Classification and Regression Trees
Motivation

Microsoft Kinect:  <link>

Higgs Boson Kaggle Challenge:  <link>
Overview

Trees are very powerful classification and regression models.

- **Fast to train** and **Fast to make predictions**
- (Generally) easy to interpret
- Efficient for **very high dimensional** feature spaces
- Efficient for **very large** amounts of training data
- Typically they don't work well individually
  - Overcome by ensemble methods (Random Forests / Boosting / etc)
What is the probability of people in this class going skiing?

Snowed this week?

Yes

Risk of avalanche

- < 20%: p = 0.9
- > 80%: p = 0.1

No

Ski station forecast

- Good snow: p = 0.7
- Not good snow: p = 0.1
Ski or not to ski?

Decision Tree

- **Snowed this week?**
  - Yes
    - Risk of avalanche:
      - < 20%: p = 0.9
      - > 80%: p = 0.6
    - No
  - No
    - Ski station forecast:
      - Good snow: p = 0.7
      - Not good snow: p = 0.1

- Split

- Leaf
Building trees

How do we build/construct a tree?

Typically top-down: *one split at a time, recursively.*

(also called greedy tree construction)
Simple example
Simple example

\[ x_2 > 8 \]
Simple example

\[ x_2 > 8 \]
\[ x_1 > 5 \]

Blue  Red
Simple example

Tree → Space Partitioning
Building trees

How do we build/construct a tree?

Typically top-down: one split at a time, recursively.

(also called greedy tree construction)

But how do we choose how to split the data?

\[ \{k^*, \tau^*\} = \arg\min_{k, \tau} I_{\text{split}}(D, k, \tau) \]

**Goal:** find a split \((k, \tau)\) that minimizes an impurity measure at the leaves

Element \(k\) of feature vector \(x\)

Threshold

\(x_k \geq \tau\)
Building trees

How do we build/construct a tree?

Typically top-down: *one split at a time, recursively.*

(also called greedy tree construction)

---

Learn split on training data $X$

**Input:** Training samples $X = \{(x_i, y_i)\}_{i=1}^N$

$x_i \in \mathbb{R}^D$

1: for $k = 1$ to $D$ do

2: Find best split for feature $k$: $\tau_k^* = \arg\min_{\tau} I_{\text{split}}(X, k, \tau)$

3: Compute cost of this split: $I_k = I_{\text{split}}(X, k, \tau^*)$

4: end for

5: return $k$ and $\tau_k$ that got the minimum impurity $I_{\text{split}}(\cdot)$
Building trees

Impurity Measures (Classification)

We are given $N$ training points $X = \{x_i, y_i\}_{i=1}^{N}$, $y_i \in \{0, 1\}$

1) Split training data

$L = \{x_i, y_i\}$

s.t. $x_{ik} > \tau$

$R = \{x_i, y_i\}$

s.t. $x_{ik} \leq \tau$

2) Leaf probabilities

$p_L = \frac{\#y_i=1}{N_L}$

$p_R = \frac{\#y_i=1}{N_R}$

3) Compute impurity

$I_{\text{split}} = N_L I(p_L) + N_R I(p_R)$
Building trees

Impurity measures for classification

\[ \text{Split impurity} = N_L I(p_L) + N_R I(p_R) \]

**Misclassification error:** \[ I(p) = 1 - \max(p, 1 - p) \]
Building trees

Impurity measures for classification

Split impurity: \[ N_L I(p_L) + N_R I(p_R) \]

Misclassification error: \[ I(p) = 1 - \max(p, 1 - p) \]

Cross-entropy: \[ I(p) = -p \log p - (1 - p) \log(1 - p) \]

Gini impurity: \[ I(p) = 2p(1 - p) \]

- Last two better behaved than misclassification (see HTF 9.2.3)
- Gini much faster to compute than cross-entropy (\( \log() \) is slow)
- Easy to extend to multi-class (see HTF 9.2.3)
Building trees

In summary

- Tree learned top-down, recursively
- To learn a split, minimize leaf impurity
  - Find best feature to split on
  - Find best threshold
- On a leaf: set its value to $p_L$ or $p_R$
Trees

Parameters?

Only one (though not easy to set):

When to stop creating more splits.

e.g.

- Stop when max tree depth reached.
- Stop when less than C number of samples available.
Good and bad points of trees

✔ Fast to train/predict, can handle very large feature spaces
✔ Easy to interpret. We can get feature importance [HTF 10.13]
✔ Can handle missing values [HTF 9.2.4]

✗ Instability: high variance [HTF 9.2.4]
✗ Lack of smoothness [HTF 9.2.4]
✗ Difficulty to capture additive structure [HTF 9.2.4]
✗ How to determine the maximum depth, or how to prune a tree
End