n^t))\, softmax(net_k^t)(\delta_{kn} - softmax(net_n^t))x_i^t}$$

32

Batch learning algorithm

start with random weights W_{ki}
do
  EPOCH:
    for all t and k
      $f_k(x^t) = softmax(\sum_i x_i^t W_{ki})$
    for all k and i
      $\Delta W_{ki} = -\alpha \sum_t \delta_k^t \cdot x_i^t$
    for all k and i
      $W_{ki} = W_{ki} + \Delta W_{ki}$
until convergence

Stochastic learning algorithm

start with random weights W_{ki}
do
  EPOCH:
    for all (x^t, y^t) in random order
      for all k
        $f_k(x^t) = softmax(\sum_i x_i^t W_{ki})$
      for all k and i
        $\Delta W_{ki} = -\alpha \cdot \delta_k^t \cdot x_i^t$
      for all k and i
        $W_{ki} = W_{ki} + \Delta W_{ki}$
until convergence

Mini-batch learning algorithm is a good tradeoff

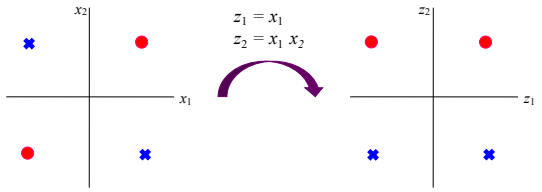
33

Adding nonlinearity

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

34

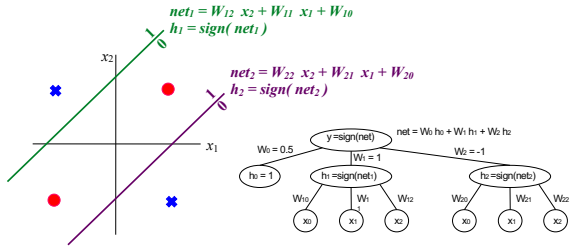
XOR problem



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

35

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)

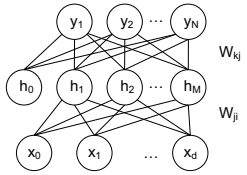
36

Neural networks

37

Multilayer perceptrons

- Also contain hidden layers in addition to input and output layers



Hidden units h_j's can be viewed as new "features" obtained by combining x_i's

A deeper architecture with nonlinear activations is more expressive than a shallow one

In this network

1. Each hidden unit computes its net activation
$$net_j^t = \sum_i x_i^t W_{ji}$$
2. Each hidden unit emits an output that is a nonlinear function (e.g., sigmoid, ReLU) of its activation
$$h_j^t = non-linear-function(net_j^t)$$
3. Each output unit computes its net activation
$$net_k^t = \sum_j h_j^t W_{kj}$$
4. Each output units emits an output (using a linear, a sigmoid or a softmax function)
$$y_k^t = output-function(net_k^t)$$

38

How to learn?

- In linear discriminants, we select the weights to minimize a loss function defined on the difference between the actual and computed outputs
- In multilayer structures, we can also select the hidden-to-output-layer weights to minimize a loss function defined on the actual and computed outputs
- However, we cannot select the input-to-hidden-layer weights in a similar way since we do not know the actual values of the hidden units
- Thus, to learn the input-to-hidden-layer weights, we propagate the loss function (defined on the outputs) from the output layer to the corresponding hidden layer → **BACKPROPAGATION ALGORITHM**

39

Backpropagation algorithm

Derive update rules for multi-class classification

$net_j^t = \sum_i x_i^t W_{ji}$

$h_j^t = \sigma(net_j^t)$ *Uses sigmoid as the non-linear function*

$net_k^t = \sum_j h_j^t W_{kj}$

$f_k(x^t) = softmax(net_k^t)$

$loss(W) = \sum_t loss^t(W)$ *Uses sum-of-squared errors*

$loss^t(W) = \frac{1}{2}\sum_{n=1}^{N} (f_n(x^t) - y_n^t)^2$

$\Delta W_{kj} = -\alpha \frac{\partial loss(W)}{\partial W_{kj}} = -\alpha \sum_t \frac{\partial loss^t(W)}{\partial W_{kj}}$

Hidden-to-output-layer weights

$$\frac{\partial loss^t(W)}{\partial W_{kj}} = \frac{\partial loss^t(W)}{\partial net_k^t} \cdot \frac{\partial net_k^t}{\partial W_{kj}}$$

$$\frac{\partial loss^t(W)}{\partial W_{kj}} = \delta_k^t \cdot h_j^t$$

$$\delta_k^t = \sum_{n=1}^{N} (softmax(net_n^t) - y_n^t) \cdot \frac{\partial softmax(net_n^t)}{\partial net_k^t}$$

40

Backpropagation algorithm

Derive update rules for multi-class classification

$net_j^t = \sum_i x_i^t W_{ji}$

$h_j^t = \sigma(net_j^t)$ *Uses sigmoid as the non-linear function*

$net_k^t = \sum_j h_j^t W_{kj}$

$f_k(x^t) = softmax(net_k^t)$

$loss(W) = \sum_t loss^t(W)$ *Uses sum-of-squared errors*

$loss^t(W) = \frac{1}{2}\sum_{k=1}^{N} (f_k(x^t) - y_k^t)^2$

$\Delta W_{ji} = -\alpha \frac{\partial loss(W)}{\partial W_{ji}} = -\alpha \sum_t \frac{\partial loss^t(W)}{\partial W_{ji}}$

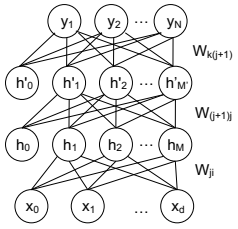
Input-to-hidden-layer weights

$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \frac{\partial loss^t(W)}{\partial net_j^t} \cdot \frac{\partial net_j^t}{\partial W_{ji}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \delta_j^t \cdot x_i^t$$

$$\delta_j^t = \sum_{k=1}^{N} \frac{\partial loss^t(W)}{\partial net_k^t} \cdot \frac{\partial net_k^t}{\partial net_j^t} = \left[\sum_{k=1}^{N} \delta_k^t \cdot W_{kj}\right] \cdot \sigma'(net_j)$$

More hidden layers



$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \frac{\partial loss^t(W)}{\partial net_j^t} \cdot \frac{\partial net_j^t}{\partial W_{ji}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \delta_j^t \cdot x_i^t$$

$$\delta_j^t = \sum_{(j+1)} \frac{\partial loss^t(W)}{\partial net_{(j+1)}^t} \cdot \frac{\partial net_{(j+1)}^t}{\partial net_j^t}$$

$$\delta_j^t = \left[\sum_{(j+1)} \delta_{(j+1)}^t \cdot W_{(j+1)j}\right] \cdot \sigma'(net_j)$$

δ_j may vanish after repeated multiplication. This makes deep architectures hard to train (when initial weights are not "good" enough)

Approaches for alleviating underfitting and overfitting problems

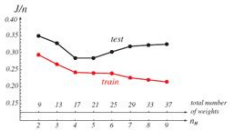
- Better network designs: Sparse connections, weight sharing, convolutional nets, long/short skip connections, activation functions, …
- Better network training: Regularization, loss function definitions, larger datasets, data augmentation, …
- Previously, layerwise pretraining (restricted Boltzmann machines, autoencoders)

Network topology

The number of hidden units and hidden layers

- It controls the expressive power of the network
- Thus, the complexity of the decision boundary
- No foolproof method to set them before training
  - Few hidden units/layers will be enough if samples are well-separated
  - More will be necessary if samples have complicated densities

We will talk about different deep network architectures later



Hidden units more than necessary

- Network is tuned to the particular training set (overfitting)
- Training error can become small, but test error is unacceptably high

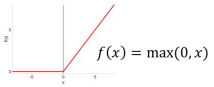
Too few hidden units

- Network does not have enough free parameters to fit the training set well
- Training and test errors are high

Some practical issues

- Commonly used activation functions

Rectified linear unit (ReLU)

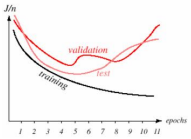


$f(x) = \max(0, x)$

Logarithmic sigmoid function

$f(x) = \frac{1}{1 + \exp(-x)}$

- When to stop?



Validation data can be used:

- Training error ultimately reaches an asymptotic value
- Error on an independent test set is expected to be higher
  - Although it usually decreases, it can also increase or oscillate
- Error on a validation set is typically used to decide when to stop

Some practical issues

1. Unbalanced class distributions
   - Classifiers typically favor the majority class(es)
   - A common practice to deal with this is to rebalance the classes or to define/use a **weighted loss**
     - ***Oversampling:*** replicate training samples from the minority class(es)
     - ***Undersampling:*** ignore some training samples from the majority class(es)

2. Small training sets
   - Data augmentation is typically useful
     - Disadvantage: memory requirements become larger, overall training becomes slower

Many of them are indeed issues not only for neural networks but also for many other classifiers

45

Some practical issues

3. Features with different orders of magnitudes
   - A neural network adjusts weights in favor of features with higher magnitudes
   - Normalization/scaling is typically useful
     - Normalize training samples, considering each feature separately
     - Use the same normalization (mean and standard deviation) for test samples

$$new(x_i^t) = \frac{x_i^t - \mu_i}{\sigma_i}$$

4. High feature values
   - May cause the exploding gradient problem
   - Normalization/scaling is typically useful

Many of them are indeed issues not only for neural networks but also for many other classifiers

46

Some practical issues

Regularization reduces sensitivity to training samples and decreases the risk of overfitting

$$loss(W) = \frac{1}{T}\sum_t loss^t(W) + \|W\|$$

$$loss(W) = \underbrace{\frac{1}{2T}\sum_t \sum_{k=1}^{N} (f_k(x^t) - y_k^t)^2}_{\text{Mean squared error}} + \underbrace{\frac{\lambda}{2T}\|W\|_2^2}_{\text{L2-regularization term}} \quad \text{where } \|W\|_2^2 = \sum_{i,j} W_{ij}^2$$

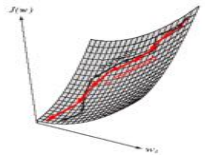
Dropout regularization

- During training, in each iteration, randomly drop out units (also their incoming and outgoing connections) with probability p to sample a "thinned" network and train it
- Training can be seen as training a collection of different thinned networks with extensive weight sharing
- In testing, consider the entire network where the weights are scaled down by multiplying them a factor of 1 – p

47

Some practical issues

- Momentum helps speed up learning especially when when there are plateaus in error surfaces



Some fraction of the previous weight updates is included into the current update rule

$$W^{(T+1)} = W^{(T)} + (1-\beta)\,\Delta W^{(T)} + \beta\,\Delta W^{(T-1)}$$

Selection of initial weights as well as selection/update of learning rate, momentum constant, dropout factor, etc. may greatly affect learning

For some, optimization methods (e.g., AdaDelta, Adam) are available

48

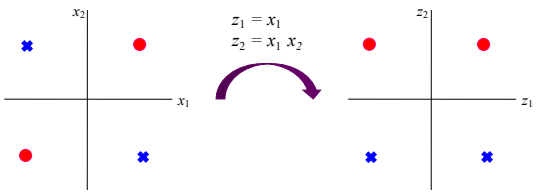
Adding nonlinearity to linear discriminants

(revisited)

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

49

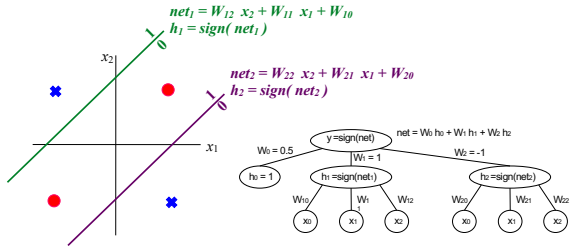
XOR problem



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

50

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)

51

Support vector machines

52

Generalized linear discriminant functions

- For two-class classification, a linear discriminant function $g(x)$ defines a linear decision boundary (*hyperplane*)

$$g(x|W) = \sum_{i=0}^{d} W_i \, x_i$$

- We can add higher-order terms to this function in order to define more complicated decision boundaries

$$g(x|W) = \sum_{i=0}^{d} W_i \, x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} W_{ij} \, x_i \, x_j$$

Quadratic discriminant function, which defines a hyperquadric decision boundary

$$g(x|V) = \sum_{m=1}^{d'} V_m \, \phi_m(x)$$

Each $\phi_m(x)$ is an arbitrary function of x

Generalized linear discriminant function

ϕ is a feature vector in d'-dimensional space and corresponds to x, for which the features are originally defined in d-dimensional space

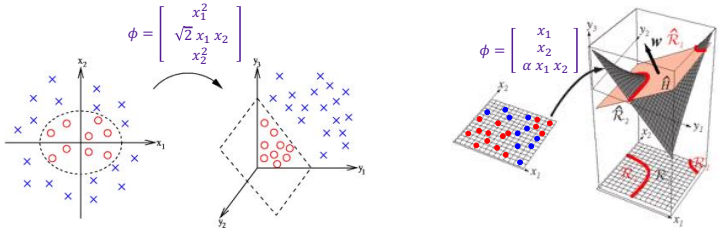
Slide credit: S. Aksoy

53

Generalized linear discriminant functions

$$\phi = \begin{bmatrix} x_1^2 \\ \sqrt{2}\, x_1 \, x_2 \\ x_2^2 \end{bmatrix}$$

$$\phi = \begin{bmatrix} x_1 \\ x_2 \\ \alpha \, x_1 \, x_2 \end{bmatrix}$$



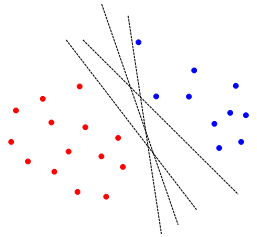
Samples of different classes are not linearly separable in the original input space. After applying the mapping ϕ, the samples become linearly separable in the new input space. Now, we can use a hyperplane for the separation.

Slide credit: S. Aksoy

54

Support vector machines

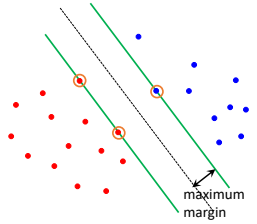
- For a linearly separable dataset, there exist an infinite number of possible linear decision boundaries (hyperplanes) that yield perfect classification



55

Support vector machines

- Among all these possibilities, there exists a unique hyperplane that maximizes the margin between the samples of different classes



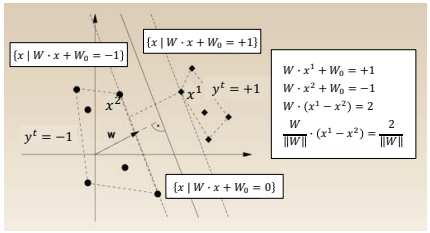
Support vector machines find this optimal hyperplane, optimizing a criterion function
(solving an optimization problem)

Margin is the perpendicular distance between the decision boundary and the closest samples

The closest samples for the maximum margin are called support vectors

56

Support vector machines



Formulate the problem as

$$y^t = \begin{cases} 1 & \text{if } x^t \in C_1 \\ -1 & \text{if } x^t \in C_2 \end{cases}$$

$W \cdot x^t + W_0 = 0$ (hyperplane)

$f(x^t) = sign(W \cdot x^t + W_0)$

For a separable dataset, there exist a weight vector W and a threshold W_0 such that

$$y^t \cdot (W \cdot x^t + W_0) > 0$$

Rescaling W and W_0, the support vectors (points closest to the hyperplane) satisfy

$$|W \cdot x^t + W_0| = 1$$

This yields the hyperplane with

$$y^t \cdot (W \cdot x^t + W_0) \geq 1$$

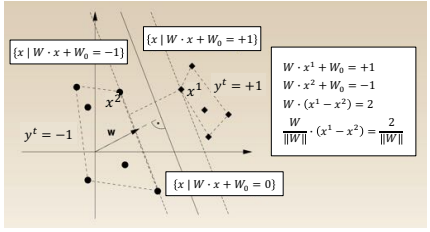
The margin, measured perpendicularly to the hyperplane, equals $2/\|W\|$

To maximize the margin,
minimize $\|W\|$
subject to $y^t \cdot (W \cdot x^t + W_0) \geq 1$

M.A. Hearst et al., "Support vector machines," IEEE Intelligent Systems, 1998. http://web.cs.iastate.edu/~honavar/hearst-svm.pdf

57

Support vector machines



Formulate the problem as

$$y^t = \begin{cases} 1 & \text{if } x^t \in C_1 \\ -1 & \text{if } x^t \in C_2 \end{cases}$$

$W \cdot x^t + W_0 = 0$ (hyperplane)

$f(x^t) = sign(W \cdot x^t + W_0)$

To maximize the margin,
minimize $\|W\|$
subject to $y^t \cdot (W \cdot x^t + W_0) \geq 1$

This is an optimization problem with a solution of

$$W = \sum_{sv} V_{sv} \cdot x^{sv}$$

W has an expansion in terms of a subset of training samples (support vectors) that lie on the margin

The final decision function is

$$f(x^t) = sign\left(\sum_{sv} V_{sv} \cdot x^{sv} \cdot x^t + W_0\right)$$

M.A. Hearst et al., "Support vector machines," IEEE Intelligent Systems, 1998. http://web.cs.iastate.edu/~honavar/hearst-svm.pdf

58

Support vector machines

This final decision function is for the case where the dataset is linearly separable

$$f(x^t) = sign\left(\sum_{sv} V_{sv} \cdot x^{sv} \cdot x^t + W_0\right)$$

Now consider the case where the dataset is not linearly separable, and we need mappings (Slides 53 and 54)

$$f(x^t) = sign\left(\sum_{sv} V_{sv} \cdot \underbrace{\phi(x^{sv}) \cdot \phi(x^t)} + W_0\right)$$

KERNEL TRICK: You do not need to know the mapping function itself but the dot product in the transformed space

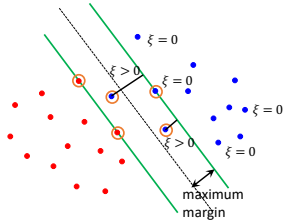
$$k(x^{sv}, x^t) = \phi(x^{sv}) \cdot \phi(x^t)$$

polynomial kernel	$k(x^{sv}, x^t) = (x^{sv} \cdot x^t)^p$
radial basis function kernel	$k(x^{sv}, x^t) = \exp(-\gamma \| x^{sv} - x^t \|^2)$
sigmoidal kernel	$k(x^{sv}, x^t) = \tanh(\kappa x^{sv} x^t + \vartheta)$

59

Support vector machines

- For non-linearly separable data, we relax the constraints



For linearly separable data

To maximize the margin,
minimize $\|W\|$
subject to $y^t \cdot (W \cdot x^t + W_0) \geq 1$

*For **non-linearly** separable data*

To maximize the margin,
minimize $\|W\| + C \sum_t \xi^t$
subject to $W \cdot x^t + W_0 \geq 1 - \xi^t$ for $y^t = +1$
$W \cdot x^t + W_0 \leq 1 + \xi^t$ for $y^t = -1$
$\xi^t \geq 0$

ξ^t is a slack variable and C is a regularization parameter
When C is small, there is a smaller penalty for misclassifications
When it is large, this penalty increases

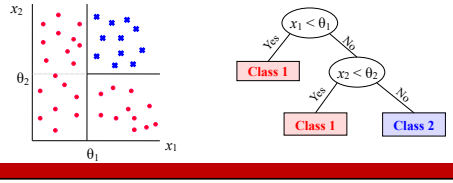
60

Decision trees

61

Decision trees

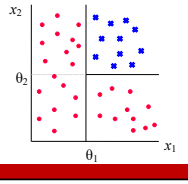
- A decision tree provides a classification or a regression model built in the form of a tree
- It is composed of internal decision nodes and leaves
  - An internal node corresponds to a test function whose discrete outcomes label the branches
  - A leaf defines a localized region (and a class for classification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

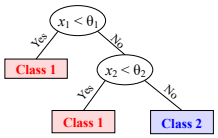


62

Decision trees

- 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



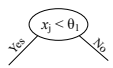


63

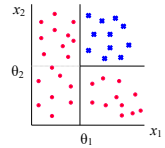
Decision trees

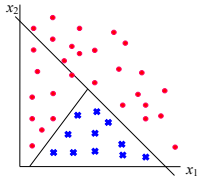
- Univariate trees
  - Test functions use one feature at a time
  - Define splits orthogonal to the coordinate axes
- Multivariate trees
  - Test functions use more than one feature at a time

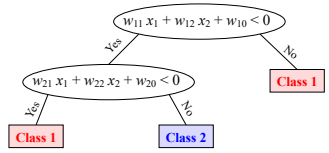
Discrete features



Continuous features







64

Classification trees

- 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

$$\underbrace{I(m)}_{\textit{Entropy at node m}} = -\sum_{k=1}^{N} P_m(C_k) \cdot \log(P_m(C_k))$$

N is the number of classes

$P_m(C_k)$ is the probability of having the k-th class at node m

$$\underbrace{I(S)}_{\textit{Entropy of binary split S}} = P_{left} \cdot I(left) + P_{right} \cdot I(right)$$

65

Classification trees

Toy example: Construct a tree for the training instances given 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

66

Classification trees

Toy example: Construct a tree for the training instances given 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:

x1 = red	x1 = green	x1 = blue
x2 ≤ 0.05	x2 ≤ 0.15	x2 ≤ 0.45

67

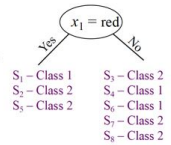
Classification trees

Toy example: Construct a tree for the training instances given below

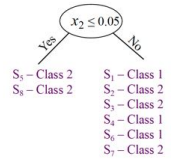
	x_1	x_2	class
S_1	red	0.5	1
S_2	red	0.2	2
S_3	green	0.5	2
S_4	blue	0.1	1
S_5	red	-0.5	2
S_6	green	0.1	1
S_7	green	0.4	2
S_8	blue	0.0	2

Calculate the entropy for all possible splits

$$I(x_1 = red) = P_{Yes}\, I(Yes) + P_{No}\, I(No)$$
$$= \frac{3}{8}\left(-\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}\right) + \frac{5}{8}\left(-\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}\right)$$
$$= 0.9512$$



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



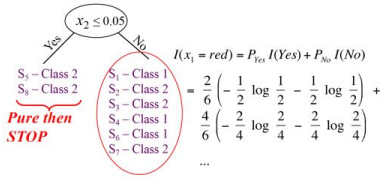
68

Classification trees

Toy example: Construct a tree for the training instances given 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



$$I(x_1 = red) = P_{Yes}\, I(Yes) + P_{No}\, I(No)$$

$$= \frac{2}{6}\left(-\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2}\right) + \frac{4}{6}\left(-\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}\right) + \ldots$$

Continue for this branch

1. *List all possible splits for this branch*
2. *Calculate the entropy for each split, considering only the training instances falling in this branch*
3. *Select the one with the minimum entropy*

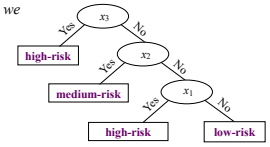
69

Rule extraction

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

Consider the following problem setting, in which we estimate the risk of being infected with covid-19

x_1: working in a hospital (Yes / No)
x_2: forget wearing mask indoors (Yes / No)
x_3: contact with a covid-19 infected patient (Yes / No)
x_4: eye color (brown / blue / green)



Rule 1: if (contact = yes) then high-risk
Rule 2: if (contact = no) and (forget mask = yes) then medium-risk
Rule 3: if (contact = no) and (forget mask = no) and (working in hospital = yes) then high-risk
Rule 4: if (contact = no) and (forget mask = no) and (working in hospital = no) then low-risk

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

70

Alternative splitting criteria

Entropy at node m

$$I(m) = -\sum_k P_m(C_k) \cdot \log(P_m(C_k))$$

$P_m(C_k)$ is the probability of having the k-th class at node m

Gini impurity at node m

$$I(m) = \sum_{k \neq j} P_m(C_k) \cdot P_m(C_j) = \frac{1}{2}\left[1 - \sum_k \left(P_m(C_k)\right)^2\right]$$

Misclassification impurity at node m

$$I(m) = 1 - \max_k P_m(C_k)$$

71

When to stop splitting

- Until all leaves are pure → **Overfitting**
- To prevent overfitting
  - Set a small threshold value in the reduction in impurity
  - Use validation (e.g., continue splitting if the validation error is decreasing)
  - Use an explicit measure of the complexity to encode the training instances 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 significantly differs from a random one)
- It is useful to keep the class frequencies/probabilities in each leaf (*classification confidence for a leaf*)

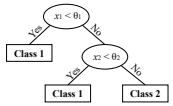
72

Pruning

- Prepruning: Stop growing the tree earlier before it overfits training samples
- Postpruning: Grow the tree until it overfits 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 validation samples
  - *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 $\underbrace{(x_1 \geq \theta_1)}_{A}$ and $\underbrace{(x_2 \geq \theta_2)}_{B}$ then Class 2

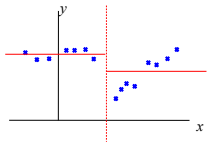
Try removing A or B and see what happens on validation samples



73

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



$$\underbrace{E(m)}_{\text{Mean squared error at node m}} = \frac{1}{|D_m|} \sum_{x^t \in D_m} \left(y^t - \hat{f}_m\right)^2$$

D_m is the set of training samples at node m

$|D_m|$ is the cardinality of this set

$\hat{f}_m$ is the estimated output at node m

$$\underbrace{E(S)}_{\text{Mean squared error of binary split S}} = P_{left} \cdot E(left) + P_{right} \cdot E(right)$$

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.)

74

Thank you!

Next time:

Basics of convolutional neural networks

75