Tree Ensembles

Jihoon Yang

Machine Learning Research Laboratory
Department of Computer Science & Engineering
Sogang University

Popular Algorithms

1. Gradient Boosting Machines (GBM)
2. Random Forests

GBM (J. Friedman, 2001)

- An additive regression model over an ensemble of trees, fitted to current residuals, gradients of the loss function, in a forward step-wise manner (GBRT(Regression Tree)/GBDT(Decision Tree))
- **Gradient Boosting = Gradient Descent + Boosting**

What is Gradient Boosting (from C. Li's notes)

Gradient Boosting

- Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, "shortcomings" are identified by gradients.
- Recall that, in Adaboost, "shortcomings" are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

Gradient Boosting for Regression (from C. Li's notes)

Let's play a game...
You are given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize square loss.
Suppose your friend wants to help you and gives you a model F.
You check his model and find the model is good but not perfect.
There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3$... How can you improve this model?
Rule of the game:

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be $F(x) + h(x)$.

Gradient Boosting for Regression (from C. Li's notes)

Simple solution:

You wish to improve the model such that

$$
\begin{aligned}
&F(x_1) + h(x_1) = y_1 \\
&F(x_2) + h(x_2) = y_2 \\
&\dots \\
&F(x_n) + h(x_n) = y_n
\end{aligned}
$$

Gradient Boosting for Regression (from C. Li's notes)

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$...$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression tree h achieve this goal perfectly?
Maybe not....
But some regression tree might be able to do this approximately.
How?
Just fit a regression tree h to data
$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), ..., (x_n, y_n - F(x_n))$
Congratulations, you get a better model!

Gradient Boosting for Regression (from C. Li's notes)

Simple solution:

$y_i - F(x_i)$ are called **residuals**. These are the parts that existing model F cannot do well.

The role of h is to compensate the shortcoming of existing model F.

If the new model $F + h$ is still not satisfactory, we can add another regression tree...

We are improving the predictions of training data, is the procedure also useful for test data?

Yes! Because we are building a model, and the model can be applied to test data as well.

How is this related to gradient descent?

Gradient Boosting for Regression (from C. Li's notes)

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

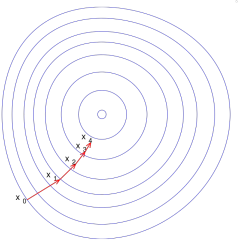


Figure: Gradient Descent. Source: `http://en.wikipedia.org/wiki/Gradient_descent`

Gradient Boosting for Regression (from C. Li's notes)

How is this related to gradient descent?

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

Gradient Boosting for Regression (from C. Li's notes)

How is this related to gradient descent?

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

Gradient Boosting for Regression (from C. Li's notes)

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**! It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**. So from now on, let's stick with gradients. The reason will be explained later.

Gradient Boosting for Regression (from C. Li's notes)

Regression with square Loss

Let us summarize the algorithm we just derived using the concept of gradients. Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
calculate negative gradients $-g(x_i)$
fit a regression tree h to negative gradients $-g(x_i)$
$F := F + \rho h$, where $\rho = 1$

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

Gradient Boosting for Regression (from C. Li's notes)

Loss Functions for Regression Problem

Square loss is:

- ✓ Easy to deal with mathematically
- × Not robust to outliers

  Outliers are heavily punished because the error is squared.

  Example:

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

  Consequence?

  Pay too much attention to outliers. Try hard to incorporate outliers into the model. Degrade the overall performance.

Gradient Boosting for Regression (from C. Li's notes)

Loss Functions for Regression Problem

- Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss($\delta = 0.5$)	0.005	0.02	0.125	1.525

Gradient Boosting for Regression (from C. Li's notes)

Regression with loss function L: general procedure

Give any differentiable loss function L

start with an initial model, say $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
- calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$
- fit a regression tree h to negative gradients $-g(x_i)$
- $F := F + \rho h$

In general,

$$\textit{negative gradients} \not\Leftrightarrow \textit{residuals}$$

We should follow negative gradients rather than residuals. Why?

Gradient Boosting for Regression (from C. Li's notes)

Negative Gradient vs Residual: An Example

Huber loss

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

Update by Negative Gradient:

$$h(x_i) = -g(x_i) = \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases}$$

Update by Residual:

$$h(x_i) = y_i - F(x_i)$$

Difference: negative gradient pays less attention to outliers.

Gradient Boosting for Regression (from C. Li's notes)

Summary of the Section

- Fit an additive model $F = \sum_t \rho_t h_t$ in a forward stage-wise manner.
- In each stage, introduce a new regression tree h to compensate the shortcomings of existing model.
- The "shortcomings" are identified by negative gradients.
- For any loss function, we can derive a gradient boosting algorithm.
- Absolute loss and Huber loss are more robust to outliers than square loss.

Things not covered

How to choose a proper learning rate for each gradient boosting algorithm. See [Friedman, 2001]
cf. Line search for ρ

GBRT

ALGORITHM 3 (LAD_TreeBoost).

$F_0(\mathbf{x}) = median\{y_i\}_1^N$

For $m = 1$ to M do:

$\tilde{y}_i = sign(y_i - F_{m-1}(\mathbf{x}_i)),\ i = 1, N$

$\{R_{jm}\}_1^J = J$-terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$

$\gamma_{jm} = \text{median}_{\mathbf{x}_i \in R_{jm}}\{y_i - F_{m-1}(\mathbf{x}_i)\},\ j = 1, J$

$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^J \gamma_{jm} 1(\mathbf{x} \in R_{jm})$

endFor

end Algorithm

Additive Training (from T. Chen's notes)

- How do we decide which f to add?
  - Optimize the objective!!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant \end{aligned}$$

Goal: find f_t to minimize this

- Consider square loss

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\ &= \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const \end{aligned}$$

This is usually called residual from previous round

Taylor Expansion Approximation of Loss (from T. Chen's notes)

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

- *If you are not comfortable with this, think of square loss*

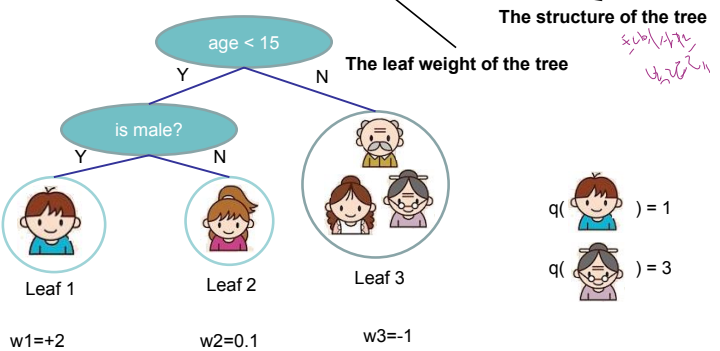
$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2$$

- Compare what we get to previous slide

Refine the Definition of Tree (from T. Chen's notes)

- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

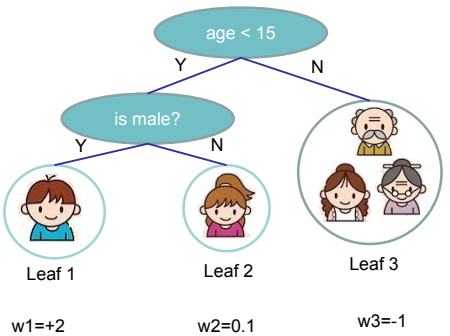


Define Complexity of a Tree (from T. Chen's notes)

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Our New Goal (from T. Chen's notes)

- Objective, with constants removed

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
  - Regroup the objective by leaf

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)} \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

  - This is sum of T independent quadratic function

The Structure Score (from T. Chen's notes)

- Two facts about single variable quadratic function

$$argmin_x \; Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}, \; H > 0 \qquad \min_x \; Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- Let us define $G_j = \sum_{i \in I_j} g_i \;\; H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

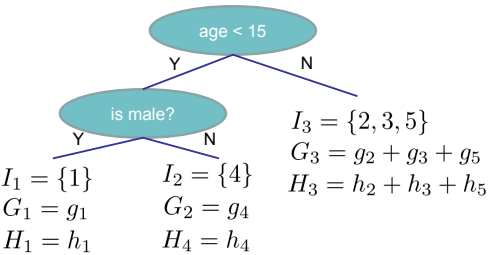
- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

This measures how good a tree structure is!

The Structure Score Calculation (from T. Chen's notes)

Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5



$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree (from T. Chen's notes)

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

- But… there can be infinite possible tree structures..

Greedy Learning of the Tree (from T. Chen's notes)

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

the score of left child

the score of right child

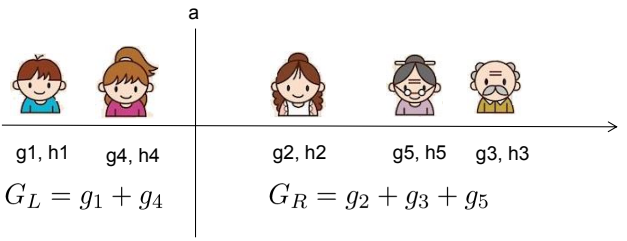
the score of if we do not split

The complexity cost by introducing additional leaf

  - Remaining question: how do we find the best split?

Efficient Finding of the Best Split (from T. Chen's notes)

- What is the gain of a split rule $x_j < a$? Say x_j is age



- All we need is sum of g and h in each side, and calculate

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

An Algorithm for Split Finding (from T. Chen's notes)

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features

- Time Complexity growing a tree of depth K
  - It is O(n d K log n): or each level, need O(n log n) time to sort There are d features, and we need to do it for K level
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

What about Categorical Variables? (from T. Chen's notes)

- Some tree learning algorithm handles categorical variable and continuous variable separately
  - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle categorical separately.
  - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a #categorical length vector

$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & otherwise \end{cases}$$

  - The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data

Pruning and Regularization (from T. Chen's notes)

- Recall the gain of split, it can be negative!

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

  - When **the training loss reduction** is smaller than **regularization**
  - Trade-off between simplicity and predictivness

- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits..

- Post-Prunning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Recap: Boosted Tree Algorithm (from T. Chen's notes)

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

Improvements

1. eXtreme Gradient Boosting (XGBoost; Chen & Guestrin, 2016)
   - Sparsity-aware split finding for fast computation (linear to number of non-missing entries in the input)
   - Weighted quantile sketch for approximate proposal of candidate split points
   - Insights on cache access patterns, data compression and sharding for building a scalable system
2. LightGBM (Ke *et al.*, 2017)
   - Gradient-based One-Side Sampling (GOSS) for excluding data instances with small gradients in the computation of information gain
   - Exclusive Feature Bundling (EFB) for bundling mutually exclusive features to reduce number of features

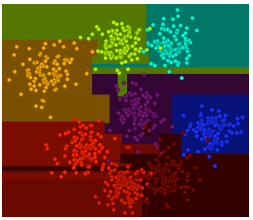
Random Forests (L. Breiman, 2001)

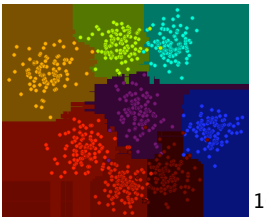
- By learning trees based on randomly chosen subset of:
  - Input variables
  - Data cases
- Bagging of random decision trees (rCART)

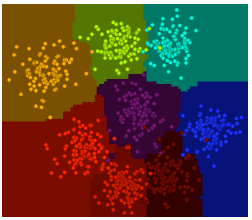
Random Forests (from Debreuve and Morpheme's notes)

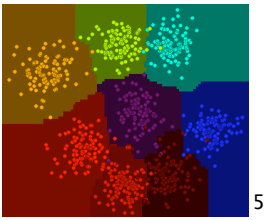
- All labeled samples initially assigned to root node
- N ← root node
- With node N do
  - Find the feature F among a random subset of features + threshold value T...
    - ... that split the samples assigned to N into 2 subsets S_{left} and S_{right}...
    - ... so as to maximize the label purity within these subsets
  - Assign (F,T) to N
  - If S_{left} and S_{right} too small to be splitted
    - Attach child leaf nodes L_{left} and L_{right} to N
    - Tag the leaves with the most present label in S_{left} and S_{right}, resp.
  - else
    - Attach child nodes N_{left} and N_{right} to N
    - Assign S_{left} and S_{right} to them, resp.
    - Repeat procedure for N = N_{left} and N = N_{right}
- Random subset of features
  - Random drawing repeated at each node
  - For D-dimensional samples, typical subset size = round(sqrt(D)) (also round(log2(x)))
  - → Increases diversity among the rCARTs + reduces computational load
- Typical purity: Gini index

Random Forests Example (from Debreuve and Morpheme's notes)


1 rCART


10 rCARTs


100 rCARTs


500 rCARTs