Decision Trees & Extensions

Why decision trees?

Several related reasons

- they are known to perform very well on **structured/rectangular** data: typically the type of financial data (if we exclude alt-data)
- they (i.e., their **extensions**) are probably the most popular solutions among winning solutions in Kaggle competitions
- they are often easy to understand and to **interpret** (at least through variable importance)

Why do tree-based models still outperform deep learning on typical tabular data?

©Guillaume Coqueret

Why again?

Forecasting with trees

Tim Januschowski [a] ✉, Yuyang Wang [a] ✉, Kari Torkkola [a] ✉, Timo Erkkilä [b] ✉, Hilaf Hasson [a] ✉, Jan Gasthaus [a] ✉

Abstract

The prevalence of approaches based on gradient boosted trees among the top contestants in the M5 competition is potentially the most eye-catching result. Tree-

Tabular data: Deep learning is not all you need

Ravid Shwartz-Ziv ✉, Amitai Armon ✉

Highlights

- Deep neural networks are not good for all type of tabular data.

- XGboost outperforms deep models on tabular data.

- It harder to optimize deep neural networks compared to XGBoost.

©Guillaume Coqueret

Agenda

1 Principle & Examples

2 Extensions

3 Variable importance

4 Wrap-up

Introduction

Difference with linear approaches

Linear approach:

$$y_i = \alpha + \beta^{(1)}x_i^{(1)} + \cdots + \beta^{(K)}x_i^{(K)} + \epsilon_i$$

The issue here is that all variables have the same impact on **all** observations!
BUT: maybe accounting ratios have different meanings across sectors, or across firm size.

There are **much more complicated functions** $y_i = f(\mathbf{x}_i) + \epsilon_i$!

$\rightarrow$ Trees are models that form clusters based on $\mathbf{x}$ such that the clusters are homogeneous in y.

In portfolio management, y is related to performance, say, return. Hence depending on $\mathbf{x}$, some clusters will have low average y (e.g., return) and others high average y. Those are the ones we will want to invest in!

supervised
learning
factor
investing

©Guillaume Coqueret

5 / 31

A basic example

y is the colour of the star (to be predicted), $\mathbf{x}^{(1)}$ is the size of the star, $\mathbf{x}^{(2)}$ is the shape (complexity) of the star.
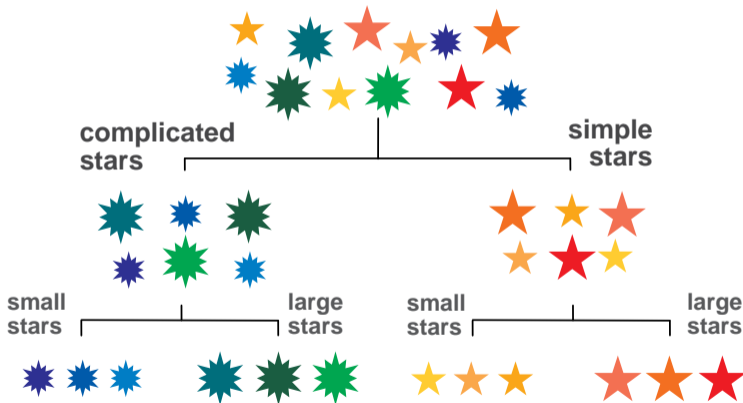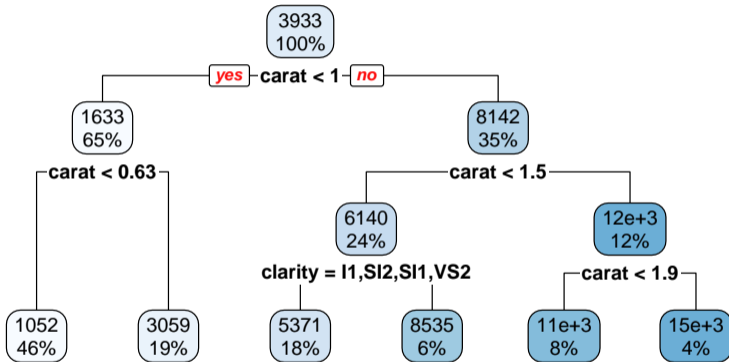
Usual non-financial example: Obama-Clinton votes from NY Times

Diamonds

Splits, **splits**, it'a all about splits!
What are the **drivers** of their prices? Cut, colour, clarity... size?

The **size**, of course, but not only!
$\rightarrow$ in rpart, from the condition under the node, it's always: TRUE to the left and FALSE to the right.

©Guillaume Coqueret

Process (1/3)

A large majority of trees work with binary splits (so will we).

A simplified case: how to choose splitting points?

Imagine we work with only one explanatory variable x. In the case of diamonds, let's assume we only consider the size (weight in carats). What's the best possible (initial) split?
Since we seek **homogeneous clusters**, total quadratic variation/dispersion is a good start:

$$V(c) = \underbrace{\sum_{i:carat>c} (p_i - \bar{p}_{carat>c})^2}_{\text{large diamond cluster } (>c)} + \underbrace{\sum_{i:carat<c} (p_i - \bar{p}_{carat<c})^2}_{\text{small diamond cluster } (<c)}$$

$\bar{p}_{carat>c}$ is the average price of all diamonds with weight larger than c.

This split yields a decomposition that is smaller than the original total variation (scaled variance):
$TV = \sum_i (p_i - \bar{p})^2$. The gain is

$$G(c) = TV - V(c)$$

supervised
learning
factor
investing

©Guillaume Coqueret

8 / 31
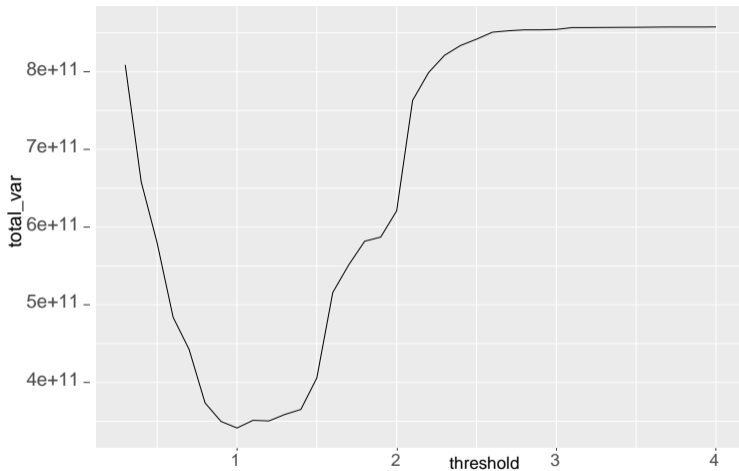
Process (2/3)

Minimum point at 1 carat. **No other feature** can achieve this low level of total dispersion.

The full splitting process can be described as:

For **each node** / **cluster**:

1. Find the best splitting point for each variable.
2. Determine which variable achieves the **highest gain** (reduction in loss or objective)
3. Split according to this variable → two **new clusters**!

This is performed until some condition is met (improvement of gain becomes too small or sufficient depth is reached).

supervised
learning
factor
investing

Categorical **output**

When working with numerical outputs, the variance is the natural dispersion metrics: it measures how (in-)**homogeneous** a cluster is. Things are more complicated for categorical data and several solutions exist. All metrics are based on the **proportions of classes** within a node: p_k.

Examples:

- the Gini impurity index: $1 - \sum_{k=1}^{K} p_k^2$
- the misclassification error: $1 - \max_{k} p_k$
- cross-entropy: $- \sum_{k=1}^{K} \log(p_k) p_k$

The tree is built by **minimising** one of these values at each split. [1]
It is also possible to weight loss according to categories (some errors are sometimes more costly than others) $\rightarrow$ loss matrix.

[1] **Note**: the Gini index is related to the Herfindahl index (portfolio diversification)

Example

Say we have 3 classes: buy, hold and sell. Inside a cluster, the proportion of signals are:

config.	buy	hold	sell	Gini coef	misclassif.
1	0.9	0.1	0.0	$1-0.9^2-0.1^2=0.18$	$1-0.9=0.1$
2	0.3	0.4	0.3	$1-2\times0.3^2-0.4^2=0.66$	$1-0.4=0.6$

Clearly, the second configuration has a higher (hence less diversified) score.
$\rightarrow$ In the first case, the decision is obvious (buy) because the cluster is quite pure, whereas in the second cas, the decision in much more complicated, as all classes are balanced.

Categorical **features**

Test all configurations!

- ▶ Splitting is only computationally feasible for a small number of classes.
- ▶ Indeed, given q classes, there are $2^{q-1} - 1$ ways to split the data.
- ▶ 10 classes imply $\sim$500 partitions - this is probably already too much, especially if there are many predictors.

In fact, there are some tricks to speed up the computation when y is numerical. The idea is to **rank** classes according to their mean in the output variable and to treat them as an ordered variable (the number of splits is greatly reduced: linear in q!).[2]

(for multiclass output, there is no simple way out)

supervised
learning
factor
investing

[2]See Sec. 9.2.4 in the *Elements of Statistical Learning* for more details.

Financial data

On rescaled features:
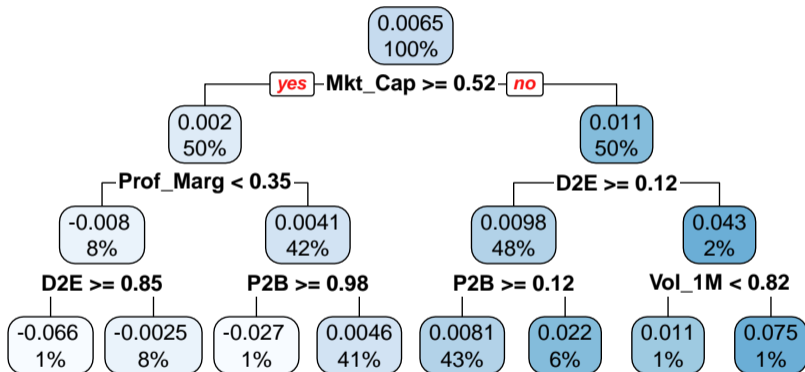
The first splitting variables are the **most important**.

Agenda

1 Principle & Examples

2 Extensions

3 Variable importance

4 Wrap-up

supervised
learning
factor
investing

©Guillaume Coqueret

Random forests (1/2)

▶ Given one splitting rule and one dataset, there is only one optimal tree.
▶ RF pick random combinations of features to build aggregations of trees. This technique is called '**bagging**'.
▶ Each output value is the average of the outputs obtained over all trees.

Tree 1   **Tree 2**   **Tree 3**

+0.1   -0.05   +0.25

(0.1-0.05+0.25)/3 = **0.1**
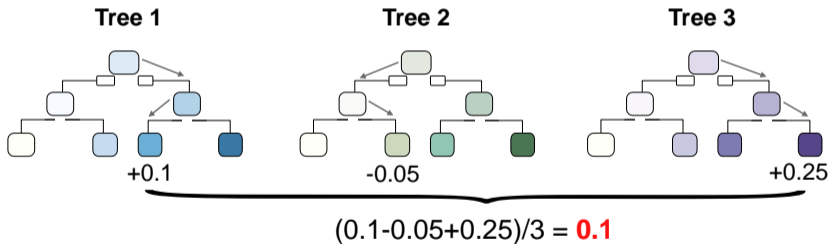
(or **majority voting** if categorical output)

supervised
learning
factor
investing

Random forests (2/2)

Options when building the forests:

- ▶ the **number** of trees!
- ▶ fix the **depth** / **complexity** of the tree (just as for simple trees);
- ▶ select the **number of features** to retain for each individual learner (tree); they are randomly selected (column subsampling);
- ▶ determine the **sample size** required to train the learners (row subsampling).

Other options: minimum number of obs. per leaf (nodesize) and maximum number of leaves (maxnodes).

Random forests are more complex and richer than simple trees.

One cool feature: RFs don't have a huge tendency to **overfit**. Thus, most of the time, their performance is higher.

supervised
learning
factor
investing

©Guillaume Coqueret

18 / 31

Boosted trees: principle (1/6)

Idea

Like RFs, **boosted trees** aggregate trees. The model can be written as:

$$\tilde{y}_i = m_J(\mathbf{x}_i) = \sum_{j=1}^{J} T_j(\mathbf{x}_i),$$

where i is the index of the occurrence (e.g., stock-date pair) and J is the **number of trees**. We thus has the following simple relationship when we add a new tree to the model:

$$m_J(\mathbf{x}_i) = \underbrace{m_{J-1}(\mathbf{x}_i)}_{\text{old trees}} + \underbrace{T_J(\mathbf{x}_i)}_{\text{new tree}}$$

When the previous model is built, how do we choose the new tree?
$\rightarrow$ the model is constructed **iteratively**.

supervised
learning
factor
investing

Boosted trees: principle (2/6)

We follow the approach of **XGBoost** (Chen & Guestrin (2016)).

Objective

What we seek to minimise (objective) is

$$O = \underbrace{\sum_{i=1}^{I} \text{loss}(y_i, \tilde{y}_i)}_{\text{error term}} + \underbrace{\sum_{j=1}^{J} \Omega(T_j)}_{\text{regularisation term}}$$

The first term (over all instances) measures the distance between the true label and the output from the model $\tilde{y}_i$. The second term (over all trees) penalises models that are too complex.

For simplicity, we will work with $\text{loss}(y, \tilde{y}) = (y - \tilde{y})^2$:

$$O = \sum_{i=1}^{I} (y_i - m_{J-1}(\boldsymbol{x}_i) - T_J(\boldsymbol{x}_i))^2 + \sum_{j=1}^{J} \Omega(T_j)$$

supervised
learning
factor
investing

Boosted trees: principle (3/6)

We have built tree T_{J-1} (and hence model m_{J-1}): how to choose tree T_J? **Decompose** the objective:

$$O = \sum_{i=1}^{I} (y_i - m_{J-1}(\boldsymbol{x}_i) - T_J(\boldsymbol{x}_i))^2 + \sum_{j=1}^{J} \Omega(T_j)$$

$$= \sum_{i=1}^{I} \left\{ y_i^2 + m_{J-1}(\boldsymbol{x}_i)^2 + T_J(\boldsymbol{x}_i)^2 \right\} + \sum_{j=1}^{J-1} \Omega(T_j) + \Omega(T_J)$$

$$- 2 \sum_{i=1}^{I} \left\{ y_i m_{J-1}(\boldsymbol{x}_i) + y_i T_J(\boldsymbol{x}_i) - m_{J-1}(\boldsymbol{x}_i) T_J(\boldsymbol{x}_i)) \right\} \quad \text{cross terms}$$

$$= \sum_{i=1}^{I} \left\{ -2 y_i T_J(\boldsymbol{x}_i) + 2 m_{J-1}(\boldsymbol{x}_i) T_J(\boldsymbol{x}_i)) + T_J(\boldsymbol{x}_i)^2 \right\} + \Omega(T_J) + c$$

The red terms are known at step J!

$\rightarrow$ things are simple with quadratic loss. For more complicated loss functions, **Taylor expansions** are used.

supervised
learning
factor
investing

We need to take care of the **penalisation** now. For a given tree T, we specify tree $T(x) = w_{q(x)}$, where w is the output value of some leaf and $q(\cdot)$ is the function that maps an input to its final leaf. We write $l = 1, \ldots, L$ for the indices of the leafs of the tree. In XGBoost, complexity is defined as:

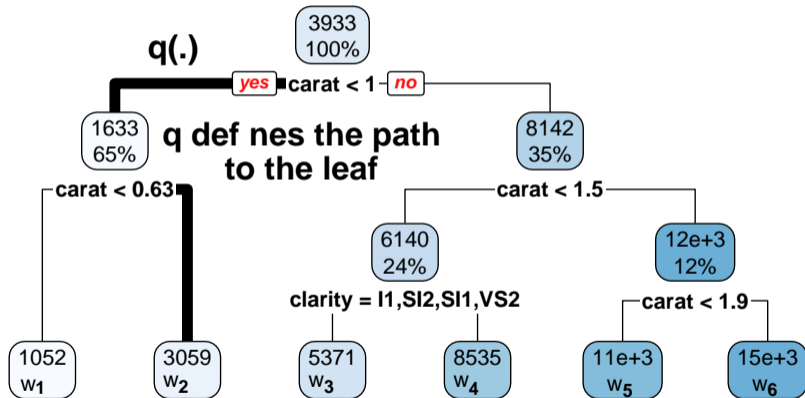$$\Omega(T) = \gamma L + \frac{\lambda}{2} \sum_{l=1}^{L} w_l^2,$$

where

▶ the first term penalises the **total number of leaves**

▶ the second term penalises the **magnitude of output values** (this helps reduce variance).

Below, we omit the index (J) of the tree of interest (i.e., the last one). Moreover, we write I_l for the set of the indices of the instances belonging to leaf l.

supervised
learning
factor
investing

Boosted trees: principle (4b/6)

Illustration of q and w:

We aggregate both sections of the objective:

$$O = 2\sum_{i=1}^{I}\left\{ y_i T_J(\boldsymbol{x}_i) - m_{J-1}(\boldsymbol{x}_i)T_J(\boldsymbol{x}_i)) + \frac{T_J(\boldsymbol{x}_i)^2}{2}\right\} + \gamma L + \frac{\lambda}{2}\sum_{l=1}^{L} w_l^2$$

$$= 2\sum_{i=1}^{I}\left\{ y_i w_{q(\boldsymbol{x}_i)} - m_{J-1}(\boldsymbol{x}_i)w_{q(\boldsymbol{x}_i)} + \frac{w_{q(\boldsymbol{x}_i)}^2}{2}\right\} + \gamma L + \frac{\lambda}{2}\sum_{l=1}^{L} w_l^2$$

$$= 2\sum_{l=1}^{L}\left(w_l \sum_{i\in I_l}(y_i - m_{J-1}(\boldsymbol{x}_i)) + \frac{w_l^2}{2}\sum_{i\in I_l}\left(1 + \frac{\lambda}{2}\right)\right) + \gamma L$$

$$\rightarrow \quad w_l^* = -\frac{\sum_{i\in I_l}(y_i - m_{J-1}(\boldsymbol{x}_i))}{\left(1 + \frac{\lambda}{2}\right)\#\{i \in I_l\}}, \quad O_L(q) = -\frac{1}{2}\sum_{l=1}^{L}\frac{\left(\sum_{i\in I_l}(y_i - m_{J-1}(\boldsymbol{x}_i))\right)^2}{\left(1 + \frac{\lambda}{2}\right)\#\{i \in I_l\}} + \gamma L$$

which gives the optimal weights (or leaf scores) and objective value.[3]

[3] $f(x) = ax + \frac{b}{2}x^2$: min at $x^* = -a/b$, min value $= -a^2/(2b)$.

Boosted trees: principle (6/6)

Final problem: the **tree structure**! This is the *q* function essentially. For instance: what's the best depth? Do we continue to grow the tree or not?

▶ we proceed node by node

▶ for each node, we look at whether a split is useful (in terms of objective) or not:

$$\text{Gain} = \frac{1}{2}\left(\text{Gain}_L + \text{Gain}_R - \text{Gain}_O\right) - \gamma$$

▶ each gain is computed with respect to the instances in each bucket:

$$\text{Gain}_{\mathcal{X}} = \frac{\left(\sum_{i \in I_{\mathcal{X}}}(y_i - m_{J-1}(\boldsymbol{x}_i))\right)^2}{\left(1 + \frac{\lambda}{2}\right)\#\{i \in I_{\mathcal{X}}\}},$$

where $I_{\mathcal{X}}$ is the set of instances within cluster $\mathcal{X}$.

Gain_O is the original gain (no split).

$-\gamma$: one unit of new leaves: two minus one!

NOTE: XGBoost also applies a learning rate: each new tree weight is multiplied by η.

supervised
learning
factor
investing

Agenda

1 Principle & Examples

2 Extensions

3 Variable importance

4 Wrap-up

supervised
learning
factor
investing

Interpretability!

Definition = gain allocation!

▶ Each time a split is performed according to one variable/**feature**, the objective function is reduced

▶ This **gain** is entirely attributed to the variable

▶ If we sum the gains obtained by all variables over all trees and normalise them in the cross-section of features, we get the **relative importance** of each feature!

$$V_f \propto \sum_{n=1}^{N} G_n(f),$$

where f is the index of the feature and $G_n(f)$ the gain associated to feature f at node n (positive if the split is performed according to f and zero otherwise).

supervised
learning
factor
investing

©Guillaume Coqueret

27 / 31

Permutation importance

An alternative definition

What happens to a set predictions if we shuffle the values of one given column (predictor)? If it is important, the performance (accuracy, MSE) should decrease sharply! If not, it will stay the same. Hence, for each feature, we can

1. Randomly shuffle the values of the predictor for some set - preferably test set
2. Evaluate the performance on this set (accuracy, MSE)
3. Average across all shuffled observations and compute the loss in accuracy/MSE

→ this is pretty **flexible** and works for any algorithm!

In the standard packages (ex: XGBoost, randomForest, even Scikit-Learn!), this is what is usually done.

Agenda

1 Principle & Examples

2 Extensions

3 Variable importance

4 Wrap-up

supervised
learning
factor
investing

Key takeaways

Trees are powerful

- ▶ they organise the data into **homogeneous clusters** in a highly nonlinear fashion
- ▶ extensions help improve the predictive performance by **aggregating several trees**
- ▶ variable importance is a first step towards **interpretability** (global in this case)

Practice is the fastest way to learn $\rightarrow$ let's go!

supervised
learning
factor
investing

©Guillaume Coqueret

30 / 31

Thank you for your attention

Any questions?