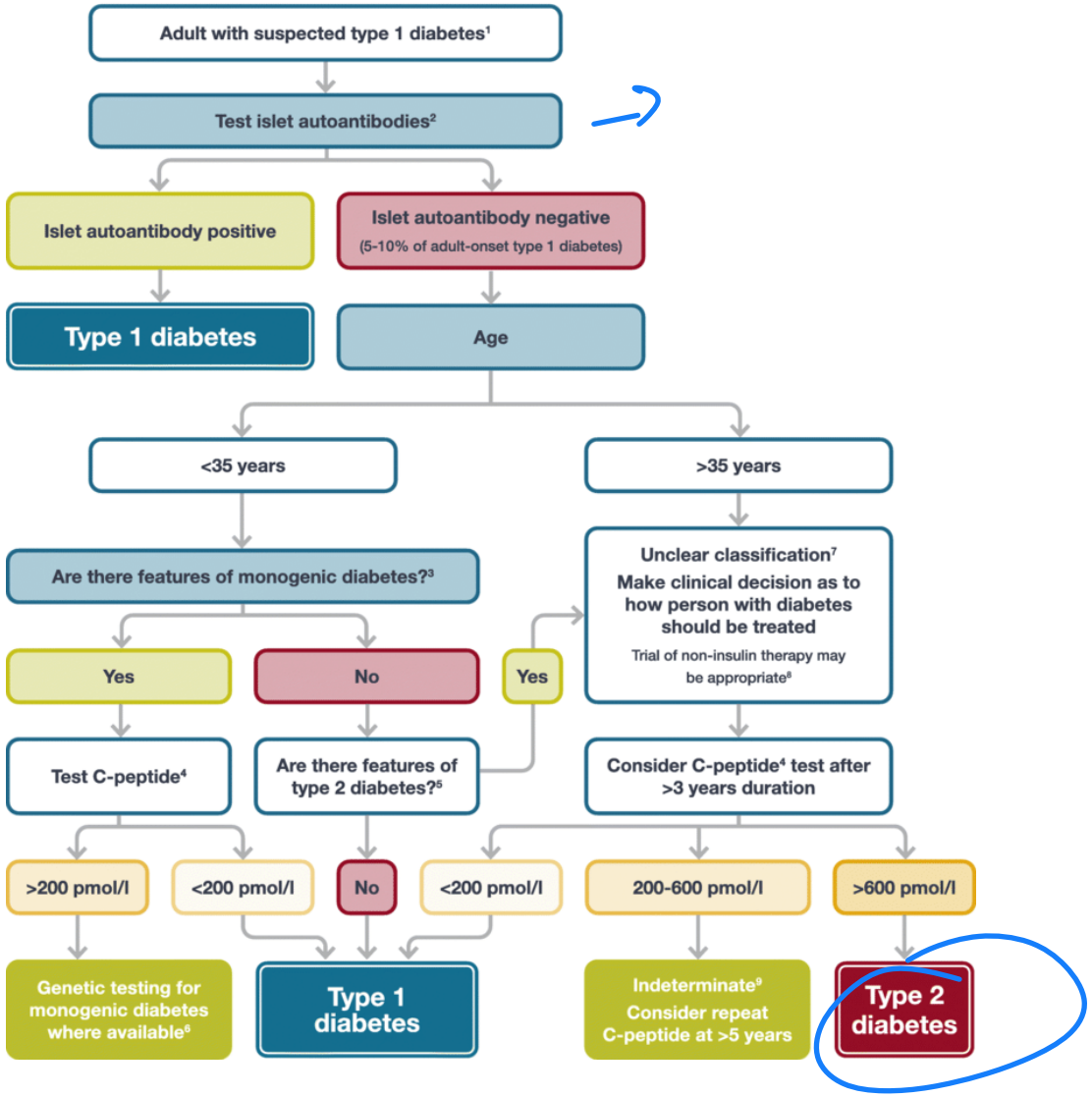
Non-parametric methods
Trees

Natasha Jaques

UNIVERSITY of WASHINGTON

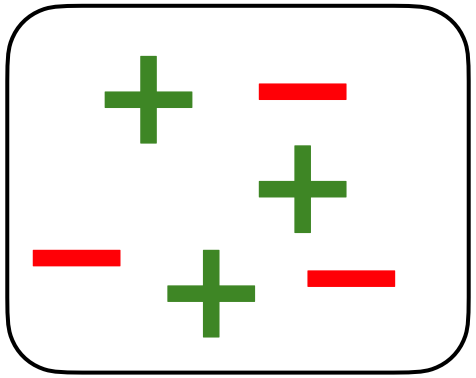
Flow chart for investigation of suspected type 1 diabetes in newly diagnosed adults, based on data from White European populations



Holt et al., 2021

Decision trees

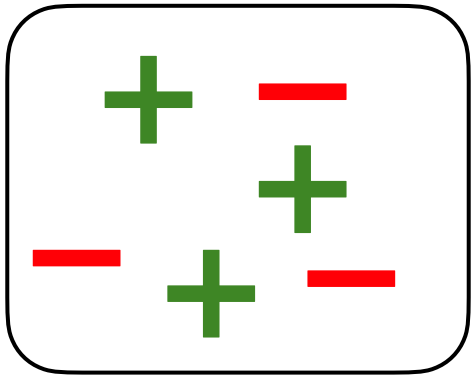
Want to classify spam from not spam using text-based features. What feature gives me the best split?



Full dataset: 50% spam, 50% not
Entropy: 1.0

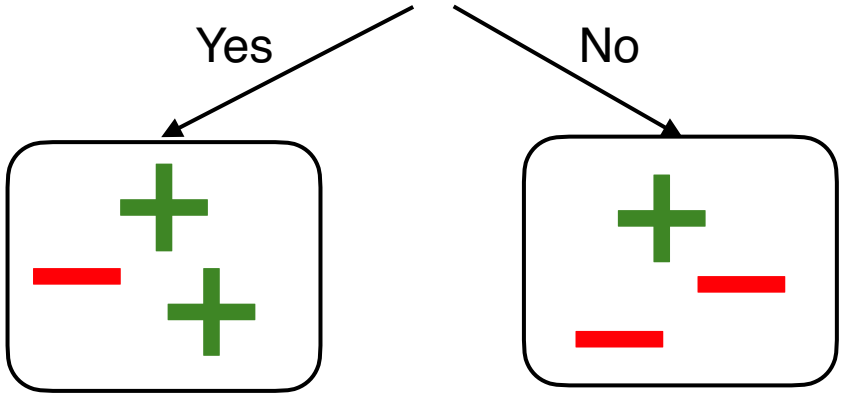
Decision trees

Want to classify spam from not spam using text-based features. What feature gives me the best split?



Full dataset: 50% spam, 50% not
Entropy: 1.0

Split on feature: contains "Limited offer"

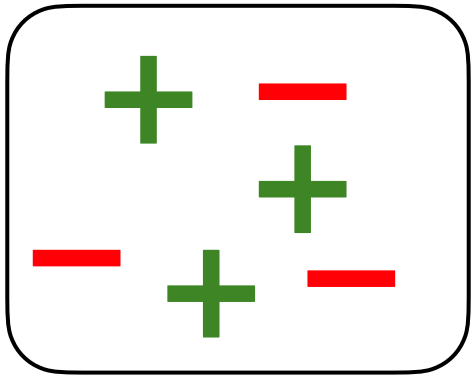


66% spam, 33% not
Entropy: 0.918

33% spam, 66% not
Entropy: 0.918

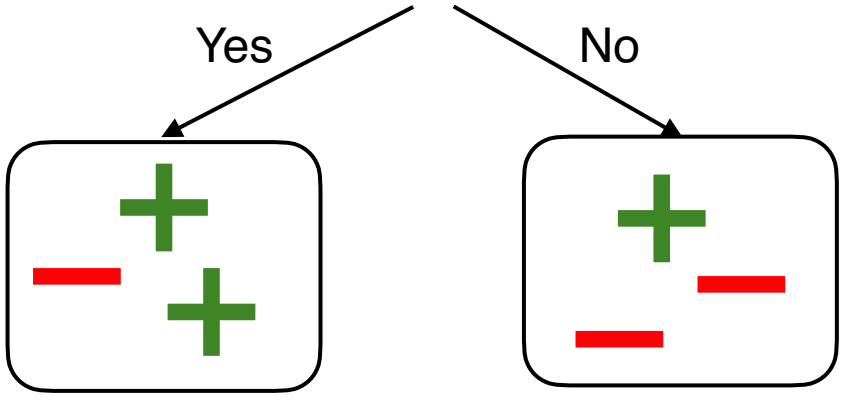
Decision trees

Want to classify spam from not spam using text-based features. What feature gives me the best split?



Full dataset: 50% spam, 50% not
Entropy: 1.0

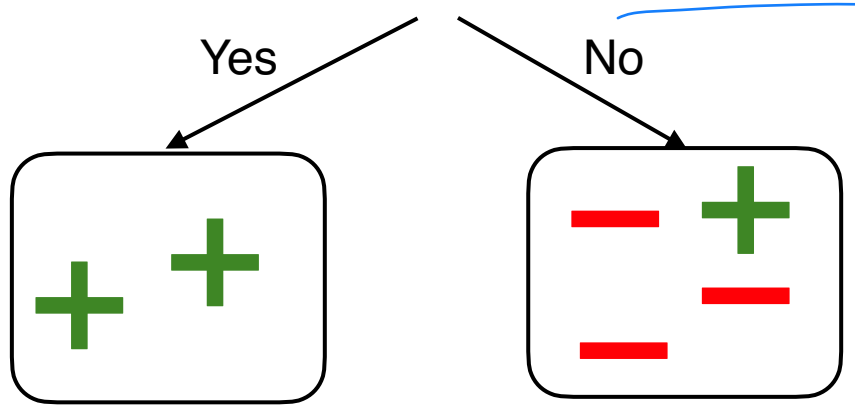
Split on feature: contains "Limited offer"



66% spam, 33% not
Entropy: 0.918

33% spam, 66% not
Entropy: 0.918

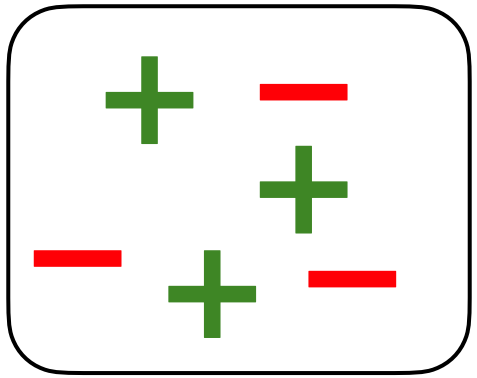
Split on feature: contains "Nigerian Prince"



100% spam, 0% not
Entropy: 0

25% spam, 75% not
Entropy: 0.811

Decision trees

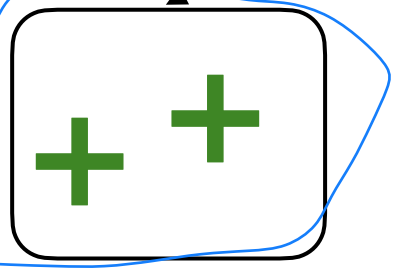


Full dataset: 50% spam, 50% not
Entropy: 1.0

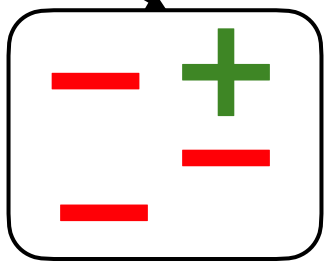
Split on feature: contains "Nigerian Prince"

Yes

No



100% spam, 0% not
Entropy: 0

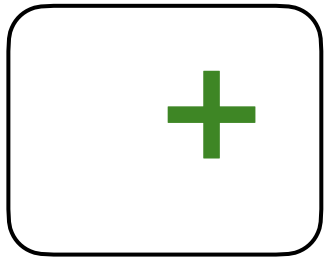


25% spam, 75% not
Entropy: 0.811

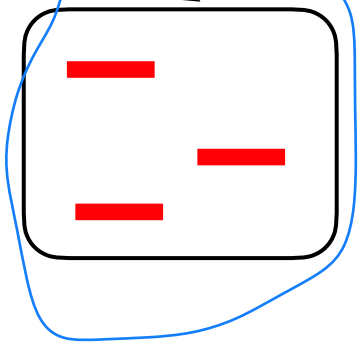
Split on feature: contains
❗ CLEARANCE ❗ ENDS SOON ❗

Yes

No



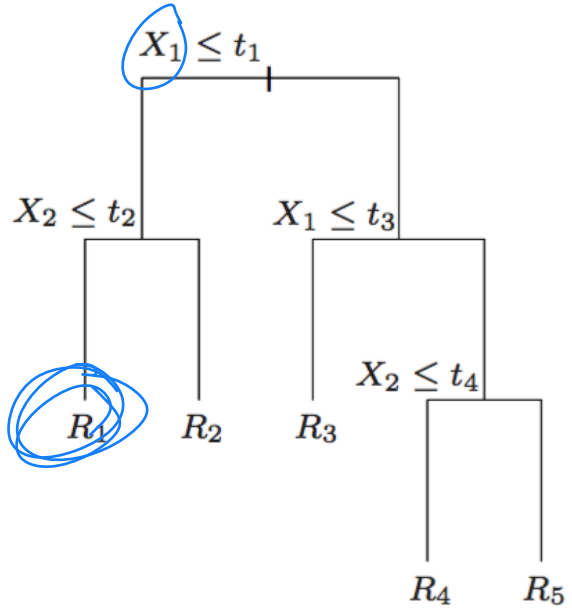
100% spam, 0% not
Entropy: 0

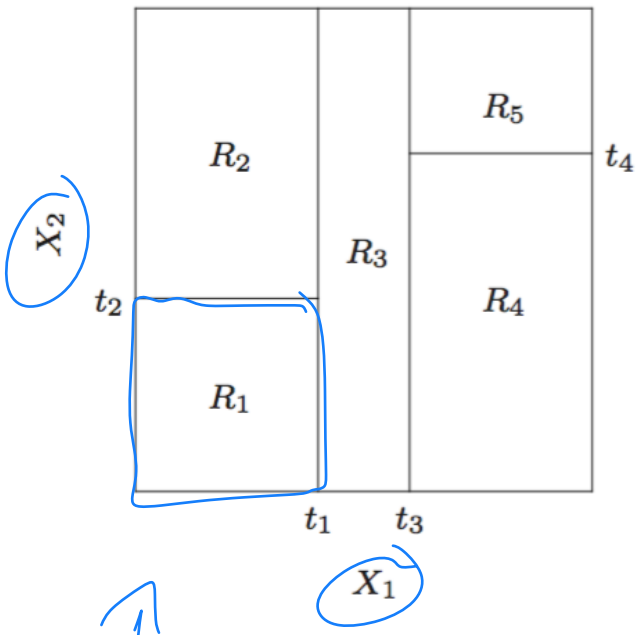


0% spam, 100% not
Entropy: 0

"axis-aligned trees"

Regression trees

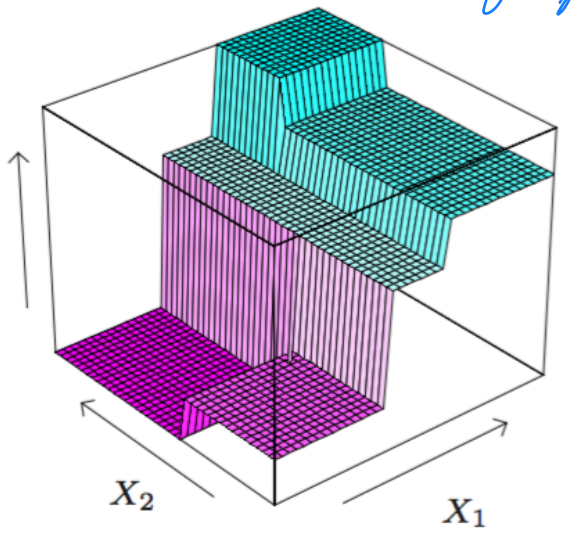




$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

c_m: actual output value for regression

I: 1 if $x \in R_m$, 0 otherwise

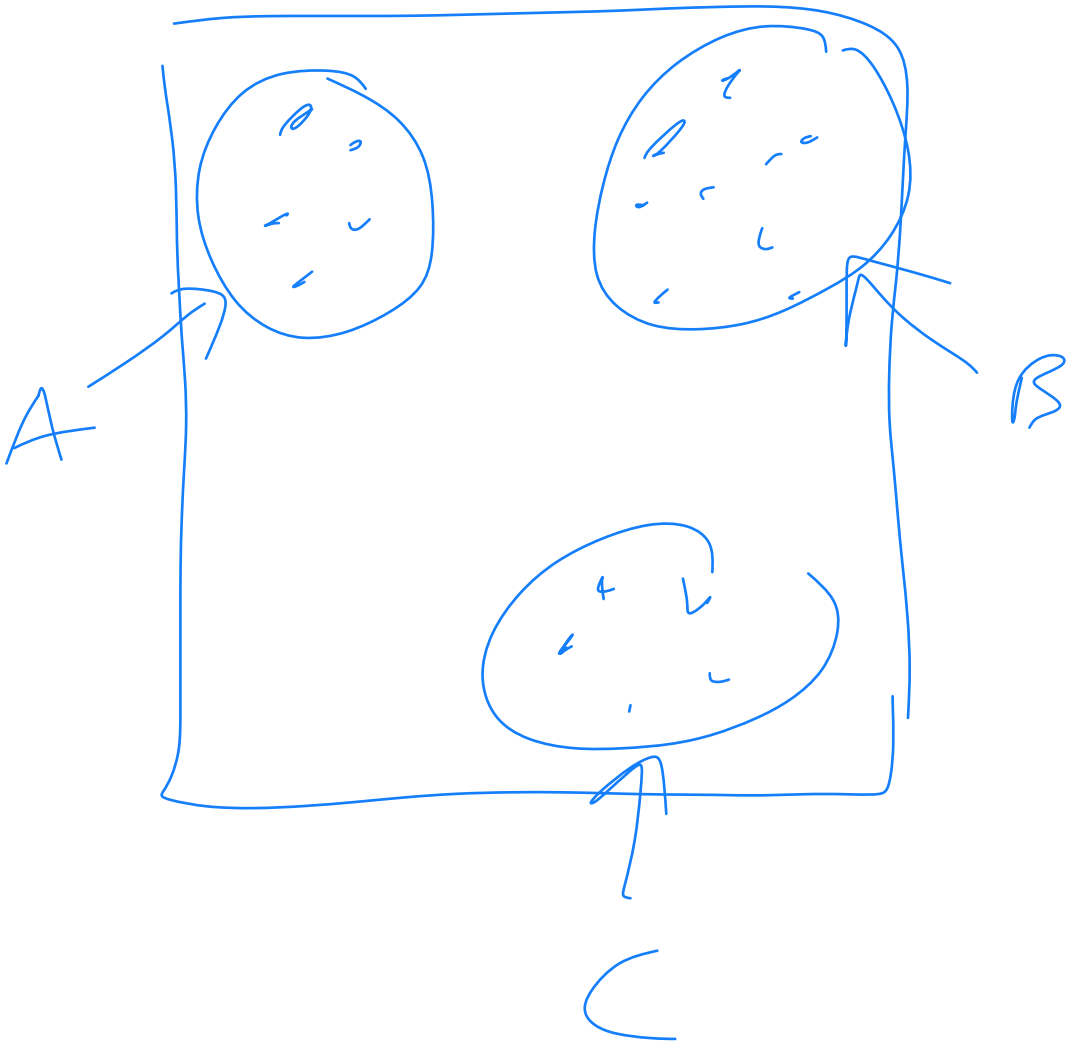


when to use?

"interpretable"

decision boundary

when not to use? pixels, super complex high-d data

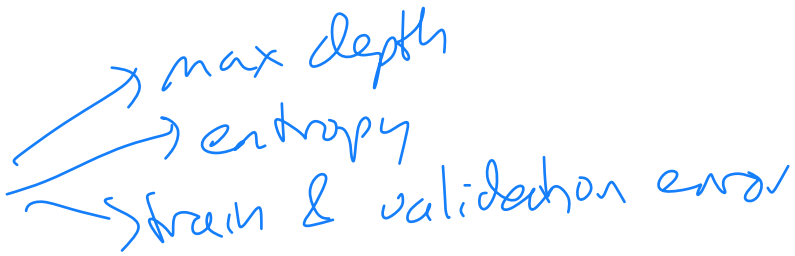

A
B
C

Generic algorithm for building trees

1. Start from empty decision tree
2. Recursively, for each node:
   - Iterate through all features and compute how good it'd be to split on each feature
   - Split on the "best" feature
3. Prune

Design choices:

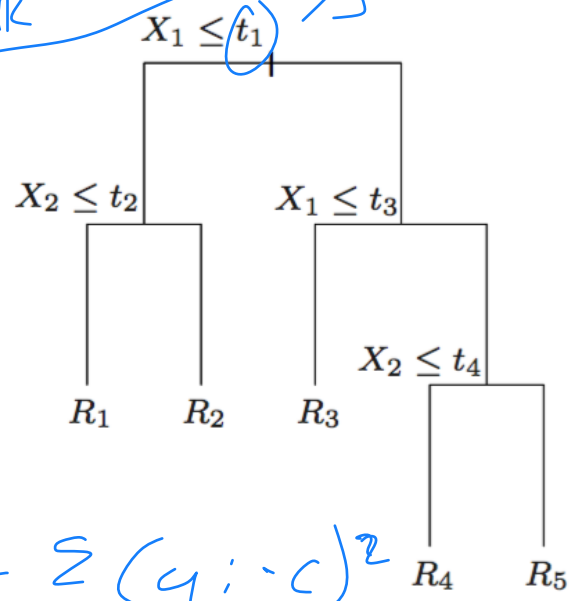
- Termination condition
- Tree complexity
- Splitting criterion
- Pruning



Splitting regression trees

$y_1, y_2 \dots y_n$

$\min_{c \in R} \sum (y_i - c)^2$



$\frac{d}{dc} \sum (y_i - c)^2$

$= -2 \sum (y_i - c) = 0$

$nc = \sum y_i \rightarrow \boxed{c = \frac{1}{n} \sum y_i}$

↳ loss func? MSE

$$f(x) = \sum_{m=1}^{M} c_m 1\{x \in R_m\}$$

Split variable x_j, split point s

$R_1(j,s) = \{x, y \mid x_j \le s\} \rightarrow c_1$

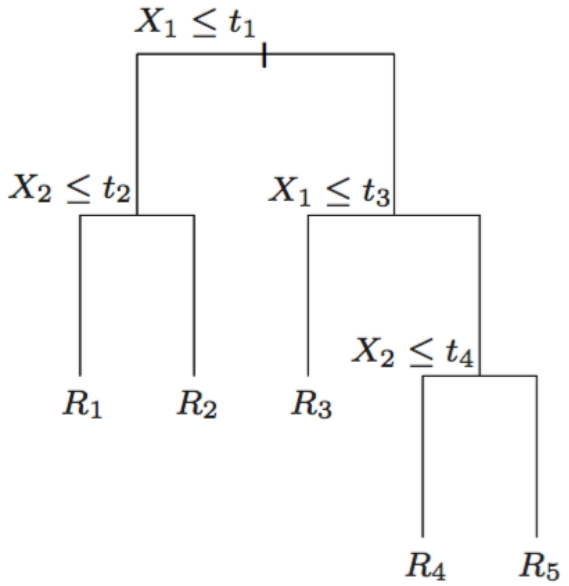
$R_2(j,s) = \{x, y \mid x_j > s\} \rightarrow c_2$

$$\underset{j,s}{\operatorname{argmin}} \left[\min_{c_1} \left[\sum_{(x,y) \in R_1(j,s)} (y - c_1)^2 \right] + \min_{c_2} \left[\sum_{(x,y) \in R_2(j,s)} (y - c_2)^2 \right] \right]$$

loss on LHS — loss on RHS

$c_1 = \frac{1}{|R_1|} \sum_{(x,y) \in R_1} y$

Splitting regression trees



→ 0-1 error / binary

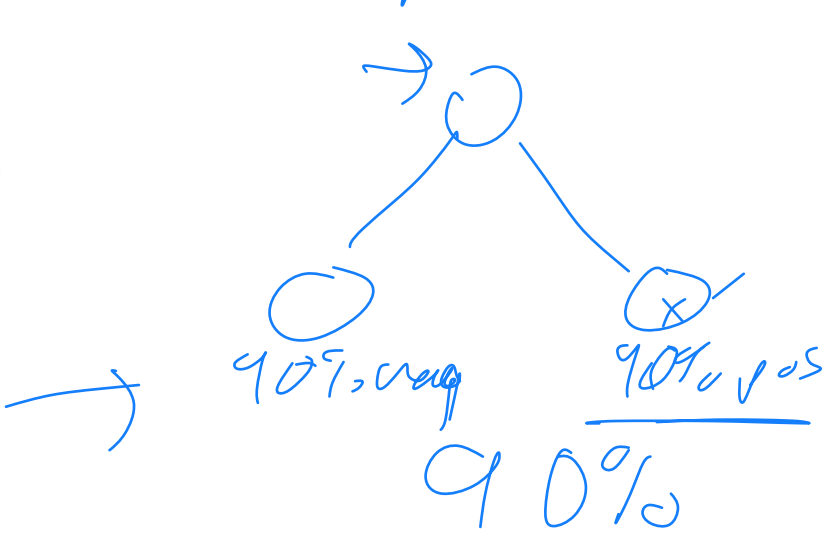
$$f(x) = \sum_{m=1}^{M} c_m 1\{x \in R_m\}$$

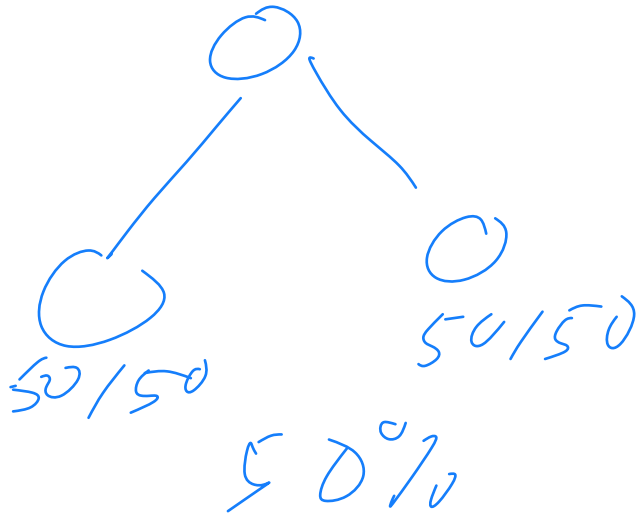
split on $x_j \leq s$, $x_j > s$

define "impurity" measure I

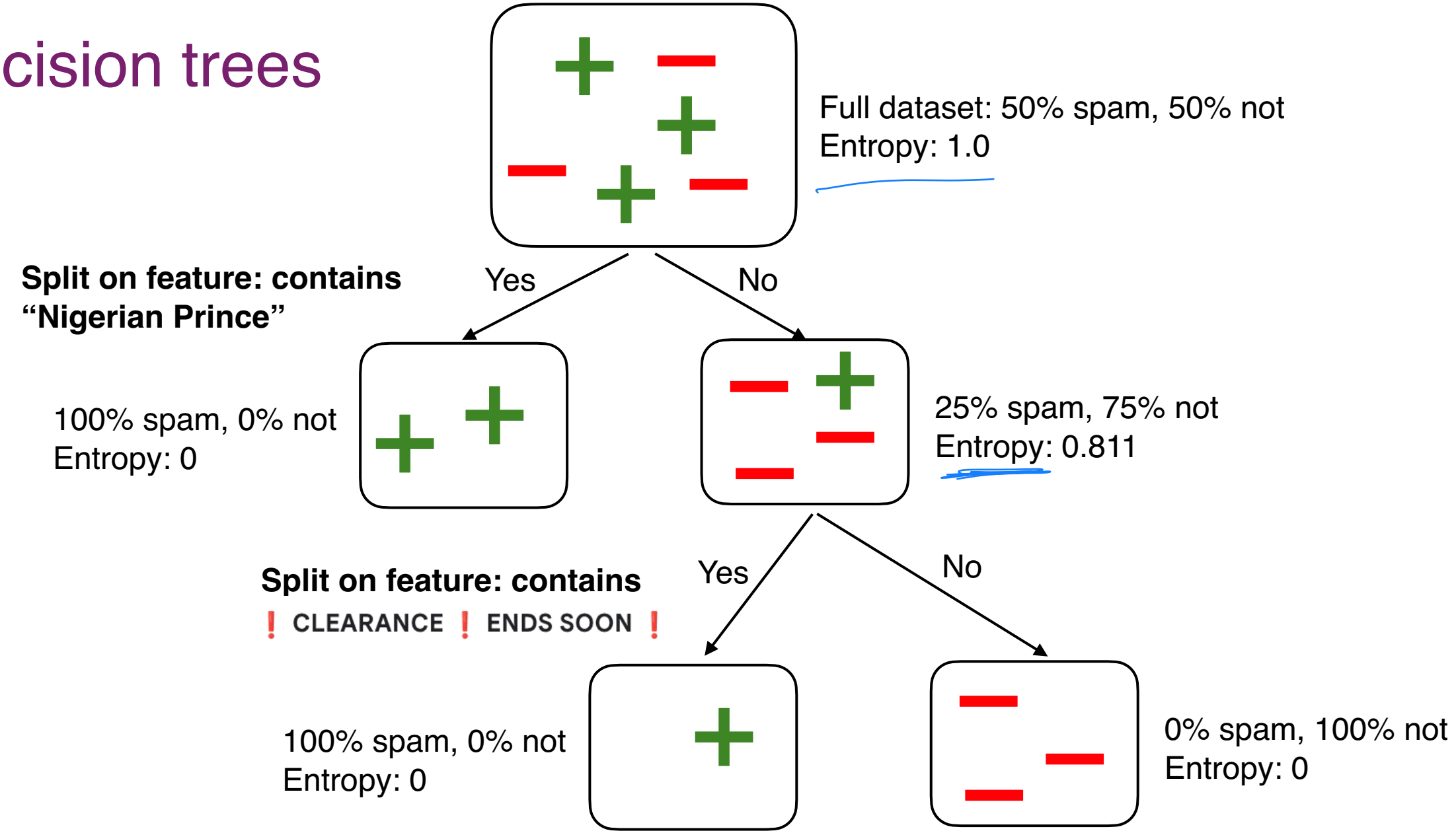
know well sorted are pos vs neg, eks.

find split that minimizes impurity.

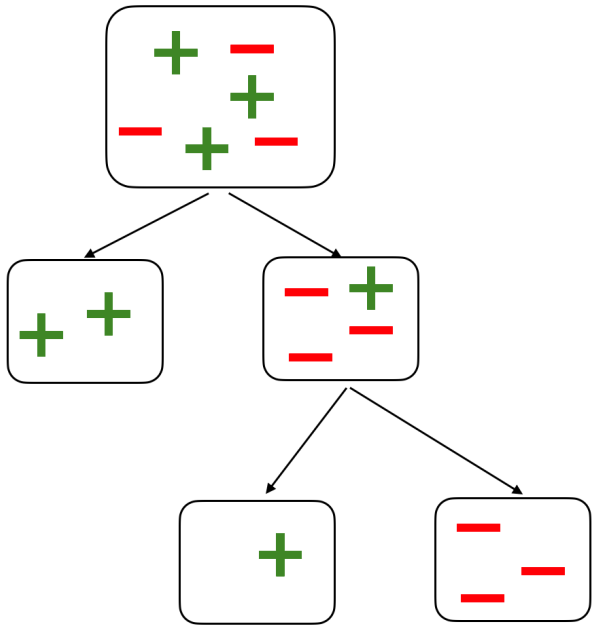




Decision trees



Splitting decision trees



impurity measures
- Gini index
- information gain

Entropy $H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$

★ → is always $\log_2$!

set $X = \{0,0,1\}$ $p_0 = \frac{2}{3}$ $p_1 = \frac{1}{3}$

$H = -\left(\frac{2}{3}\log\frac{2}{3} + \frac{1}{3}\log\frac{1}{3}\right) \simeq 0.918$

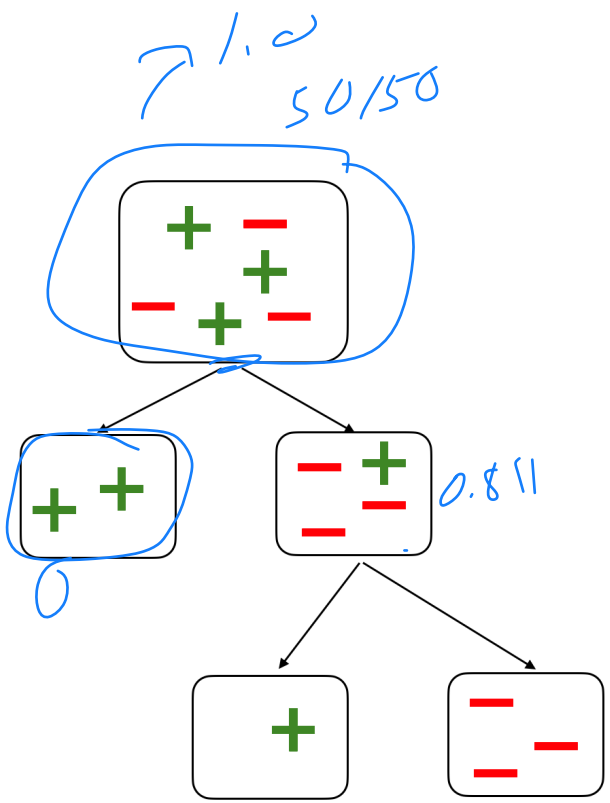
from 0 → 1

0 → perfectly sorted

1 → uniform distribution

label	
A	1.0
B	0.0
C	0.0
D	0.0

Splitting decision trees



Information gain:

$$I(X, a) = H(X) - H(X \mid a)$$

set, variable

Decision trees

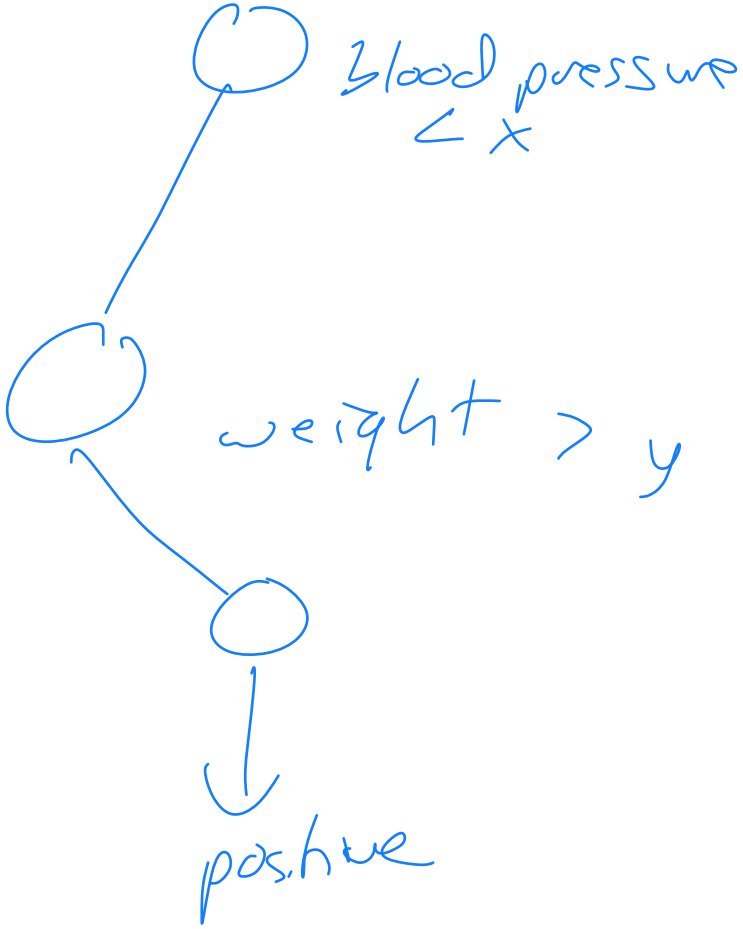
decision criteria for splitting based on a $\rightarrow$ $X_j = S_j$ split n points

$$= H(X) - \sum_{i=1}^{k} \frac{|R_i(j,s)|}{n} H(R_i(j,s))$$

Interpreting trees

Trees are “easy” to interpret:

- You can explain how the classifier came to the conclusion it did



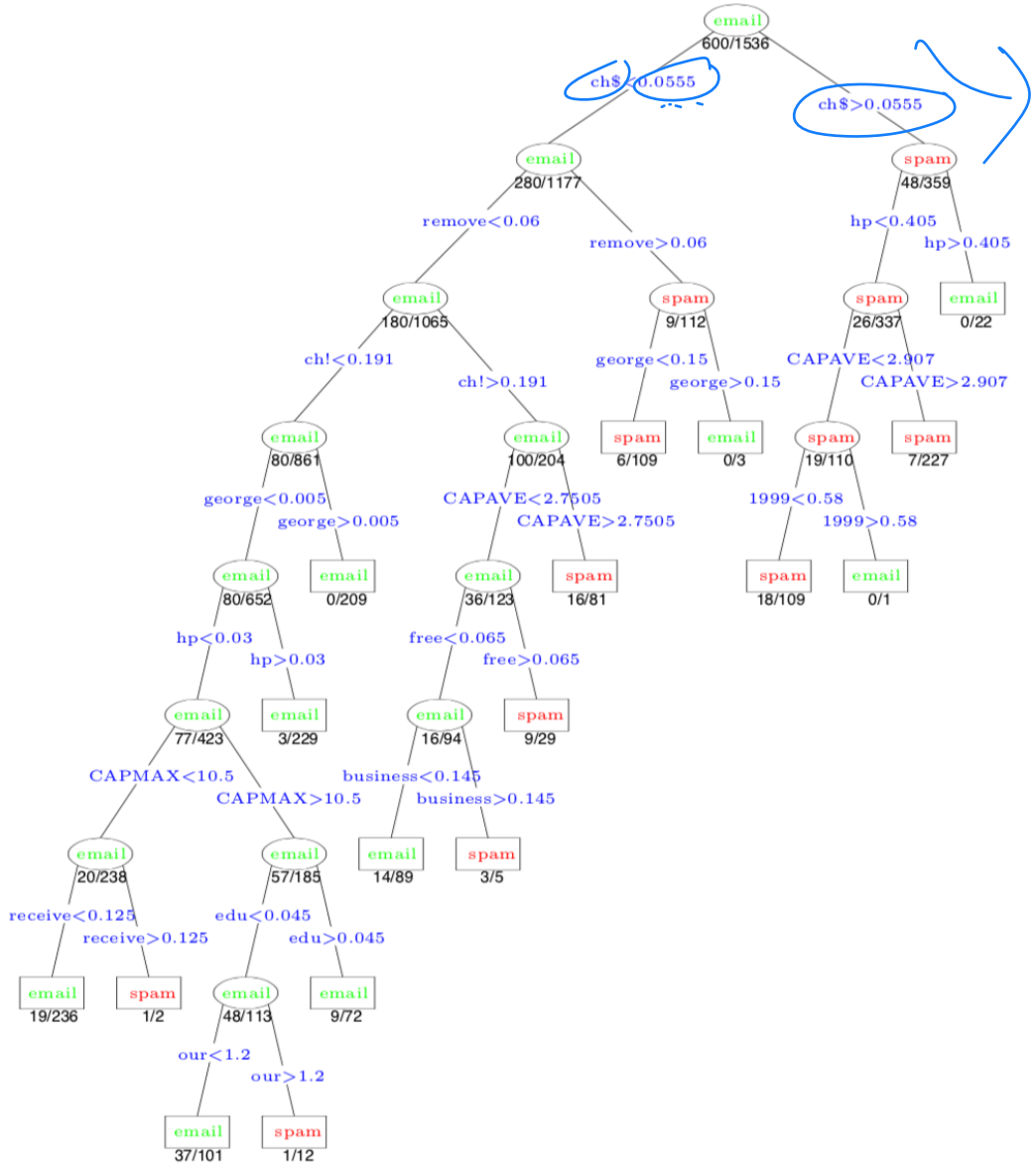
Interpreting trees

Trees are “easy” to interpret:

- You can explain how the classifier came to the conclusion it did

But they can be complex

- Small changes in data can result in large difference in trees



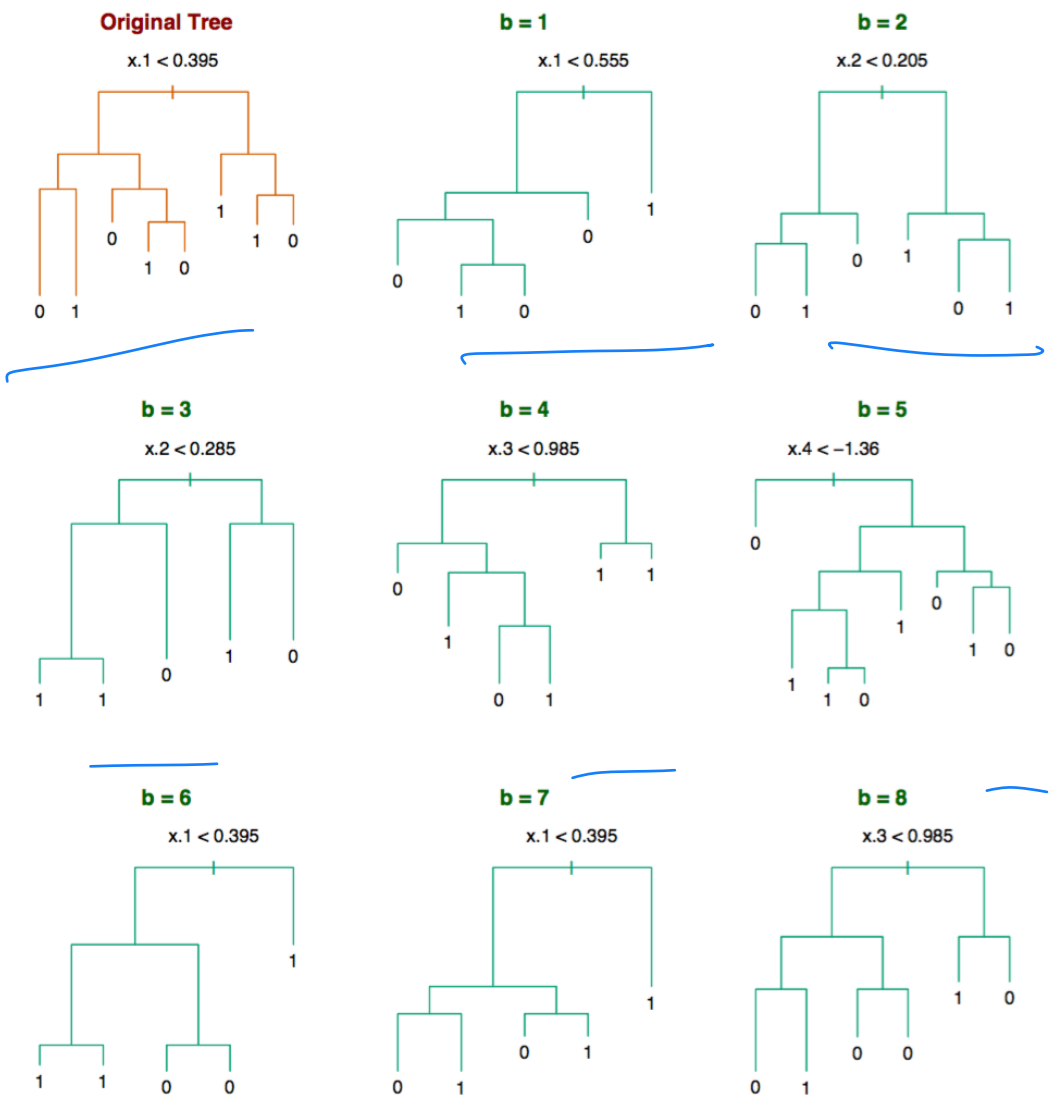
Summary so far

- Trees have ↓ bias, ↑ variance
- Deal with categorial variables well
- Intuitive, "interpretable"
- Good software exists
- Some theoretical guarantees

Random forests

- Forest = many trees
- Tree methods have low bias but high variance
- We can reduce variance by constructing many "lightly correlated" trees and averaging them

- Bagging: Bootstrap aggregating



Random forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

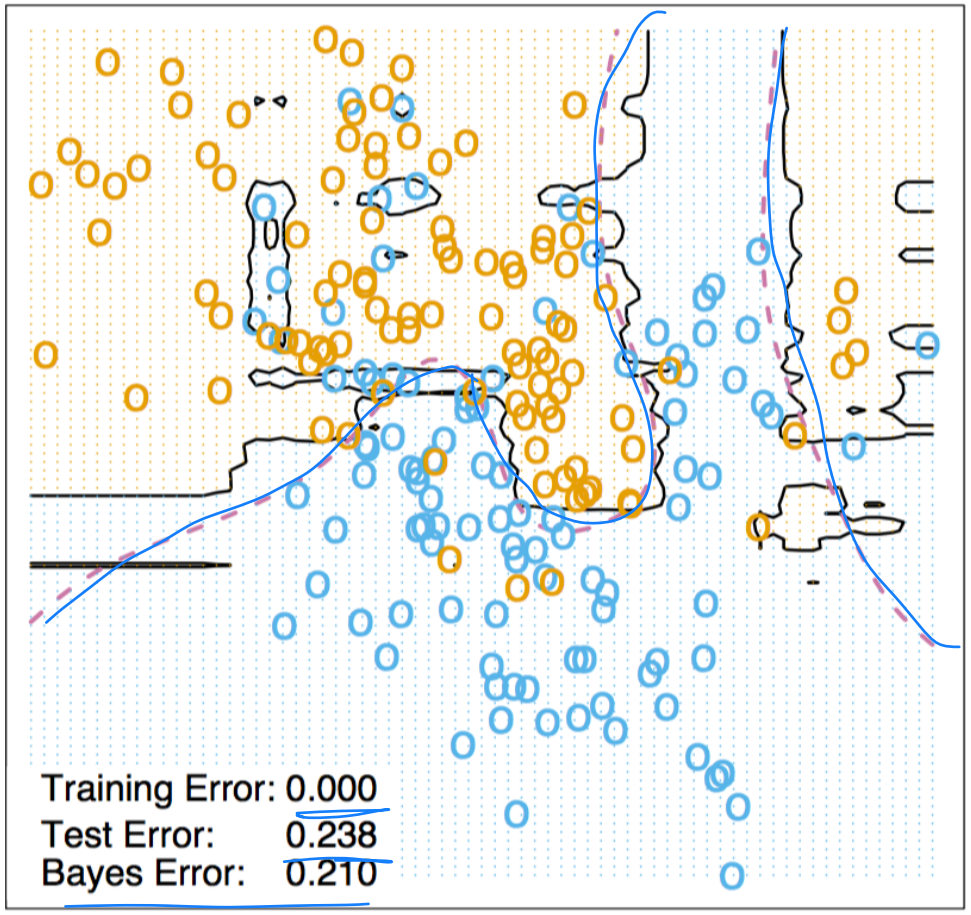
1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random forests: Decision boundary example



→ Bayes optimal

Random forests

↗ trees

Average of lightly correlated variables has lower variance

Given random variables $Y_1, Y_2, \ldots, Y_B$ with
$\mathbb{E}[Y_i] = y$, $\mathbb{E}[(Y_i - y)^2] = \sigma^2$, $\mathbb{E}[(Y_i - y)(Y_j - y)] = \rho\sigma^2$

correlation coefficient $0 < \rho \leq 1$ (no relationship ↓, duplicate ↓)

unbiased — variance — correlation b/w predictors

$$\mathbb{E}[(\frac{1}{B}\sum_{i=1}^{B} Y_i - y)^2] = E\left[\left(\frac{1}{B}\sum_{i=1}^{B}(Y_i - y)\right)^2\right]$$

B → # of trees

$$= E\left[\frac{1}{B^2}\sum_{i=1}^{B}(Y_i - y)^2\right] + E\left[\frac{1}{B^2}\sum_{i \neq j}(Y_i - y)(Y_j - y)\right]$$

$$= \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2$$

if $\rho = 1$: $\frac{\sigma^2}{B} + \frac{(B-1)}{B}\sigma^2$

$\frac{B}{B}\sigma^2$

if $\rho < 1$: can reduce variance by increasing B

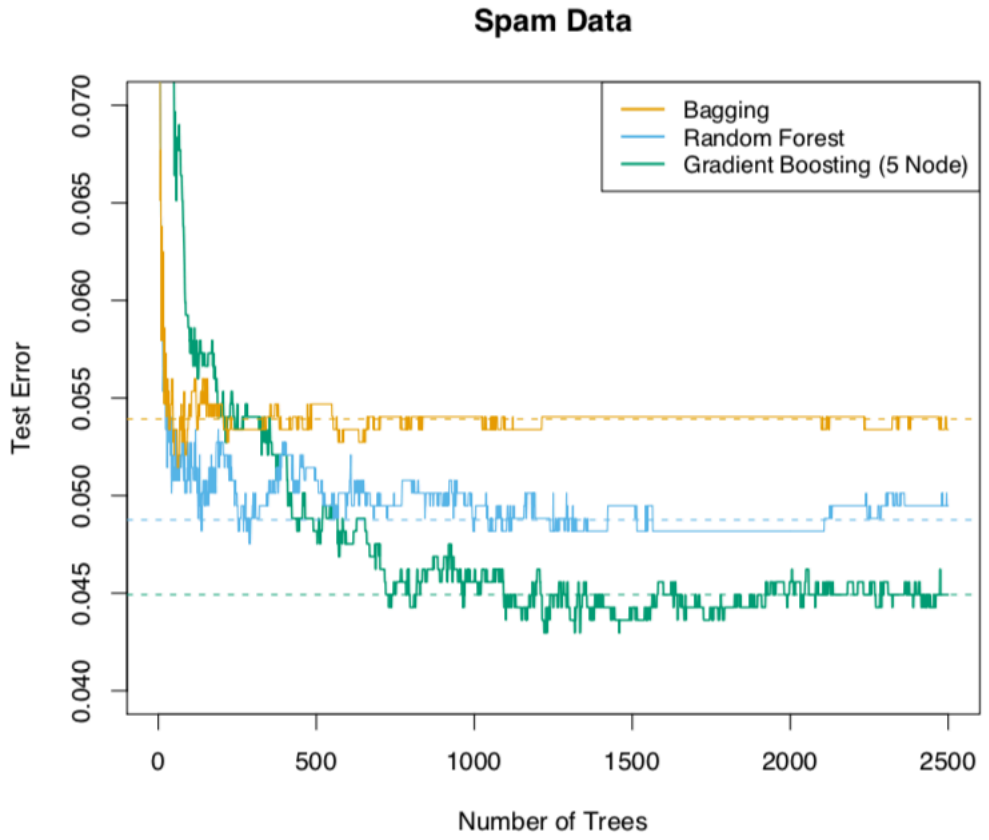
Ensembles are powerful

For binary classification, if you have N weak classifiers, but each one is slightly better than random chance (gets right answer with $p > 0.5$), what happens if you take the majority vote?

As $N \to \infty$, what is the probability that the majority vote gets the right answer?

- Marquis de Condorcet, "Essay on the Application of Analysis to the Probability of Majority Decisions" (1785). Known as the Condorcet Jury Theorem.
- Schapire, "**The Strength of Weak Learnability**" (1990)

The power of weakly correlated predictors



Bagging: Averaged trees on bootstrapped datasets using all d features

Random forest: Averaged trees on bootstrapped datasets using m randomly selected features

→ less correlated

Takeaways:

- Reducing correlation improves performance
- Ensembles are powerful

Summary so far

- Random forests have ↓ bias, ↓ variance
- Deal with categorial variables well
- Not that intuitive nor “interpretable”
- Gives some notion of confidence estimates
- Good software exists
- Some theoretical guarantees

Boosting and additive models

Instead of ensembling bootstrapped models, can we:

- Keep the idea of ensembling / combining simpler models, but
- Not necessarily have the models be identically distributed?

Key idea: Given a current collection of models, add a new model that focuses on what the previous models got wrong

Boosting

Additive models

classification

Given $\{(x_i, y_i)\}_{i=1}^n$ $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$

$b_m : \mathbb{R}^d \to \mathbb{R}$ $m = 1 \dots M$ // a model

$x \to y$

$\beta_m \in \mathbb{R}^m$ // weight ~~of a particular model~~ params

$$\hat{\beta}_1 \hat{b}_1 \dots \hat{\beta}_M \hat{b}_M = \underset{\dots}{\operatorname{argmin}} \sum_{i=1}^n l\left(y_i, \sum_{m=1}^M \beta_m b_m(x)\right)$$

// too hard!

Simultaneous opt. of so many params

Sequentially learn each new model (greedily)

one at a time.

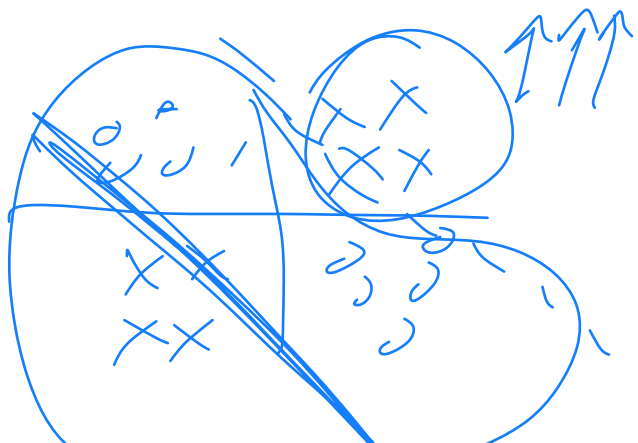
Forward stagewise additive models

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.



AdaBoost → binary classifiers

$b(x, r) : \mathbb{R}^d \to \{-1, 1\}$

$L(y, f(x)) = e^{-y f(x)}$

// upweight points the previous ensemble got wrong

↳ "boosts" the weight on mistakes

Forward stagewise additive models

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Boosted regression trees

$b(x, r)$ = regression tree

$L(y, f(x)) = (y - f(x))^2$

$L(y_i, f_{m-1}(x_i) + \beta b(x_i, r))$ = $(y_i - f_{m-1}(x_i) - \beta b(x_i, \gamma))^2$

previous ensemble — new model — residual

$(r_{im} - \beta b(x_i, \gamma))^2$

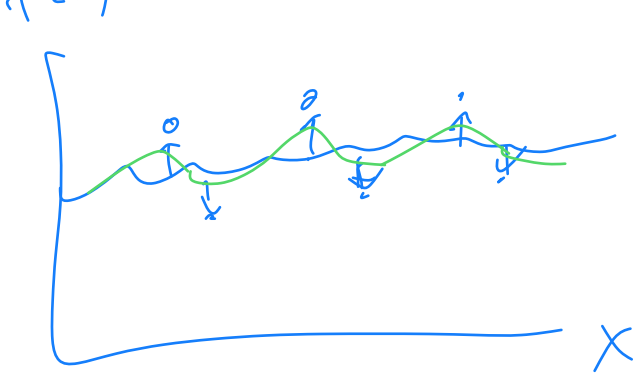
Gradient boosting

take gradient steps to minimize loss "in a function space"

$L(y_i, f(x_i)) = L(y_i, f_i) = (y_i - f_i)^2$

$\frac{\partial L(y_i, f_i)}{\partial f_i} = -2(y_i - f_i)$

$f_i := f + \beta b$ → step size, → gradient direction

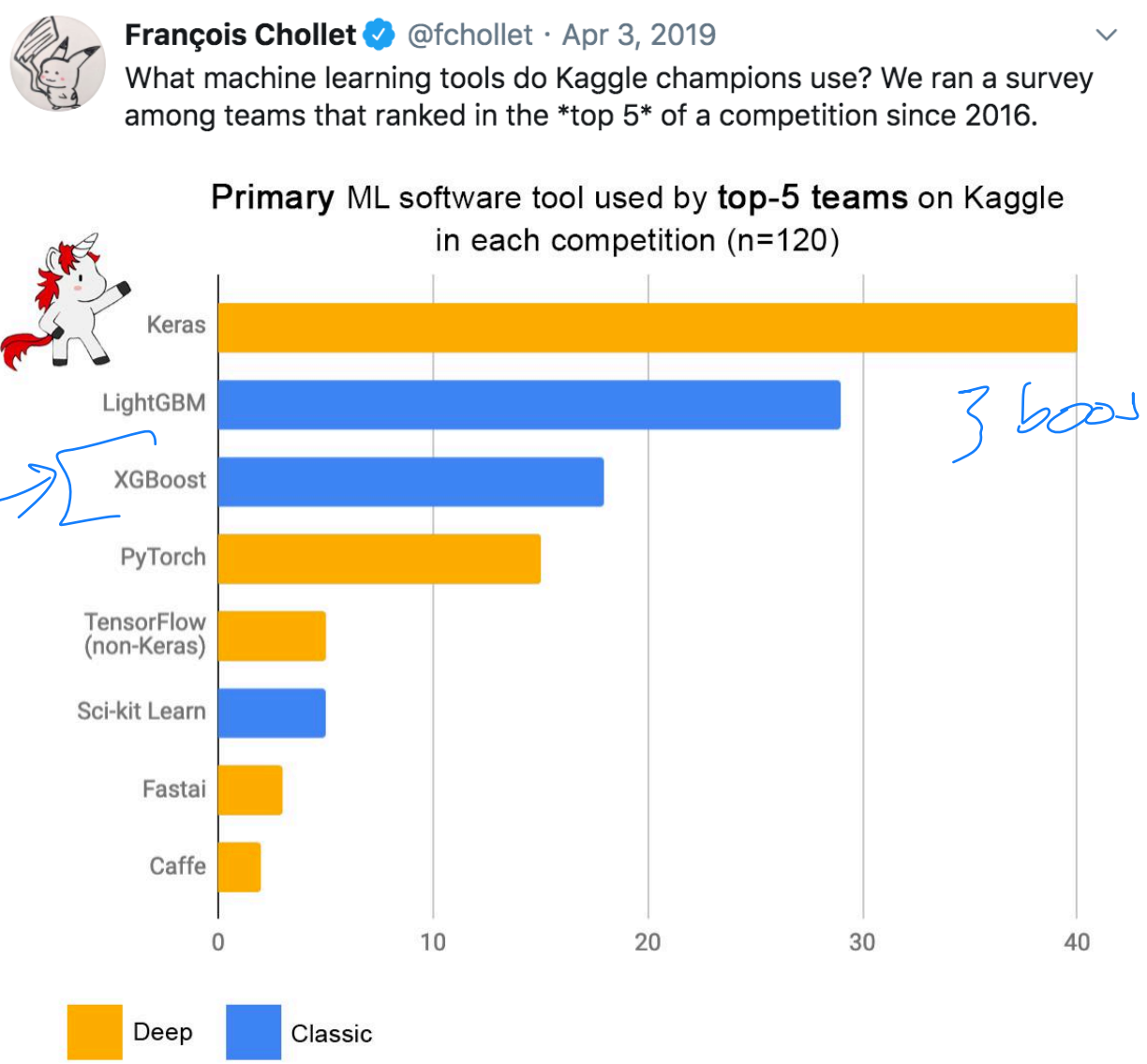


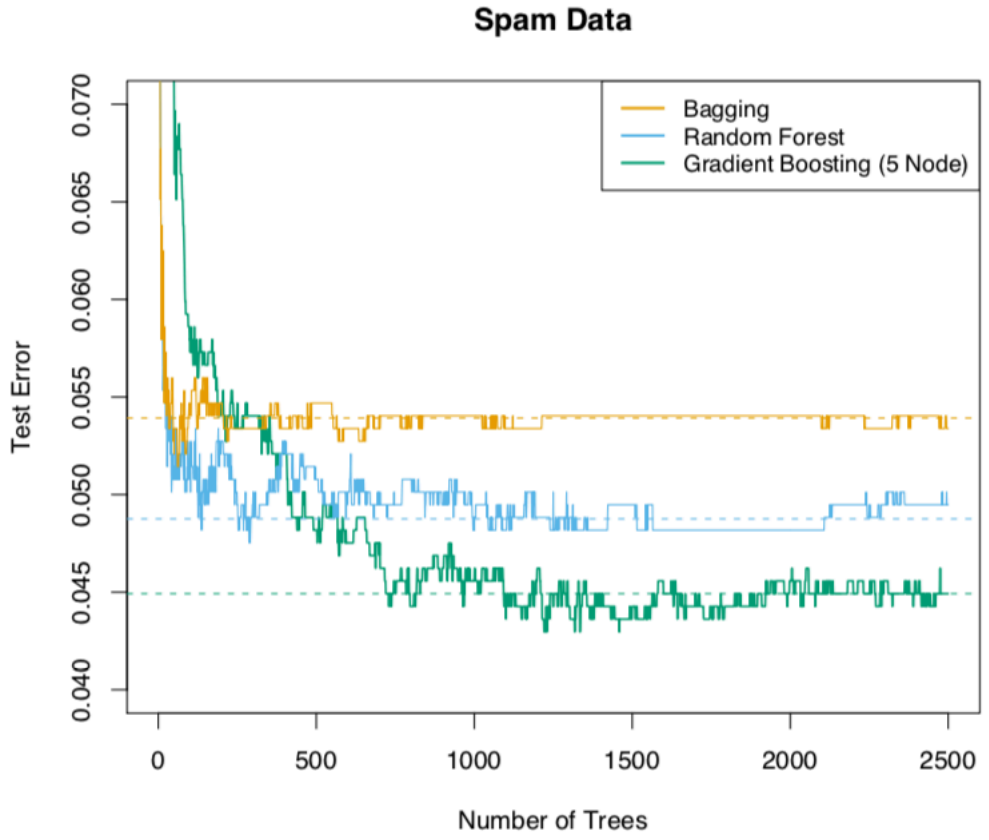
→ gradient of the func at the pts where you have data; move func towards those points

Gradient boosting generalizes boosting to arbitrary loss funcs

A brief history of boosting

- 1988 Kearns and Valiant: “Can weak learners be combined to create a strong learner?”
- 1990 Schapire: “Yup, in theory”
- 1995 Schapire and Freund: “Practical for 0/1 loss” -> AdaBoost
- 2001 Friedman: “Practical for arbitrary losses”
- 2014 Tianqi Chen: “Scale it up!” -> XGBoost


François Chollet @fchollet · Apr 3, 2019
What machine learning tools do Kaggle champions use? We ran a survey among teams that ranked in the *top 5* of a competition since 2016.
Primary ML software tool used by top-5 teams on Kaggle in each competition (n=120)
Keras
LightGBM
XGBoost
PyTorch
TensorFlow (non-Keras)
Sci-kit Learn
Fastai
Caffe
0
10
20
30
40
Deep
Classic



Bagging: Averaged trees on bootstrapped datasets using all d features

Random forest: Averaged trees on bootstrapped datasets using m randomly selected features

Boosting: Learned combinations of trees

Takeaways

- Single trees: low bias, high variance
- Ensembles: low bias, (relatively) low variance
- Bagging averages many lightly dependent models to reduce variance
  - Random forests: same but with random subset of features
- Boosting learns a linear combination of high bias, highly dependent classifiers to reduce error
- Gradient boosted trees are commonly used for categorical data