

An Introduction to Tree-based ML models

Timothy Daley

Tree models can be thought of as rule sets

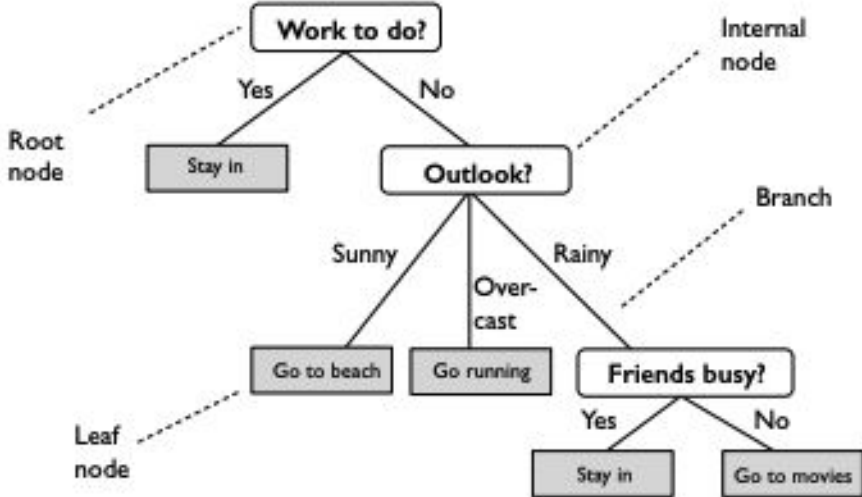


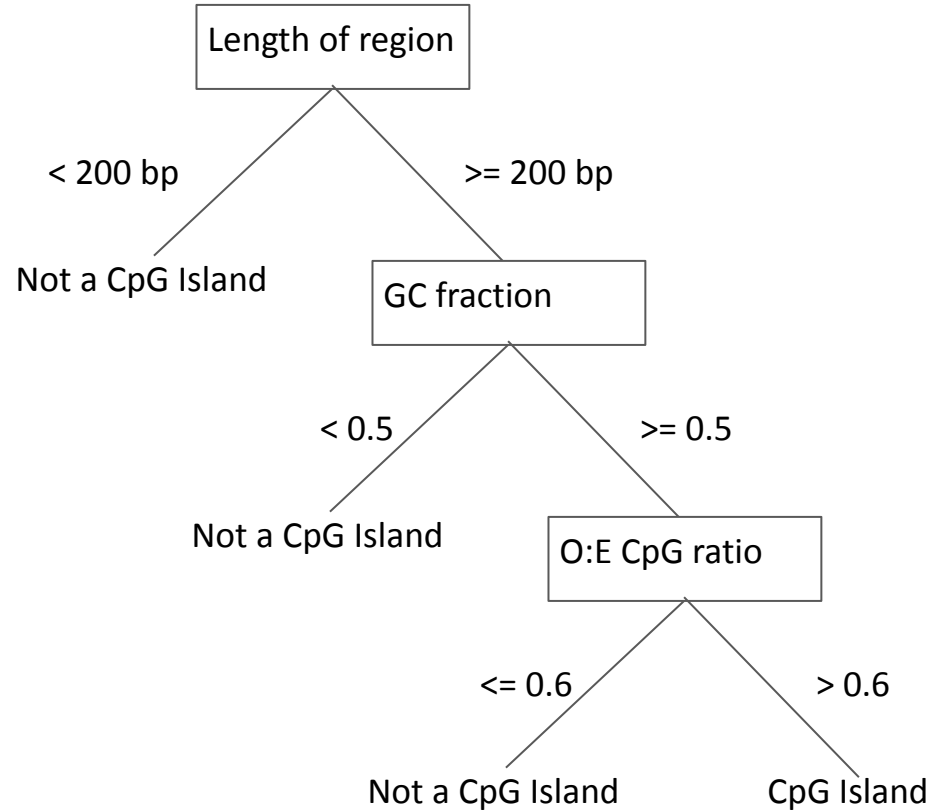
Figure 1: Example of a non-binary decision tree with categorical features.

Example: CpG Islands

Definition:

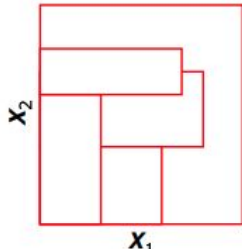
- region with at least 200 bp,
- GC percentage greater than 50%,
- an observed-to-expected CpG ratio greater than 60%.

https://en.wikipedia.org/wiki/CpG_site#CpG_islands

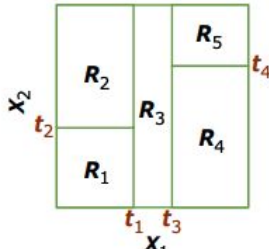


Tree models can be thought of as partitioning the feature space

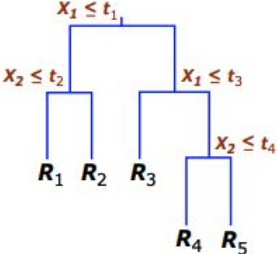
Partitions and CART



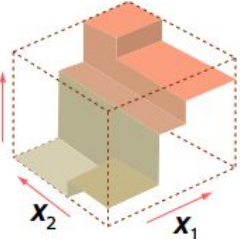
(a) General partition that cannot be obtained from recursive binary splitting.



(b) Partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data.



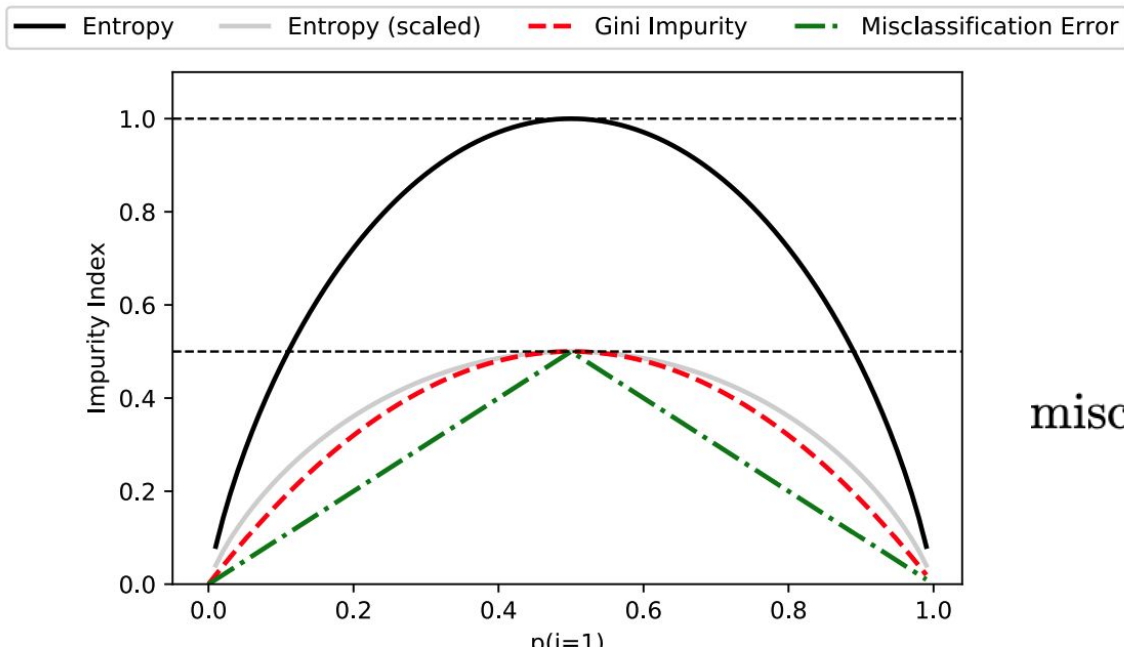
(c) Tree corresponding to the partition in the top right panel.



(d) A perspective plot of the prediction surface.

Image by MIT OpenCourseWare, adapted from Hastie et al., *The Elements of Statistical Learning*, Springer, 2009.

How to compute splits (classification)



Entropy or Gini impurity is used over classification accuracy because the former are differentiable (important for boosting) and tend to lead to better trees (example in ESL)

$$\text{misclassification error} : 1 - \hat{p}$$

$$\text{Gini index} : \sum \hat{p}(1 - \hat{p})$$

$$\text{cross-entropy} : - \sum \hat{p} \log \hat{p}$$

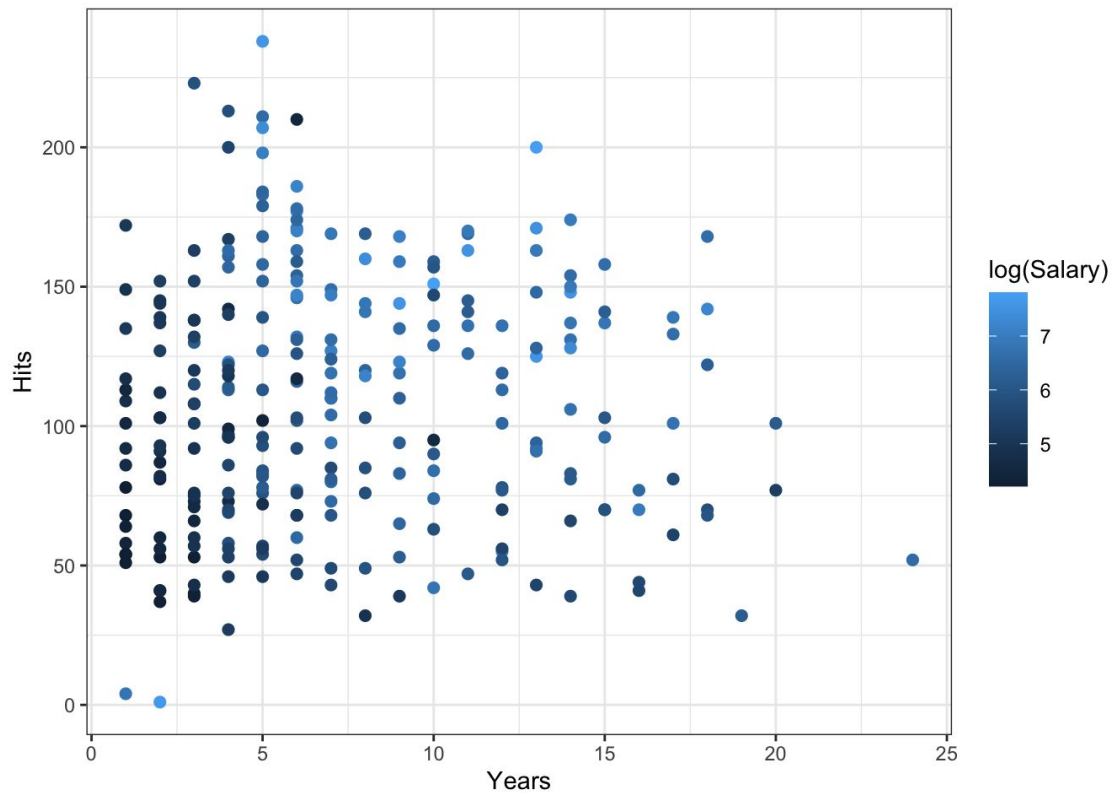
How to compute splits (regression)

Split by total within group variance

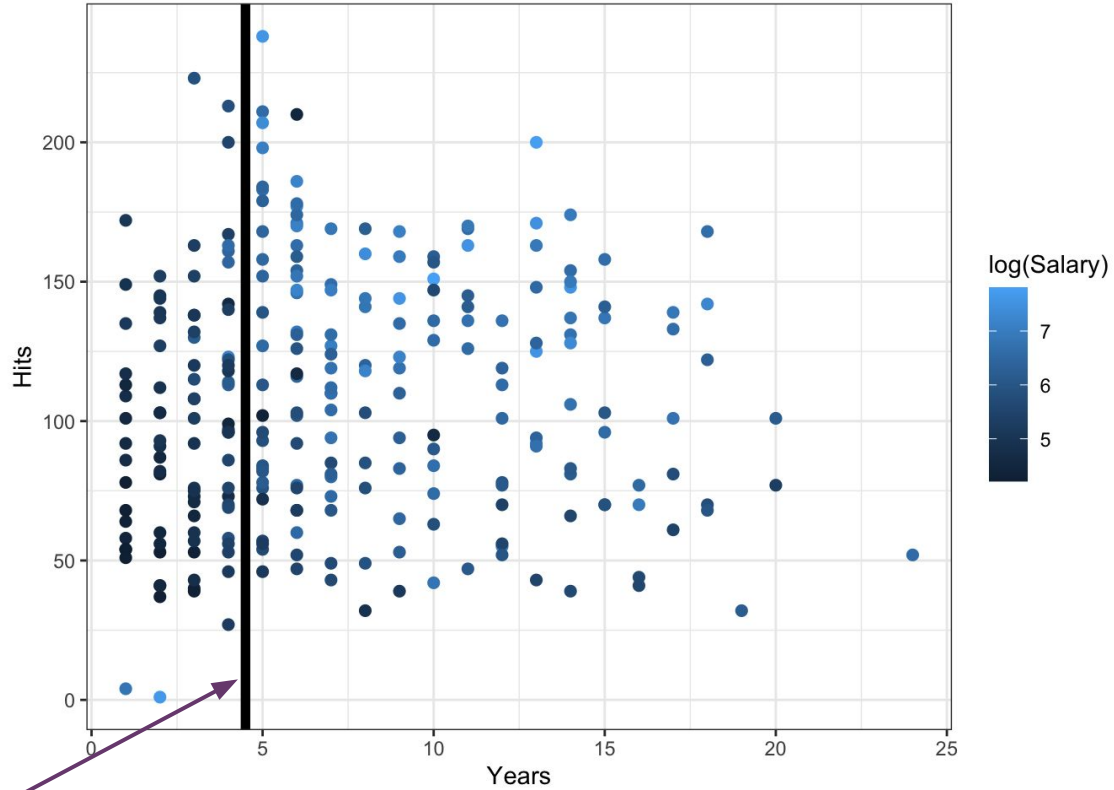
choose splitting point s s.t.

$$\arg \min_s \sum_{i:x_i < s} (y_i - \bar{y}_{x.<s})^2 + \sum_{i:x_i \geq s} (y_i - \bar{y}_{x.\geq s})^2$$

Example: Hitters

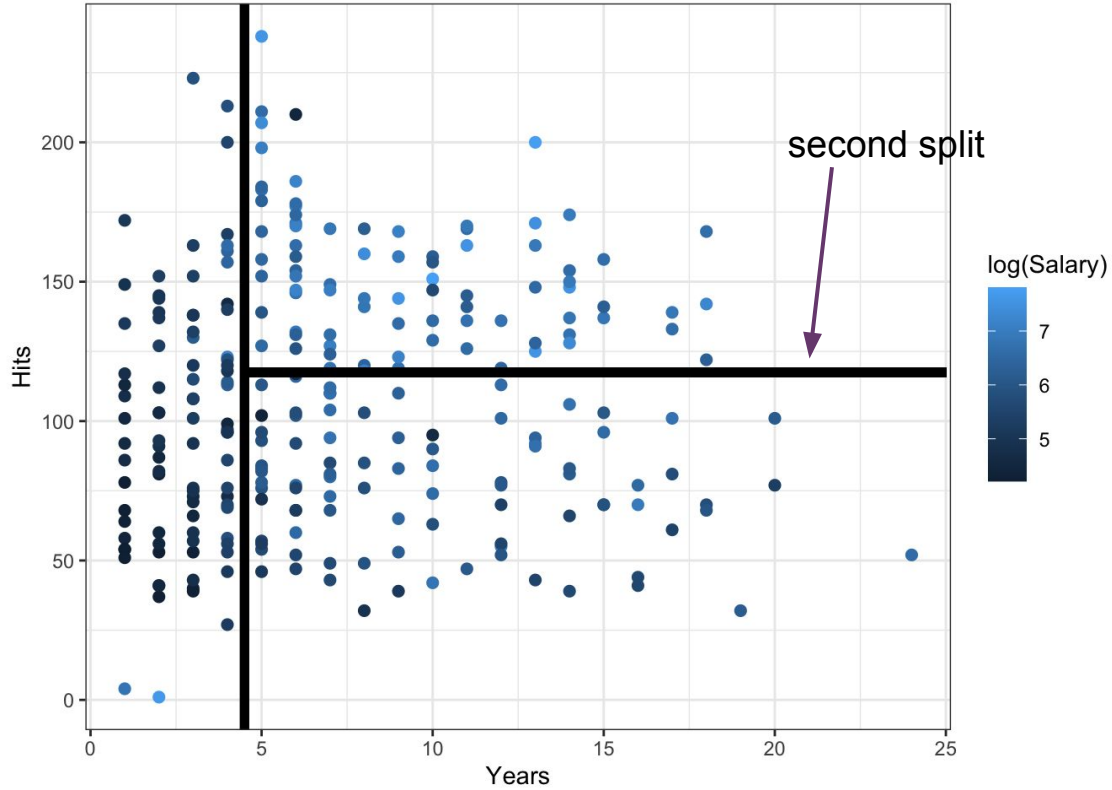


Example: Hitters

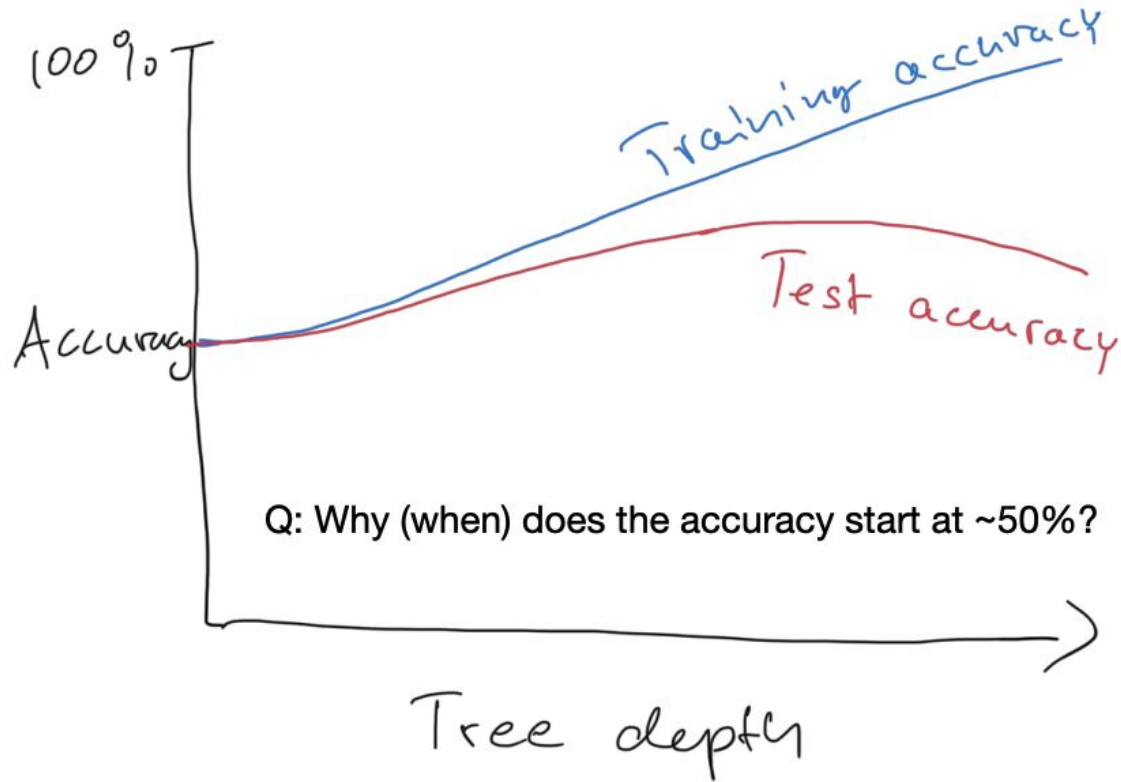


first split

Example: Hitters



How tall/deep should trees be?



Tree pruning

- Adding more branches can't decrease training performance
- 2 options:
 - Stopping criteria
 - Grow a big tree and trim
- Most implementations do the latter

Missing values

- Several ways to handle missing values:
 - Create an indicator variable of missing or not
 - Set missing values to a very large positive or negative number
 - Imputation

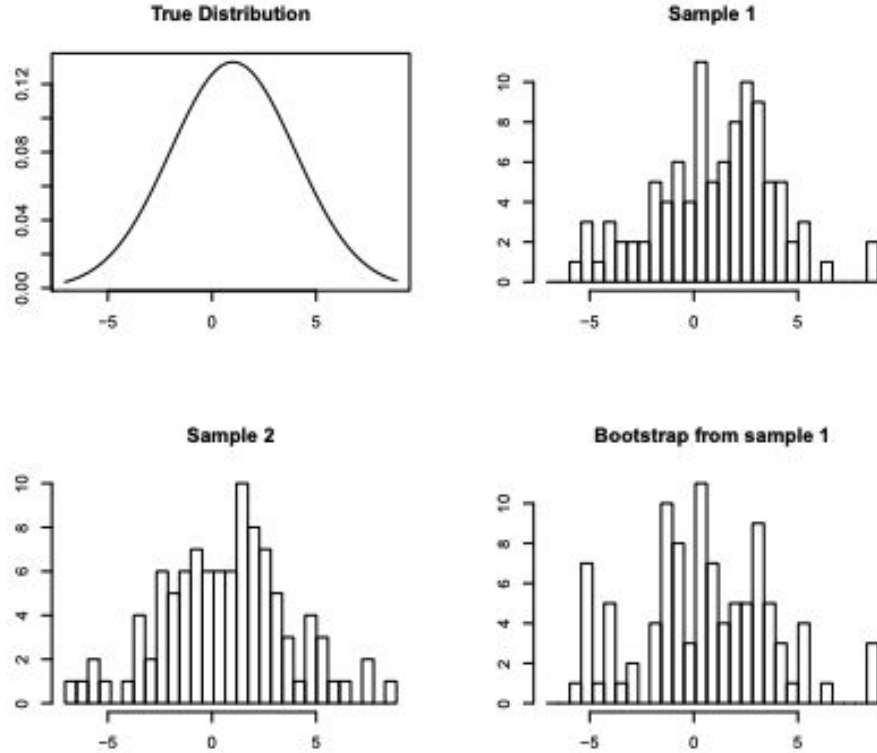
Missing values

- Several ways to handle missing values:
 - Create an indicator variable of missing or not
 - Set missing values to a very large positive or negative number
 - ~~○ Imputation~~
 - In practice most missing data is not missing at random, so imputation will introduce bias into the model.

Random forests

- Trees are “weak” learners
 - They don’t perform well individually
 - High variance
- Advantages:
 - Simple to interpret
 - Simple to calculate
 - Can capture non-linear behavior
- Bootstrap aggregating:
 - Sample the data, create a bunch tree, average the result
 - Wisdom of the crowd

Bootstrapping



Source: https://www.andrew.cmu.edu/user/achoulde/95791/lectures/lecture05/lecture05_95791.pdf

~63.2% of observations will appear in the bootstrapped sample

Random forests

- Want variability in features, don't want the same features every time
 - Subset features for each bootstrap
 - Works to de-correlate trees
- OOB: Out of bag performance
 - For a given observation take all bootstraps that didn't use that observation
 - Pseudo-test set
- Feature importance
 - Permute values of a feature, measure performance
 - Higher error means feature is more important

(Gradient) Boosting

- Idea:
 - Take a weak learner
 - Iteratively improve the learner

$$\hat{y}_i^{(0)} = b$$

$$\hat{y}_i^{(1)} = b + f_1(x_i)$$

$$\hat{y}_i^{(2)} = b + f_1(x_i) + f_2(x_i)$$

⋮

$$\hat{y}_i^{(t)} = \sum_{k=0}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Gradient Boosting

How to determine $f_t(x)$?

$$\text{Objective}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

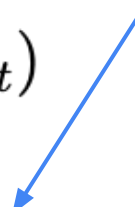
$$\begin{aligned} \text{Objective}^{(t)} \approx \sum_{i=1}^n & \left(l(y_i, \hat{y}_i^{(t-1)}) + \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) f_t(x_i) \right. \\ & \left. + \frac{1}{2} \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)}) f_t^2(x_i) \right) + \Omega(f_t) \end{aligned}$$

Gradient Boosting

How to determine $f_t(x)$?

$$\text{Objective}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

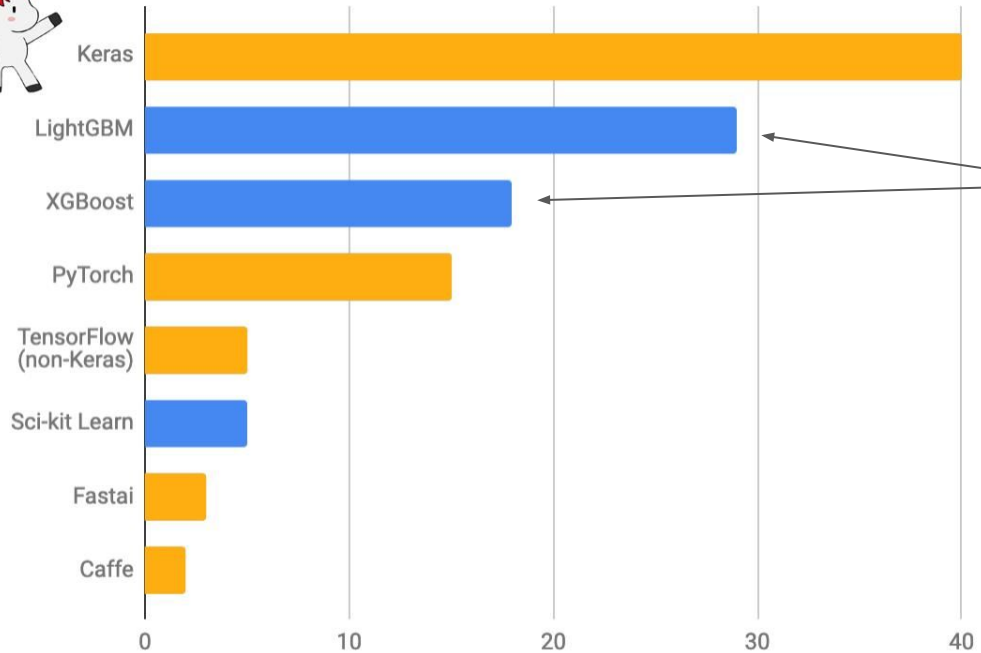
gradient!



$$\begin{aligned} \text{Objective}^{(t)} \approx \sum_{i=1}^n & \left(l(y_i, \hat{y}_i^{(t-1)}) + \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) f_t(x_i) \right. \\ & \left. + \frac{1}{2} \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)}) f_t^2(x_i) \right) + \Omega(f_t) \end{aligned}$$

Success of Gradient Boosting

Primary ML software tool used by **top-5 teams** on Kaggle in each competition (n=120)



Gradient boosted tree methods

Deep Classic

Application to real data: predicting outcome of cervical cancer biopsies

```
load('-/Downloads/cervical.RData')
head(cervical)
```

```
## Age Number.of.sexual.partners First.sexual.intercourse Num.of.pregnancies
## 1 18 4 15 1
## 2 15 1 14 1
## 3 34 1 15 1
## 4 52 5 16 4
## 5 46 3 21 4
## 6 42 3 23 2
## Smokes Smokes..years. Hormonal.Contraceptives Hormonal.Contraceptives..years.
## 1 0 0 0 0
## 2 0 0 0 0
## 3 0 0 0 0
## 4 1 37 1 3
## 5 0 0 1 15
## 6 0 0 0 0
## IUD IUD..years. STDs STDs..number. STDs..Number.of.diagnosis
## 1 0 0 0 0
## 2 0 0 0 0
## 3 0 0 0 0
## 4 0 0 0 0
## 5 0 0 0 0
## 6 0 0 0 0
## STDs..Time.since.first.diagnosis STDs..Time.since.last.diagnosis Biopsy
## 1 1 1 Healthy
## 2 1 1 Healthy
## 3 1 1 Healthy
## 4 1 1 Healthy
## 5 1 1 Healthy
## 6 1 1 Healthy
```

```
table(cervical$Biopsy)
```

```
##
## Cancer Healthy
## 55 803
```

Random forest model


```
cervical$Biopsy = factor(cervical$Biopsy, levels = c("Healthy", "Cancer"))
train_ind = rbinom(dim(cervical)[1], 1, 0.8)
cerv_train = cervical[which(train_ind == 1), ]
cerv_test = cervical[which(train_ind == 0), ]
cerv_rf = randomForest::randomForest(subset(cerv_train, select=-c(Biopsy)), y = cerv_train$Biopsy)
cerv_rf_pred = predict(cerv_rf, cerv_train)
table(data.frame(cerv_rf_pred, cerv_train$Biopsy))
```

```
##           cerv_train.Biopsy
## cerv_rf_pred Healthy Cancer
##   Healthy      647      28
##   Cancer         0      16
```

```
cerv_rf_pred = predict(cerv_rf, cerv_test)
table(data.frame(cerv_rf_pred, cerv_test$Biopsy))
```

```
##           cerv_test.Biopsy
## cerv_rf_pred Healthy Cancer
##   Healthy      156      11
##   Cancer         0         0
```

Oops! Everything is predicted as healthy
Class imbalance to blame!



Random forest model, upsampling minority class


```
cerv_rf = randomForest::randomForest(subset(cerv_train, select=-c(Biopsy)), y = cerv_train$Biopsy, classwt = c(1, 2))  
cerv_rf_pred = predict(cerv_rf, cerv_train)  
table(data.frame(cerv_rf_pred, cerv_train$Biopsy))
```

```
##           cerv_train.Biopsy  
## cerv_rf_pred Healthy Cancer  
##   Healthy     638      0  
##   Cancer       17     47
```

```
cerv_rf_pred = predict(cerv_rf, cerv_test)  
table(data.frame(cerv_rf_pred, cerv_test$Biopsy))
```

```
##           cerv_test.Biopsy  
## cerv_rf_pred Healthy Cancer  
##   Healthy     144      6  
##   Cancer        4      2
```

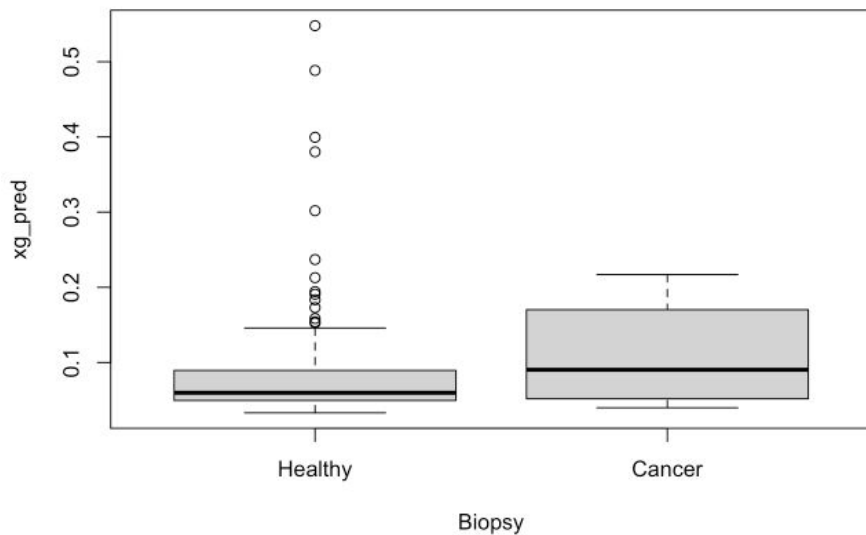
Better performance on predicting Cancer correctly, worse on predicting Healthy correctly



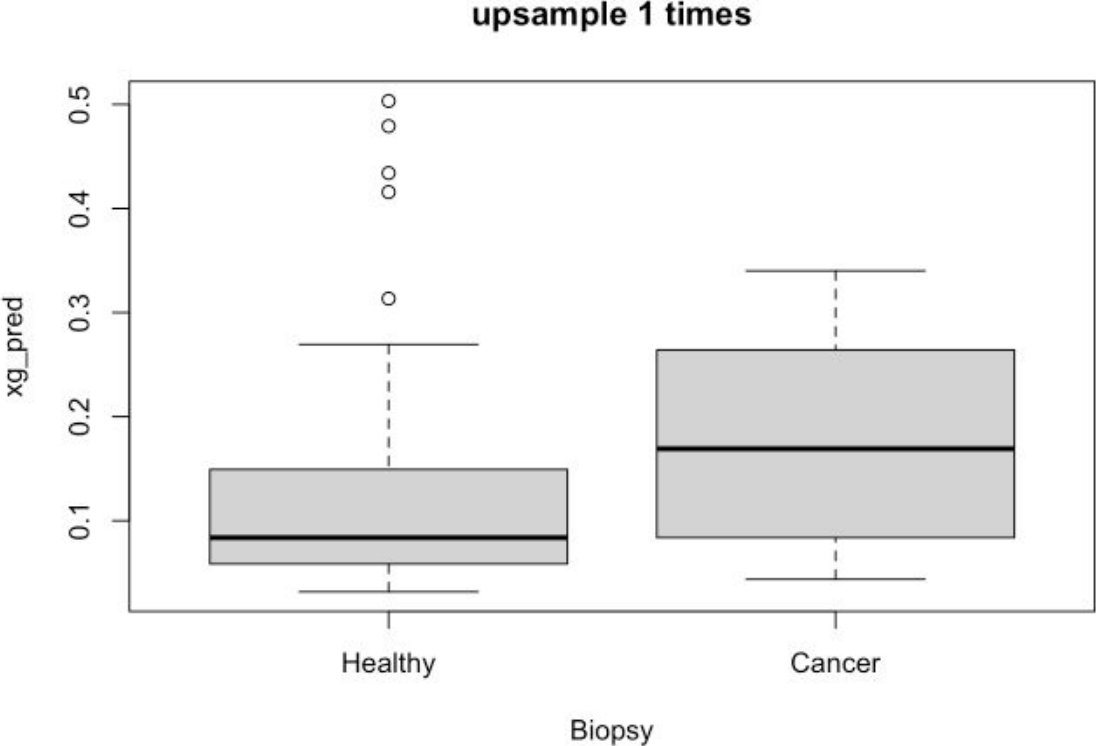
XgBoost model

```
cerv_xg = xgboost::xgboost(data = as.matrix(subset(cerv_train, select=-c(Biopsy))), label = as.numeric(cerv_train$Biopsy) - 1, objective = "binary:logistic", nrounds = 10)
```

```
cerv_xg_pred = predict(cerv_xg, as.matrix(subset(cerv_test, select=-c(Biopsy))))  
boxplot(xg_pred ~ Biopsy, data.frame(xg_pred = cerv_xg_pred, Biopsy = cerv_test$Biopsy))
```

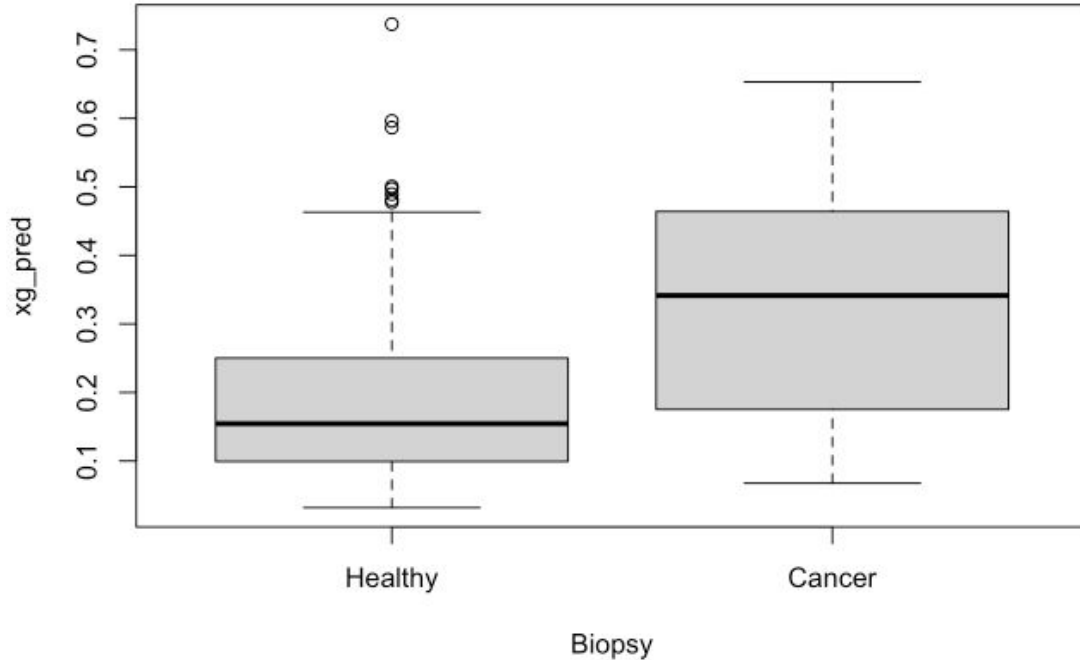


XgBoost model: upsample minority class



XgBoost model: upsample minority class

upsample 5 times



Upsampling the minority class:

- improves the prediction of the minority class
- worsens performance on the majority class
- loss of calibration of the model predictions

Ultimately, it's a question of trade-offs.

XgBoost model: upsample minority class

