Comparison of Conformal Prediction and Bootstrapping on a Regression Problem with Random Forests

732A76 Research Project

David Hrabovszki
Linköping University

04 January 2021

Introduction

Conformal prediction is a relatively new framework for obtaining prediction intervals. Its main advantage is that it guarantees that the probability of making accurate predictions is larger than or equal to a user specified confidence level. Conformal prediction can be built on an arbitrary model that results in point prediction, and it will create prediction intervals from it. Inductive conformal prediction was suggested in (Vovk, Gammerman, and Shafer 2005), which has a very computationally efficient way of calculating intervals by only training the underlying model once. Therefore, this paper also uses this approach.

In this experiment, conformal prediction and parametric bootstrapping algorithms are implemented and their performances are compared on a regression task, where the underlying model is a random forest and the data used is artificially generated. The experiments are run with different sizes of datasets as well. The code used can be found on https://github.com/hradave/Research_project. Standard R packages were used where possible, but the conformal prediction was implemented from scratch building on the papers (Johansson et al. 2014) and (Boström et al. 2016).

The competing methods are evaluated along three metrics: coverage rate, mean size of prediction regions, runtime. Coverage rate is the ratio of the test instances that fall into the prediction region, mean size of prediction regions is referred to as efficiency in (Johansson et al. 2014) and is simply the average of the prediction intervals over the test set. Runtime is the CPU time needed to train the models, and to calibrate the nonconformity scores in the case of conformal prediction. Prediction times are not included in this measure.

Methods

Conformal Prediction

The use of conformal prediction allows us to obtain a prediction region instead of a single-point prediction, where the regions can be different sizes for each instance. It also guarantees that the probability of making an error (the test instance being outside the prediction interval) is bounded by a confidence level set by the user, this makes all conformal predictors valid (Johansson et al. 2014). Conformal prediction can be built on an arbitrary prediction model, in this paper a random forest is used for this purpose.

In this paper, similarly to the one referenced above, inductive conformal prediction is implemented, where the underlying model only has to be trained once on the training set, which can be used for prediction on the test set. In this case however, a separate dataset is required to calibrate the nonconformity scores.

The nonconformity scores are calculated on the calibration and test sets using the nonconformity function, which is usually related to the absolute error of the prediction for each instance:

$$\alpha_i = |y_i - \hat{y_i}|$$

The scores represent how different the new example is from the old ones. If it is relatively small, then that instance fits well into the model that we trained on the training set. The role of the calibration set is to find the smallest nonconformity score, compared to which, at least 95% (or other confidence level) of the calibration data points' scores are smaller. Taking this smallest score (bound) $\alpha_{s(\delta)}$, we can create prediction regions for the test instances by adding this bound to and subtracting it from the predicted value:

$$\hat{Y}_j^{\delta} = \hat{y}_j \pm \alpha_{s(\delta)}$$

Since this method does not depend on the individual test observation, the regions will have the same size, which would cancel one of the main advantages of conformal prediction. Therefore, this paper employs a normalized nonconformity function instead, where the scores are calculated by dividing the absolute prediction error by σ_i, which is the estimated difficulty of the random forest model for making a correct prediction for instance y_i.

$$\alpha_i = \frac{|y_i - \hat{y_i}|}{\sigma_i}$$

The prediction intervals then become:

$$\hat{Y}_j^{\delta} = \hat{y}_j \pm \alpha_{s(\delta)} * \sigma_j$$

There are several very different ways to calculate σ, but this paper builds on the conclusions of the experiments in (Boström et al. 2016), and implements the method which proved to be the least computationally expensive, but equally efficient approach, the variance-based measure. This method takes the variance of the predictions of the individual trees in the random forest as the difficulty of each instance, because it assumes that there is a larger understanding (smaller variance) between the trees, if the instance is easy to predict. The nonconformity scores are calculated this way:

$$\alpha_i = \frac{|y_i - \hat{y_i}|}{\mu_i + \beta}$$

where β is a parameter that controls the sensitivity of the nonconformity measure and μ_i is the variance of the individual predictions for instance i.

Bootstrap

Bootstrapping is a widely used algorithm for creating confidence and prediction intervals using datasets randomly sampled with replacement (non-parametric) or generated from their distributions (parametric). Since this paper experiments with artificially simulated data, so the distribution of the response value is known, the more precise parametric version can be implemented.

A specified number (R) of training set replicates are generated, the random forest model is trained on each of them, and the predictions are made for the test test in every iteration. This results in a distribution of prediction values for every test instance, and the quantile method can be used to obtain 95% (or any other) confidence or prediction bands.

Empirical Evaluation

Experimental Setup

Data Simulation

Empirical experiments are often conducted on real world datasets, but because this research focuses on the parametric version of bootstrapping, we opted for an artificially simulated dataset instead. This means that we can be certain about the distribution of the response variable, which is necessary for parametric bootstrap. The implementation was based on a previous paper that uses an additive error model (Zhang et al. 2020):

$Y = m(X) + \epsilon$, where $X \sim \mathcal{N}(0, \Sigma_{50})$ and Σ_{50} is an AR(1) covariance matrix with $\rho = 0.6$ and diagonal values equal to 1. The error term ϵ is Gaussian noise and the $m(X)$ mean function is nonlinear with interaction between the predictors to make the data more complex. 100,000 instances were simulated from this model. The response variable is normalized, so that the resulting interval sizes can be more easily interpreted. An interval size of 0.5 means that the region covers 50% of the range the variable can take.

Data Splits

The experiments use the holdout method for evaluation with 70% - 30% split between training and test sets. The first 70,000 instances were used as the pool for training and the last 30,000 for testing. For each different data size, the appropriate number of observations were sampled from these pools randomly to ensure that no test instance has been used for training or for validation while optimizing the parameters. It is important to note that while bootstrapping uses all training examples for training the random forest, conformal prediction utilizes only a part of it for proper training and the rest for calibration. A possible approach would have been to make the number of instances used for training equal for both methods, but since calibration is an important step in obtaining prediction intervals, I made the decision to use a subset of the training data for calibration. The number of calibration instances was set to

$$q = 100 * \left\lfloor \frac{|Z|}{400} \right\rfloor - 1$$

according to (Johansson et al. 2014), where Z is the full training set. This means that around 20 - 25% of the full training set was used for calibration.

Computer Specifications

The experiments were run on a notebook with an Intel Core i5-8250U processor (4-core 1.6 GHz base frequency) and 12GB of RAM. The conformal predictions were run on a single core, but the bootstrapping utilized all cores in parallel, which is straightforward to implement with the boot() function on a Linux system.

Chosen Parameters

Three parameters were optimized using grid search: the number of trees in a random forest (ntree), the sensitivity of the nonconformity measure in conformal prediction (β) and the number of bootstrap replicates (R). 10,000 instances were used from the training pool for the grid search, where **ntree** was chosen to be **125**, because this value resulted in a "low enough" mean squared error. A higher number of trees would make the predictions slightly more accurate, but only at the a significantly larger computation cost. β was chosen to be **0.01**, because the mean of conformal prediction region sizes was the smallest with this value, and the runtime was slightly shorter than with others. The confidence level set for the interval was barely affected by β. The number of bootstrap replicates should be as large as possible, but increasing this parameter increases the runtime linearly, so the optimal value was set to a low value **(R = 200)** that still results in reasonably good coverage rates and mean region sizes.

Results

Prediction intervals were constructed using conformal prediction and bootstrapping at 95% confidence level on artificially simulated datasets of sizes 1,000 - 10,000. In each run, we monitor the coverage rate, the mean interval size and the time required to train the models.

Theoretically, the coverage rate of the conformal prediction should be around the confidence level (Johansson et al. 2014), because the prediction intervals produced are always valid. This is exactly what we observe here as well, the coverage rate is slightly larger than 0.95, but it seems to converge to it as the number of observations grows. Bootstrapping, however, resulted in much lower coverage rates in all the runs, and there seems to be no indication that a larger dataset would solve this problem (Figure 1 top left).

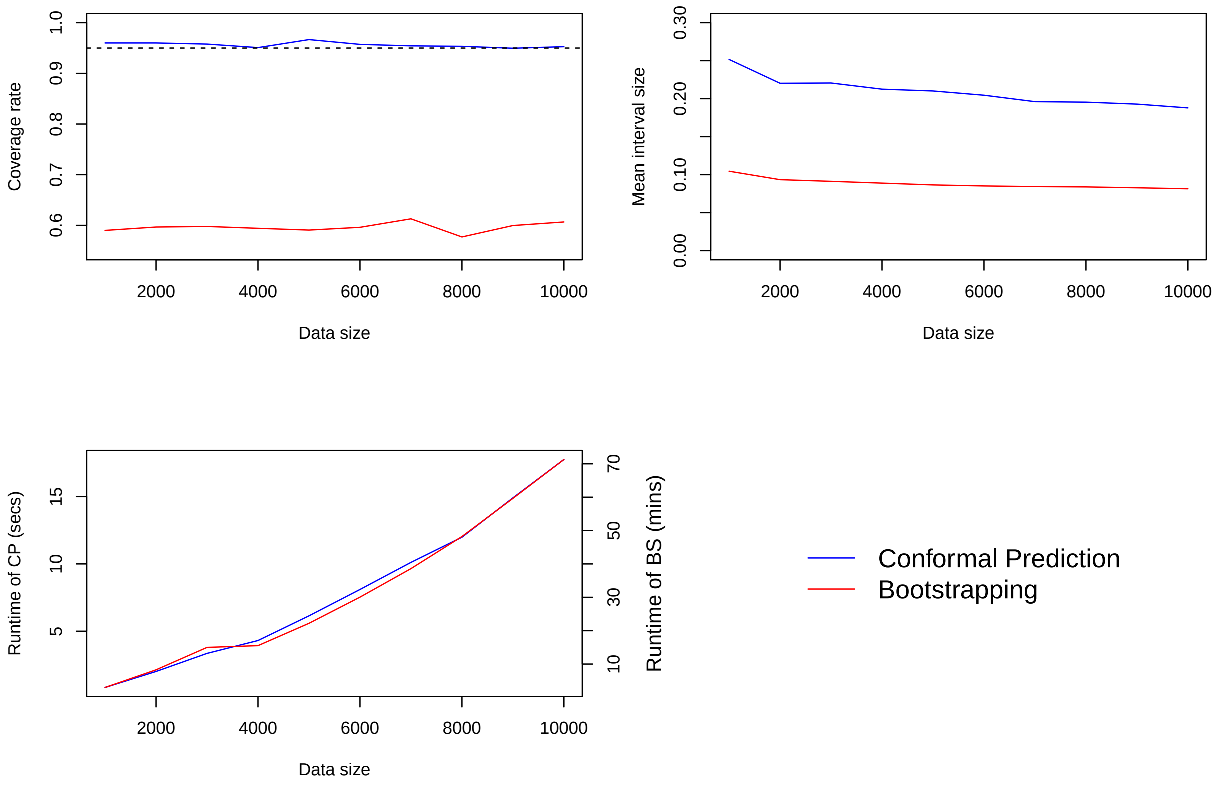

Figure 1: Results

The mean size of the prediction regions decreases in both cases as the dataset grows, but the bands resulting from bootstrapping are much narrower, around half the size of the conformal prediction bands. The prediction regions cover around 10% of the whole range with bootstrapping, and 20% - 25% with conformal prediction (Figure 1 top right).

There is a significant difference between the runtimes of the two methods, even though it increases linearly in both cases as we use more data. Conformal prediction in general seems to be much faster, than bootstrapping. Conformal prediction is measured in seconds, and bootstrapping is measured in minutes on Figure 1 (bottom left).

Conclusion

The fact that prediction regions from conformal prediction are valid is no surprise, but the regions from bootstrapping are clearly worse than expected, because they do not include 95% of the test instances.

The mean sizes of the intervals are therefore much narrower with bootstrapping than with conformal prediction, but I would not draw conclusions from this, because this also means that the intervals do not have the confidence levels that we set.

The difference in runtime between the two methods is very conspicuous, conformal prediction is significanlty faster. This is fairly easy to explain, because bootstrapping trains the random forest model R (number of bootstrap replicates) times (+1 for the initial estimates), whereas conformal prediction only trains once. This is a huge advantage of the conformal method, because training the model is by far the most computationally expensive task in prediction. Bootstrapping also has to perform random data generation for every replicate. Conformal prediction has to calibrate the bound of the non-conformity scores on the calibration set, but this only includes one apply() function call, some vector operations and one sorting. The overhead costs of conformal predictions are minute relative to the cost of training the model R times, therefore this paper concludes that conformal prediction is much faster, than bootstrapping, while producing more accurate (valid) prediction intervals.

Future Improvements

Several modifications could be made to improve this study of comparing two different methods. The parameters of the random forest and the number of bootstrap replicates could be further optimizied for different data sizes, but in this paper I intended to keep a setup unchanged for more straigthforward comparison. We could look at out-of-bag predictions of the random forests instead of using a separate set for either calibration or testing, but this would not necessarily improve predictive quality. It would be worth an experiment though, because then the available dataset would be better utilized. The data simulation process could be refined, and real world datasets could be used as well to see how the two methods compare against each other in a more real world setting.

References

Boström, Henrik, Henrik Linusson, Tuve Löfström, and Ulf Johansson. 2016. “Evaluation of a Variance-Based Nonconformity Measure for Regression Forests.” *COPA 2016*.

Johansson, Ulf, Henrik Boström, Tuve Löfström, and Henrik Linusson. 2014. “Regression Conformal Prediction with Random Forests.” *Machine Learning*.

Vovk, Vladimir, Alex Gammerman, and Glenn Shafer. 2005. “Algorithmic Learning in a Random World.” In *Algorithmic Learning in a Random World*. https://doi.org/10.1007/b106715.

Zhang, Haozhe, Joshua Zimmerman, Dan Nettleton, and Daniel J. Nordman. 2020. “Random Forest Prediction Intervals.” *The American Statistician* 74 (4): 392–406. https://doi.org/10.1080/00031305.2019.1585288.

Appendix

Table 1: Results of experimental runs

Size	CP coverage rate	CP region size	CP runtime	BS coverage rate	BS region size	BS runtime
1000	0.960	0.252	0.8	0.590	0.105	178.8
2000	0.960	0.220	2.0	0.597	0.093	496.2
3000	0.958	0.221	3.4	0.598	0.091	899.4
4000	0.951	0.213	4.3	0.594	0.089	930.9
5000	0.967	0.210	6.1	0.591	0.087	1332.6
6000	0.957	0.205	8.1	0.596	0.085	1802.8
7000	0.954	0.196	10.1	0.613	0.084	2315.5
8000	0.953	0.195	12.0	0.577	0.084	2892.9
9000	0.950	0.193	14.9	0.600	0.083	3579.0
10000	0.953	0.188	17.8	0.607	0.081	4277.5