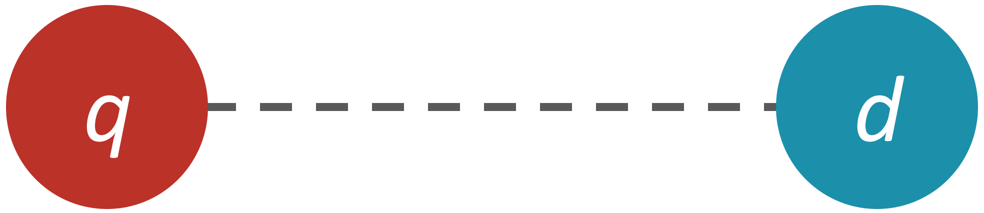
UFMG UNIVERSIDADE FEDERAL DE MINAS GERAIS

Information Retrieval

Learning to Rank: Pointwise

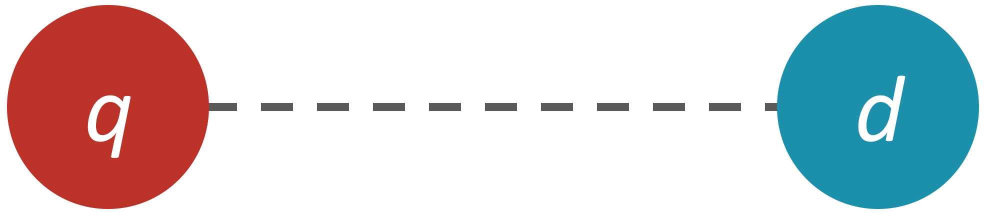
Rodrygo L. T. Santos
rodrygo@dcc.ufmg.br

The ranking problem



$$f(q, d)$$

Learning to rank



$$\boldsymbol{f}(\mathbf{x})$$

Learning to rank

Feature-based representation

- Individual models as ranking ”features”

Discriminative learning

- Effective models learned from data
- Aka machine-learned ranking

Building blocks

Goal is to learn a ranking model

- $f : \mathcal{X} \rightarrow \mathcal{Y}$

That minimizes some loss function

- $\mathcal{L} : f(\mathcal{X}) \times \mathcal{Y} \rightarrow \mathcal{R}$

Ideally, we would like a low test error

- We settle for a good training error / capacity trade-off

What is a learning algorithm?

Given

- A family of functions $\mathcal{F}$ (e.g., linear, trees, neural nets)
- A measure of loss $\mathcal{L}$ (error + capacity)

Learning can be cast as an optimization problem

- $$\begin{aligned} f^* &= \operatorname{argmin}_{f \in \mathcal{F}} \mathcal{L}(f, \text{train}) \\ &= \operatorname{argmin}_{f \in \mathcal{F}} \operatorname{Err}(f, \text{train}) + \lambda \operatorname{Reg}(f) \end{aligned}$$

Classical algorithms

Linear learning algorithms

- Regression (LASSO, Ridge reg.)
- Classification (logistic reg., linear SVM, AdaBoost)

Non-linear learning algorithms

- Neural networks
- (Boosted) regression trees

Linear learning algorithms

$\mathcal{F}$ is the set of linear functions

- $f_w(x) = w^T x$, where $x \in \mathcal{R}^d$, $w \in \mathcal{R}^d$
  - x: d-dimens. feature vector describing an example
  - w: weight vector defining the function f_w
  - $f_w(x)$ is the prediction of f_w for example x

Linear learning algorithms

Most typical regularizers

- L2 regularization, $||w||_2^2 = \sum_j w_j^2$
  (prefer ”flat” weights)
- L1 regularization, $||w||_1 = \sum_j |w_j|$
  (prefer ”sparse” weights)

Classical algorithms

Linear learning algorithms

- **Regression (LASSO, Ridge reg.)**
- Classification (logistic reg., linear SVM, AdaBoost)

Non-linear learning algorithms

- Neural networks
- (Boosted) regression trees

Linear learning for regression

Ordinary least squares

- $\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$

Ridge regression

- Add L2 regularization ($+\lambda \, \|w\|_2^2$)

LASSO

- Add L1 regularization ($+\lambda \, \|w\|_1$)

Linear learning for regression

Ordinary least squares

- $\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$

Typical optimization strategy

- Compute the gradient of $\mathcal{L}(\cdot)$ as $\nabla\mathcal{L}(\cdot)$
- Take a step (iterate) in the opposite direction
  $w = w - \alpha\, \nabla\mathcal{L}(w)$, where α is the learning rate

Computing gradients

Gradient as a d-dimens. vector of partial derivatives

- $\nabla\mathcal{L}(w) = \left(\frac{\partial\mathcal{L}(w)}{\partial w_1}, \frac{\partial\mathcal{L}(w)}{\partial w_2}, \dots, \frac{\partial\mathcal{L}(w)}{\partial w_d}\right)$

How to compute each partial derivative?

- Manual derivation… d may be very large!
- Automatically

Computing gradients

Ordinary least squares

- $\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$

For a given dimension k

- $\frac{\partial}{\partial w_k}\mathcal{L}(w) = \frac{\partial}{\partial w_k}\frac{1}{2}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$

Computing gradients

$$\frac{\partial}{\partial w_k}\mathcal{L}(w) = \frac{\partial}{\partial w_k}\frac{1}{2}\sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$$

$$= \sum_{i=1}^{m}\frac{\partial}{\partial w_k}\frac{1}{2}\left(f_w(x^{(i)}) - y^{(i)}\right)^2$$

$$= \sum_{i=1}^{m} 2\,\frac{1}{2}\left(f_w(x^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial w_k}\left(f_w(x^{(i)}) - y^{(i)}\right)$$

$$= \sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial w_k}f_w(x^{(i)})$$

Computing gradients

$$
\begin{aligned}
\frac{\partial}{\partial w_k} f_w\left(x^{(i)}\right) &= \frac{\partial}{\partial w_k} \mathrm{w}^T x^{(i)} \\
&= \frac{\partial}{\partial w_k} \sum_{j=1}^{d} w_j x_j^{(i)} \\
&= \frac{\partial}{\partial w_k} \left(w_k x_k^{(i)} + \sum_{j \neq k} w_j x_j^{(i)} \right) \\
&= x_k^{(i)}
\end{aligned}
$$

Computing gradients

$$\frac{\partial}{\partial w_k}\mathcal{L}(w) = \sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial w_k}f_w(x^{(i)})$$

$$= \sum_{i=1}^{m}\underbrace{\left(f_w(x^{(i)}) - y^{(i)}\right)}_{\textit{prediction error}}\underbrace{x_k^{(i)}}_{k^{th}\textit{ feature score}}$$

Computing gradients

$$\frac{\partial}{\partial w_k}\mathcal{L}(w) = \sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial w_k} f_w(x^{(i)})$$

$$= \sum_{i=1}^{m}\left(f_w(x^{(i)}) - y^{(i)}\right) x_k^{(i)}$$

- $m = |D|$: batch gradient descent
- $m = c \ll |D|$: mini-batch gradient descent
- $m = 1$: stochastic gradient descent

Classical algorithms

Linear learning algorithms

- Regression (LASSO, Ridge reg.)
- **<u>Classification (logistic reg., linear SVM, AdaBoost)</u>**

Non-linear learning algorithms

- Neural networks
- (Boosted) regression trees

Linear learning for classification

Prediction

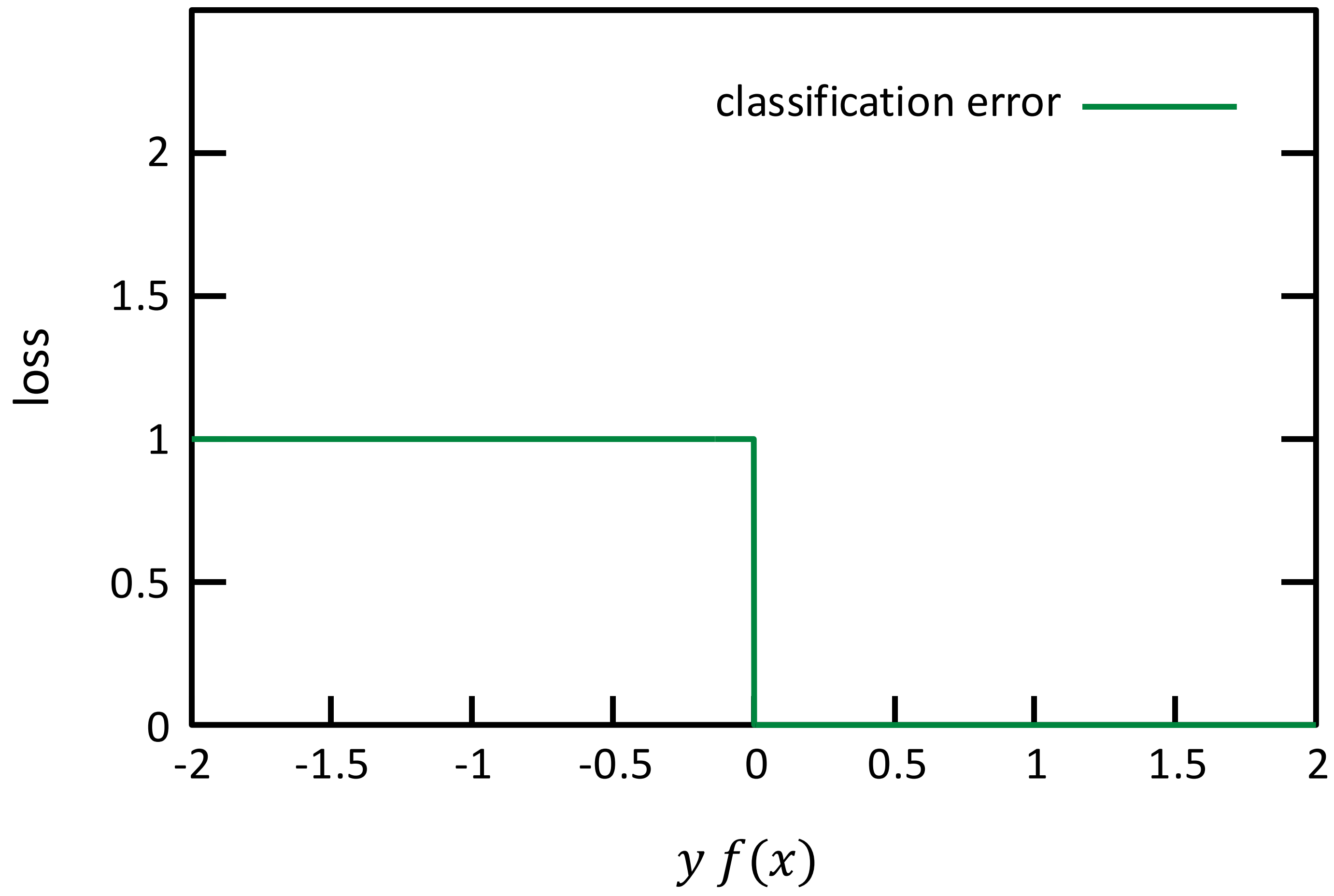
- $f_w(x) = w^T x$ $\quad$ > 0 predicts $+1$

  ≤ 0 predicts -1

Classification error

- $\mathcal{L}(w) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\left(y^{(i)} f_w\left(x^{(i)}\right) < 0\right)$
- **Problem:** non-differentiable!

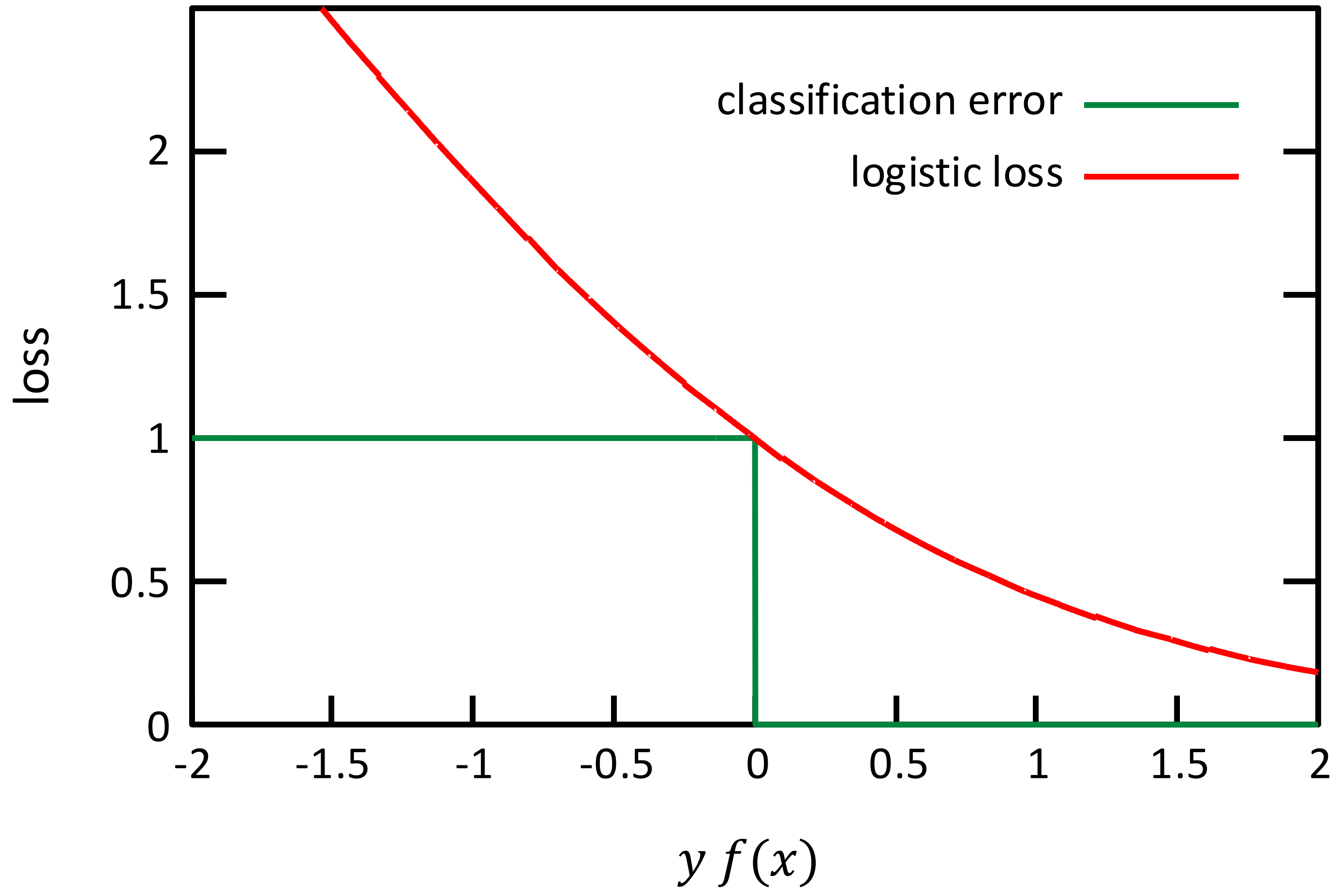
Upper-bounding classification error



Classification error

$\mathbf{1}(y f_w(x) < 0)$

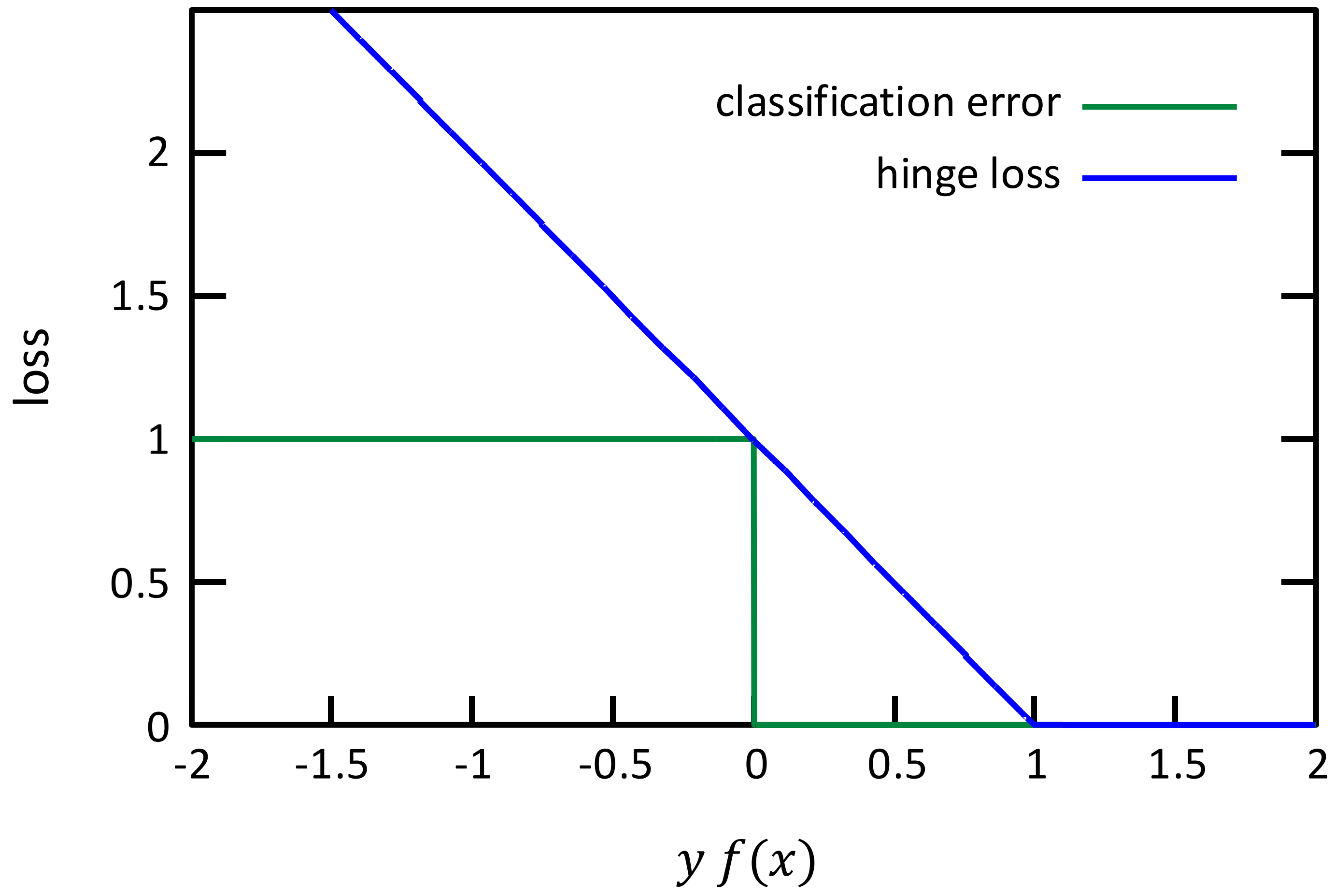
Upper-bounding classification error



Logistic loss

$$\log\left(1 + \exp(-y f_w(x))\right)$$

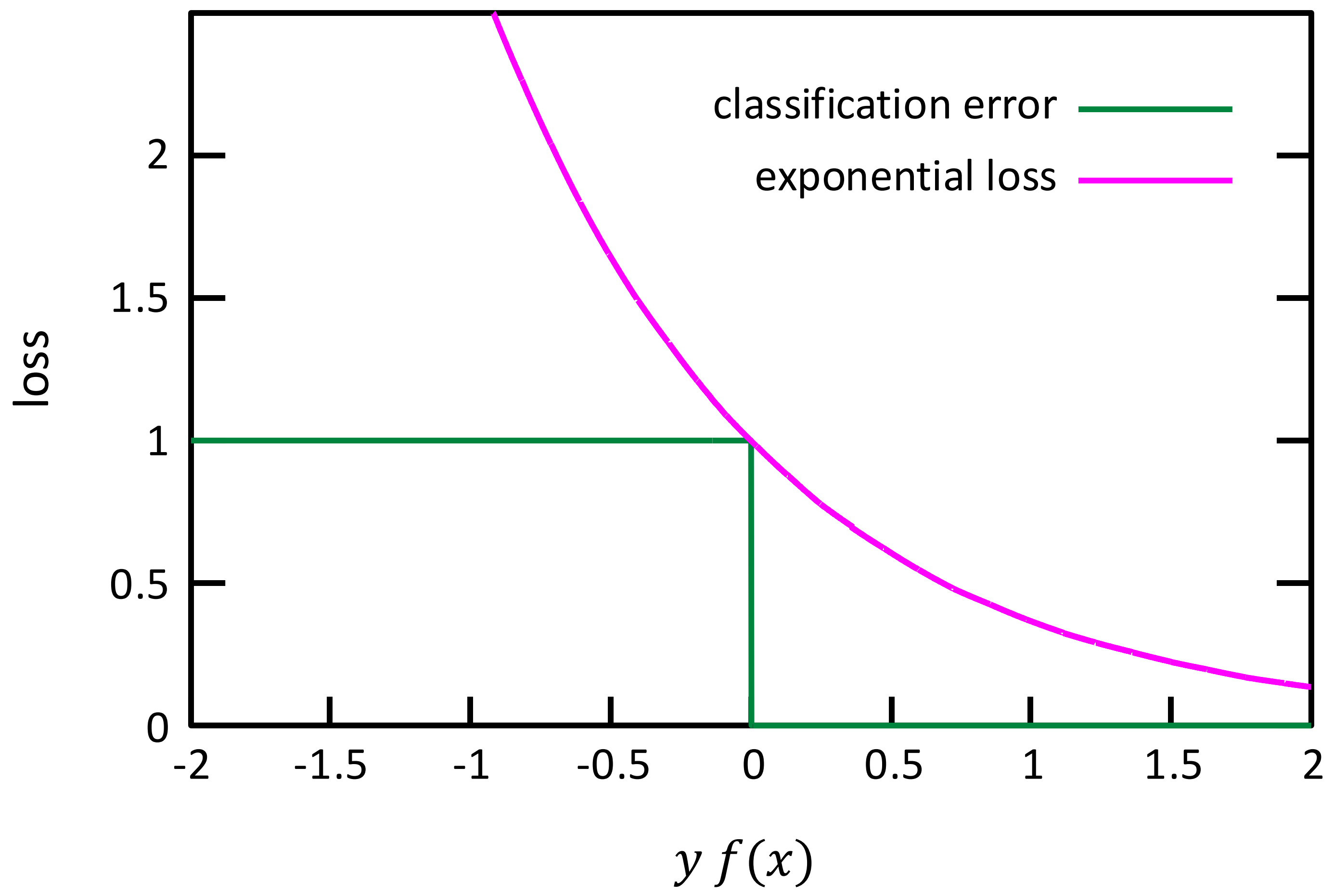
Upper-bounding classification error



Hinge loss

$$\max\left(0, 1 - y f_w(x)\right)$$

Upper-bounding classification error



Exponential loss

$\exp(-y f_w(x))$

Linear learning for classification

Logistic regression

- Logistic loss + L1 or L2 regularizer

Support Vector Machines (SVM)

- Hinge loss + L2 regularizer

AdaBoost

- Exponential loss + L1 regularizer

Classical algorithms

Linear learning algorithms

- Regression (LASSO, Ridge reg.)
- Classification (logistic reg., linear SVM, AdaBoost)

Non-linear learning algorithms

- **Neural networks**
- (Boosted) regression trees

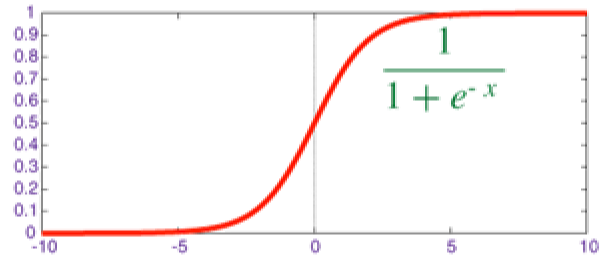
Neural networks

Composable functions through several layers

- Inspired by biological neural networks (your brain)

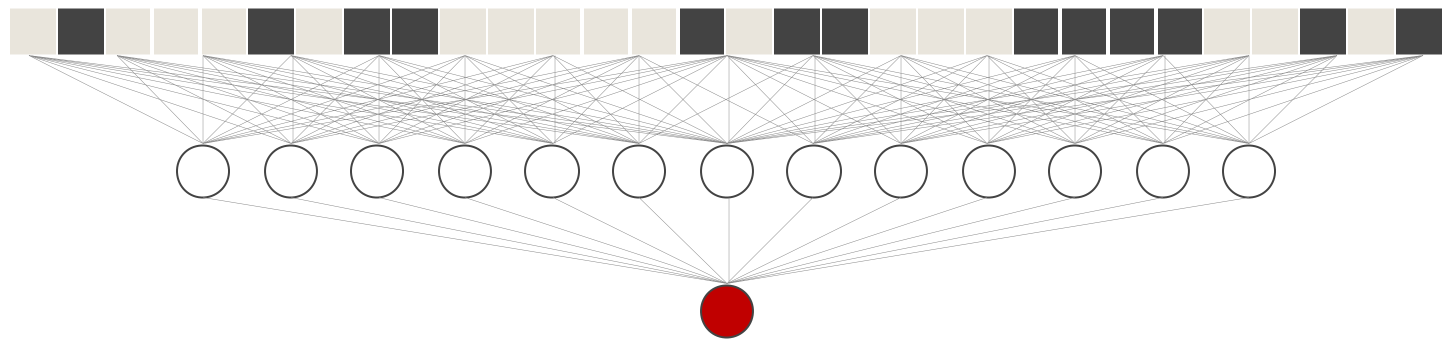
Each layer performs a linear operation potentially followed by a non-linear activation

- $\ell_0 = x$ and $\ell_i = \underset{\textit{activation}}{\underline{\sigma_i}}(\underset{\textit{weights}}{\underline{W_i}}\ell_{i-1} + \underset{\textit{biases}}{\underline{b_i}})$



sigmoid function

Example: shallow (2-layer) network



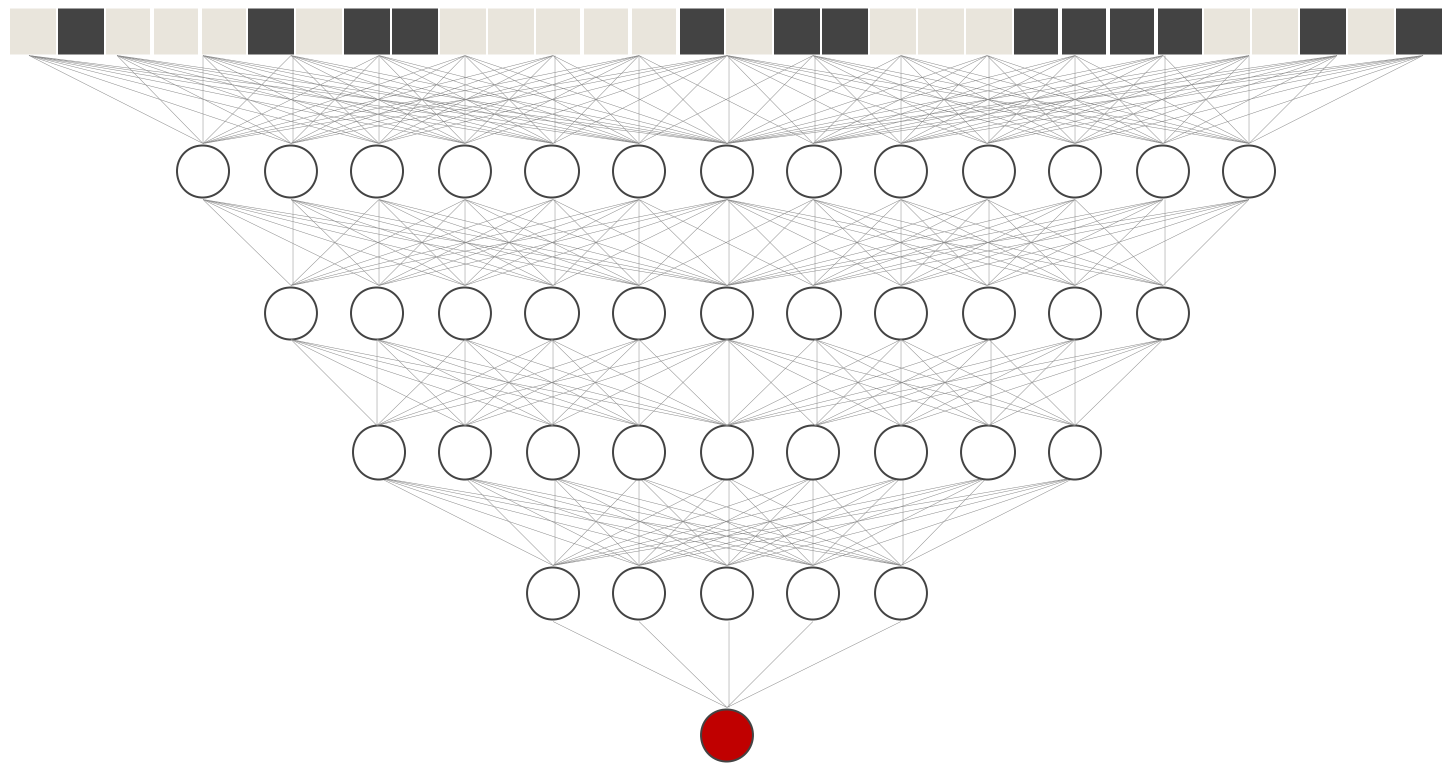
$f_w(x) = \ell_2$

$$\underset{[d \times 1]}{\ell_0} = x$$

$$\underset{[h_1 \times 1]}{\ell_1} = \sigma_1(\underset{[h_1 \times d]}{W_1}\underset{[d \times 1]}{\ell_0} + \underset{[h_1 \times 1]}{b_1})$$

$$\underset{[1 \times 1]}{\ell_2} = \sigma_2(\underset{[1 \times h_1]}{W_2}\underset{[h_1 \times 1]}{\ell_1} + \underset{[1 \times 1]}{b_2})$$

Example: deep (5-layer) network



$$\underset{[d\times 1]}{\ell_0} = x$$

$$\underset{[h_1\times 1]}{\ell_1} = \sigma_1(\underset{[h_1\times d]}{W_1}\underset{[d\times 1]}{\ell_0} + \underset{[h_1\times 1]}{b_1})$$

$$\underset{[h_2\times 1]}{\ell_2} = \sigma_2(\underset{[h_2\times h_1]}{W_2}\underset{[h_1\times 1]}{\ell_1} + \underset{[h_2\times 1]}{b_2})$$

$$\underset{[h_3\times 1]}{\ell_3} = \sigma_3(\underset{[h_3\times h_2]}{W_3}\underset{[h_2\times 1]}{\ell_2} + \underset{[h_3\times 1]}{b_3})$$

$$\underset{[h_4\times 1]}{\ell_4} = \sigma_4(\underset{[h_4\times h_3]}{W_4}\underset{[h_3\times 1]}{\ell_3} + \underset{[h_4\times 1]}{b_4})$$

$$\underset{[1\times 1]}{\ell_5} = \sigma_5(\underset{[1\times h_4]}{W_5}\underset{[h_4\times 1]}{\ell_4} + \underset{[1\times 1]}{b_5})$$

Neural network learning

Parameters (W_i's and b_i's) learned via gradient descent

- Find $\{W_i, b_i\}$ which minimize loss

Works with any differentiable loss

- Cross-entropy commonly used
- L2 regularization on $\{W_i, b_i\}$

Architecture selected using validation data

Classical algorithms

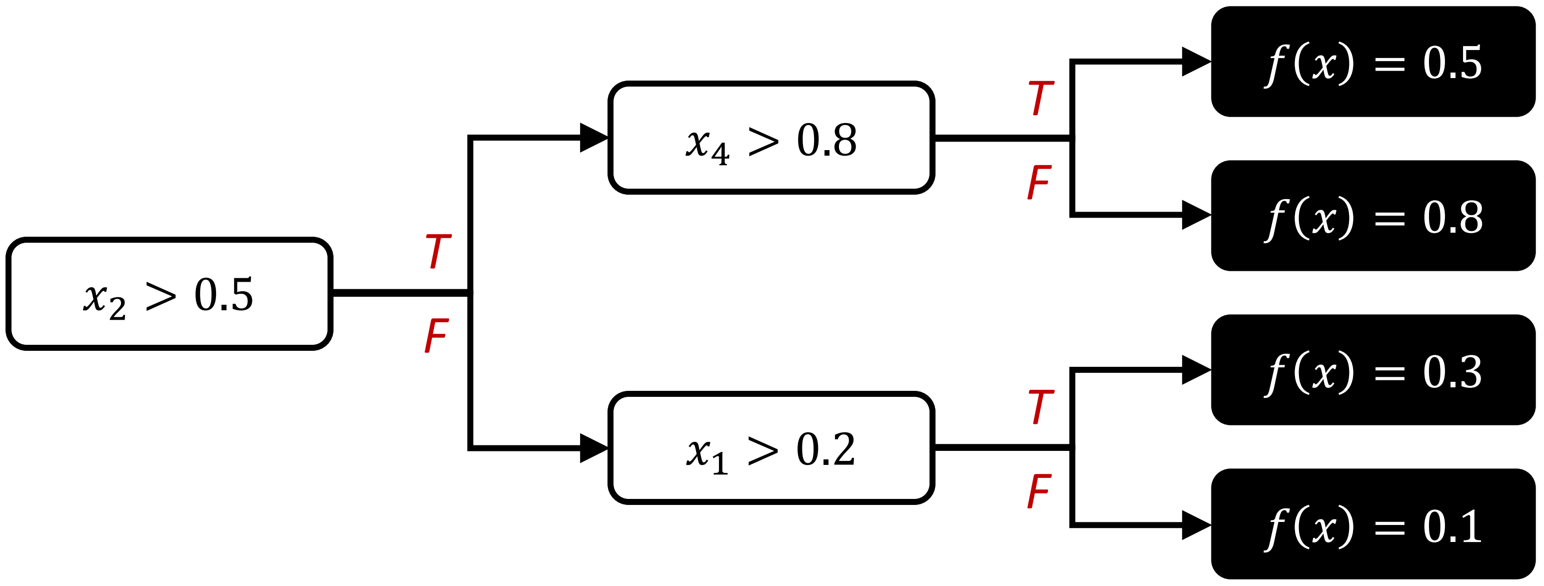
Linear learning algorithms

- Regression (LASSO, Ridge reg.)
- Classification (logistic reg., linear SVM, AdaBoost)

Non-linear learning algorithms

- Neural networks
- **(Boosted) regression trees**

Regression trees



$x = (0.3, 0.6, 0.2, 0.5)$

$x_2 > 0.5$? ***T***

$x_4 > 0.8$? ***F***

$\therefore f(x) = 0.8$

Examples travel the tree from the root to one leaf

Each node performs a test on the input x

Each leaf corresponds to a prediction $f(x)$

Greedy learning

1. Start with a tree containing only the root
2. Splitting: for each node with depth < max depth
   - Find best test and create two leaves from the node
   - Find best prediction for these two leaves
3. Rep. #2 until no more nodes with depth < max depth

Greedy learning

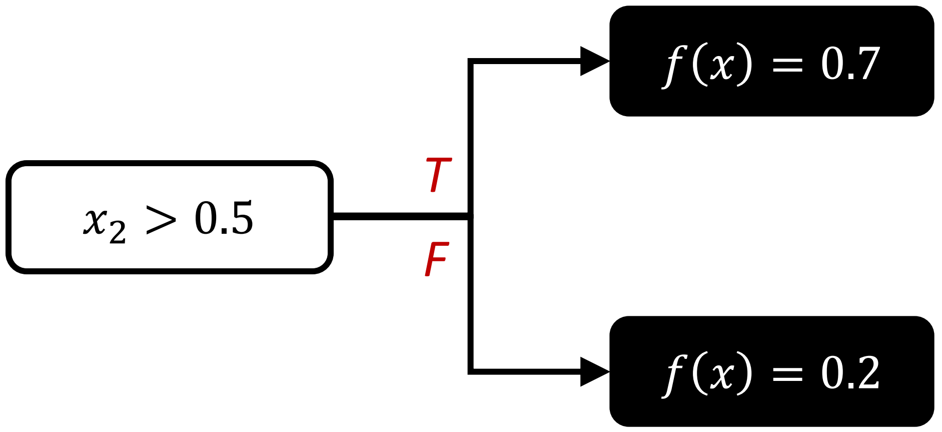
$\boxed{f(x) = 0.5}$

Best split: $x_2 > 0.5$

Best predictions: $f(x) = 0.7 \text{ if } x_2 > 0.5$

$f(x) = 0.2 \text{ otherwise}$

Greedy learning

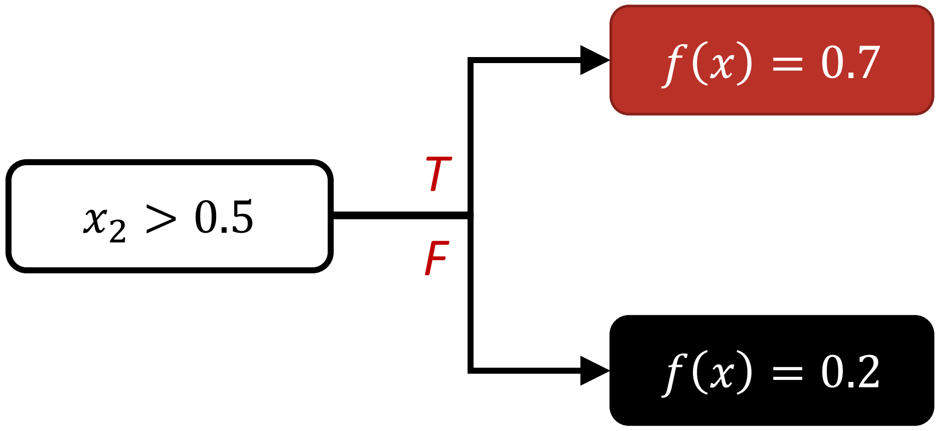


Best split: $x_2 > 0.5$

Best predictions: $f(x) = 0.7$ if $x_2 > 0.5$

$f(x) = 0.2$ otherwise

Greedy learning

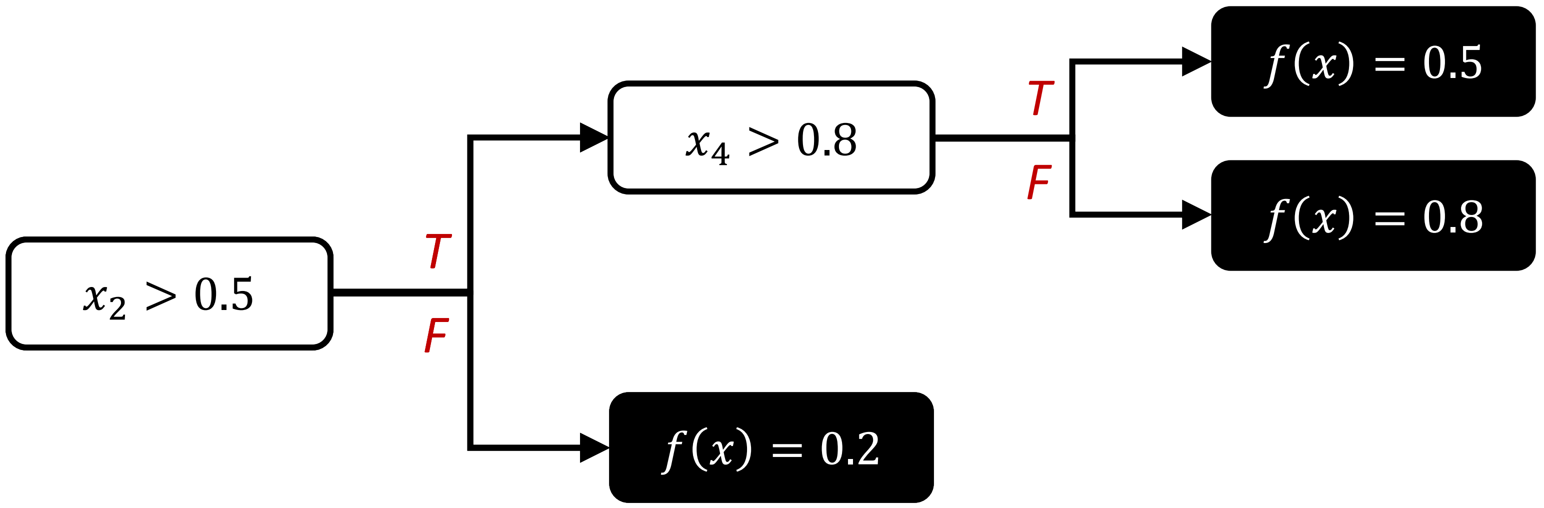


Best split: $x_4 > 0.8$

Best predictions: $f(x) = 0.5 \text{ if } x_4 > 0.8$

$f(x) = 0.8 \text{ otherwise}$

Greedy learning

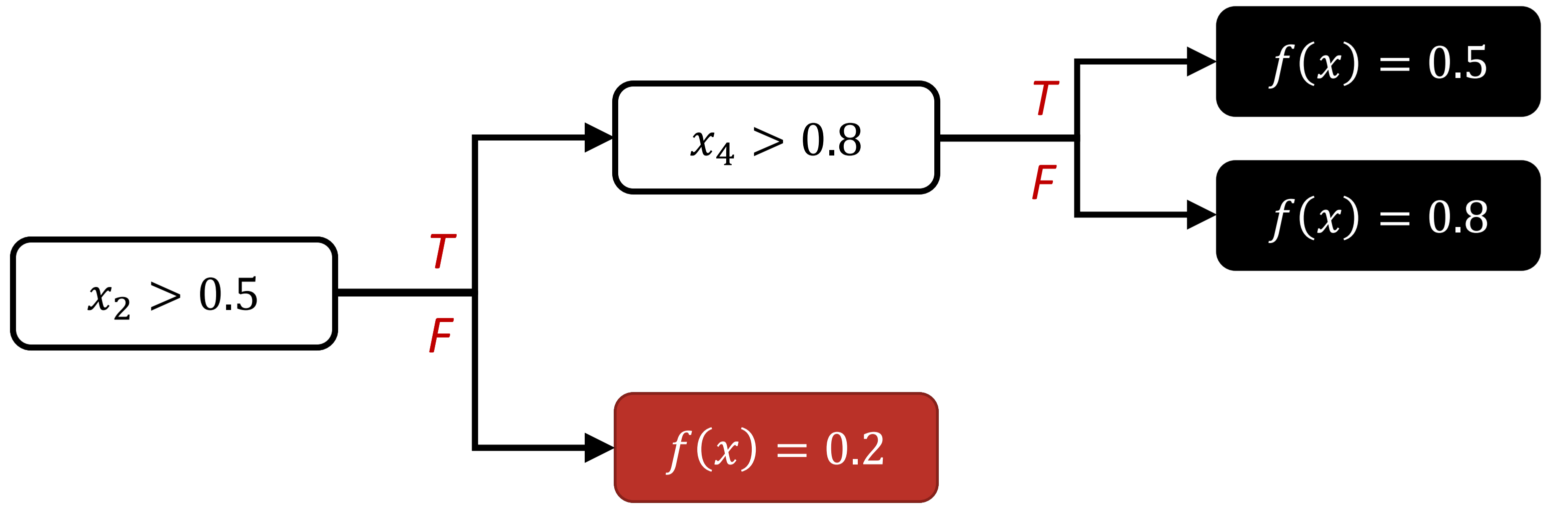


Best split: $x_4 > 0.8$

Best predictions: $f(x) = 0.5$ if $x_4 > 0.8$

$f(x) = 0.8$ otherwise

Greedy learning

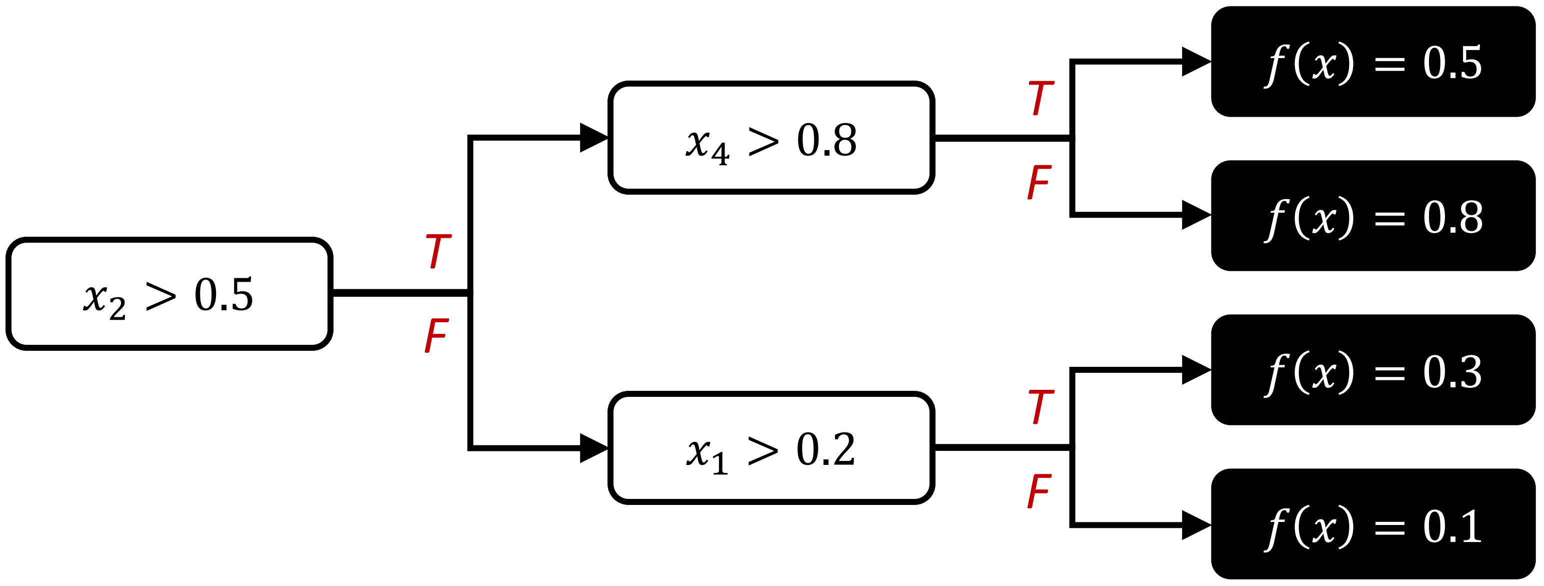


Best split: $x_1 > 0.2$

Best predictions: $f(x) = 0.3$ if $x_1 > 0.2$

$f(x) = 0.1$ otherwise

Greedy learning



Best split: $x_1 > 0.2$

Best predictions: $f(x) = 0.3$ if $x_1 > 0.2$

$f(x) = 0.1$ otherwise

Greedy learning a regression tree

1. Start with a tree containing only the root

2. Splitting: for each node with depth < max depth
   - Find best test and create two leaves from the node
   - Find best prediction for these two leaves

3. Rep. #2 until no more nodes with depth < max depth

Splitting a node

Given all S_k examples reaching node k, find:

- i, τ: defining the test $x_i > \tau$

Let $S_k^l \equiv S_k : x_i \leq \tau$ and $S_k^r \equiv S_k : x_i > \tau$

Minimize: $\mathcal{L} = \sum_{(x,y)\in S_k^l} \left(\bar{y}^l - y\right)^2 + \sum_{(x,y)\in S_{k^r}} (\bar{y}^r - y)^2$

- $\underbrace{\bar{y}^l \equiv \frac{1}{|S_k^l|} \sum_{(x,y)\in S_k^l} y}_{f(x) \text{ in the left branch}}$ and $\underbrace{\bar{y}^r \equiv \frac{1}{|S_k^r|} \sum_{(x,y)\in S_k^r} y}_{f(x) \text{ in the right branch}}$

Regression trees

Advantages

- Low computation cost for prediction
- Easy to interpret

Disadvantages

- Only models piecewise constant functions
- Learning deep trees requires lots of examples

Gradient boosted regression trees (GBRTs)

A GBRT is an ensemble of regression trees

- $f(x) = \sum_{t=1}^{T} h_t(x)$, where h_t is a regression tree

Advantages

- Models more complex functions than a single tree
- Learning many shallow trees requires less training data than learning one deep tree

Gradient boosting

For $t = 1, \dots, T$

- Current model: $f_t = \sum_{i=1}^{t-1} h_i$
- Learn h_t^* to "correct" f_t
  - $f_{t+1} = f_t + h_t^* \approx y$

    $\therefore h_t^* \approx \underline{y - f_t \qquad \textit{residual}}$
  - $h_t^* = \operatorname{argmin}_{h_t} \sum_{(x,y)} (h_t(x) - (y - f_t(x)))^2$

    $= \operatorname{argmin}_{h_t} \sum_{(x,y)} (h_t(x) - (-\alpha \nabla \mathcal{L}(f_t)))^2$

Summary

Pointwise approaches borrowed from other tasks

- Regression, classification, ordinal classification

Straightforward, yet not quite suited for ranking

- Ranking requires getting relative scores right
- Higher positions matter more than lower positions
- All queries are equally important

References

Learning to rank for information retrieval
Liu, FnTIR 2009

Learning to rank for information retrieval
Liu, 2011

Learning to rank for information retrieval and natural language processing
Li, 2014

References

[Learning for Web rankings]
Grangier and Paiement, 2011

[Stanford's CS229 lecture notes]
Ng, 2017

[TensorFlow and deep learning without a PhD]
Gorner, 2017

References

Mathematics for machine learning
Deisenroth et al., 2018

UFMG UNIVERSIDADE FEDERAL DE MINAS GERAIS

Coming next...

Learning to Rank: Pairwise and Listwise

Rodrygo L. T. Santos
rodrygo@dcc.ufmg.br