Decision Trees + Ensemble Methods

Decision Trees are good to use with ensembles. Tree ensembles often win many Kaggle competitions.

Decision Trees

- Non-linear (vs. SVMs)
- Ski example: linear model bad
- Split into regions. How?
- Picking Regions:
  - Greedy, Top-Down, Recursive Partitioning
  - 20 questions, draw out tree, branch vs. leaf
  - Formally, given node $p$ (parent node), covering region $R_p$, we can define a split $s_p$ on $j$-th feature with threshold $t$ as
    \[ s_p(j, t) = \{ x | x_j < t, x \in R_p \}, \{ x | x_j \geq t, x \in R_p \} \]

How to choose split?

- Intuitively, trying to isolate + and -
- Useful to define $L(R)$ on region
- For now define $L(R)$ as misclassification loss
  - Given $C$ total classes, define $\hat{P}_c$ as proportion of examples in $R$ that are of class $c$
    \[ L_{\text{misclass}} = 1 - \max_c (\hat{P}_c) \]
  - Loss of parent node is: $L(R_p)$
  - Loss of children: $L(R_1) + L(R_2)$
  - Want to maximize reduction in loss: $L(R_p) - (L(R_1) + L(R_2))$
  - $j$, $t$ for $s_p$ to do $\geq 0$
Misclassification Loss Has Problems

\[ R_p \downarrow 900/100 \quad R_p \downarrow 900/100 \]

Vs.

\[ R_1' \downarrow 400/100 \quad R_2' \downarrow 500/100 \]

\[ L(R_1) + L(R_2) = 100 \]

\[ L(R_1') + L(R_2') = 100 \]

In fact, \( L(R_p) = 100 \)

Instead, define cross-entropy loss

\[ L_{\text{cross}} = -\sum_c \hat{p}_c \log_2 \hat{p}_c \quad (\hat{p}_c \log_2 \hat{p}_c = 0 \text{ if } \hat{p}_c = 0) \]

Number of bits needed to specify outcome given distribution is known.

\[ \text{Cross-entropy} \]

Gini loss has similar shape \( L_{\text{gini}} = \sum_c p_c (1 - p_c) \)

Now, let's cover some extensions of DTs...

Regression Trees

Instead of predicting majority class, predict mean of values in \( R_m: \hat{y}_m = \frac{\sum_{i \in R_m} y_i}{|R_m|} \)

Minimize squared loss

\[ L_{\text{squared}} = \frac{\sum_{i \in R_m} (y_i - \hat{y}_m)^2}{|R_m|} \]

Ski example snowfall
Categorical Variables

North, South Hemisphere, Equator

Location
N +++ +++++
E +++++ ---
S +++++ ++

Loc {
E \ N} N

\% month < 3 ?

For q categories, 2^q splits possible so does not scale well, except for binary classification case.

Regularization

Decision trees are high variance, low bias. Use heuristics for regularization
1) Min leaf size
2) Max depth
3) Max # nodes
4) Min decrease in loss dangerous since might be higher b~ order interactions!
5) Pruning use validation set, measure misclassification or squared error

Runtime - Binary Classification
n examples, f features, d depth
Test time is \(O(d^n)\), if balanced \(O(\log n)\)
Train time, each point in \(O(d)\) nodes, costs \(O(f)\) at each node
So \(O(nfd)\) nf is size of design matrix and d often \(\log n\)
No additive structure

![Graph showing additive structure](image)

Linear models on the other hand are great at additive structure!

**Recap**
- Easy to explain
- Interpretable
- Categorical Var
- Fast

Will solve these issues via ... ensembling!

**Ensembling**

Take independent, identically distributed (i.i.d) random variables (RV) \( X_i \). Say \( \text{Var}(X_i) = \sigma^{-2} \)

\[
\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n} \sum X_i\right) = \frac{\sigma^{-2}}{n}
\]

Now drop independence assumption, so only i.d.
Say \( X_i \)'s are correlated by \( p \)

Then \( \text{Var}(\bar{X}) = p \sigma^{-2} + \frac{1-p}{n} \sigma^{-2} \) as \( n \to \infty \)

**Ways to ensemble**
1) Use different algorithms
2) Use different training sets
3) Bagging (RF)
4) Boosting (AdaBoost, xgboost)
Bagging

- Stands for bootstrap aggregation, variance reduction method
- Bootstrap-method from statistics to measure uncertainty
- Have true population P, which training set S was drawn from. Write \( S \sim P \).
- Ideally, have \( S_1, S_2, \ldots \) all drawn from \( P \).
- Now, assume \( S = P \).
- Can now draw new samples \( Z \) from \( S ! \) \( Z \sim S \) (sample w/ replacement, \( |Z| = |S| \) ) \( Z_1, Z_2, \ldots Z_M \)
- Train model \( G_m \) on \( Z_m \). Can then look at variability in predictions.
- However, can also define new aggregate predictor \( G_\ast = \frac{1}{M} \sum_{m=1}^{M} G_m(x) \)

Bias-Variance Analysis

- Recall \( \text{Var}(\bar{X}) = \frac{1}{M} \text{Var}(X) \) for \( M \) predictors.
- Bagging creates less correlated predictors, \( \delta \), thereby reducing Variance.
- Note that Bias increases a bit due to reduced sampling, but \( \frac{1}{M} \text{Var} \) typically outweighs \( \delta \) Bias.
- Note too that \( \frac{1}{M} \) can't hurt, since just decrease Var more.
Decision Trees + Bagging

Recall DT are high variance, low bias \( \rightarrow \) ideal fit for variance reduction of bagging!

Note bagging has another benefit for DT:
If a feature is missing, simply don't use trees in ensemble that contain that feature!
Missing feat \( \rightarrow \) Ignore \( GM \)'s splitting on it!

Out-of-bag estimation

Additional benefit of bagging: \( \frac{2}{3} \) free validation set.
On average, \( Z \) will contain \( \frac{2}{3} \) of \( S \). Use other \( \frac{1}{3} \) to estimate error, call OOB error. In limit as \( M \rightarrow \infty \) OOB gives equivalent results to LOOCV.

Variable Importance Measure
Do lose some interpretability.

For each feature, find each split that uses it in the ensemble. Measure decrease in loss, average.
Note doesn't measure degradation in perf if didn't have feature, since other features might be able to substitute.

Random Forest

Intuition: one very strong predictor \( \rightarrow \) correlated trees

Instead, at each split only allow subset of features to be used (ex: \( u = \sqrt{F} \)). Decreases Var, slight increase in Bias.

Again \( \text{Var}(\bar{x}) = \sigma^2 + \frac{1-P}{M} \sigma^2 \) as \( \frac{1}{M} \rightarrow 0 \) \( \text{Var} \)
(assuming \( M \) large)

Just what RF does! Works even better for missing values...
Recap - Bagging

- \( \downarrow \text{Var} \) (even more so for RF)
- \( \uparrow \text{bias} \) (even more so for RF)
- Better accuracy
- Deal with missing values
- Harder to interpret
- Still not additive
- More expensive

Boosting

- Bagging was variance reducing, boosting bias reducing

Since reducing bias, want high bias, low variance models, weak learners

In terms of DT, just use decision stump

Intuition via Example

\[ \begin{align*}
  x_1 & \quad x_2 \\
  \times & \quad \times \\
  \times & \quad \times \\
  0 & \quad 0 \\
  \times & \quad \times \\
  0 & \quad 0 \\
  \times & \quad \times
\end{align*} \]

Boost weight of negatives

AdaBoost

1. Training data \((x_i, y_i) \ldots (x_N, y_N)\)
2. Set \( w_i = \frac{1}{N} \) for \( i = 1 \ldots N \)
3. For \( m = 1 \ldots M \):
   a. Fit weak classifier \( G_m \) to weighted training data
   b. Compute weighted error \( \epsilon_m = \frac{\sum w_i I(y_i \neq G_m(x_i))}{\sum w_i} \)
   c. Compute weight \( \alpha_m = \log \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \)
   d. Set \( w_i = w_i \cdot \exp \left[ -\alpha_m I(y_i \neq G_m(x_i)) \right] \)
4. Return \( f_\theta(x) = \text{sign} \left( \sum_m \alpha_m G_m \right) \)
An Adaboost — cont'd

Additive — combine together prediction of many weak models. Often times beats single strong model. No longer independent of previous models in sequence. L∞ can overfit.

More general framework: Forward Stagewise Additive Modeling (FSAM)

Though derived on its own, Adaboost can be thought of as special case of FSAM. Still want to create classifier

f

1. Initialize f0(x) = 0
2. For m = 1, ..., M
   a) Compute (βm, ym) = arg min β,y β,y \sum L(yi; f m-1(x) + β \theta m b(x; y))
   b) Set fm(x) = f m-1 + βm b(x; y m)
3. Return f(x) = fm(x)

Adaboost is FSAM with exponential loss and 2-class classification

L(y, f(x)) = \exp(-y f(x)) . Proof in ESL.

For squared loss for regression, FSAM is same as fitting individual tree to residual y - f m-1(x)

Gradient Boosting

For more general losses, can't always write out nice closed-form solution to minimization problem. Turn to numerical optimization. One way: Take derivative, do gradient descent. But restricted to taking steps that are in model class.
Compute gradient at each training point \( i \): wrt current predictor \( f \)

\[
 g_i = \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}
\]

New predictor simply tries to train \( \text{regression tree} \)

to match this gradient. In FSAM:

\[
 y = \arg\min_{\gamma} \frac{1}{N} \sum_{i=1}^{N} (g_i - b(x_i; \gamma))^2
\]

Add in, is your gradient step:

\[ \gamma \]

Recap - Boosting

\[ \text{\# bias} \quad \text{\#1 var} \quad \text{\# Prone to overfitting} \]

\[ \text{\# Very good accuracy} \]