A Comparison Between Machine Learning and Classic Vector Autoregressive for GDP forecast

Author:
Kenny Xavier Monar Aguilar

Director:
Joaquim Gabarro

A thesis presented for the degree of
MSc in Statistics

Departamento de Estadística e Investigación Operativa
Universitat Politècnica de Catalunya
Barcelona - Spain

Contents

1 Summary

GDP Forecasting: Machine Learning, Linear or Autoregression? −¿ revisar y añadir a la bibliografía

In the recent years there has been an explosive increase in the number of research papers using machine learning methods for forecasting. In this work, I will focus on comparing the estimation of GDP using classical and machine learning methods. In particular, I am interested in analyzing if the index S&P 500 can help forecast the GDP, therefore I will take as a basis a work where a VAR model is used to analyze the relationship between GDP and S&P 500. From there I use three recently developed implementations of the gradient boosting decision tree to model GDP and S&P 500. The recent implementations are XGBoost, LightGBM and CatBoost and are very famous because they are widely used in winner solutions in Kaggle competitions. The metric I use to do the comparisson is the Mean Squared Error and by using the three machine learning algorithms, I find that they provide better results than the Vector Autoregressive. I also perform grid search with several parameters, taking into account regularization, to avoid overfitting and obtain the lowest mean squared error. To select the best model I consider the evaluation results along with the overfitting measure and finally I take a look to the prediction charts of all the algorithms. In my opinion XGBoost offers the best predictions in this exercise.

2 Introduction

In this work, I will reference the paper published by Ros(2019)[1] as it estimates a VAR for GDP and S&P 500, therefore I will follow his methodology to obtain the data and determine the order of the VAR model. Thus, the GDP calculated using the expenditure approach will be used.

It is interesting to verify if the S&P 500 can be used to forecast GDP since it is the most commonly used benchmark for determining the state of the overall economy in the U.S [2] and it has a higher frequency which makes it an interesting candidate to be used as leading indicator of the GDP through the quarter.

The main reference related to the estimation of the VAR is in the paper by Ros(2019)[1]. In the mentioned document a VAR of order 2 is estimated and S&P seems to be helpful in forecasting GDP growth.

I have collected a few articles where machine learning methods are compared with classic methods. Those articles have very interesting conclusions regarding the use of machine learning to study economic variables. However, I was not able to find a comparisson between gradient boosting and VAR analyzing the relationship between the US GDP and the stock market, therefore I think it would be an interesting exercise.

2.1 Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction 2014

In the paper by Adebiyi [3], the authors use historical daily stock prices for the study. The data consists on open price, low price, high price, close price and volume traded. The authors select the closing price to represent the price of the index to be modeled. The index selected is the Dell Inc. stock data from 1988 until 2011, which correspond to 5680 observations. The mentioned series have random walk pattern and doesn't show global trend or seasonality pattern as expected in financial series.

To determine the best ARIMA model, the criteria used is the to pick the p and q values that correspond to the lowest Bayesian Information Criterion (BIC) and Standard Error of Regression (SER)and a high adjusted R^2. In this case, the ARIMA(1,0,0) is selected.

Regarding the neural network, the authors employ a three layer (one hidden layer) multilayer perceptron model trained with the back propagation algorithm. The model was trained with 1000, 2000 and 5000 epochs while minimizing the mean squared error during training. The model with 10 input neurons, 17 hidden neurons and 1 output neuron was the most accurate in the experiments.

The authors conclude that both ARIMA and ANN models can achieve good forecast in real life problems. However, the predictive model using the ANN approach shows better performance over the ARIMA models. The reason is that the actual and predicted values obtained with the ANN model are very close while the ARIMA forecasting is directional.

2.2 How is Machine Learning Useful for Macroeconomic Forecasting? 2019

In the paper by Coulombe [4], the authors study different horizons and variables to understand the usefulness of the features driving ML gains over econometric methodologies. They analyze four features: non linearities, regularization, cross validation and alternative loss function. The effect in data rich and data poor environments is studied. The authors designed experiments to allow easy identification of the treatment effects of interest.

The authors try to improve the understanding of each ML method properties rather than the selection of a single winner model for a specific target. The main question is "What are the key features of ML modeling that improve the macroeconomic prediction?". The exercise followed by the authors consists of an extensive forecasting race between many models that differ regarding nonlinearity, regularization, hyperparameter selection and loss function.

The five macroeconomic variables analyzed are: industrial production, unemployment rate, consumer price index, difference between 10 year treasury constant maturity rate and federal funds rate and housing starts. Those are considered as representative of US economy and almost 40 years of data are used in the study.

Regarding the results, first, non linearities have benefits for industrial production, unemployment rate, term spread and increase with longer horizons, especially when combined with factor models. Nonlinearity is harmul for inflation and housing starts.

Second, regarding big data, alternative regularization methods such as Lasso, Ridge, Elastic-net don't improve over the factor model. This suggests that the factor representation of the economy is accurate as a mean of dimensionality reduction.

Third, the hyper parameter selection using k-fold cross validation does better on average that any other criterion and the next best criteria is BIC. It suggests that ignoring information criteria is not harmful when analyzing more complex ML models.

Fourth, replacing the standard in sample quadratic loss function by the e-intensive loss function in support vector regressions is not useful. Finaly, the marginal effects of big data are positive and significant or real activity series and term spread and increase with longer horizons.

2.3 GDP Forecasting: Machine Learning, Linear or Autoregression? 2021

A recent work, from Maccarrone 2021[5], compares classical models such as SARIMAX and linear regression with K-Nearest Neighbour. It utilizes two forecasting strategies: one-step-ahead and multi-step-ahead forecast and evaluate the performance using the mean squared error.

The authors compare the predictive power of different models to forecast the real U.S. GDP. With quarterly data going from 1976 until 2020, the authors conclude that the model KNN captures the self predictive ability of the U.S. GDP and the mentioned ML technique performs better than traditional time series analysis.

The authors also include macroeconomic variables in order to increase the level of forecasting accuracy. They find that the predictions are improved only when considering long forecast horizons. The authors also conclude that the use of machine learning algorithm provides additional guidance for data-driven decision making.

Regarding methodology, the authors use AR and SARIMA as a benchmark for time series analysis and then forecast the U.S. GDP with models ARX, SARIMAX and linear regression to be able to add some measures of economic activity. Finally the authors use the k-nearest neighbour methodology. The main goal of the authors is to achieve forecasts with high accuracy and with high degree of explainability.

The strategies used to study the accuracy of the GDP predictions are one-step-ahead and multi-step ahead forecasting.

On one hand, the KNN achieves the best performance for the one-step-ahead strategy suggesting that the repetitive patterns in the GDP increases the forecast accuracy. On the other hand, it loses predictive power when the forecast is performed for a longer horizon. SARIMA performs poorly in both strategies, but when including covariates, SARIMAX obtains a lower error in the one-step-ahead strategy. ARX achieves the best forecasting performance in one-step ahead when using proxies for the yield curve. The Linear Regression achieves the best performance in the multi-step-ahead forecast using proxies for the yield curve and macro variables. The results suggest to use KNN model for one-step-ahead forecast and Linear Regression with financial variables for multi-step ahead forecasts. In that way, the multi-step provides a long-term vision for planning in advance investments, monetary policy, etc. while on the other hand, the one-step-ahead might support possible refinement around the decisions taken.

2.4 Boosted Embeddings for Time Series Forecasting 2021

In the paper by Karingula[6], the authors propose a time series forecast model called DeepGB. They formulate and implement a Gradient boosting where the weak learners are deep neural networks. In the paper, the authors demonstrate that the model outperformns the comparable state of the art models using real world sensor data and public dataset.

The main idea is that the neural networks are able to represent non linear relationships well. It is known by the universal approximation theorem that neural networks can approximate any non linear region although it might need an exponential number of nodes/features for the approximation to work.

On the other hand, gradient boosting can't handle complex arithmetic relationships such as multiplication of elements. The authors combine different models by creating small parallel models for a single task. Then the next model will attempt to solve only the gradient of the loss function. In this way, the process is simpler than performing a full regression on the output range.

The authors experimented using data available in Kaggle to monitor device health and wikipedia data. Regarding device monitor, the authors used the connections per seconds from devices during a month. It is a difficut rate to forecast. Regarding wikipedia data, the authors attempt to forecast the number of page access per day.

The authors first went through ARIMA, DeepAR, neural prophet and seq2seq models. Finally the authors conclude, based on SMAPE, that DeepGB outperformed the rest of the models in the two datasets analyzed.

2.5 Exploring what stock markets tell us about GDP in theory and practice 2021

In the paper by Ball [7], the authors study the relationship between stock market movements and GDP in real time. They use a simple theoretical model to clarify the relationship between both series, and finally explore the US GDP and S&P 500 series through different tools such as correlations, regressions, extreme value calculations, etc.

The GDP vintage and revised data used in the study go from 1999 until 2020 and it comes from the OECD database. The stock market data was obtained from Yahoo finance. The series are detrended and quarterly data is used.

For the regression analysis of real GDP and S&P 500, the authors considered as explanatory variables the contemporaneous value and up to 5 lags of real GDP and S&P 500. Among the findings, the authors see that lagged GDP values have the largest effects with negative coefficients which suggests mean reversion. On the other hand, the results show that an increase in the S&P 500 is associated with an increase of 0.05 of real GDP. However, considering the three significant lags, the cumulative effect of an increase in the S&P 500, over the past year, is associated with an increase of 0.46 increase in real GDP. When running the same regressions for vintage GDP the results were similar but less significant.

The main findings of the authors are that S&P 500 is weakly correlated with real GDP, but strongly and significantly correlated with one lag as the theory predicts. The authors also find that the S&P 500 is more closely related to final, revised GDP numbers than to vintage GDP estimates, it suggests that stock market trends are informative about the true GDP.

3 Models

3.1 Vector Autoregressive

The VAR model used to determine the relationship between the growth rate of the GDP and the growth rate of the index S&P 500:

$$dGDP_t = \beta_{1,11} dGDP_{t-1} + \cdots + \beta_{p,11} dGDP_{t-p} + \beta_{1,12} dSP500_{t-1} + \cdots$$
$$+\beta_{p,12} dSP500_{t-p} + \epsilon_{1,t}$$
$$dSP500_t = \beta_{1,22} dSP500_{t-1} + \cdots + \beta_{p,22} dSP500_{t-p} + \beta_{1,21} dGDP_{t-1} + \cdots +$$
$$+\beta_{p,21} dGDP_{t-p} + \epsilon_{2,t}$$

To determine the order of the VAR model, I will use Akaike's information criteria (AIC). The model estimated with the lowest AIC corresponds to the optimal lag p. The results of the Bayesian Information Criteria(BIC), the Hannan and Quinn criterion (HQIC) and final prediction error (FPE) are also calculated.

3.2 Gradient boosting

As commented by Brownie[8], gradient boosting refers to a type of ensemble machine learning algorithms used for regression or classification problems.

Any arbitrary and differentiable loss function is used to fit the model using the gradient descent optimization algorithm. Boosting consists in combining several decision trees, called weak learners, sequentially to obtain a powerful model.

Such ensemble algorithms are based on decision trees. The process is as follows: first a decision tree model is fit to explain the target variable using the explanatory variables and an error prediction is calculated. In the next step another decision tree model is fit to explain the prediction error from the previous step and a new error is calculated. This last step is repeated several times.

The idea behind this process is that the model learns slowly[9]. In every iteration the new trees fit the residuals from the previous step, improving the areas where the model has not performed well.

Gradient boosting is a machine learning algorithm frequently used to win machine learning competitions (like Kaggle) on tabular and similar structured datasets.

I will use the most recent implementations of the Gradient Boosting Decision Trees such as XGBoost, Light GBM and CatBoost as they have been reported in the forecasting community including Kaggle competitions as top options to tackle forecasting problems.

3.2.1 XGBoost

XGBoost[10] stands for eXtreme Gradient Boosting and it's an implementation of the gradient boosting decision trees algorithm. It is very popular in competitions such as Kaggle because it has a high predictive power and it is very simple to parameterize. It can be used for regression and classification problems.

According to the authors[11], the success of XGBoost lies in the scalability as it runs several times faster than other solutions. The innovations that allow XGBoost to scale include: a novel tree learning algorithm for handling sparse data, a weighted quantile sketch for efficient proposal calculation and a sparsity aware algorithm for parallel learning.

3.2.2 LightGBM

LightGBM[12] was developed after XGBoost and it tries to improve the scalability when data size and feature dimensions are high. One of the differences with respect to XGBoost is that instead of scanning all data instances to estimate the information gain of all possible points, LightGBM proposes two techniques called Gradient based one side sampling (GOSS) and exclusive feature bungling (EFB). With GOSS, a proportion of the data is excluded and only the remaining is used to estimate the information gain. With EFB, mutually exclusive features are bundled to reduce the number of features.

According to the authors' experiments, the LightGBM is 20 times faster than other implementations achieving almost the same accuracy.

3.2.3 Catboost

Catboost can be used for solving problems, such as regression, classification, multi-class classification and ranking. It was developed after XGBoost and LightGBM. The authors[13] demonstrate that the existing implementations of gradient boosting face a statistical issue called prediction shift, caused by a kind of target leakage. To solve this issue, the authors introduced two advances called ordered boosting and an algorithm for processing categorical features.

Ordered boosting is a modification of standard grading boosting algorithm

4 Experimental setup

4.1 Data

4.1.1 U.S. GDP

The GDP data is obtained from OECD[14]. The GDP growth rate is used for the analysis and it goes from Q2-1947 to Q4-2021.

In the chart we can see that the serie is stationary in mean if we don't consider the year 2020 where we had the covid lock down. For the purpose of this analysis I will use data only until 2019-Q4.

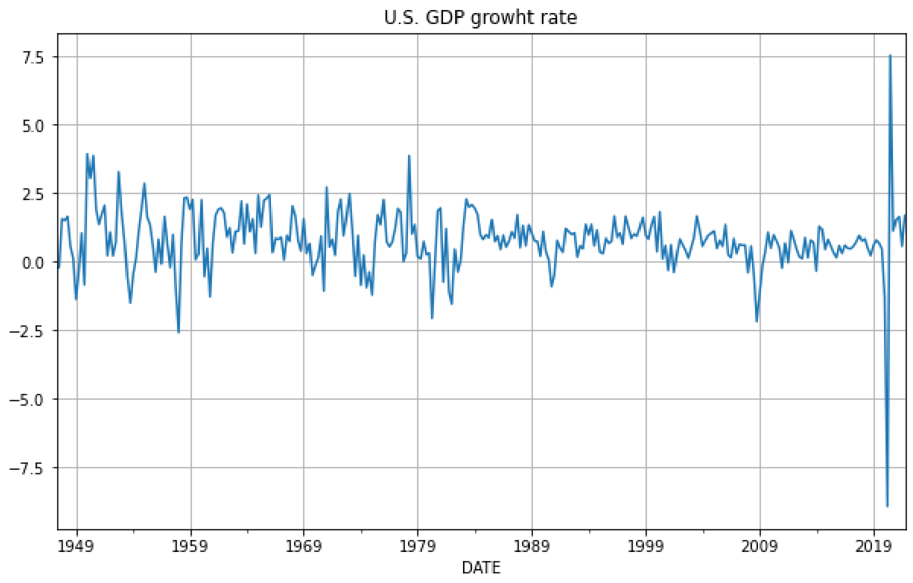

Figure 1: U.S. GDP ($dGDP_t$) 1947-Q2:2021-Q4

The series is $dGDP_t$ and it denotes the real growth of U.S. GDP at quarter t compared with the previous quarter t-1.

4.1.2 S&P 500

The S&P 500 data is obtained from the home page of Nobel Prize Laureate Rober J. Shiller[15]. Professor Shiller collects data of the Monthly Averages of Daily Closing Prices from 1871 for the S&P 500, consumer price index, etc.

First we need to calculate the real value of the S&P 500 index. We obtain it dividing the S&P 500 by the price index:

Let t denote the set of quarter time series:

t = 1947-Q2, 1947-Q3, ... , 2018-Q2, 2019-Q4

M_t denotes the Average Price per Share in Month Ending Quarter:

M_t = Monthly Average Price per Share at t

CPI_t denotes the consumer price index at quarter t:

CPI_t = consumer price index at t

P_t denotes the price index and measures how much prices have changed in any given year compared to a base year:

$$P_t = CPI_t/CPI_{base_period}$$

The final step is to divide M_t by P_t. Taking 1983-Q3 as the base period:

SP_t = Real Monthly Average Price per Share at t (chained 1983 dollar)

$$SP_t = M_t \ / \ P_t$$

Now as shown in the first chart of Figure 2, S&P 500 is not stationary. In the second chart, the growth rate dSP_t is shown and it is stationary. It is calculated as the log-return of the index at quarter t compared with the previous quarter t-1:

$$dSP_t = \log(SP_t) \text{ - } \log(SP_{t-1})$$

In the case of the index S&P 500, in the first chart of the Figure 2, the effect of the financial crisis of 2008 is very clear but we can't see a big effect in the case of the covid lockdown.

See below some statistics of the growth rate of the U.S GDP vs the growth rate of the S&P 500.

	count	mean	std	min	max
dGDP	291.0	0.778720	0.935266	-2.595440	3.930098
dSP	291.0	0.977199	6.704694	-29.018388	20.778526

Table 1: U.S. GDP ($dGDP_t$) and S&P500 (dSP_t)1947-Q2:2019-Q4

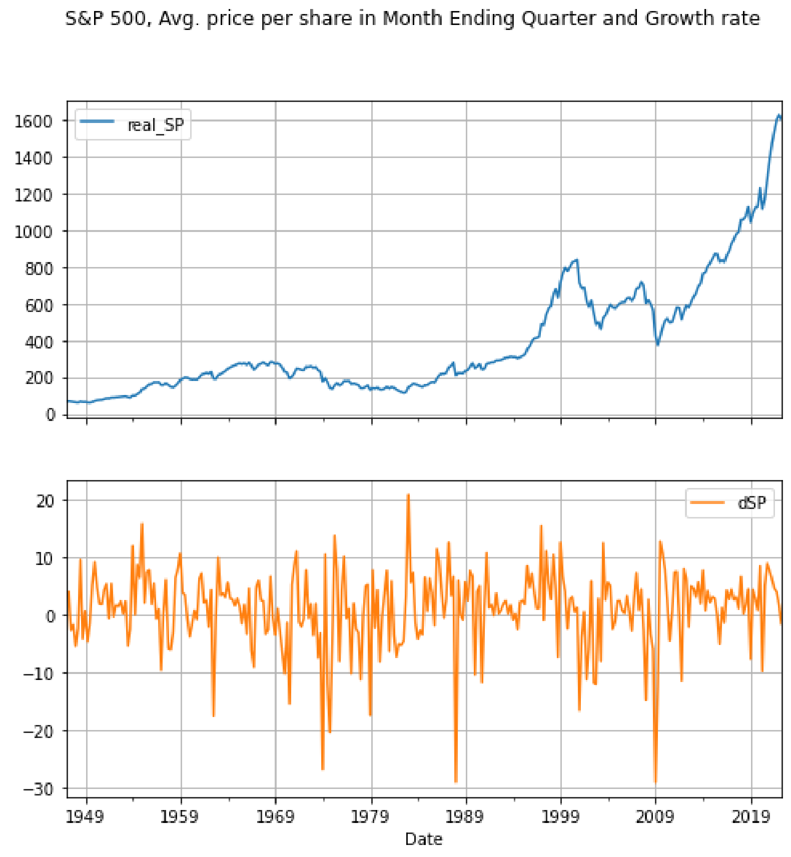

Figure 2: S&P 500 (SPt) and growth rate ($dSPt$) 1947-Q2:2021-Q4

The dataset is split between train and test datasets. The train data goes from 1947-Q2 until 2007-Q1 and the test data goes from 2007-Q2 until 2019-Q4. The comparissons among all algorithms are based only on one-step-ahead forecasting.

4.2 Hyperparameters tuning

As mentioned by Burkov(2019)[16], the hyperparameters have to be tuned by the data analyst. The idea is to find the best combination of hyperparameters that fit well the training data while at the same time ensuring that the model generalizes well.

Nowadays there are several frameworks to perform the hyperparameter tuning such as hyperopt and optuna. In this work I will be using grid_search by sklearn library. It consists on defining a set of values per each hyperparameter, then grid_search will combine all those values and train and validate as many models as combinations were created.

4.2.1 XGBoost

The hyperparameters I will check in XGBoost are:

max_depth:

max_depth sets the maximum depth of the decision trees.

max_leaves:

max_leaves is the maximum number of nodes to be added.

num_estimators:

num_estimators is the number of boosting rounds. It is equal to the number of boosted trees to use. Higher number of estimators translates in higher risk of overfitting.

reg_alpha and reg_lambda:

reg_alpha and reg_lambda are L1 and L2 regularisation terms, respectively. The greater these numbers, the model will be less prone to overfitting but might end up underfitting.

4.2.2 LightGBM

In this section I comment the hyperparameters I will be evaluating and the definitions come from the LightGBM web documentation[17].

num_leaves:

num_leaves is the main parameter to control the complexity of the tree model. In theory,by using the formula $num\_leaves = 2^{(max_depth)}$ we can obtain the same number of leaves as depth-wise tree. However, this simple conversion is not good in practice, thus, we should hyper tune this parameter along with max_depth.

max_depth:

max_depth can be used to limit the tree depth explicitly. It is specially useful to reduce the risk of overfitting when the data size is small.

num_estimators:

num_estimators also known as num_iterations and several other aliases. It specifies the number of boosting iterations (trees to build). As usual with decision tree models, the more trees translate to higher risk of overfitting.

reg_alpha:

reg_alpha corresponds to lambda_l1 and l1_regularization is constrained to values greater or equal than 0 and is used to deal with overfitting.

reg_lambda:

reg_lambda corresponds to lambda_l2 and l2_regularization is constrained to values greater or equal than 0 and is used to deal with overfitting.

4.2.3 CatBoost

In this section, I include the hyperparameters I analyzed. The definitions come from the website[18]

depth:

In most cases, the optimal depth ranges from 4 to 10. Values in the range from 6 to 10 are recommended. The maximum value allowed is 15.

l2_leaf_reg:

l2_leaf_reg is the coefficient at the L2 regularization term of the cost function. Any positive value is allowed.

learning_rate:

learning_rate is used for reducing the gradient step. By default, the learning rate is defined automatically based on the dataset properties and the number of iterations.

The possible ways of adjusting the learning rate depending on the overfitting results are:

- If there is no overfitting in the last iterations of the training then increase the learning rate.
- If overfitting is detected then decrease the learning rate.

num_estimators:

num_estimators is also known as iterations or num_trees. By default, CatBoost builds 1000 trees but I will reduce the number of iterations to speed up the training.

When the number of iterations decreases, the learning rate has to be increased. The default learning rate can be tuned to get the best possible quality. To tune the learning rate is advisable to look at the evaluation metric values on each iteration:

- Decrease the learning rate if overfitting is observed.
- Increase the learning rate if there is no overfitting and the error on the evaluation dataset still was reduced on the last iteration.

4.3 Model selection and assessment

As indicated by Hastie and Tibschirani [19], model selection refers to estimating the performance of different models in order to choose the best one. On the other hand, model assessment is, having chosen a final model, estimating its prediction error (generalization error) on new data. In this exercise , I will be using a split 80% for training and 20% for testing.

4.4 Cross Validation on Time Series and grid search

I will follow one of the most usual approachs to deal with time series and I am defining 5-fold cross validation process. As explained by Shrivastava(2019)[20], I will create 5 pairs of training/test sets that follow below rules:

- Every test set contains unique observations.
- Observations from the training set occur before their corresponding test set.

As a result, the 5 pairs of training/test sets are:

- Fold 1: Training: [1] Test: [2].
- Fold 2: Training: [1, 2] Test: [3].
- Fold 3: Training: [1, 2, 3] Test: [4]
- Fold 4: Training: [1, 2, 3, 4] Test: [5]
- Fold 5: Training: [1, 2, 3, 4, 5] Test: [6]

I will build the models iteratively following the mentioned procedure and compute the value of the mean squared error in each Test set, from folds 1 to 5. Then I will take the average of the five values of the metric to get the final cross validation mean squared error.

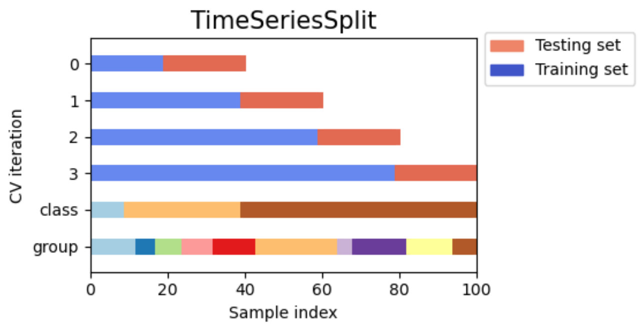

Figure 3: TSCV: Time series split[21]

Grid search consist in creating a model per each combination of the hyper-parameters that will be proved. I will use grid search with cross-validation to find the best hyperparameter values for the model. The final model will be built by training the whole training dataset with the best hyperparameters found via cross validation. Finally, I will assess the model using the test set.

4.5 Validation strategy: One-step-ahead

One-step ahead forecasts are computed sequentially, for each data point in the test dataset, by predicting the values using in period t+1 all the data known until period t.

Forecast error is computed by subtracting forecast value (estimated at the previous point) from the observed value at the current point. Overall validation error, is computed as an average value of all the squared errors in the test dataset.

4.6 Models comparisson methodology

I will calculate the mean squared error of the train and test datasets for each of the four models analyzed. I will be checking if the models are underfitting or overfitting and which of the models is better at generalization. Finally I will also be checking the forecasting vs actual values chart, to identify if besides the metrics it can help identify the best model.

5 Results

5.1 VAR estimation

I use data starting in Q2-1947 which is a longer period compared to Ros(2019)[1], the justification is that as I will be using machine learning models I prefer to have as much data as possible and also the series are stationary.

The VAR has lowest AIC and BIC for p=1 as noted in table 2, therefore I will be using the model with one lag for the comparisson. This is a difference compared with the decision taken by Ros(2019)[1] where they used p=2.

lags	**AIC**	**BIC**	**FPE**	**HQIC**
0	3.649	3.680	38.45	3.662
1	3.506*	3.597*	33.33*	3.543*
2	3.527	3.677	34.02	3.588
3	3.523	3.734	33.88	3.608
4	3.524	3.795	33.93	3.633
5	3.546	3.877	34.68	3.680
6	3.554	3.945	34.95	3.711
7	3.580	4.031	35.87	3.762
8	3.605	4.116	36.80	3.811
9	3.632	4.203	37.80	3.862
10	3.638	4.270	38.07	3.893
11	3.664	4.356	39.07	3.943
12	3.632	4.384	37.85	3.935

Table 2: VAR order selection. * highlights the minimums

5.2 Gradient boosting

5.2.1 Hyperparameters

In the next tables are shown the grid search values that were analyzed in each implementation of the gradient boosting and also the best hyperparameters obtained. Those best values will be used for the estimation of the final model and validation.

	Grid search values	**Best parameters**
max_depth	5,10,20	10
max_leaves	5,10,20,30	5
n_estimators	5,10,25	5
reg_alpha	3,4,5,	4
reg_lambda	2,3,4	3

Table 3: Hyperparameter optimization XGBoost

	Grid search values	**Best parameters**
max_depth	5,10,20	10
num_leaves	5,10,20,30	5
n_estimators	5,10,25	5
reg_alpha	3,4,5,	4
reg_lambda	2,3,4	3

Table 4: Hyperparameter optimization LightGBM

	Grid search values	**Best parameters**
depth	13,14,15	15
l2_leaf_reg	19,20,21	19
n_estimators	15,20,25,100	15
learning_rate	0.05,0.01	0.05

Table 5: Hyperparameter optimization CatBoost

5.2.2 Validation

The table reports the average mean squared error of the 4 models in the train and test periods. All the three implementations of the gradient boosting model perform better than the VAR. On the other hand, the model with the lowest Test MSE is Light GBM and it is also the one with lowest overfitting. The gradient boosting models have a mean squared error which is roughly half the one achieved by the VAR model in the test period. According to the test MSE of the three gradient boosting algorithms, they are very similar regarding generalization.

	CV_score	**Train MSE**	**Test MSE**
VAR		44.79	47.98
XGBoost	27.27	11.25	24.17
LightGBM	27.07	18.88	23.42
Catboost	27.33	21.8	23.45

Table 6: Train and test scores

5.3 Charts comparisson

Finally we can see the charts with the forecasts obtained with all four models in the test period. Regarding the serie dGDP, The VAR model seems to overestimate it similarly than Catboost. LightGBM reports values way higher than the dGDP range in the test period. Lastly the XGBoost is the one that moves closer to the actual series without overestimating it.

Regarding dSP all four models seem to underestimate several periods pushed by the falls that can be observed in the years 2009, 2012, 2016 and 2019.

It's also worth noting that some movements in the growth of the S&P are not followed by the growth of the GDP. We have an example at the end of 2015. In that year, we see negative a growth rate in 2015 which implies a decrease in the value of the index, however such decline is not followed by a decrease in the GDP, although we see a smaller growth rate.

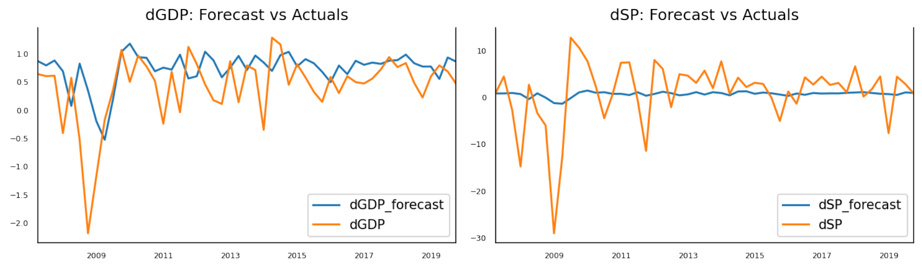

Figure 4: VAR(1) prediction

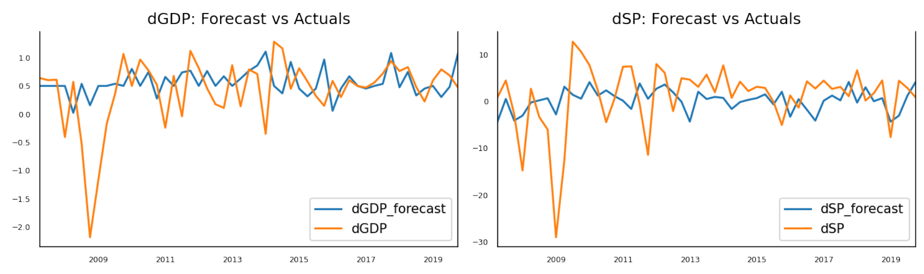

Figure 5: XGBoost prediction

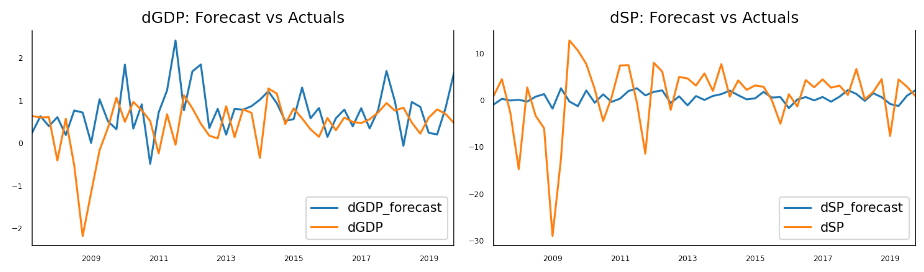

Figure 6: LightGBM prediction

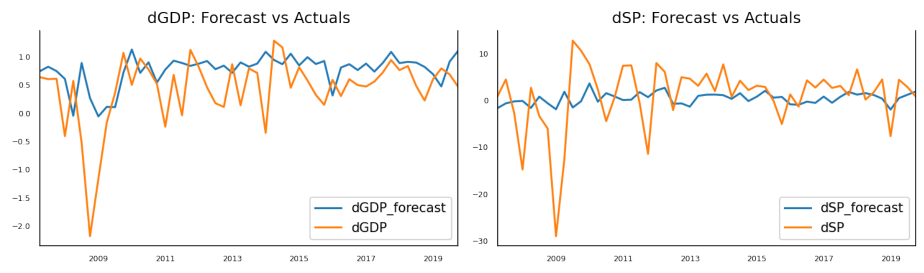

Figure 7: CatBoost prediction

6 Conclusions

In this work I provided a comparison about the predictive ability of Gradient Boosting Decision Tree algorithms and its comparisson with the classical Vector Autoregressive. The GBDT implementations XGBoost, LGBM and CatBoost perform better predictions as shown in the one-step-ahead strategy. In particular, LGBM has the lowest Test MSE and also the lowest overfitting. However, after analyzing the forecasts of the test period in the charts, we can see that the models over predict the serie dGDP except for XGBoost. With all mentioned points the winner of this exercise is XGBoost.

For next experiments it would be interesting to add macroeconomic variables using current and lagged values to try to reduce the forecasting errors that remained high in some periods. Taking as example the work presented in the bibliography, some good candidates for future analysis are industrial production, unemployment rate, housing starts and the difference between 10 year treasury constant maturity rate and federal funds rate as they are representative of U.S. economy.

References

[1] J. M. Ros, "An Inquiry into the Nature and Causes of the Econometric Relation between GDP and Stock Market in the U.S." 2019.

[2] B. Beers, "Why Do Investors Use the S&P 500 as a Benchmark?" 2022. [Online]. Available: https://www.investopedia.com/ask/answers/041315/what-are-pros-and-cons-using-sp-500-benchmark.asp

[3] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, "Comparison of arima and artificial neural networks models for stock price prediction," *Journal of Applied Mathematics*, vol. 2014, pp. 9–11, 2014.

[4] P. G. Coulombe, M. Leroux, D. Stevanovic, and S. Surprenant, "How is machine learning useful for macroeconomic forecasting?" pp. 1–52, 2020. [Online]. Available: http://arxiv.org/abs/2008.12477

[5] G. Maccarrone, G. Morelli, and S. Spadaccini, "Gdp forecasting: Machine learning, linear or autoregression?" *Frontiers in Artificial Intelligence*, vol. 4, pp. 1–9, 2021.

[6] S. R. Karingula, N. Ramanan, R. Tahmasbi, M. Amjadi, D. Jung, R. Si, C. Thimmisetty, L. P. Cabrera, M. Sayer, and C. N. Coelho, "Boosted embeddings for time series forecasting," 4 2021.

[7] C. Ball and J. French, "Exploring what stock markets tell us about gdp in theory and practice," *Research in Economics*, vol. 75, pp. 330–344, 2021. [Online]. Available: https://doi.org/10.1016/j.rie.2021.09.002

[8] B. Jason, "Gradient boosting with scikit-learn, xgboost, lightgbm, and catboost," *https://machinelearningmastery.com/gradient-boosting-with-scikit-learn-xgboost-lightgbm-and-catboost/*.

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/

[10] A. Mello, “Xgboost: theory and practice,” *https://towardsdatascience.com/xgboost-theory-and-practice-fb8912930ad6*.

[11] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *https://arxiv.org/abs/1603.02754*, 2016.

[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 3147–3155, 2017.

[13] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: Unbiased boosting with categorical features,” *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 6638–6648, 2018.

[14] “OECD Data.” [Online]. Available: https://data.oecd.org/gdp/quarterly-gdp.htm#indicator-chart

[15] R. Shiller, “ONLINE DATA ROBERT SHILLER,” data file available as http://www.econ.yale.edu/~shiller/data/ie_data.xls. [Online]. Available: http://www.econ.yale.edu/~shiller/data.htm

[16] A. Burkov, *The Hundred-Page Machine Learning Book*, 2019. [Online]. Available: http://themlbook.com/

[17] “LightGBM website.” [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Parameters.html

[18] “Catboost website.” [Online]. Available: https://catboost.ai/en/docs/concepts/speed-up-training

[19] T. Hastie, R. Tibshirani, and J. Riedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, 2008.

[20] S. Shrivastava, “Cross Validation in Time Series.” [Online]. Available: https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

[21] “Visualizing cross-validation behavior in scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py

Appendix

```
# Libraries
import pandas as pd
import numpy as np
import math
from xgboost import XGBRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
import warnings
from dateutil.relativedelta import relativedelta
from datetime import date, timedelta,datetime
import matplotlib.pyplot as plt
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic



warnings.filterwarnings("ignore", category=DeprecationWarning)

# %% [markdown]
# # Import S&P data
# %%
parser = lambda x: pd.to_datetime(x, format='%Y,%m')
data=pd.read_csv('./datasets/sp.csv',sep=';', date_parser=parser,
                                  parse_dates=['Date'],index_col='
                                  Date',decimal=',')
data.head(6)

# %%
data_quarter=pd.DataFrame(data['S&P'].resample('Q-JAN',convention='
                                  start').agg('mean'))
data_quarter['CPI']=data['CPI'].resample('Q-JAN', convention='start
                                  ').agg('last')
data_quarter.reset_index(inplace=True)
data_quarter['Date']=data_quarter['Date'].dt.to_period('M').dt.
                                  to_timestamp()
data_quarter.set_index('Date',inplace=True)
data_quarter

# %%
data_quarter['real_SP']=data_quarter['S&P']/data_quarter['CPI']*(
                                  data_quarter.iloc[[145]]['CPI'].
                                  values)
data_quarter['lSP']=np.log(data_quarter['real_SP'])
data_quarter['dSP']=(data_quarter['lSP']-data_quarter['lSP'].shift(
                                  1))*100
data_quarter.tail(20)

# %%
data_quarter[['real_SP','dSP']].plot(subplots=True,figsize=(10,8),
                                  grid=True,title='S&P 500, Avg.
```

```
                                    price per share in Month Ending
                                    Quarter and Growth rate')

# %% [markdown]
# # Import GDP

# %%
df=pd.read_csv('./datasets/DP_LIVE_30042022183016541.csv',
                                    parse_dates=['DATE'],index_col='
                                    DATE',usecols=['DATE', 'Value'])
df=df.rename(columns={'Value':'dGDP'})
df.head()

# %%
#plt.figure()
df['dGDP'].plot(grid=True,figsize=(10,6),title='U.S. GDP growht
                                    rate')
#plt.suptitle('title_string', y=1.05, fontsize=18)

# %%
plot_acf(df['dGDP']['1947-04-01':'2019-10-01'],zero=False)

# %%
plot_pacf(df['dGDP']['1947-04-01':'2019-10-01'],zero=False)

# %% [markdown]
# # Merge data

# %%
df_merged_data=df['dGDP'].reset_index()
df_merged_data['target_dGDP']=1
df_merged_data.rename(columns={'dGDP': 'target'}, inplace=True)
df_merged_data['dGDP_1']=df_merged_data['target'].shift(1)
df_merged_data

# %%
df_tmp=data_quarter['dSP'].reset_index()
df_tmp['target_dGDP']=0
df_tmp.rename(columns={'dSP': 'target','Date':'DATE'}, inplace=True
                                    )
df_tmp['dSP_1']=df_tmp['target'].shift(1)
df_tmp

# %%
#Join dSP as feature of dGDP as target
df_merged_data=df_merged_data.merge(df_tmp[['DATE','dSP_1']],
                                    left_on='DATE',right_on='DATE')
df_merged_data.head()

# %%
#Join dGDP as feature of dSP as target
df_tmp=df_tmp.merge(df_merged_data[['DATE','dGDP_1']],left_on='DATE
                                    ',right_on='DATE')
df_tmp.head()

# %%
#Concatenate datasets dGDP and dSP
```

```
df_merged_data=pd.concat([df_merged_data,df_tmp], ignore_index=True
                                    )
df_merged_data.head()

# %%
df_merged_data.shape

# %%
df_merged_data[df_merged_data['DATE']=='1947-07-01']

# %% [markdown]
# # Train/test split

# %%
end_train_data="2007-01-01"
end_test_data='2019-10-01'
test_period=pd.date_range(datetime.strptime(end_train_data, '%Y-%m-
                                    %d')+ pd.DateOffset(60),
                                    end_test_data,freq=pd.offsets.
                                    MonthBegin(3)     )
test_period


# %%
# to avoid deprecation warning
df_merged_data.index=pd.Index(df_merged_data.index, dtype="uint64")
df_merged_data.index

# %%
X_train=df_merged_data[df_merged_data['DATE'] < end_train_data].
                                    drop(columns=['DATE','target'])
X_test=df_merged_data[df_merged_data['DATE'].isin(test_period)].
                                    drop(columns=['DATE','target'])
y_train=df_merged_data.loc[df_merged_data['DATE'] < end_train_data,
                                    'target']
y_test=df_merged_data.loc[df_merged_data['DATE'].isin(test_period),
                                    'target']

# %%
display(X_train.head())
display(y_train.head())

# %% [markdown]
# # VAR

# %%
data_var=data_quarter.merge(df,left_index=True,right_index=True)[['
                                    dGDP','dSP']].dropna()
X_var_train=data_var[:end_train_data]
X_var_test=data_var.loc[test_period]
nobs=len(X_var_test)
X_var_train.tail(),X_var_test.head()

# %%
data_var.to_csv('./data_var.csv')

# %%
```

```
table_desc=data_var['1947-04-01':'2018-07-01'].describe().transpose
                                  ()
print(table_desc.to_latex(index=True))

# %%
table_desc=data_var['1947-04-01':'2018-10-01'].describe().transpose
                                  ()
print(table_desc.to_latex(index=True))

# %%
table_desc=data_var['1947-04-01':'2019-12-01'].describe().transpose
                                  ()[['count','mean','std','min','
                                  max']]
print(table_desc.to_latex(index=True))

# %%
table_desc

# %%
# Import Statsmodels
model = VAR(X_var_train)

# %%
x = model.select_order(maxlags=12)
x.summary()

# %% [markdown]
# ## Train the VAR model of selected order (p)

# %%
model_fitted = model.fit(1)
model_fitted.summary()

# %%
#train error
np.sum(np.sum(model_fitted.resid**2)/len(model_fitted.resid))

# %% [markdown]
# ## Forecast

# %%
# Get the lag order
lag_order = model_fitted.k_ar
print(lag_order)  #> 4

# %%
#in sample forecast
mod = VAR(data_var[end_train_data:end_test_data].values)
forecasts = mod.predict(model_fitted.params)
df_forecast = pd.DataFrame(forecasts, index=test_period, columns=
                                  data_var.columns+ '_forecast')
df_forecast.tail()


# %%
def plot_actual_forecast(df_forecast,df_test):
    fig, axes = plt.subplots(nrows=int(len(df_forecast.columns)/2),
                                         ncols=2, dpi=150, figsize=(
                                        10,3))
    for i, (col,ax) in enumerate(zip(df_test.columns, axes.flatten
                                        ())):
        df_forecast[col+'_forecast'].plot(legend=True, ax=ax).
                                            autoscale(axis='x',tight=
                                            True)
        df_test[col][test_period].plot(legend=True, ax=ax);
        ax.set_title(col + ": Forecast vs Actuals")
        ax.xaxis.set_ticks_position('none')
        ax.yaxis.set_ticks_position('none')
        ax.spines["top"].set_alpha(0)
        ax.tick_params(labelsize=6)

    plt.tight_layout();
plot_actual_forecast(df_forecast,X_var_test)

# %% [markdown]
# ## Evaluate the forecast

# %%
df_eval=pd.DataFrame()
df_eval['forecast']=pd.concat([df_forecast['dGDP_forecast'],
                                  df_forecast['dSP_forecast'] ])
df_eval['real']=X_var_test['dGDP']-X_var_test['dSP']
df_eval['squared_errors']=(df_eval['forecast']-df_eval['real'])**2
-np.mean(df_eval['squared_errors'])

# %% [markdown]
# # Pipeline

# %%
tscv = TimeSeriesSplit(n_splits=5,)

# %% [markdown]
# ## XGBoost

# %%
xgb= XGBRegressor()

# %%
pipe = Pipeline([
     ('model',  xgb)
])
param_grid = {
    'model__max_depth': [ 5,10,20],
    'model__max_leaves':[5,10,20,30],
    'model__n_estimators': [5,10, 25],
    'model__reg_alpha' : [ 3,4,5],
    'model__reg_lambda' : [ 2,3,4 ]

}


# %%
grid=None
```

```
grid = GridSearchCV(pipe, cv=tscv,
                    param_grid=param_grid,n_jobs=-1,scoring='
                                                              neg_mean_squared_error
                                                              ')
grid.fit(X_train,y_train)

# %%
grid.best_params_

# %%
print("cv score: %.4f" % grid.best_score_)
print("train score: %.4f" % grid.score(X_train, y_train))
print("test score: %.4f" % grid.score(X_test, y_test))

# %%
df_cv_xgb=pd.DataFrame(grid.cv_results_)
df_cv_xgb.sort_values(by='rank_test_score').head()

# %%
plt.plot(df_cv_xgb.index,df_cv_xgb['mean_test_score'],'-o')
plt.errorbar(df_cv_xgb.index, df_cv_xgb['mean_test_score'], yerr=
                                  df_cv_xgb['std_test_score'],
                                  ecolor='g')

# %%
df_cv_xgb['mean_test_score'].hist()

# %%
forecasts_xgb = np.reshape(grid.predict(X_test),(-1,2),order='F')
df_forecast_xgb = pd.DataFrame(forecasts_xgb, index=test_period,
                                  columns=data_var.columns+ '
                                  _forecast')
df_forecast_xgb.tail()

# %%
plot_actual_forecast(df_forecast_xgb,X_var_test)

# %% [markdown]
# ## LGBM

# %%
lgbm = LGBMRegressor(objective="regression")#,device='GPU')

# %%
pipe2 = Pipeline([
    ('model', lgbm)
])
param_grid2 = {
    'model__max_depth': [ 9,10,11,15],
    'model__n_estimators': [11,12,13,14,25,50],
    'model__linear_tree':[False,True],
    'model__num_leaves':[13,14,15,16,17,25,50],
    'model__learning_rate':[0.06,0.07,0.08,0.1],
    'model__reg_alpha':[3,4,5],
    'model__reg_lambda':[1,2,3]

}
```

```
# %%
grid2=None
grid2 = GridSearchCV(pipe2, cv=tscv,
                     param_grid=param_grid2,n_jobs=-1,
                     scoring='neg_mean_squared_error',verbose=3)
grid2.fit(X_train,y_train)

# %%
grid2.best_params_

# %%
print("cv score: %.4f" % grid2.best_score_)
print("train score: %.4f" % grid2.score(X_train, y_train))
print("test score: %.4f" % grid2.score(X_test, y_test))

# %%
forecasts_lgbm = np.reshape(grid2.predict(X_test),(-1,2),order='F')
df_forecast_lgbm = pd.DataFrame(forecasts_lgbm, index=test_period,
columns=data_var.columns+ '_forecast')
df_forecast_lgbm.tail()

# %%
plot_actual_forecast(df_forecast_lgbm,X_var_test)

# %% [markdown]
# ## Catboost

# %%
catboost = CatBoostRegressor(task_type="GPU")
#catboost = CatBoostRegressor(iterations=150)#,task_type="GPU",
                                  devices='0:1')

# %%
pipe3= Pipeline([
     ('model',  catboost)
])
param_grid3= {
    'model__depth': [13,14,15],
    'model__l2_leaf_reg': [19,20,21],
    'model__n_estimators':[15,20,25,100],
    #'model__iterations':[10,20,30,40]
    'model__learning_rate':[0.05,0.1]
}

# %%
grid3=None
grid3 = GridSearchCV(pipe3, cv=tscv,
                     param_grid=param_grid3,n_jobs=1,
                     scoring='neg_mean_squared_error')#n_jobs=1 for
                                                      using gpu
grid3.fit(X_train,y_train)

# %%
grid3.best_params_

# %%
```

```
print("cv score: %.4f" % grid3.best_score_)
print("train score: %.4f" % grid3.score(X_train, y_train))
print("test score: %.4f" % grid3.score(X_test, y_test))

# %%
forecasts_cb = np.reshape(grid3.predict(X_test),(-1,2),order='F')
df_forecast_cb = pd.DataFrame(forecasts_cb, index=test_period,
columns=data_var.columns+ '_forecast')
df_forecast_cb.tail()

# %%
X_train.shape,X_test.shape

# %%
plot_actual_forecast(df_forecast_cb,X_var_test)

# %%
print("train score: %.4f" % grid.score(X_train, y_train))
print("test score: %.4f" % grid.score(X_test, y_test))
print("train score: %.4f" % grid2.score(X_train, y_train))
print("test score: %.4f" % grid2.score(X_test, y_test))
print("train score: %.4f" % grid3.score(X_train, y_train))
print("test score: %.4f" % grid3.score(X_test, y_test))
```