# Introducción al aprendizaje automático

● ● ●

#3. Funciones de costo y optimización. Árboles de decisión

Regresión polinomial

- En verde se ilustra la función "verdadera" (inaccesible)

- Las muestras son uniformes en x y poseen ruido en y

- Utilizaremos una **<u>función de costo</u>** (error cuadrático) que mida el error en la predicción de y mediante $y(x, \mathrm{w})$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$
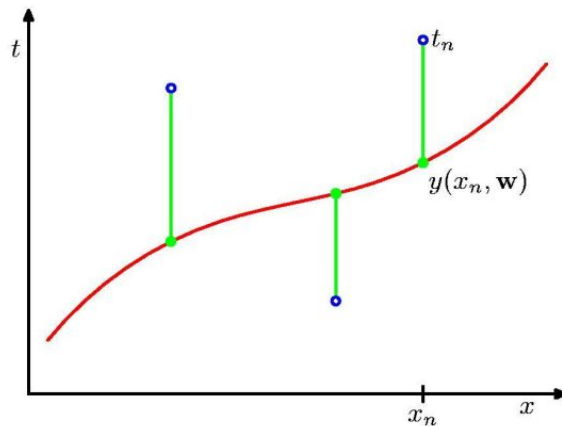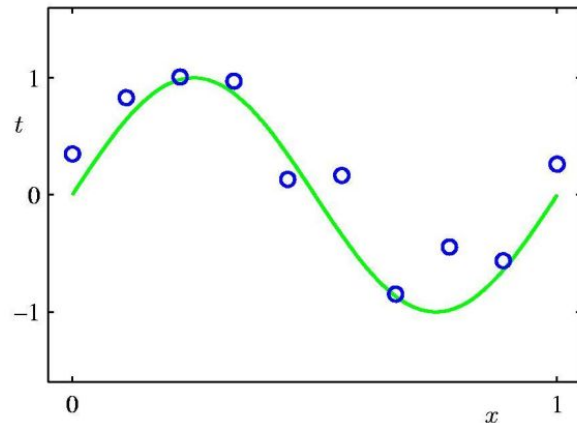
Regresión polinomial

- Función de predicción: $y(x; w) = \sum_{j=0}^{M} w_j x^j = w^T \tilde{x}, \quad \tilde{x} = (1, x, \ldots, x^M)^T$

- Función de costo: $L(w) = \dfrac{1}{2} \sum_{n=0}^{N} [y_n - y(x; w)]^2$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N, \quad \mathbf{X} = \begin{bmatrix} \tilde{x}_1^T \\ \vdots \\ \tilde{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times (M+1)}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_M \end{bmatrix} \in \mathbb{R}^{(M+1)}$$
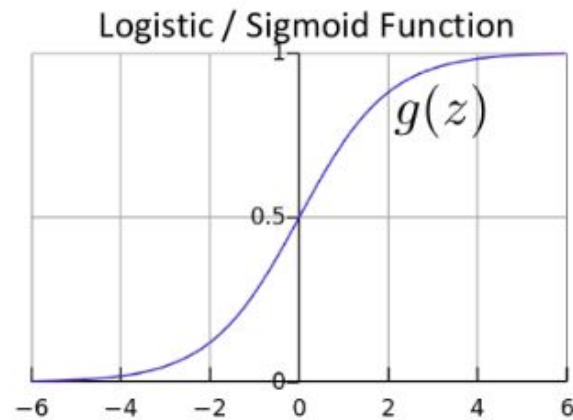
$$L(w) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \quad \Rightarrow \quad \mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$L(w) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w} \quad \Rightarrow \quad \mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Petersen & Pedersen, The matrix cookbook (2012)

Regresión logística

- Dados $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, con $x_i \in \mathbb{R}^n$, $y_i \in \{0, 1\}$

- Modelo: $p(y=1|x) = h_w(x)$

$$h_w(x) = \frac{1}{1 + exp(-w^T x))}$$

Logistic / Sigmoid Function

$g(z)$

- Función de costo

$$L(w) = -\sum_{i=1}^{N} y_i \log\left(h_w(x_i)\right) + (1 - y_i) \log\left(1 - h_w(x_i)\right)$$

- $h_w(x)$ no lineal → <u>no admite solución en forma cerrada</u>

Optimización y aprendizaje

- Un problema típico en ML se puede escribir como:

$$L(w) = \sum_{i=1}^{N} \ell(y_i, f_w(x_i)) + \lambda R(w)$$
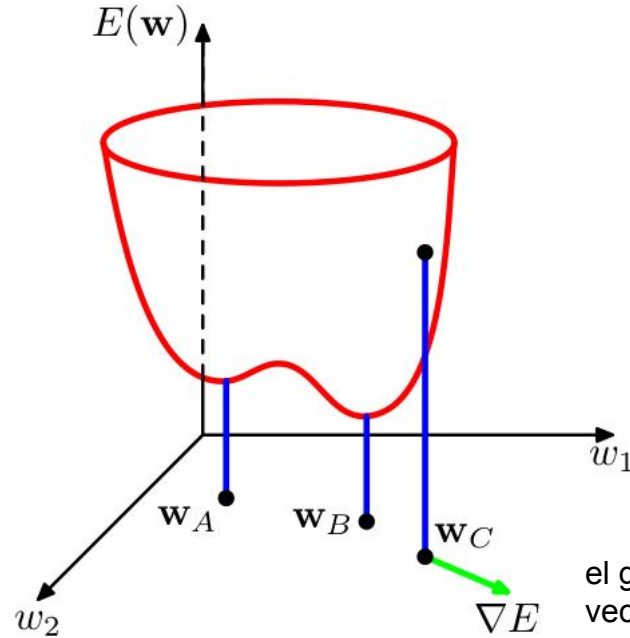
costo de predicción del par (x_i, y_i) regularización

- "Aprender" significa resolver:

$$w^* = \arg \max_{w} L(w)$$

.. aún cuando no existan soluciones en forma cerrada.

Optimización

el gradiente de $E(w)$ evaluado en w_C es un vector que apunta en la dirección de máximo crecimiento de la función si me paro en w_C.

- ¿La solución es única?

- Empleando algoritmos iterativos, ¿la solución depende del punto de inicio?
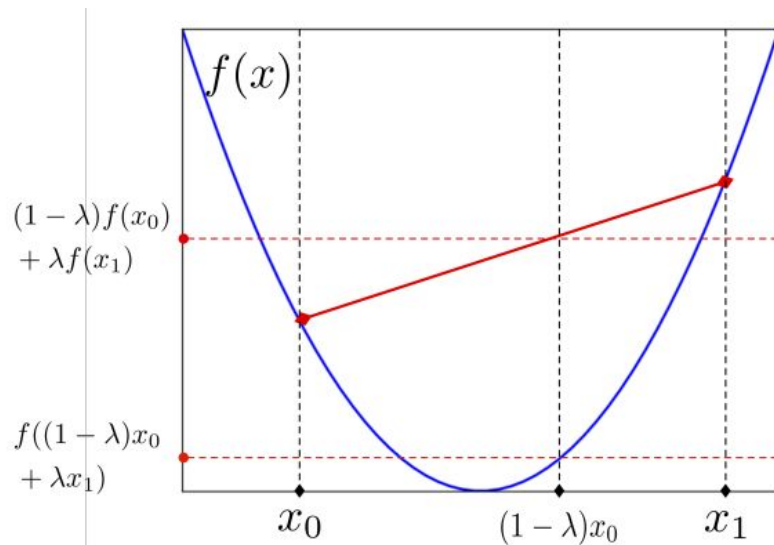
Funciones y conjuntos convexos

- Una **función** f es **convexa** si para cualquier x_0, x_1 en el dominio de f,

$$f((1 - \lambda)x_0 + \lambda x_1) \leq (1 - \lambda)f(x_0) + \lambda f(x_1), \quad 0 \leq \lambda \leq 1$$
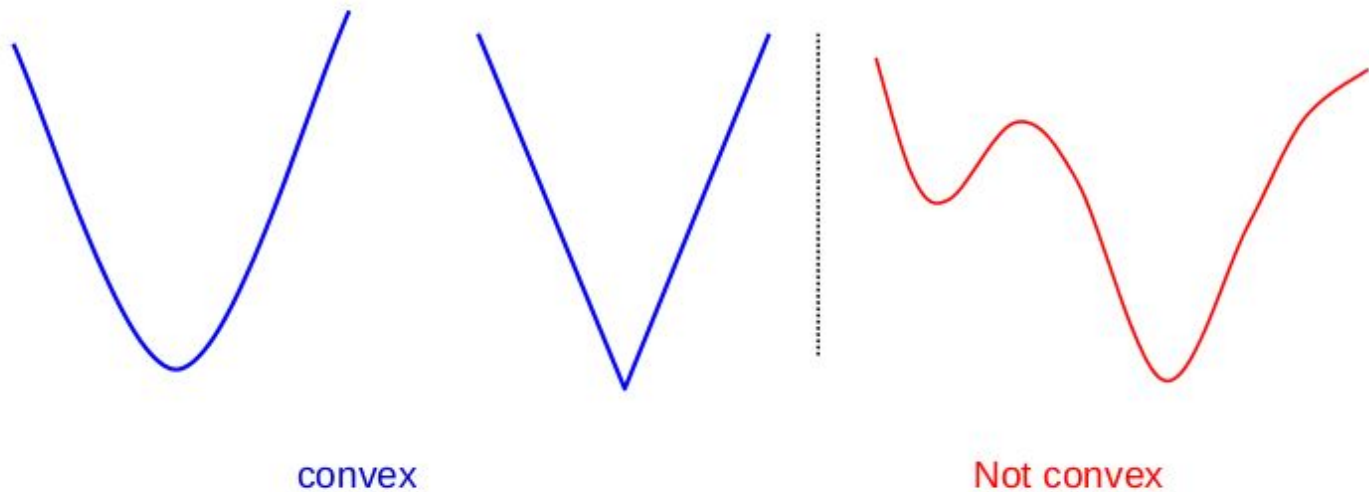
- Un **conjunto** S es **convexo** si para cualquier x_0, x_1 en S,

$$(1 - \lambda)x_0 + \lambda x_1 \in S$$

- Intuitivamente la función tiene forma de "cuenco"

Convex function examples

convex

Not convex

A non-negative sum of convex functions is convex

SVM

$$\min_{\mathbf{w} \in \mathbb{R}^d} C \sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$

convex

Why we care about convexity

- All critical points are minima
- Gradient descent finds the optimal solution (more on this in a later lecture)

Descent Methods

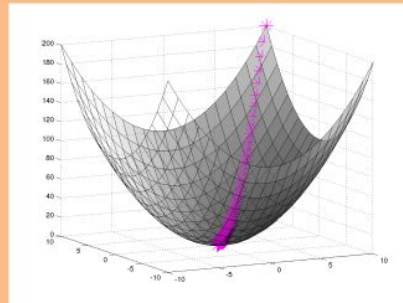- The typical strategy for optimization problems of this sort is a descent method:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \Delta\boldsymbol{w}^{(\tau)}$$

- These come in many flavours
  - Gradient descent $\nabla E(\boldsymbol{w}^{(\tau)})$
  - Stochastic gradient descent $\nabla E_n(\boldsymbol{w}^{(\tau)})$
  - Newton-Raphson (second order) ∇^2

Descenso de gradiente

Algorithm 1 Gradient Descent

1: **procedure** $\mathrm{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$
2: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$ **while** not converged **do**
4: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$ **return** $\boldsymbol{\theta}$

In order to apply GD to Logistic Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_N} J(\boldsymbol{\theta}) \end{bmatrix}$$

Gradiente en regresión logística

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

$$p \equiv \frac{1}{1 + e^{-\mathbf{x} \cdot \mathbf{w}}} = \frac{1}{1 + \exp(-\sum_j x^j w^j)}$$

We're going to dive into this thing here: d/dw(p)

$$(\log f)' = \frac{1}{f} f'$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p} \frac{\partial}{\partial w^j} p & \text{if } y = 1 \\ \frac{1}{1-p} (-\frac{\partial}{\partial w^j} p) & \text{if } y = 0 \end{cases}$$

Gradiente en regresión logística

$$\frac{\partial}{\partial w^j} p = \frac{\partial}{\partial w^j}(1 + \exp(-\sum_j x^j w^j))^{-1}$$

$$(f^n)' = nf^{n-1} \cdot f'$$
$$(e^f)' = e^f f'$$

$$= (-1)(1 + \exp(-\sum_j x^j w^j))^{-2} \frac{\partial}{\partial w^j} \exp(-\sum_j x^j w^j)$$

$$= (-1)(1 + \exp(-\sum_j x^j w^j))^{-2} \exp(-\sum_j x^j w^j)(-x^j)$$

$$= \underbrace{\frac{1}{1 + \exp(-\sum_j x^j w^j)}}_{p} \underbrace{\frac{\exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)}}_{1-p} x^j$$

$$\frac{\partial}{\partial w^j} p = p(1-p)x^j$$

Gradiente en regresión logística

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p} p(1-p)x^j = (1-p)x^j & \text{if } y = 1 \\ \frac{1}{1-p}(-1)p(1-p)x^j = -px^j & \text{if } y = 0 \end{cases}$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

- Regla de actualización en regresión logística:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y - p)\mathbf{x}$$

Details: Picking learning rate

- Use grid-search in log-space over small values on a tuning set:
  - e.g., 0.01, 0.001, …

- Sometimes, decrease after each pass:
  - e.g factor of $1/(1 + dt)$, t=epoch
  - sometimes $1/t^2$

- Fancier techniques I won't talk about:
 - Adaptive gradient: scale gradient differently for each dimension (Adagrad, ADAM, ….)

The Machine Learners Job

(1) Get the labeled data: $(x^1, y^1), \ldots, (x^n, y^n)$

(2) Choose a parametrization for hypothesis: $h_w(x)$

(3) Choose a loss function: $\ell(h_w(x), y) \geq 0$

(4) Solve the *training problem*:

$$\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell\left(h_w(x^i), y^i\right) + \lambda R(w)$$

(5) Test and cross-validate. If fail, go back a few steps
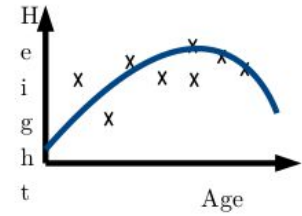
Parametrizing the Hypothesis

Linear:
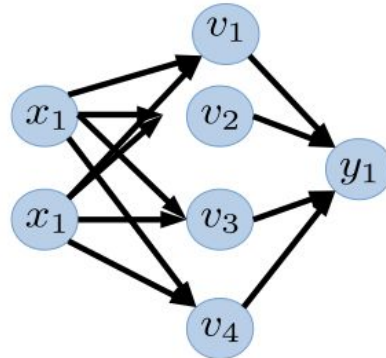$$h_w(x) = \sum_{i=0}^{d} w_i x_i$$
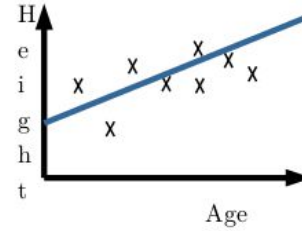
Polinomial:
$$h_w(x) = \sum_{i,j=0}^{d} w_{ij} x_i x_j$$
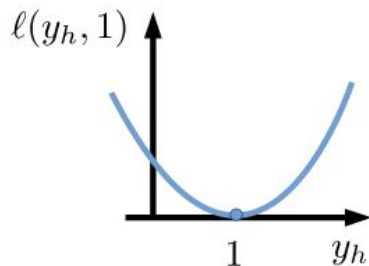

Neural Net:

$exe:$

$$v_1 = \text{sign}(w_{11} x_1 + w_{12} x_2)$$

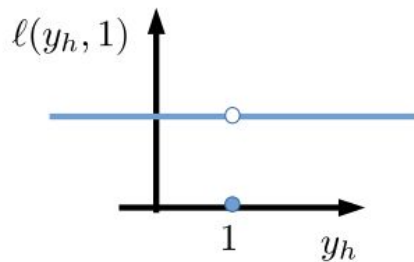$$v_4 = 1/\left(1 + \exp(w_{41} x_1 + w_{42} x_2)\right)$$

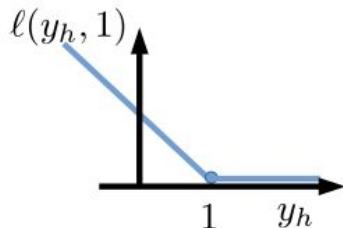# Choosing the Loss Function

Let $y_h := h_w(x)$

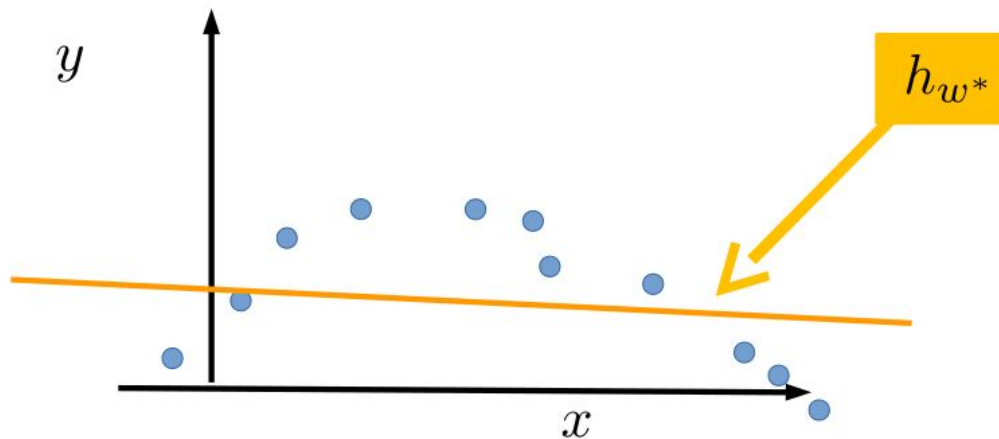Quadratic Loss $\quad \ell(y_h, y) = (y_h - y)^2$

Binary Loss $\quad \ell(y_h, y) = \begin{cases} 0 & \text{if } y_h = y \\ 1 & \text{if } y_h \neq y \end{cases}$

Hinge Loss $\quad \ell(y_h, y) = \max\{0, 1 - y_h y\}$
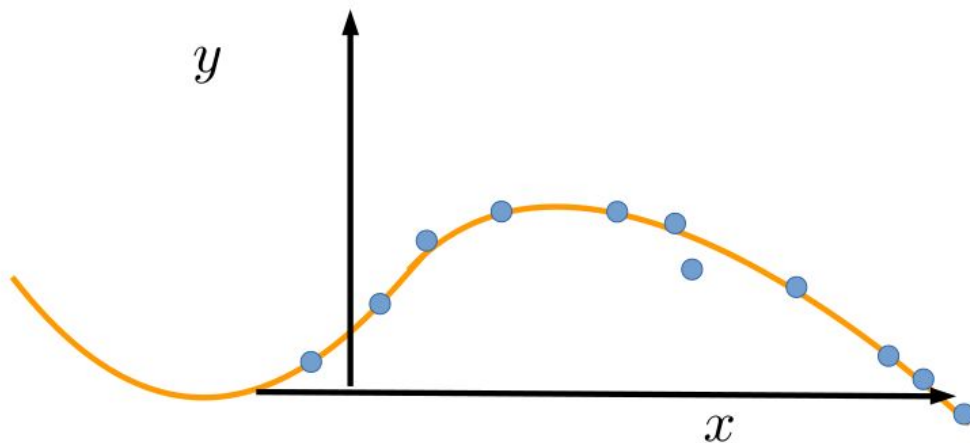
Overfitting and Model Complexity

Fitting 1ˢᵗ order polynomial

$$h_w = \langle w, x \rangle$$

$$w^* = \arg \min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(h_w(x^i) - y^i \right)^2$$

Overfitting and Model Complexity

Fitting 3ʳᵈ order polynomial

$$h_w = \sum_{i=0}^{3} w_i x^i$$

$$w^* = \arg \min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(h_w(x^i) - y^i \right)^2$$

Overfitting and Model Complexity
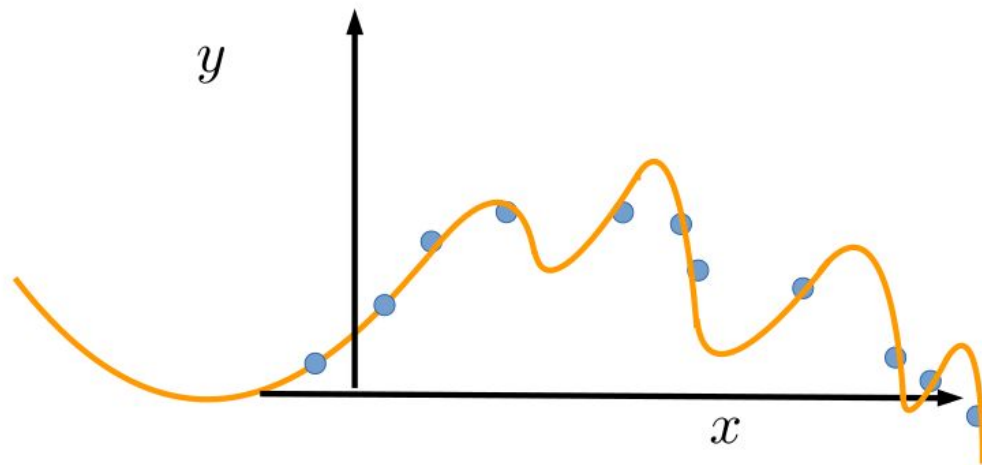

Fitting 9ᵗʰ order polynomial

$$h_w = \sum_{i=0}^{9} w_i x^i$$

$$w^* = \arg \min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(h_w(x^i) - y^i \right)^2$$

Regularization

Regularizor Functions

$$R: \quad \mathbf{R}^d \quad \rightarrow \quad \mathbf{R}_+$$
$$w \quad \rightarrow \quad R(w)$$

Controls tradeoff between fit and complexity

General Training Problem

$$\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell\left(h_w(x^i), y^i\right) + \lambda R(w)$$

Goodness of fit, fidelity term ...etc

Penlizes complexity

Exe: Ridge Regression

Linear hypothesis
$$h_w(x) = \langle w, x \rangle$$

+

L2 regularizor
$$R(w) = ||w||_2^2$$

L2 loss
$$\ell(y_h, y) = (y_h - y)^2$$

Ridge Regression
$$\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y^i - \langle w, x^i \rangle)^2 + \lambda ||w||_2^2$$

Exe: Logistic Regression

Linear hypothesis
$$h_w(x) = \langle w, x \rangle$$

+

L2 regularizor
$$R(w) = ||w||_2^2$$

Logistic loss
$$\ell(y_h, y) = \ln(1 + e^{-yy_h})$$

Logistic Regression
$$\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \langle w, x^i \rangle}) + \lambda ||w||_2^2$$

The Machine Learners Job

(1)   Get the labeled data: $(x^1, y^1), \ldots, (x^n, y^n)$

(2)   Choose a parametrization for hypothesis: $h_w(x)$

(3)   Choose a loss function: $\ell(h_w(x), y) \geq 0$

(4)   Solve the *training problem*:

$$\min_{w \in \mathbf{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell\left(h_w(x^i), y^i\right) + \lambda R(w)$$

(5)   Test and cross-validate. If fail, go back a few steps

Árboles de decisión

Hypotheses: decision trees $f : X \rightarrow Y$

- Each internal node tests an attribute x_i

- One branch for each possible attribute value $x_i = v$

- Each leaf assigns a class y

- To classify input x: traverse the tree from root to leaf, output the labeled y
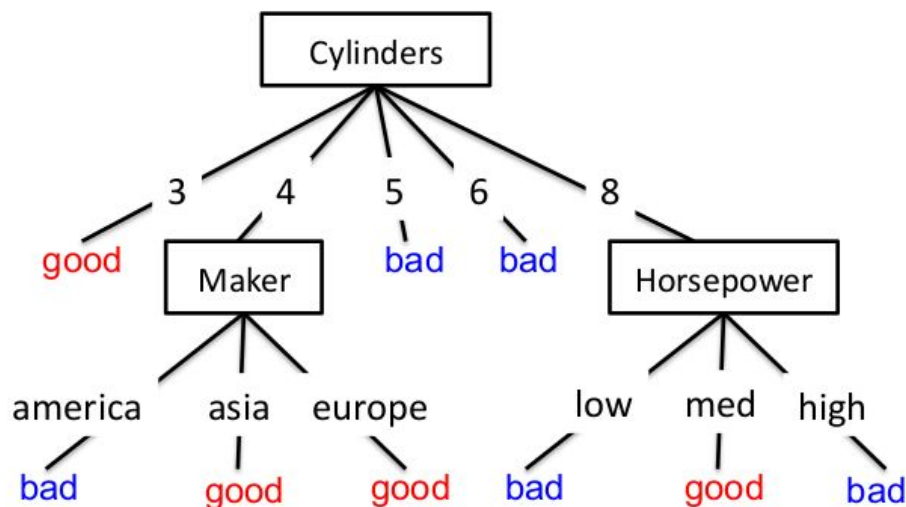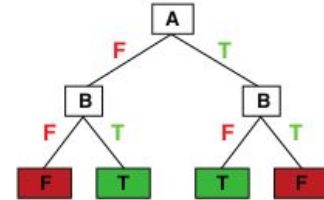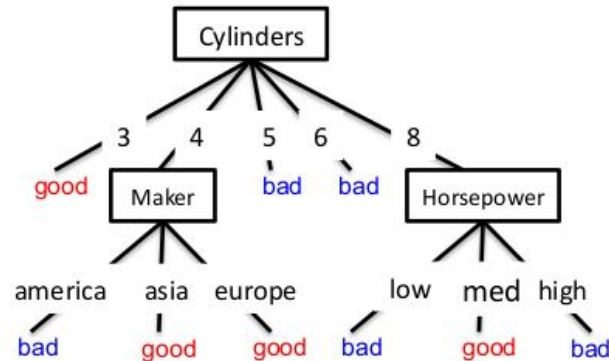

Human interpretable!

What functions can be represented?

- Decision trees can represent any function of the input attributes!

- For Boolean functions, path to leaf gives truth table row

- Could require exponentially many nodes

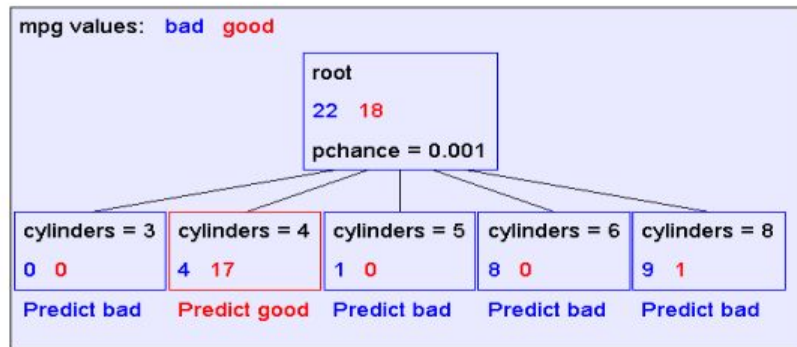| A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

(Figure from Stuart Russell)

Learning *simplest* decision tree is NP-hard

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest '76]

- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on **next best attribute (feature)**
  - Recurse

Key idea: Greedily learn trees using **recursion**

mpg values: bad   good

root
22   18
pchance = 0.001

cylinders = 3	cylinders = 4	cylinders = 5	cylinders = 6	cylinders = 8
0 0	4 17	1 0	8 0	9 1
Predict bad	Predict good	Predict bad	Predict bad	Predict bad

Take the Original Dataset..

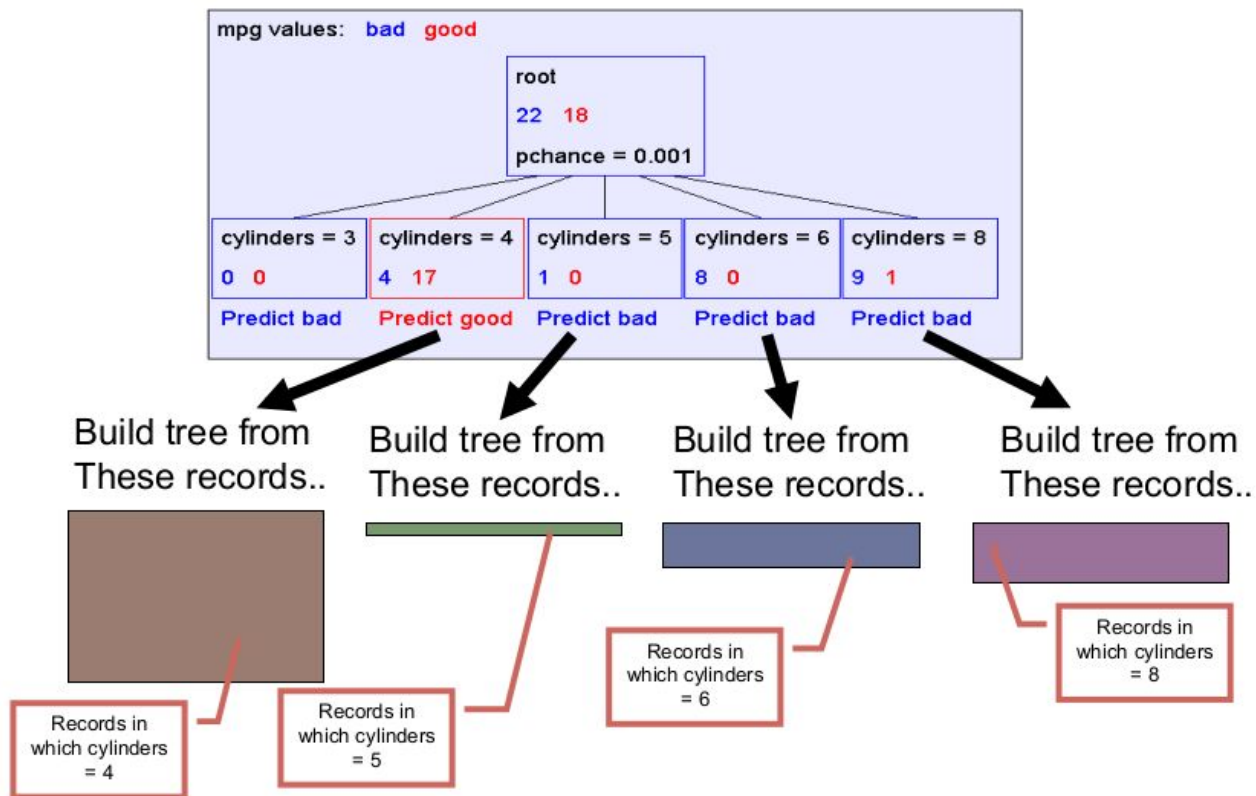And partition it according to the value of the attribute we split on
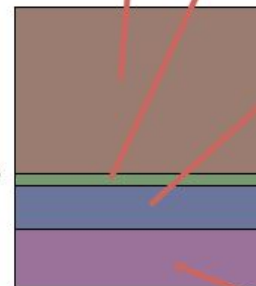
Records in which cylinders = 4

Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8

Recursive Step

Splitting: choosing a good attribute

Would we prefer to split on X_1 or X_2?

X_1
t   f

Y=t : 4   Y=t : 1
Y=f : 0   Y=f : 3

X_2
t   f

Y=t : 3   Y=t : 2
Y=f : 1   Y=f : 2

Idea: use counts at leaves to define probability distributions, so we can measure uncertainty!
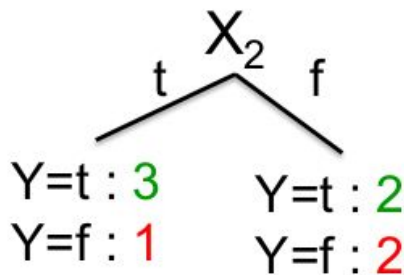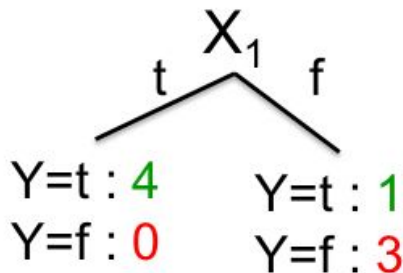
X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Measuring uncertainty

- Good split if we are more certain about classification after split
  - Deterministic good (all true or all false)
  - Uniform distribution bad
  - What about distributions in between?

P(Y=A) = 1/2	P(Y=B) = 1/4	P(Y=C) = 1/8	P(Y=D) = 1/8

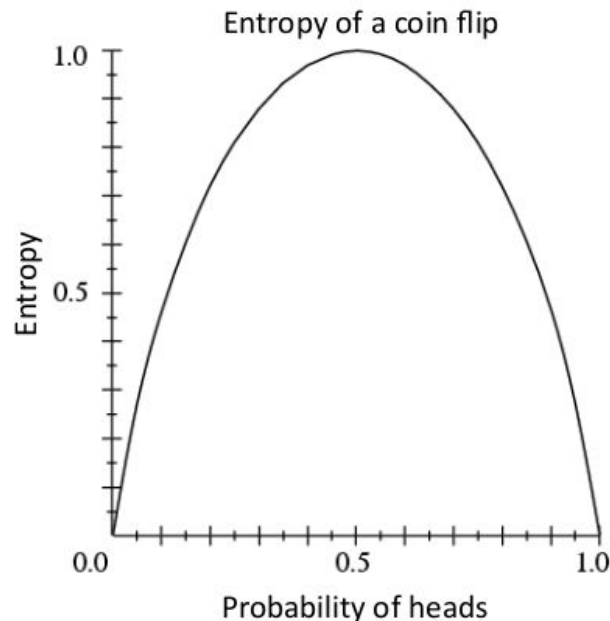| P(Y=A) = 1/4	P(Y=B) = 1/4	P(Y=C) = 1/4	P(Y=D) = 1/4

Entropy

Entropy $H(Y)$ of a random variable Y

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation:
$H(Y)$ is the expected number of bits needed to encode a randomly drawn value of Y (under most efficient code)

Entropy of a coin flip

Entropy vs. Probability of heads

High, Low Entropy

- **"High Entropy"**
  - Y is from a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable

- **"Low Entropy"**
  - Y is from a varied (peaks and valleys) distribution
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

Entropy Example

Entropy of a coin flip

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

P(Y=t) = 5/6

P(Y=f) = 1/6

H(Y) = - 5/6 log$_2$ 5/6 - 1/6 log$_2$ 1/6

= 0.65
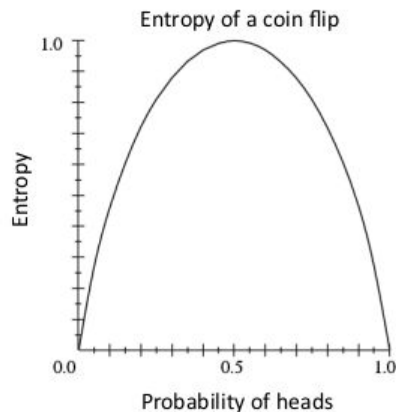
X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Conditional Entropy

Conditional Entropy $H(Y|X)$ of a random variable Y conditioned on a random variable X

$$H(Y \mid X) = - \sum_{j=1}^{v} P(X = x_j) \sum_{i=1}^{k} P(Y = y_i \mid X = x_j) \log_2 P(Y = y_i \mid X = x_j)$$

Example:

X_1

t   f

$P(X_1=t) = 4/6$

$P(X_1=f) = 2/6$

Y=t : 4   Y=t : 1

Y=f : 0   Y=f : 1
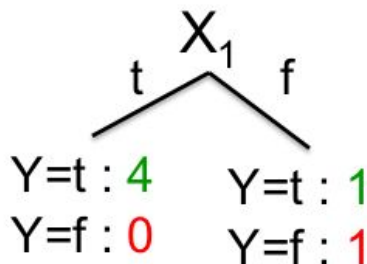
$H(Y|X_1)$ = - 4/6 (1 $\log_2$ 1 + 0 $\log_2$ 0)

               - 2/6 (1/2 $\log_2$ 1/2 + 1/2 $\log_2$ 1/2)

      = 2/6

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Information gain

- Decrease in entropy (uncertainty) after splitting

$$IG(X) = H(Y) - H(Y \mid X)$$

In our running example:

$IG(X_1) = H(Y) - H(Y|X_1)$
$\quad\quad = 0.65 - 0.33$

$IG(X_1) > 0 \rightarrow$ we prefer the split!

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Learning decision trees

- Start from empty decision tree

- Split on **next best attribute (feature)**

  - Use, for example, information gain to select attribute:

$$\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$$

- Recurse

Decision trees will overfit

- Standard decision trees have no learning bias
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Must introduce some bias towards simpler trees
- Many strategies for picking simpler trees
  - Fixed depth
 - Minimum number of samples per leaf
- Random forests

Real-Valued inputs

What should we do if some of the inputs are real-valued?

Infinite number of possible split values!!!

| mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

Threshold splits

- **Binary tree:** split on attribute X at value t
  - One branch: X < t
  - Other branch: X ≥ t

- **Requires small change**
 - Allow repeated splits on same variable **along a path**
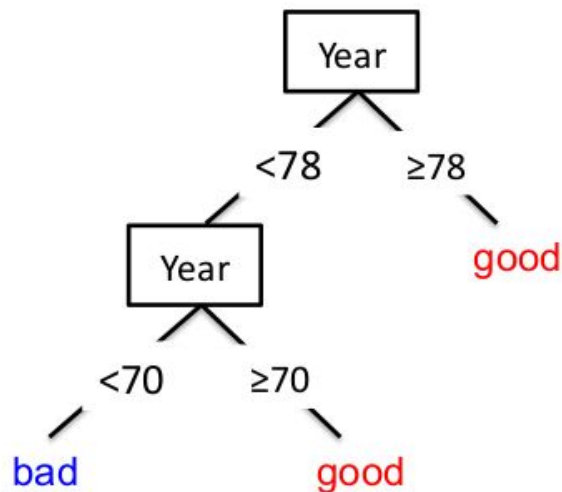
The set of possible thresholds

- Binary tree, split on attribute X
  - One branch: X < t
  - Other branch: X ≥ t
- Search through possible values of *t*
  - Seems hard!!!
- But only a finite number of *t*'s are important:
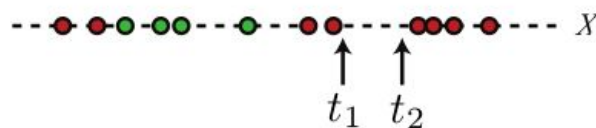
  - Sort data according to X into $\{x_1,\ldots,x_m\}$
  - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$
  - Morever, only splits between examples of different classes matter!
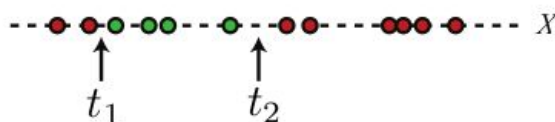
(Figures from Stuart Russell)

Picking the best threshold

- Suppose X is real valued with threshold t

- Want **IG(Y | X:t)**, the information gain for Y when testing if X is greater than or less than t

- Define:
  - $H(Y|X{:}t) = p(X < t)\, H(Y|X < t) + p(X >= t)\, H(Y|X >= t)$
  - $IG(Y|X{:}t) = H(Y) - H(Y|X{:}t)$
  - $IG^*(Y|X) = \max_t IG(Y|X{:}t)$

- Use: $IG^*(Y|X)$ for continuous variables

What you need to know about decision trees

- Decision trees are one of the most popular ML tools
  - Easy to understand, implement, and use
  - Computationally cheap (to solve heuristically)

- Information gain to select attributes (ID3, C4.5,…)

- Presented for classification, can be used for regression and density estimation too

- Decision trees will overfit!!!
  - Must use tricks to find "simple trees", e.g.,
    - Fixed depth/Early stopping
    - Pruning
  - Or, use ensembles of different trees (random forests)