Final Project: "Tree-based models for political science data"

CS112 - Professor Diamond

Oren Dar

2018–04–20

Header

TO: *Board of Curriculum Design, Department of Economics, Harvard*

FROM: *Oren Dar, CS112 Student*

SUBJECT: *Overview and comparison of predictive inference methods in political science*

1 Introduction

Distinguished Board of Curriculum Design & Faculty,

I am honored to present to you the results of my replication and extension of "Tree-based models for political science data", followed by my personal analysis as to the efficacy and application of the methods discussed below.

Motivation

I chose to replicate and extend this paper for several reasons:

1. It was published recently, and so it is unlikely that it has ever been replicated or extended yet.

2. I found the paper to be clear and concise conceptually, and so I thought it would be an appropriate project for me.

3. I strongly support the authors' goal of adding more models to the toolbox of political science data analysis and predictive inference, and I have some experience with other models and libraries which would be a natural extension.

2 Replication

2.1 Goal

My chosen goal was to replicate the second part of Table 2 in Section 5 (page 22), labeled *Out-of-sample fit statistics*, where each row represents a specific model, as I believe the code for reproducing this table can serve as a template for producing such a table for any domain and any dataset, which will inform model selection. The models included in the paper are a generalized additive model (GAM), a gradient-boosted machine (GBM), and Bayesian additive regression trees (BART).

The GAM, which is a linear model, was smoothed over the predictors which the authors considered the most relevant - in other words, the model is purposely better than it would realistically be in the usual case in data analysis where there is no prior information (or, as the authors have put it, "unrealistically well specified"). This was done in order to

showcase the effectiveness of the other models, which did not receive this prior information, relative to an ideal linear model.

The GBM and BART models are tree-based boosting ensembles. Both models fit decision trees sequentially, with each subsequent tree fit in order to minimize the current error. The GBM model utilizes gradient-based optimization, where subsequent trees are fit to the negative of the gradient of the current loss, whereas the BART model utilizes Bayesian optimization - and so requires a prior and a likelihood at initialization, and inference is based on an iterative Markov chain Monte Carlo (MCMC) algorithm which generates samples from the posterior.

The columns are evaluations of the predictions of each model. In order: the correlation of the predictions with the predictions of the GAM, the Brier score, the area under the Receiver Operating Characteristic (ROC) curve, the area under the Sensitivity curve, and the area under the Specificity curve.

2.2 Process

The code and data are shared in their entirety on dataverse, and come with a readme file and a pdf which explain what to run and how to replicate their results. However, almost every single line in the relevant file returned an error, or simply did nothing - I believe that it is because the syntax is from an older version of R.

The authors have also kindly included their environment setup and package versions in the readme file, so I considered setting up the same R environment as them to get the file to run, but in the interest of replicating their results in a manner which will allow future researchers to utilize the same code and methods, I chose to use the up-to-date versions

of R and relevant packages, and spent 2 days debugging the entire script and eventually got the same results as in the paper.

Several notes:

1. I have not replicated the first part of Table 2, as that included a 10-fold within-sample cross-validation which the authors ran on hundreds of cores and was too computationally expensive for my personal computer.

2. I considered the 'correlation with GAM' column to not be very informative in terms of the relative performance of the models, and so I have replaced it with 'Accuracy' (percent of observations which were classified correctly), as I believe this is a common metric to show when evaluating several models on a binary classification task with relatively balanced classes.

3. Brier score is mathematically equivalent to mean squared error, and furthermore the authors are actually showing the square root of that (as can be confirmed in the code). Therefore, I have replaced the column name with 'RMSE', which is both the technically correct name and a more common one, but the calculation is exactly the same.

4. I use the prefix 'AU' to denote 'Area under (curve)', as is common in the literature.

5. I have added an additional column, 'LogLoss', which corresponds to logistic loss, or equivalently binary cross-entropy loss or negative log likelihood, as it is the standard metric for binary classification, as it is sensitive to the class probabilities the model predicts and accurately evaluates the model's confidence in its predictions - a slightly worse model may only have slightly lower accuracy or slightly higher RMSE, but it will have a significantly higher log loss. Since all models discussed

here perform well and the differences are relatively minor, this metric is critical and should be present.

6. I did not perform my own cross-validation on the hyperparameters of the random forest, as it was too computationally expensive. Instead I used the hyperparameters mentioned in the article in order to facilitate replication.

7. The train-test split was produced following the same procedures as the authors (1035 observations in the training set, 135 observations in the test set), and using the same random seed - as the GAM results are all exactly the same, I am reasonably certain my train and test data is exactly the same as the authors'. There are minor differences in the results of the GBM and BART models, which is due to the updated versions of R and the packages, as well as the fact that the BART process does not take a fixed random seed.

	Accuracy	RMSE	AUROC	AUSen	AUSpec	LogLoss
GAM	0.839	0.355	0.866	0.610	0.713	0.428
GBM	0.830	0.333	0.906	0.622	0.740	0.356
BART	0.843	0.341	0.907	0.636	0.771	0.637

As can be seen in the table above, my results indeed align with the paper's results with regards to RMSE, AUROC, AUSen and AUSpec - the tree-based models beat the GAM on every one of these metrics, and the authors specifically point out the performance of the BART model. However, when we look at the LogLoss column, we can see that the BART model has significantly higher LogLoss than both the GBM and the GAM models.

2.3 Evaluation on a 75/25 train-test split

As mentioned in the previous section, I could only replicate the second part of Table 2, which is out-of-sample evaluation with a 90/10 train-test split, leaving about 100 observations in the test set. This is usually considered too small of a test set, as fixed statistical noise can have a large impact on a small test set (for example, each change of label is 1% accuracy). The authors have chosen to address that problem using an expensive 10-fold in-sample cross-validation to evaluate the models on the training data. I have chosen a more practical and scalable approach - simply use a more conventional 75/25 train-test split for the dataset, which will allow for more robust evaluations of model performance while still allowing the models to train on most of the data. I then produced the exact same table as above using the exact same code, only with a 75/25 train-test split (863 observations in the training set, 287 observations in the test set).

	Accuracy	RMSE	AUROC	AUSen	AUSpec	LogLoss
GAM	0.872	0.313	0.918	0.613	0.704	0.330
GBM	0.872	0.302	0.930	0.617	0.713	0.307
BART	0.882	0.299	0.937	0.654	0.780	0.513

As can be seen in the table above, all three models perform quite significantly better on the bigger test set. This is not usually the case, as a bigger test set implies smaller training set and therefore less information, and so the only explanation I can think of is that previously, the statistical noise had a negative bias on the evaluation of the models, and therefore when we use a bigger test set and mitigate the effect of statistical noise we get a less biased and more realistic evaluation of how the models perform on unseen data. Although again the BART model seems to perform slightly better than the others

on the metrics the authors chose, it has a significantly higher LogLoss.

3 Extension

3.1 Goal

The goal of the extension part of this project is twofold: to allow researchers and data scientists who use Python (and are not proficient with R) to also have a template for running and evaluating several powerful models on any domain problem and any dataset; and to allow me to evaluate the performance of neural networks on the data - the authors claimed they have tried it and it did not perform well (as can be seen in Table 1), but I suspected that it is likely that the authors are not experts at neural networks, which require more fiddling in order to work well, and so I wanted to see what I can achieve if I put some thought and effort into it.

I only used the 75/25 train-test split dataset for this section.

3.2 Python with authors' preprocessing

I exported the final datasets for training and test, which were preprocessed by the authors, from R and imported it in Python. As area under sensitivity & specificity curves is not a common metric and I have been unable to find them on Python, I have omitted them from the table. I have tested several models from the popular Scikit-learn package:

1. GBM - similar to R's GBM model.

2. Random Forest (RF) - the authors have tested RF in R and found that it did not perform as well (Table 1). However, I have chosen to include it in order to test it in

Python - it is also easy to use and generally trains quickly and performs well even out-of-the-box and on most types and scales of data.

3. Extra-Trees (ET) - extra-trees refers to extremely randomized trees, where each tree is only fitted to a subsample of the data and is given a specific feature and cut-off point for each split - essentially a more strongly constrained and randomized version of RF, which leads to a further decrease in bias and increase in variance (which can help prevent overfitting, especially given the small dataset).

4. KerasTF-NN - A deep neural network with a single hidden layer, using the Keras API with the Tensorflow backend. I standardized the data and tweaked the parameters of the network a little until I achieved a result which seemed reasonable to me and left it there, in the interest of time - I am certain that with more tweaking, and especially with a bigger dataset, better results can be achieved, as the main problem is preventing overfitting. I used a Dropout layer to force it to forget some of the activations with every batch in order to generalize better, and I used a cyclical learning rate scheduler (Smith ,2017) implementation I found online.

5. PyTorch-NN - A deep neural network with several embedding layers followed by linear layers, using PyTorch. Again, I standardized the data and used cyclical learning rate scheduler and high amounts of dropout to avoid overfitting and tweaked the network a little until I achieved reasonable results and left it there, and further tweaking and especially more data is guaranteed to improve results.

	Accuracy	RMSE	AUROC	LogLoss
GBM	0.880	0.291	0.940	0.282
RF	0.869	0.300	0.933	0.303
ET	0.858	0.304	0.935	0.294
KerasTF-NN	0.861	0.303	0.942	0.292
PyTorch-NN	0.875	0.300	0.950	0.283

As can be seen in the table above, all Python models perform as well or slightly better than their R counterparts with no tweaking, hyperparameter selection or cross-validation at all.

3.3 Python with own preprocessing

The authors preprocess the data as part of the script that produces Table 2. In the previous section I used the final dataset after being preprocessed by the authors. In this section, I want to see whether preprocessing it myself will affect the results.
Specifically, I performed the following steps using the fastai library:

1. I standardized all continuous variables in the training set to have mean 0, variance 1 by subtracting the mean and dividing by the standard deviation, and then subtracted and divided by the same numbers for the corresponding columns in the test set (unnecessary for tree-based models but crucial for linear models and NNs).

2. For each variable which had any missing values, I replaced those with an indicative value (like -1) and added a column which indicates which rows had a missing value for that variable.

3. I did not one-hot-encode any variables, as tree-based models do not require one-

hot-encoded categorical variables since they only care about the order and not the actual numerical value, and one-hot-encoded variables do not leverage the embedding layers in the PyTorch NN.

I then ran the exact same models as before:

	Accuracy	RMSE	AUROC	LogLoss
GBM	0.895	0.289	0.944	0.280
RF	0.878	0.296	0.941	0.296
ET	0.868	0.299	0.941	0.290
KerasTF-NN	0.861	0.298	0.946	0.284
PyTorch-NN	0.878	0.288	0.951	0.270

As can be seen in the above table, my preprocessing improved the results for every single model (higher AUROC and lower RMSE and LogLoss). The PyTorch NN has seen the biggest improvement in RMSE and LogLoss and is clearly the winner, as it has the lowest RMSE and LogLoss and highest AUROC, likely because it can now fully leverage its embedding layers for the categorical variables and so use a richer representation than any of the other models.

4 Conclusions

When compared to the well-informed GAM, which is likely significantly more accurate than the standard linear model used in most analyses, the GBM model reduced the LogLoss error by 14%. The GBM model is also easier and faster to use, as it can easily handle a large dataset and many covariates, as it identifies and uses the best predictors while being unlikely to overfit and robust to outliers. Implementations also usually in-

cludes a variety of hyperparameters which may be tuned in order to provide the best performance for the task at hand.

The PyTorch-NN (together with slightly better preprocessing) reduced the LogLoss error by 18%. However, it is more complicated to use, and is unlikely to massively outperform tree-based models unless significant amounts of labeled data are readily available or the relationships in the data are very complex.

In my opinion and as is evident in this analysis, **all scientists or researchers who work with data - especially if they are interested in predictive inference - should receive comprehensive data science training in order to have the appropriate toolbox for their analyses**, including being able to perform their own data preprocessing and intelligent model selection, which should extend beyond linear models, considering how most real data is inherently nonlinear.

Understanding how tree-based models work is not complicated, and using such models is usually a few lines of code in whichever language the researchers utilize, and so I believe that this distinguished Board will easily be able to add more data science into the curriculum at the Harvard School of Economics, to the benefit of the entire discipline and consequently the world.

5 References

1. Chipman, H. A., George, E. I., & McCulloch, R. E. (2010). BART: Bayesian additive regression trees. The Annals of Applied Statistics, 4(1), 266-298.

2. Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Ma-

chine learning, 63(1), 3-42.

3. Montgomery, J. M., & Olivella, S. (2016). Tree-based models for political science data. http://pages.wustl.edu/montgomery/trees. Replication Data: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/8ZJBLI.

4. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

5. Smith, L. N. (2017, March). Cyclical learning rates for training neural networks. In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on (pp. 464-472). IEEE.

6 Appendix

In this folder, you will find all the code that was used in order to produce the tables shown here, as well as a readme file detailing the contents of each of the code files. I apologize for the lack of comments and documentation - I ended up not having enough time to pretty it up.