Using Boosted Regression Trees in Insurance

Allianz Lebensversicherungs AG

Jakob Gerstenlauer

June 7, 2019

The Challenges of Boosted Regression Trees

Why you want to use them.

Gradient boosted regression trees offer many attractive traits:

- high predictive accuracy,
- no need to specify the model structure,
- automatically detect interactions,
- variety of objective functions,
- ability to cope with incomplete data.

The Challenges of Boosted Regression Trees

Why applying them to insurance data is tricky.

However, gradient boosted regression trees pose crucial challenges:

- difficult to tune the many hyper-parameters,
- lack of global model coefficients,
- features may have both positive and negative effects,
- a confusing jumble of tools for their interpretation,
- many potential pitfalls in model interpretation.

Preprocessing for XGBoost

Feature selection and missing data imputation.

Recommended processing steps:

- Remove inputs which only add noise to the data set using entropy-based filter methods.
- Test if missing values are missing at random.
- Data preparation of metric inputs:
  - If interpretability is key, use a simple linear regression model to remove numeric features with maximum VIF (see this blog)[5].
  - Z-transform inputs if you want to compare effect sizes.
- Data preparation of categorical inputs:
  - Find and remove redundant features based on Cramers V.
  - Merge rare levels of categorical variables.
  - Are there sensible reference levels?
  - Yes: Create levels - 1 dummy variables for each category.
  - No: Create levels dummy variables for each category.

Hyperparameter Tuning

Tips for hyperparameter tuning.

Tuning of hyperparameters is important, because XGBoost is highly tunable and default values are far from optimal [4].

Recommended steps:

1. Draw a balanced subsample (undersampling).
2. Use recommended defaults [4, Table 7] for all hyperparameters, set $\eta = 0.3$, tune *nrounds* with *xgboost.cv()*.
3. Tune *tree-depth* with a simple tree (code).
4. Define a search space with recommended parameter ranges for all other hyperparameters [4, Table 7], set-up a tuning experiment in mlr using nested resampling.
5. Set $\eta = 0.031$ and tune *nrounds* with *xgboost.cv()*.
6. Fit the model using the *xgboost()* function with η=0.031 and the optimal number of iterations.

Understanding the results

From global to individual importance measures.

- In tree-based ML methods, the effect of a feature depends on the values of other features.
- Instead of using global feature importance measures, why not use individualized feature importance measures?
- This is the idea behind SHAP (SHappley Additive exPlanation) values [3]:
    - SHAP values describe the additive individual effects of inputs on the output.
    - They are guaranteed to show local accuracy: the sum of the feature attributions is equal to the model prediction.
    - They are a model-agnostic method.

Understanding the results

From global to individual importance measures.

SHAP values describe the individual effects of inputs on the output. They have the following advantages:

1. There is a solid theory behind them showing that they unify various existing feature importance measures [3].
2. They can be extracted from all modern boosting algorithms in a computationally efficient way (SHAP values for higher order interactions are only available in the SHAP python package).
3. Once calculated, it is possible to estimate both mean global effects and visualize interactions between inputs.

Understanding the results

Retrieving global coefficients from SHAP values

Approach to estimate global coefficients from SHAP values of **dummy** variables:

1. Retrieve SHAP values for level.
2. Retrieve probability of reference level.
3. Backtransform to probability scale from log odds-ratio.
4. Calculate summary function (mean, median, quantile).

Understanding the results

Retrieving global coefficients from SHAP values

Approach to estimate global coefficients from SHAP values of **numeric** variables:

1. Retrieve SHAP values for feature.
2. Backtransform to probability scale from log odds-ratio.
3. Retrieve x values for feature.
4. Calculate slope between x-values and probabilities.

Understanding the results

Looking at both global and local effects.

Based on this approach, we are able to estimate:

1. *global mean effects* based on median SHAP values,
2. *variability of effects* based on quantiles of SHAP values,
3. *local effects*, based on individual SHAP values.

In the insurance industry, these global effect estimates are key to effectively communicate the results of data science projects.

Conclusions

The challenges of using ensemble methods.

Boosted regression tree algorithms can only be successfully applied and understood, if:

1. data preparation steps are followed meticulously,
2. hyperparameter tuning is done in a systematic way,
3. audience is ready to accept lack of true global coefficients,
4. both global mean and variance of effects are estimated.

References

[1] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[2] Brandon M Greenwell. “pdp: an R Package for constructing partial dependence plots”. In: *The R Journal* 9.1 (2017), pp. 421–436.

[3] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. “Consistent individualized feature attribution for tree ensembles”. In: *arXiv preprint arXiv:1802.03888* (2018).

[4] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. “Tunability: importance of hyperparameters of machine learning algorithms”. In: *arXiv preprint arXiv:1802.09596* (2018).

[5] Alain F Zuur, Elena N Ieno, and Chris S Elphick. “A protocol for data exploration to avoid common statistical problems”. In: *Methods in ecology and evolution* 1.1 (2010), pp. 3–14.

Appendix: Gradient Boosting in Detail.

Boosting as a meta-algorithm.

Boosting* can be regarded as *gradient descent in model space*:

1. Select your base model (e.g. a regression tree)
2. Decide on the hyper-parameters of your model (tree depth, minimum size for split, ...)
3. Decide on the hyper-parameters of your boosting algorithm (η: learning rate, I: number of iterations)
4. While the current iteration $i < I$ do:
   4.1 Fit your base model to the training data.
   4.2 Replace the original output $y := y - \eta \hat{y}$.

***Caveat**: The explanation above is referring to the dominant flavour of boosting which is also more precisely called gradient boosting to differentiate it from the *Adaboost* algorithm where only the weights of instances are recalculated.

Appendix: Comparison of boosting packages

Different gradient boosting algorithms.

Comparison of popular packages for gradient boosting in R:

package	categories	parallel	missings
gbm	factors	no	yes
XGBoost	dummy	yes	yes
LightGBM	dummy	yes	yes
catboost	numeric*	yes	yes

*Catboost transforms categorical inputs into numeric vectors replacing each category by the expected target value.

Appendix: Comparison of boosting packages

Handling missing values.

While all packages are able to deal with missing values, the details differ significantly:

1. catboost: Missing values are replaced by either the minimum or maximum value of the feature (docs).
2. XGBoost: Decides on a default direction if value is missing or zero [1].
3. gbm: Fits a separate node for missing values at each split (link).
4. LightGBM: It is possible to switch between treating zeros in sparse matrix as missings and using explicit NAs (docs)

While the mechanism of catboost seems to be the most lacking, it is unclear which of the other packages is best at treating missing values.

Appendix: Understanding the results

Tools to peek into the black box.

One drawback of trees and especially ensemble methods build on trees is that they **lack global model coefficients**. Traditionally, the following tools have been used to peek into the black box:

1. Global feature importance measures:
   - Gain: total reduction of loss or impurity contributed by all splits using the input.
   - Split count: frequency of using input to split.
   - Permutation: random permutation of the values of an input to assess the increase of the model error.
2. Plots of predicted values.
3. Marginal plots or partial dependence plots showing the average of the predicted response for each x-value holding all other covariates constant (e.g. *plot.gbm()* or the *pdp* package which can be used in combination with gbm and xgboost to create partial dependence plots [2]).

Appendix: Understanding the results

...and why they are inadequate.

But these different approaches have their drawbacks:

1. All except permutation based global feature importance measures have been found to be inconsistent [3]. Even permutation based measures are not able to assess the *direction of effects*.
2. Plots of predicted values are unable to isolate the effect of a specific feature.
3. Marginal plots or partial dependence plots are difficult to interpret. They are not always available for a given algorithm. They can't be used to display interactions.

Appendix: Understanding the results

Retrieving global coefficients from SHAP values

How to backtransform from log odds-ratio scale:

1. odds-ratio reference level: $r_0 = \frac{p}{1-p}$
2. odds-ratio treatment level: $r_1 = \frac{p_{x1}}{1-p_{x1}}$
3. $\beta_1 = log(\frac{r_1}{r_0})$
4. $e^{\beta_1} = \frac{r_1}{r_0}$
5. $p_{x1} = \frac{e^{\beta_1} r_0}{e^{\beta_1} r_0 + 1}$

Given the probability p of the reference level, we can calculate the increase/decrease in probability given the specific level. For numeric features, use the global mean probability as reference.

Appendix: Understanding the results

Retrieving global coefficients from SHAP values

Another way of phrasing the problem:

1. The linear predictor for a given observation: $y = x_1\beta_1 + x_2\beta_2$
2. There is no intercept term β_0!
3. If we want to quantify the effect of feature x_1, we have to measure the effect in reference to a certain reference probability p and use it as a replacement for the missing intercept.
4. $y = \beta_0 + x_1\beta_1$
5. $\beta_0 = log(\frac{p}{p-1})$
6. Odds scale: $r_1 = e^y = e^{log(\frac{p}{1-p})+\beta_1} = e^{\beta_1}\frac{p}{1-p} = e^{\beta_1}r_0$
7. Probability scale: $p_{x1} = \frac{e^{\beta_1}r_0}{e^{\beta_1}r_0+1}$
8. Percentage increase or decrease: $\frac{p_{x1}}{p}$