Aix Marseille School of Economics

Master 2: Econometrics, Big Data and statistics

Prediction methods: Mid-term evaluation

Author:
Coraline BEST

Professor: Pierre MICHEL

February 6, 2024

Table of contents

1 Introduction

The XGBoost algorithm was first introduced by T. Chen and Guestrin 2016. It is an extension of the Gradient Boosting algorithm (GB) proposed by Friedman 2001. This machine learning algorithm serves for both prediction and classification purposes. It builds on Classification and Regression Trees (CART) using the GB framework and augmenting it with an improved way to penalize the complexity of the tree. The method raised to popularity due to its success in multiple forecasting competitions, especially on Kaggle.[1]

One can wonder why did XGBoost became such a popular model compared to GB while being so similar to it. The differences between the two can be summarized in two main points. First, XGBoost has more parameters allowing for refined tuning, coming from its regularization of the variables. These new parameters allow XGBoost to be less prone to overfitting and allow for better generalization. Second, the XGBoost package comes with multiple advantages. It allows for parallel computing making it faster to fit. It also has built in capacities to deal with missing values. These two characteristics, variable regularization and added features of the package make XGBoost suited for most prediction exercises and relatively cost efficient compared to other machine learning models. This makes XGBoost a Swiss army knife for forecasting often used as benchmark model.

Applications in economics
Because of its performances and popularity the XGBoost algorithm has made its way into social science and economic research. Its applications are becoming more and more widespread and cover multiple domains.

One popular application represents the usage of XGBoost estimations as benchmark for other forecasting models like in Flachaire et al. 2022 or Lolić, Sorić, and Logarušić 2022 who additionally leverage their results to identify new features for economic policy uncertainty forecasting. Secondly, it has been used in more complex models for specific estimation problems like in Viviano and Bradic 2023 and J. Chen and Ritzwoller 2023. Here, for instance, an XGBoost model is integrated in a larger algorithm to estimate treatment effects in specific contexts. Other applications go even further and use XGBoost for direct estimation of economic indicators like Blumenstock 2018, who tries to predict wealth using survey data collected from cellphone digital footprints. Due to sparse official data, Qureshi, Chu, and Demers 2020 propose using XGBoost to forecast the Canadian GDP growth based on Google Trends data. XGBoost also finds numerous applications in Finance. For example, Deng et al. 2023 propose the algorithm for stock index direction predictions using investor sentiments. Even intergovernmental organisation such as the OECD 2019 advocate the utility of XGBoost for economic modelling.

2 Materials and methods

2.1 Presentation of the algorithm

The XGBoost model is a supervised algorithm based on boosted Classification and Regression Trees. Before presenting the mathematical derivation, I first outline a more intuitive explanation of the

[1] Link to a non exhaustive list of successful Kaggle applications

algorithm using the Gradient Boosting algorithm (GB) as reference.

Intuitive presentation of the algorithm
Step 1 Initialization: XGBoost assigns a random initial prediction value $\hat{y}_0$ of the target variable to *all* training observations $i = 1, ..., n$.

Step 2 Growing trees: Similar to GB, XGBoost fits a tree model based on the prediction errors (pseudo residuals) of the previous predictions $\hat{y}_{t-1}$. More precisely, at each t^{th} iteration, starting from the initial prediction value $\hat{y}_0$, we calculate the residuals $\delta_t^{(i)} = y_{true}^{(i)} - \hat{y}_{t-1}^{(i)}$ for all i observations and create a tree that fits all variables $x_1, ..., x_p$ on the computed pseudo residuals δ_t.

However, when growing the tree, unlike Gradient Boosting, XGBoost does not use an impurity measurement (e.g., squared residuals or Gini Impurity) to choose the best decision rule among all candidate splits (s, j).[2] Instead, it decides upon the best split using a measurement of similarity *Gain* for each candidate split (s, j). Based on similarity scores of the parent and each child node, the *Gain* quantifies the improvement in similarity of the residuals within each node when splitting the parent node into two child nodes.[3] More precisely, the *Gain* is the difference between the similarity score of the parent node and the sum of similarity scores of the potential child nodes. At each iteration, the candidate split $(s, j)^*$ providing the highest *Gain* is chosen as decision rule for the m^{th} branch in the tree.

To avoid overfitting, the XGBoost algorithm includes an automatic pruning method. More concretely, we only keep those branches (binary splits) that exceed a minimum threshold in similarity *Gain* denoted by hyperparameter γ. Moreover, a regularization hyperparameter λ is included to scale the similarity scores and control for leaves containing only few training samples which otherwise would have misleadingly high similarity scores and thus higher *Gain* measures, indicating a good decision rule.

Step 3 Estimating: Analogously to GB, combining the initial estimation with the estimated pseudo residuals multiplied by learning rate η gives us a new estimation $\hat{y}_t = \hat{y}_{t-1} + \eta\ f_t(x)$ with $\hat{\delta}_t = f_t(x)$. The learning rate scales the tree's estimation and ensures a slow convergence of the ensemble model towards y_{true}, and thus avoids overfitting.

Step 4 Iterating: Based on the new estimations $\hat{y}_{t=1}$, we compute the new pseudo residuals $\delta_{t=2}$ on which we fit our new model $f_{t=2}(x) = \hat{\delta}_{t=2}$. Put differently, based on the new residuals, we repeat steps 2-3 and obtain $\hat{y}_{t=2} = \hat{y}_0 + \eta\ [f_{t=1}(x) + f_{t=2}(x)]$.

Step 5 Update and Convergence: We repeat steps 2-4 until the desired maximum number of XGBoost trees T are estimated or the pseudo residuals, i.e. increases in accuracy, become insignificantly small. More concretely, we get $\hat{y} = \hat{y}_{t-1} + \eta\ \sum_{t=1}^{T} f_t(x)$.

[2] with $s \in S(x_j)$ denoting all possible thresholds for the respective variable j

[3] For that matter, the similarity score quantifies the homogeneity of the residual values in each node and is computed by dividing the squared sum of residuals by the number of residuals in node m multiplied by a regularization parameter λ: $similarity\ score_m = \frac{1}{n_m\ \lambda} \left[\sum_{i=1}^{n_m}\ \delta_i\right]^2$ with n_m denoting the total number of training samples per node.

Mathematical derivation of the algorithm

This is a single slow learner tree that is progressively updated on the data instead of averaging the prediction of multiple different trees like in a Random Forest. The slow learner prediction is given by $\hat{y}_i^t = \sum_{k=1}^{n} f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i)$. Simply put we see that the prediction at step t is the updated prediction of step $t-1$ by the new tree $f_t(x_i)$. To find the optimal parameters for the best forecast, we have the following objective function to minimize :

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{i=1}^{n} \omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}^{t-1} + f_t(x_i)) + \sum_{i=1}^{t} \omega(f_i) \tag{1}$$

In this formula we try to minimize $\sum_{i=1}^{n} l(y_i, \hat{y}_i^t)$, the sum of all individual prediction error based on a given loss function. This problem is augmented by $\sum_{i=1}^{t} \omega(f_i)$ which constrains the complexity of the trees. The first trick of the XGBoost algorithm is to approximate the loss by its Taylor expansion up to the second order. Indeed, besides the Mean Squared Error, many loss functions can be cumbersome to optimize because they might not be convex. To solve this, XGBoost directly takes the Taylor expansion of $l(y_i, \hat{y}^{t-1} + f_t(x_i))$ up to the second degree and plugs it in equation 1. After removing the constant terms we find :

$$\mathcal{L} = \sum_{i=1}^{n} \left[\partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{t-1}) f_t(x_i) + \frac{1}{2} \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{t-1}) f_t^2(x_i) \right] + \omega(f_i) \tag{2}$$

This way we find that the objective function only depends on the two partial derivatives, and makes it able to support any loss function. The function is now approximated around 0 by a quadratic, convex, function. This also improves the computational cost of the model. Another specificity of the XGBoost algorithm is the use of a penalization parameter on the complexity of the tree, noted by $\omega(f_i)$.

$$\omega(f_i) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2 \tag{3}$$

This term gives us two new hyper-parameters, γ and λ. For a tree f_i its number of leaves T is constrained by γ. Constraining the number of leaves will force the tree to be more general by grouping more observations in the same leaf. The second variable, ω_j, is the score (meaning the prediction) of leaf j. The λ term will be better understood when we will look at equation 8. By augmenting the loss function by the squared sum of predictions, it will force the model to only create new split if the change in the sum of prediction is significantly large.

We now see how equation 2 can be rearranged to give a closed form to expression to the parameters of interest. We first note that the prediction of leaf t from a given tree is noted as $f_t(x) = \omega_{q(x)}$, q being the function mapping each data point to one leaf. Therefore a tree $f(x)$ can be seen as the sum of the prediction from each leaf, represented by the sum $\sum_{j=1}^{T}$. From this we rewrite the objective function from equation 2, using the notation $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{t-1})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{t-1})$:

$$\mathcal{L} = \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) \omega_j + \frac{1}{2} ((\sum_{i \in I_j} h_i) \omega_j^2) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2 \tag{4}$$

$$= \sum_{j=1}^{T} \left[G_j \omega_j + \frac{1}{2} ((H_j + \lambda) \omega_j^2) \right] + \gamma T \tag{5}$$

Note that in equation 5 the expression was further simplified using $\sum_{i \in I_j} g_i = G_J$ and $\sum_{i \in I_j} h_i = H_J$. We first notice when comparing equations 2 and 4 the change of index of the total sum. We go from the sum on all observations to the sum of all leaves. We use the fact that all the observations in a single leaf have the same prediction to make this change. This is made clearer by the index $i \in I_j$, which indicates all the observations falling in leaf j. This notation appears for the sums on g_i and h_i. In equation 5, all the terms depend on the tree structure given by q, selected endogenously. Because of that they are independent from each others. In addition the expression is quadratic, depending only on G_j, H_j, λ. For a structure q, we can easily find the expression for the score for each leaf w_j^* minimizing the loss and the optimal loss $\mathcal{L}^*$.

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{6}$$

$$\mathcal{L}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{7}$$

Even with these results, it is not possible to test them for any tree structure. Instead, like said in the introduction the trees are grown step by step. With equations 7 we can construct the splitting rule:

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \tag{8}$$

Simply put, if the gain is positive or above a given value we accept this new split. $\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}$ represents the increase in score from the new left and right leaves (the two first fractions) minus the score of the old leaf (third fraction) compared to the complexity parameter γ.

From the precedent equations we only see γ and λ. The first one, γ directly controls the number of leaves by restraining the expression of the gain as seen in equation 8. The second, λ, controls the regularization strength. Increasing it will decrease the impact of single tree on the gain, making the model more conservative. In addition to these two we can add a new one coming from the boosting structure. Boosting models are slow learning models that are continuously updated on the residuals. One way to control for overfitting is to directly reduce the impact that a new tree has on the prediction. This is done with the learning rate parameter η as we see it in $\hat{y}^t = \hat{y}^{t-1} + \eta f_t(x_i)$. By reducing the impact of one new split on the prediction we avoid to put too much importance on a split that might be bad or too specific to our training data. Finally some other parameters could be considered as hyperparamters. Indeed, as we use a regression tree, instead of using λ, γ, η to control overfitting we can directly control the shape of the tree. These parameters can be maximum tree depth, maximum number of leaves or minimum number of observations per leafs. Directly controlling the shape forces the tree to be less specific and therefore less specific to one training sample.

The second question concerns how exactly splits are determined. The Gain equation 8 and the different hyperparameters tell if we can accept a new split or the total number of splits allowed. However, because not all possible combinations can be tested, especially with continuous variables, the way a splitting variable and splitting threshold is chosen is algorithmic. As it is a computer science problem I leave it to the reader looking deeper in T. Chen and Guestrin 2016.

2.2 Data used

To implement the XGBoost algorithm I use the California House Price dataset, which comprises data from the 1990 California census. The objective of this project is to train an XGBoost model that is able to predict median house values in California. The dataset consists of 20,640 observations and encompasses 10 numerical features:

Longitude: indicates how far west a house is, higher values represent a more westerly location.

Latitude: indicates how far north a house is, higher values represent a more northerly location.

Housing median age: median age of a house within a block, lower values indicate newer buildings.

Total rooms: total number of rooms within a block.

Total bedrooms: total number of bedrooms within a block.

Population: total number of people residing within a block.

Households: total number of households within a block, where a household is a group of people residing within a home unit.

Median income: median income for households within a block (measured in tens of thousands of US dollars).

Median house value: median house value for households within a block (measured in US dollars).

Ocean proximity: indicates the location of the house with respect to the ocean sea.

The XGboost algorithm is robust to the scale of features, eliminating the need for data standardization. This resilience is attributed to XGBoost's reliance on decision trees, where splitting points are determined by relative order rather than absolute values. After preprocessing the dataset, I partition it into two subsets: a 80% training set and a 20% testing set. I conduct 5-fold cross-validation, training the XGBoost model and assessing its performance on each fold. I subsequently predict on the test set. To evaluate prediction performance, I employ the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{9}$$

calculated on the test sample. The objective is to minimize the RMSE, signifying optimal predictive performance. Additionally, I perform grid search, exploring various values for the learning rate η, regularization parameter λ and the maximum depth to fine-tune the model.

3 Results

This section begins by outlining descriptive statistics for the dataset, followed by a presentation of the results obtained through the utilization of the XGBoost model.

3.1 Descriptive statistics

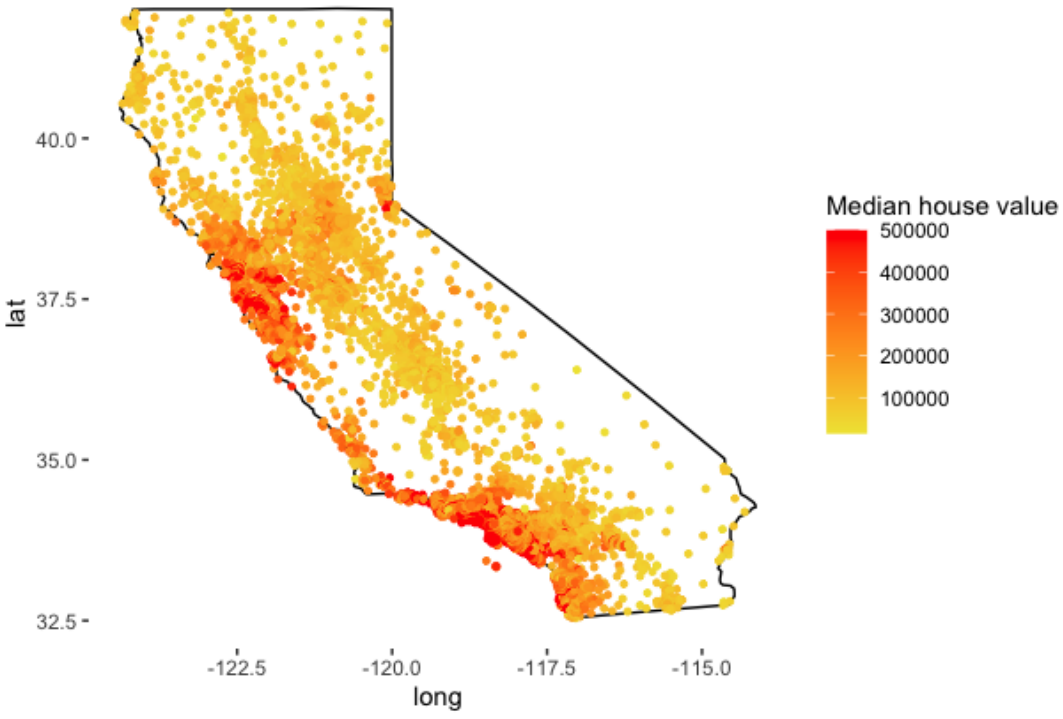
Table 1 displays descriptive statistics about the data, such as the min, max, median and mean.

Table 1: Descriptive Statistics

	Min.	Median	Mean	Max.
longitude	-124.3	-118.5	-119.6	-114.3
latitude	32.54	34.26	35.63	41.95
housing_median_age	1.00	29.00	28.64	52.00
total_rooms	2	2127	2636	39320
total_bedrooms	0.0	431.0	532.5	6445.0
population	3	1166	1425	35682
households	1.0	409.0	499.5	6082.0
median_income	0.4999	3.5348	3.8707	15.0001
ocean_proximity	-	-	-	-
median_house_value	14999	179700	206856	500001

The following map illustrates the geographical distribution of house values in California. The red points represent the most expensive houses, predominantly situated along the coastal areas near the ocean. On the other hand, the yellow points denote the least expensive houses, primarily found inland.

Figure 1: Distribution of Median House Prices in California



3.2 Interpretation of the results

In this subsection, I will delve into the results and the selection of hyperparameter values.
Regarding the objective function, I opted for the default choice which is regression with squared loss. I used a grid search for the learning rate (eta), a parameter governing the step size shrinkage during updates to prevent overfitting. Eta scales down the feature weights after each boosting step, contributing to a more conservative boosting process. While the default value for eta is typically 0.3, I conducted experiments with eta values of 0.01, 0.1, 0.2, and 0.3. Additionally, I performed a grid search for the max_depth parameter, representing the maximum depth of a tree. Increasing this value enhances model complexity but also raises the risk of overfitting; a value of 0 implies no depth limit. I explored max_depth values of 3, 6, and 9. I also did a grid search for lambda with values 1 and 5. I set alpha to 0.1, representing the L1 regularization term on weights, which makes the model more conservative with higher values. RMSE gauges the average difference between predicted and actual values, providing insights into prediction performance. Optimal model performance is associated with lower RMSE values, indicative of minimal errors between predicted and true outcomes. In my specific case, the lowest RMSE is achieved when eta is set to 0.2, lambda equal to 5 and the maximum depth of the tree is 9. The table illustrates that an excessively small eta value leads to a significantly elevated RMSE, signaling subpar prediction performance. Smaller eta values, like 0.01, suggest a more conservative learning approach, where the model adjusts parameters cautiously, taking smaller steps during training. Conversely, larger eta values, such as 0.30, prompt a more aggressive learning process, with the model adapting rapidly and taking larger steps in parameter space during training. The choice of eta is pivotal, striking a balance between model stability and its ability to capture intricate patterns in the data. Additionally, a maximum depth of 3 yields to a higher RMSE as it implies a simpler tree. Conversely, higher values like 9 result in a more complex tree, potentially capturing more intricate patterns but also risking overfitting. The selection of max depth involves a trade-off between model complexity and the risk of overfitting.

The table generated below is presenting the Root Mean Squared Error (RMSE) results obtained from training the XGBoost model with different combinations of hyperparameters:

Table 2: RMSE Results

eta	**max_depth**	**lambda**	**RMSE**
0.01	3	1	113767.15
0.10	3	1	55284.04
0.20	3	1	51891.96
0.30	3	1	51083.82
0.01	6	1	106748.08
0.10	6	1	49122.12
0.20	6	1	48929.04
0.30	6	1	49234.27
0.01	9	1	103236.08
0.10	9	1	48560.69
0.20	9	1	49000.99
0.30	9	1	50102.46
0.01	3	5	113994.47
0.10	3	5	55430.00
0.20	3	5	52115.58
0.30	3	5	51280.52
0.01	6	5	108017.10
0.10	6	5	49312.46
0.20	6	5	48513.93
0.30	6	5	48900.56
0.01	9	5	105732.60
0.10	9	5	48342.00
0.20	9	5	**48294.00**
0.30	9	5	49002.28

When using the best combination of hyperparameters I obtain an out of sample RMSE of 45228.43 on the test data. It is a surprising result to find a test RMSE smaller than the train RMSE but it can be explained in a few possible ways. First of all this result can happen just because of chance, but might not be reproduced consistently. Also the final model uses more data for training than the cross validation models, which improves its performance. Finally the specific shuffling of the data between training and testing might have caused this result if the distribution of the characteristics are different between training and test data.

4 Conclusion

XGBoost is a powerful and widely used machine learning algorithm, but like any method, it has both advantages and disadvantages. It often stands out comparing to different other tree-based algorithm due to its superior performance, regularization techniques, and versatility. Particularly noteworthy is XGBoost's tendency to outperform Random Forests in predictive accuracy, attributed to its adept handling of gradient boosting and incorporation of effective regularization features. Moreover XGBoost is very easy to implement and can be run efficiently due to its parallel processing, making it significantly faster than traditional gradient boosting algorithms. XGBoost has also a built-in capability to handle missing data, eliminating the need for imputation or pre-processing steps.

XGBoost includes regularization terms (L1 and L2 regularization) which help prevent overfitting. XGBoost can handle a variety of data types, including numerical and categorical features. It also supports different loss functions and evaluation metrics, making it versatile for various types of problems. While not covered in the homework XGBoost provides a feature importance score, allowing users to understand which features contribute the most to the model's predictions. The algorithm's pruning step further refines the model by eliminating branches that contribute minimally to predictive power, reducing the final model's complexity. However, it's important to note that XGBoost comes with its challenges. XGBoost has several hyperparameters, and finding the optimal combination can be challenging. As I have seen, the performance of XGBoost is highly dependent on the choice of hyperparameters. Selecting the right combination requires careful consideration and experimentation. The models created by XGBoost, especially with deep trees, can become complex and challenging to interpret compared to simpler models like linear regression. In conclusion, XGBoost's advantages contribute to its popularity in various machine learning tasks. However, users should be aware of its complexity, potential challenges in interpretability, and the importance of parameter tuning. Depending on the specific characteristics of the data and the problem at hand, other decision-tree-based methods like Random Forests may also be considered.

References

Blumenstock, Joshua E. (2018). "Estimating Economic Characteristics with Phone Data". In: *AEA Papers and Proceedings* 108, pp. 72–76.

Chen, Jiafeng and David M. Ritzwoller (2023). "Semiparametric estimation of long-term treatment effects". In: *Journal of Econometrics* 237.2, Part A, p. 105545.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754.

Deng, Shangkun et al. (2023). "Stock index direction forecasting using an explainable eXtreme Gradient Boosting and investor sentiments". In: *The North American Journal of Economics and Finance* 64, p. 101848. ISSN: 1062-9408. DOI: https://doi.org/10.1016/j.najef.2022.101848. URL: https://www.sciencedirect.com/science/article/pii/S1062940822001838.

Flachaire, Emmanuel et al. (2022). *GAM(L)A: An econometric model for interpretable Machine Learning*. arXiv: 2203.11691 [stat.ML].

Friedman, Jerome H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5, pp. 1189–1232.

Lolić, Ivana, Petar Sorić, and Marija Logarušić (2022). "Economic Policy Uncertainty Index Meets Ensemble Learning". In: *Computational Economics* 60, pp. 401–437. DOI: https://doi.org/10.1007/s10614-021-10153-2. URL: https://link.springer.com/article/10.1007/s10614-021-10153-2#citeas.

OECD (2019). *Economic Modelling Machine Learning*. World Economic and Financial Surveys. Organisation for Economic Co-operation and Development.

Qureshi, Shafiullah, Ba M. Chu, and Fanny S. Demers (2020). *Forecasting Canadian GDP growth using XGBoost*. Carleton Economic Papers 20-14. Carleton University, Department of Economics. URL: https://ideas.repec.org/p/car/carecp/20-14.html.

Viviano, Davide and Jelena Bradic (2023). "Synthetic Learner: Model-free inference on treatments over time". In: *Journal of Econometrics* 234.2, pp. 691–713.