DOI: 10.1478/AAPP.1012A12

AAPP | Atti della Accademia Peloritana dei Pericolanti
Classe di Scienze Fisiche, Matematiche e Naturali
ISSN 1825-1242

Vol. 101, No. 2, A12 (2023)

A DIFFERENT APPROACH FOR CAUSAL IMPACT ANALYSIS ON PYTHON WITH BAYESIAN STRUCTURAL TIME-SERIES AND BIDIRECTIONAL LSTM MODELS

PASQUALE FOTIA [a] AND MASSIMILIANO FERRARA [ab*]

ABSTRACT. In this paper, we propose using a combination of two models, the Google model, CasualImpact, and a Bidirectional LSTM, along with an Incremental Difference in Difference model, to infer information from time series data in order to analyse the causal impact of an event on a particular phenomenon over time. Our method identifies the causal influence of a treatment on an outcome variable by comparing the difference in result seen after "sham" therapy, but before genuine treatment. The models are regularly applied to each period of the time series to assess not only the effect of the event that occurred at a certain time t, but also the behaviour of the phenomenon around t. The Google model (CausalImpact) is based on a Bayesian framework, whereas the Bidirectional LSTM model use a neural network. The implementation of the two models in Python is easy to execute, but the cost of data processing time can be substantial. The findings from these two models must then be analysed in conjunction with the incremental difference in difference model in order to assess if the target event at time t had a greater influence than the impact seen for values close to time t. This study aims to integrate the use of these three models and iterate this approach in order to give a thorough analysis of the temporal causal influence of an event on a phenomena.

1. Introduction

Analysing the causal effect of an intervention or policy on a particular result of interest is causal impact analysis. It is a statistical method used to determine the amount to which an intervention or policy has caused a change in the outcome of interest, as opposed to changes that would have occurred anyhow. Causal impact analysis is essential for comprehending the efficacy of interventions and policies and making well-informed judgments regarding future interventions and policies. For causal effect analysis, numerous procedures are available, including randomized controlled trials (RCTs), propensity score matching (PSM), and difference-in-differences (DiD) techniques (Caliendo and Kopeinig 2008; Wing, Simon, and Bello-Gomez 2018). The choice of methodology is determined by the precise research issue and the accessible data type. RCTs are regarded as the gold standard for causal effect analysis because they permit random assignment of participants to a treatment or control group, which helps to account for any potential confounding variables. However,

RCTs are not always practicable or ethical, and alternate approaches such as PSM and DiD may be employed when RCTs cannot be conducted. Persons in the treatment group are matched with individuals in the control group based on their observed characteristics, such as age, gender, and income. This helps to adjust for any potential confounders and permits the estimation of the intervention's or policy's causal effect. DiD is a strategy that compares, over time, the changes in the outcome of interest for a treatment group (i.e., the group exposed to the intervention or policy) and a control group (i.e., the group not exposed to the intervention or policy). The incremental difference-in-differences (IDD) (Dolton, Bondibene, and Wadsworth 2010) is a version of the difference-in-differences (DiD) method that enables the assessment of treatment effects over time by employing numerous pre-treatment and post-treatment periods. In general, causal impact analysis is an excellent method for determining the efficacy of interventions and policies. It enables researchers to evaluate treatment outcomes, account for potential confounding variables, and make informed decisions regarding future treatments or policies. Combining the concepts of Bayesian inference and causal inference, causal impact analysis with a Bayesian framework estimates the causal influence of an intervention or policy on a particular outcome of interest (Cervantes and Rambaud 2020). Bayesian inference is a statistical inference technique that employs a prior probability distribution to revise the likelihood of a hypothesis in light of fresh data. The process of evaluating the causal influence of an intervention or policy on a particular result of interest is known as causal inference. A prior probability distribution is used to model the uncertainty in the causal effect of an intervention or policy in a Bayesian approach to causal impact analysis. The prior probability distribution is then revised in light of fresh data to estimate the intervention's or policy's causal influence. The Bayesian method permits the incorporation of prior information and the quantification of uncertainty in the estimation of the causal impact. The usage of a Bidirectional LSTM model is an additional method employed in this work. To integrate causal impact analysis with Bi-LSTM for forecasting, one can use the Bi-LSTM model to estimate the causal effect of the intervention on the time series outcome, and then incorporate the estimated causal effect into the forecasting model. The forecasting model can then use the estimated causal influence in conjunction with other pertinent data, such as previous values of the outcome and exogenous factors, to predict future values of the outcome. In this work therefore, through the creation of fictitious data, we want to test two different approaches, which do not require a control group, Bayesian Causal Impact and the Bi-LSTM neural network, in comparison with the Incremental Difference in Difference approach , which instead needs one.

2. Related work

2.1. Incremental differences-in-differences (IDiD). Dolton, Bondibene, and Wadsworth (2010) present a retrospective overview of the effects of the national minimum wage (NMW) on labour market performance in the United Kingdom since its introduction in 1999. They employ a 'incremental differences-in-differences' (IDiD) estimate to examine the effects of the NMW in each year via its differential impact on local labour markets. The NMW is related with a substantial decline in pay disparity in the lowest half of the distribution, according to them findings. The pay inequality has decreased more in regions where the

NMW has a greater impact than elsewhere. The benefit of employing the IDiD estimation method is that it helps the estimation of the incremental effects of each year's uprating from year to year. Even though the average effect across all years is not statistically different from zero, this does not imply that the effect of any given year's adjustment in the minimum wage is also zero. This assessment based on the differences-in-differences technique, allows to estimate the marginal (interaction) effect of each year's change in the NMW.

2.2. Causal impact Bayesian structure. State-space models differentiate between a state equation, which explains the transition of a collection of latent variables from one time point to the next, and an observation equation, which specifies how a given system state translates into measurements. This distinction renders them highly adaptable and potent (see Leeflang *et al.* 2009 for a discussion of this in the context of marketing research). The state-space paradigm is inherited by the technique outlined by Brodersen *et al.* (2015) in three significant ways. First, it allows to incorporate a variety of hypotheses regarding the latent state and emission mechanisms underlying the observed data, such as local trends and seasonality. Second, Brodersen *et al.* (2015) infer the time evolution of counterfactual activity and incremental impact using a completely Bayesian methodology. The flexibility with which posterior inferences can be summarized is an advantage of this method. Third, Brodersen *et al.* (2015) employ a regression component that forbids a rigid commitment to a particular set of controls by integrating out posterior uncertainty about the influence of each predictor and uncertainty about which predictors to include in the first place, therefore preventing overfitting. They also apply a Markov chain Monte Carlo-based stochastic algorithm for posterior inference (MCMC). The CausalImpact R package provides an implementation of this approach. Also, in Python is available a version of this package.

2.3. Bidirectional LSTM. Deep learning is a subcategory of machine learning that utilizes neural network-based deep architectures for learning. One of the challenges of deep time series prediction problems is the ability to learn long-term dependencies. However, traditional recurrent neural networks (RNNs) are not optimal for learning long-term dependencies due to vanishing/exploding gradient concerns (Nielsen 2015). Hochreiter and Schmidhuber (1997) proposed Long Short-Term Memory (LSTM) networks which are a type of deep learning architecture that is well-suited for time series prediction tasks because they are effective at capturing long-term dependencies (Goodfellow, Bengio, and Courville 2016). LSTMs overcome the vanishing/exploding gradient problem by using gates within each cell to control the flow of information, allowing them to store and process relevant past information. LSTMs and Gated Recurrent Units (GRUs) are two types of RNNs that are commonly used for time series prediction. While both LSTMs and GRUs are effective for time series tasks, LSTMs are typically selected for larger datasets (Zhang, Bi, and Qiu 2019). Additionally, a variation of LSTMs known as Bidirectional LSTMs (BiDLSTMs) can further improve prediction accuracy by considering both previous and future observations (Hu, Wang, and Ma 2015; Cui *et al.* 2018). BiDLSTMs process data in both the forward and reverse directions, unlike traditional LSTMs which only process information in one direction. This allows BiDLSTMs to extract additional features and improve overall learning. The model can be implemented in Python using the Keras library.

3. Methodology

The technique employed in this work comprises of multiple phases in order to assess the effect of a certain policy on a particular outcome. To test the model and replicate the beneficial effects of a policy, we generated a collection of fictitious data. The data were created on the assumption that the policy in question has a substantial positive effect on the outcome of interest. Statistical methods were then applied to the simulated data to evaluate the policy's efficacy and quantify the policy's impact. We utilized an Incremental Difference in Difference model to examine the data. This methodology allows us to analyse the impact of the policy by comparing the pre-intervention and post-intervention periods. We utilized a Google-developed, Bayesian-structured causal effect model and applied it regularly during the whole study period. This model employs the Bayesian Structural Time Series approach, which permits us to estimate the causal effect of the policy by comparing the counterfactual scenario (what would have occurred without the policy) to the observed data. After the tuning phase, a Bi-LSTM model was executed. For the duration of the period, the same approach was utilized to assure the reliability of our results. The Bi-LSTM model is an effective deep learning technique that enables us to describe the temporal dependencies in the data and increase prediction accuracy.

4. Libraries

The following libraries are used in the code for the model creation:

(1) `pandas` and `numpy` are popular Python libraries for data manipulation, frequently used in machine learning and data science. While `numpy` offers support for arrays and mathematical operations, `pandas` offers data structures and data analysis tools.

(2) The functionality of the code is supported by four libraries: `os`, `random`, `copy`, and `warnings`. The `os` library enables interaction with the operating system, `random` creates random numbers, `copy` gives methods for duplicating objects, and `warnings` alerts the user to potential problems. These libraries are utilized for a variety of purposes, including dealing with file and directory paths, communicating with the file system, generating random numbers, duplicating objects, and sending warnings.

(3) `matplotlib` is a Python data visualization library that allows you to make plots, charts, and other visualizations.

(4) Python's `statsmodels` package has a sublibrary named `statsmodels.formula.api`. It provides a high-level interface for fitting statistical models with formulas written in the R language. The `ols` function is used to fit linear regression models with ordinary least squares. The `formula` parameter in `ols` is used to express the model in R-style formula notation, with the response variable on the left and the predictor variables on the right of the symbol. This library is utilized to estimate a variety of statistical models, such as linear regression, extended linear models, robust linear models, etc.

(5) `causalimpact` is a Python library that compares the counterfactual scenario (what would have occurred if the treatment hadn't been applied) to the actual data in order to estimate the causal impact of a treatment or intervention.

(6) `tensorflow` and `keras` are libraries for deep learning development and training. `keras` is a high-level neural networks API that provides a user-friendly interface for constructing and training deep learning models, whereas `tensorflow` is a numerical computation framework that can conduct large-scale calculations on many devices. The `keras.layers` module offers the Bidirectional wrapper that may be added to an LSTM layer to produce a bidirectional LSTM. This enables the network to forecast based on both past and future context, making it an effective tool for tasks such as natural language processing.

(7) `tensorflow_probability` is a Python library that provides probabilistic modeling and inference in tensorflow.

These libraries are commonly used in data science, statistical modeling, machine learning, and deep learning projects.

5. Dataset

We utilized a synthetic dataset to facilitate the controlled and reproducible evaluation and illustration of the algorithm. By utilizing a fabricated dataset, we are able to precisely define the fundamental patterns and characteristics, isolating and analyzing particular aspects of the algorithm's behavior. This artificial dataset provides a valuable basis for experimentation and benchmarking, allowing us to gain insight into the capabilities and performance of the algorithm under controlled conditions. We create two sets of random data and graph them together. It begins by configuring the seed for the random number generator, ensuring that the same random numbers are created each time the code is executed. Using the `linspace` function, the code then generates two arrays, `x` and `y`. The `x` array consists of 200 values ranging from 1 to 200, whereas the `y` array has 100 values ranging from 30 to 100. A new array, `yconc`, is produced by concatenating an array of 100 values, all of which are 30, with the `y` array. The treated group's `yfinal` array is then created by adding random noise to each value in the `yconc` array. To represent the control group, the algorithm additionally generates a 200-element array of ones with the value 20. The `ycontrfin` array for the control group is created by adding random noise to each value in this array. The code then presents both sets of data on the same graph (Appendix A), with `yfinal` data displayed in red and `ycontrfin` data displayed in blue. In addition to x and y axis labels, a caption, and a grid, the plot contains axis labels. We have generated two lists, `A` and `B`, each containing 200 members, each of which is a string consisting of either "A" or "B." The two lists are then joined to form a single list called `ListAB`. Then, the two NumPy arrays, `yfinal` and `ycontrfin`, are concatenated into `ylist`, and the two arrays `x` are concatenated into `xlist`. A pandas DataFrame, `df`, with three columns is then created: `AB` includes the contents of `ListAB`, `value` contains the elements of `ylist` as a list, and `time` contains the elements of `xlist` as a list. This DataFrame contains a record of all the values and groups that were used in the plot, which can be useful for further analysis or manipulation. For all information related to the code explained above, refer to Appendix A.

6. Incremental Difference in Difference

The code shown in Fig. 1, implements the Incremental Difference-in-Differences (IDID) technique, which estimates the causal influence of a treatment on an outcome variable.

The `IDID` function requires three parameters: a data frame `df`, a threshold time `t`, and a treatment group `c`. The function produces a deep duplicate of the input dataframe `df` such that the original dataframe is not modified. Then, three additional columns are added to the duplicated dataframe: `time1`, `treated`, and `did`. The `time1` column is a binary variable whose value is 1 if the original dataframe's time is higher than or equal to the threshold time `t`, and 0 otherwise. The `treated` column is a binary variable whose value is 1 if the treatment group in the original dataframe matches the specified treatment group `c`, and 0 otherwise. The product of the `time1` and `treatment` columns yields the `did` column. The function then pulls the `time1`, `treatment`, and `did` columns from the cloned dataframe, as well as the result variable `value`. It utilizes these variables to execute an ordinary least squares (OLS) regression using the `statsmodels` package, a toolkit for estimating different statistical models. The regression formula is `y ~ didd + timed + treatedd`, where `y` is the dependent variable and `didd`, `timed`, and `treatedd` are the newly formed columns. The function takes the p-value of the `did` coefficient and the model's coefficients and saves them in the variables `pv` and `cofidid`, respectively, after fitting the model. The function then returns the coefficient `did` and the p-value. The code then iterates over the items of the list `x[1:]` and executes the `IDID` function for each element, appending the `did` coefficient and p-value to the lists `listalldid` and `listpv`, respectively.

```
import copy
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

def IDID(df, t, c):
   df1 = copy.deepcopy(df)

   df1['time1'] = np.where(df1['time'] >= t, 1, 0)
   df1['treated'] = np.where(df1['AB'] == c, 1, 0)
   df1['did'] = df1['time1'] * df1['treated']

   timed = df1['time1']
   treatedd = df1['treated']
   didd = df1['did']
   y = df1['value']

   res = smf.ols(formula='y ~ didd + timed + treatedd', data
       =df1).fit()

   pv = res.pvalues[1]
   cofidid = res.params.tolist()
```

```
    return cofidid[1], pv


listalldid = []
listpv = []

for i in x[1:]:
    alldid = IDID(df, i, 'A')
    listalldid.append(alldid[0])
    listpv.append(alldid[1])
```

FIGURE 1. Implementing the Incremental Difference-in-Differences technique with OLS regression.

7. Causal impact

This code in Fig. 2 uses the `CausalImpact` class from the `causalimpact` package to perform causal impact analysis on a dataset. The script starts by measuring the running time using the `time` module. It then initializes an empty list called `listci`. The script then loops over a range of values between 3 and 196 with a step of 1. For each iteration, it creates a `CausalImpact` object, with the `post_period` starting from the current iteration value, and appends the object to the `listci` list. The script then prints out the current iteration value and the total running time of the loop at the end. After that, the script loops over the `listci`, and for each element in the list, it extracts the summary of the analysis, prints a specific part of the summary, and then prints another specific part of the summary by calling the `output = 'report'` argument.

```
import time
from causalimpact import CausalImpact

start_time = time.time()

listci = []

for i in np.linspace(3, 196, 194):
    print(i)
    ci = CausalImpact(yfinal, [0, int(i)], [int(i) + 1, 199],
         model_args={'fit_method': 'hmc'})
    listci.append(ci)

print("---␣%s␣seconds␣---" % (time.time() - start_time))
```

```
z = 0
for ci in listci:
   string = ci.summary()
   print(string)

   z = z + 1

z = 0
for ci in listci:
   string = ci.summary(output='report')
   print(string)

   z = z + 1

for ci in listci:
   ci.plot()
```

FIGURE 2. Performing Causal Impact Analysis with the CausalImpact package

8. Bidirectional LSTM

The three supplied functions are incorporated into the development of a Bidirectional LSTM (Long Short-Term Memory) model for causal effect analysis. The model is trained to identify the appropriate analytical parameters.

```
import tensorflow as tf

def custom_loss(y_true, y_pred):
   max1 = tf.math.maximum(y_true, y_pred)
   max2 = tf.reduce_sum(max1)
   min1 = tf.math.minimum(y_true, y_pred)
   min2 = tf.reduce_sum(min1)
   jaccard = (1 - (min2 / max2)) * 100

   return jaccard
```

FIGURE 3. Custom loss function based on Jaccard Similarity Index for model training.

The custom loss in Fig. 3 is a loss function used to train the model. The function accepts two parameters, `y_true` and `y_pred`, which represent the true and predicted values, respectively. Within the function, it computes the Jaccard similarity index, a measure of the similarity between two datasets, between the true and predicted values. The Jaccard index is calculated as `(1 - (min2 / max2))* 100`, where `min2` is the sum of the minimum values between the true and predicted sets, and `max2` is the total of the highest values. Higher values of the Jaccard index indicate a greater degree of similarity between the true and predicted sets.

```
import numpy as np

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
   X, y = list(), list()
   for i in range(len(sequence)):
      % find the end of this pattern
      end_ix = i + n_steps
      % check if we are beyond the sequence
      if end_ix > len(sequence)-1:
         break
      % gather input and output parts of the pattern
      seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
      X.append(seq_x)
      y.append(seq_y)
   return np.array(X), np.array(y)
```

FIGURE 4. Split sequence function for dividing univariate time series into samples.

The `split_sequence` function in Fig. 4 is used to divide a univariate sequence into samples. `sequence` is the sequence to be divided, while `n_steps` specify the number of time steps needed to separate the series. The function divides the sequence into samples, each of which consists of `n_steps` time steps. `X` is a list of input samples, whereas `y` is a list of output samples. This code of `Bi-LSTM` model for forecasting time series is in Appendix B. The function for the model accepts the parameters `treatment_period`, `seed`, and `data`. The variable `treatment_period` is used to provide the range of data that will be utilized for training the model. The variable `seed` is used to initialize the random number generator in order to ensure repeatability of the findings. The function begins by defining two periods, `pre_period` and `post_period`, where `pre_period` is the range of data used for model training and `post_period` is the range of data used for model testing. It also initializes the list `list_steps` variable, which contains the number of steps the model will employ while producing predictions. The function then populates a number of empty lists with values for future use: `bestepochvalue`: will

be used to hold each iteration's minimal validation loss. `bestepochcicle`: will be used to hold the index of each iteration's smallest validation loss. `listvalmaelist`: will be used to keep each iteration's validation mean absolute error. This variable will be used to hold the model's final predictions for each iteration. The code then enters a `for` loop that iterates through each step in the `list_steps` list. The function sets the seed values for the Python hash seed environment variable within the loop. It then defines the input sequence by slicing the data array from the start to the `treatment_period`, selects the number of time steps (`n_steps`) equal to the current step in the iteration, and separates the sequence into samples (`X,y`) using the `split_sequence` function. Then, it transforms the samples from `[samples, timesteps]` to `[samples, timesteps , features]` and sets the variable `n_features` to 1 to indicate that the model will only use one feature. The method then constructs an early stopping callback that checks validation loss and terminates training if validation loss does not improve after a defined number of iterations (`patience = 40`). The model is then defined as a sequential model that consists of a bidirectional LSTM layer and a dense layer. The LSTM layer has 50 units and is activated with LeakyReLU. The Bidirectional wrapper is utilized to make the LSTM layer bidirectional, allowing it to handle input in both the forward and reverse orientations. The model is then built with an Adam optimizer and a custom loss function (`custom_loss`), which is not specified but is presumably used to provide a custom loss function for the model's compilation. The model is also constructed with a mean absolute error (MAE) measure that will be used to evaluate the model's performance. The model is then fitted to the data using the specified number of epochs (`epochs = 100`), a validation split of 0.33, and the previously stated early termination callback. The fit's history is recorded under the variable `history`. The code then takes the validation loss and validation mean absolute error from the `history` object and puts them accordingly in the `listvaloss` and `listvalossmae`. The minimal validation loss and its index are subsequently stored in the `xmin` and `yindex` variables, respectively. This value is added to `listvalmaelist`. `yindex` is added to `bestepochcicle`, whereas `xmin` is added to `bestepochvalue`. This is accomplished by setting the number of epochs in the `fit` function to `yindex+1`. The function then refits the model using the number of epochs that corresponds to the smallest validation loss. This upgraded version is stored in `model1`. The predictions are then stored in the variable `yhat` using the adjusted model and input data. The code then initializes a prediction list that will be used to hold the model's final predictions. The code then starts a 10-iteration loop in which, for each iteration, a prediction is made on the current batch using the model and added to the prediction list. The latest forecast is then applied to the current batch. The prediction list is thereafter appended to the `listaresultfin` list. After the `for` loop, the function returns the `listaresultfin` list containing the final predictions.

```
listann = x[33:-15]

result_final = []
for j in listann:
   range_seed = [123]
   result_seed = []
   for i in range_seed:
      print(j, '
         ------------------------------------------------
         ')
      best_ep, best_cicle, val_mae_best, list_result_fin =
         bi_lstm_iteration(int(j), i, listadd)
      print(list_result_fin)
      parameter = list_result_fin.index(min(list_result_fin)
         )
      min_best_loss = best_ep[parameter]
      mi_val_mae_best = val_mae_best[parameter]
      best_cicle_select = best_cicle[parameter] + 1
      list_cod_step = [8, 9, 10, 11, 12, 13, 14, 15, 16]
      best_step_select = list_cod_step[parameter]
      print(min(list_result_fin), min_best_loss,
         mi_val_mae_best, best_cicle_select,
         best_step_select, i, j)
      result_seed.append([min(list_result_fin),
         min_best_loss, mi_val_mae_best, best_cicle_select,
          best_step_select, i, j])

   best_i = []
   for i in result_seed:
      best_i.append(i[0])

   re = best_i.index(min(best_i))
   result_final.append(result_seed[re])
```

FIGURE 5. Post-processing results of LSTM iteration function for time series prediction optimization.

This code, in Fig. 5, does extra processing on the preceding `lstm_iteration` function's output. The first line constructs the list `listann` by slicing the array `x` from index `33` to the second-to-last element `(-15)`. This list contains the values that will be used as the treatment period in the `lstm_iteration` function. The code then initializes a

list `resultfinal` that will be used to hold the processing's final results. It begins a `for` loop that iterates over each element in `listann`. For each element, it initializes a list `rangeseed` with a single element, `123`, and a list `resultseed` containing nothing, which will be used to hold the results of the `lstm_iteration` function for each seed value. In each iteration, the `lstm_iteration` function is called using the current value of `j` from the outer loop as the treatment period, the current value of `i` from the inner loop as the seed, and the `listadd` array as the data. Four values are returned by the function: `bestep`, `bestcicle`, `valmaebest`, and `listresultfin`. The code then retrieves the index of the minimal value in `listresultfin` and puts it in the variable `parameter`. The index is then used to retrieve the respective values of `minbestlost`, `mivalmaebest`, `bestcileselect`, and `beststepselect` from the `bestep`, `valmaebest`, `bestcicle`, and `listacodstep` lists. The code then displays `minbestlost`, `mivalmaebest`, `bestcileselect`, `beststepselect`, the current seed value `i`, and the current treatment period value `j`. Then, a list containing these values is appended to the `resultseed` list. After the inner loop, the code produces a new list containing just the minimal values from the `resultseed` list and puts the index of the minimum value in the variable `re`. Finally, the element at index `re` in the `resultseed` list is appended to the `resultfinal` list. After the outer loop, the `resultfinal` list will include the processing's final results, including the minimum value from the `listreslтfin`, `minbestlost`, `mivalmaebest`, `bestcileselect`, `beststepselect`, the seed value, and the treatment period value for the best result. This is used to establish the optimal parameters for the Bi-LSTM model to be utilized in a causal impact study; it assists in locating the most accurate prediction results.

9. Discussion of the results

The findings indicate that the three utilized models (Incremental Difference in Difference, CausalImpact, and Bidirectional LSTM) predict the treatment's causal effect for each period (Fig. 6). The values appear to grow steadily over time, indicating that the treatment group is exerting a greater influence with time. The fact that the trend shifts after 100 periods shows that there is a change point or threshold in the data at which the treatment's causal effect begins to rise. Given that the three models employ distinct methodologies for estimating causal effects, it is probable that each model captures the trend shift in a somewhat different way. For example, the incremental difference in difference model may be more sensitive to changes in the early periods, while the CausalImpact model might be more sensitive to changes in the later periods and the Bidirectional LSTM might be more accurate in general. This study's objective was to assess the efficacy of several models for evaluating the impact of actions on a system. In this study, the Incremental Difference in Difference approach serves as a benchmark for the other analytical techniques. As indicated by the low p-values (Fig. 7) for each period in the report, this strategy offers excellent insights into the effects of treatments. Nonetheless, locating an appropriate control group for the research is one of the obstacles of using real-world data. In spite of this difficulty, the incremental change in difference methodology provides a useful method for evaluating the genuine impact of interventions in real-world circumstances.

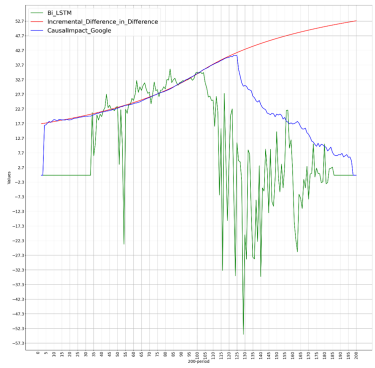

FIGURE 6. Comparison of predicted causal effects from Incremental Difference in Difference, CausalImpact, and Bidirectional LSTM models for time series data.

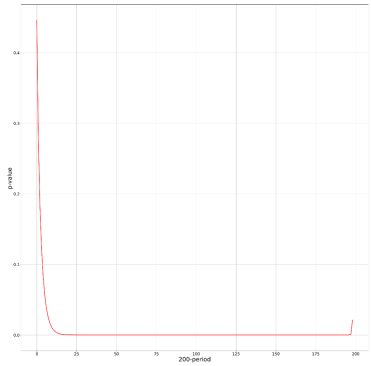

FIGURE 7. Assessing the significance of treatment effects through p-value analysis in Incremental Difference in Difference model.

In the context of causal impact analysis, the results of the Google Causal Impact model and the Bi-LSTM model provide important insights on the efficacy of various methodologies for evaluating the impact of interventions on a particular system. With a probability of 0 for all periods (Fig. 8), the findings of the GoogleCausal Impact model indicate that the model can properly reflect the causal effect of interventions on the investigated system. This makes the Google Causal Impact model a helpful tool for applications that demand a clear knowledge of an intervention's causal impact. In contrast, the results of the Bi-LSTM model based on the metrics for training the neural network for each period of the Time Series Data (Fig. 9), show that it may be better appropriate for situations when the objective is not to exactly capture the causal influence of an intervention but rather to offer a broad knowledge of the system's trends and patterns. In addition to their accuracy, the findings of the Google Causal Impact model and the Bi-LSTM model also provide the time necessary to conduct a causal impact study using each technique. In this instance, the

```
Analysis report {CausalImpact}


During the post-intervention period, the response variable had
an average value of approx. 64.59. By contrast, in the absence of an
intervention, we would have expected an average response of 29.82.
The 95% interval of this counterfactual prediction is [29.03, 30.58].
Subtracting this prediction from the observed response yields
an estimate of the causal effect the intervention had on the
response variable. This effect is 34.77 with a 95% interval of
[34.01, 35.56]. For a discussion of the significance of this effect,
see below.


Summing up the individual data points during the post-intervention
period (which can only sometimes be meaningfully interpreted), the
response variable had an overall value of 6458.76.
By contrast, had the intervention not taken place, we would have expected
a sum of 2982.13. The 95% interval of this prediction is [2902.91, 3057.8].


The above results are given in terms of absolute numbers. In relative
terms, the response variable showed an increase of +116.58%. The 95%
interval of this percentage is [114.04%, 119.24%].


This means that the positive effect observed during the intervention
period is statistically significant and unlikely to be due to random
fluctuations. It should be noted, however, that the question of whether
this increase also bears substantive significance can only be answered
by comparing the absolute effect (34.77) to the original goal
of the underlying intervention.


The probability of obtaining this effect by chance is very small
(Bayesian one-sided tail-area probability p = 0.0).
This means the causal effect can be considered statistically
significant.
```

FIGURE 8. Significance of causal effect: Results of CausalImpact analysis.

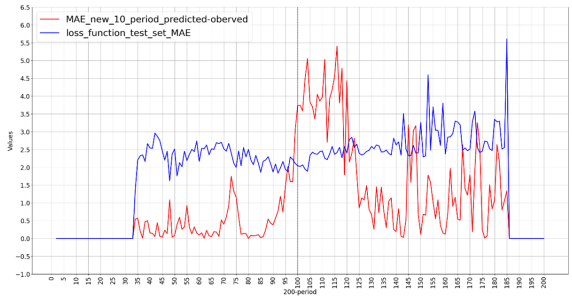

FIGURE 9. Evaluating the performance of Bi-LSTM for causal impact analysis: a comparison of next 10 period prediction MAE with test set MAE for each period of time series data.

Google Causal Impact model required around 9 hours, but the Bi-LSTM model required approximately 4 hours. It is crucial to note that the Bi-LSTM model's training period may be lengthened by adding additional parameters to the neural network, resulting in a more accurate prediction. However, this enhanced precision may need more time and processing resources. In summary, the findings of the Google Causal Impact model and the Bidirectional LSTM model give important information on the precision and time necessary

for causal impact analysis. The decision between the two approaches will rely on the analysis's particular criteria and objectives, as well as the available resources and limits.

10. Conclusion

This study finishes with a thorough evaluation of two unique Causal Impact Analysis techniques. The first technique is based on a Google software that leverages a Bayesian framework, which offers a precise confidence assessment but limits parameter adjustment. The second technique is based on a Bidirectional LSTM, which provides greater flexibility in parameter adjustment and enhances forecasting. The findings of this study suggest that both techniques are capable of detecting the onset of therapy, and that the strategy used will depend on the application and the user's priorities. One of the key advantages of adopting the Google package is that it is based on a Bayesian framework that offers a clear level of confidence; this is especially advantageous when you want a clear understanding of the results' certainty. In contrast, the Bidirectional LSTM may be a preferable choice when more parameter change flexibility is required, which is crucial when striving to improve prediction. Notably, these two approaches are not mutually exclusive, and the combination of numerous procedures might yield more exact and complete results. The results of this study demonstrate the need of studying several approaches for Causal Impact Analysis and the potential benefits of merging multiple methods for more precise and thorough results. This study offers valuable insights into the numerous approaches for Causal Impact Analysis, as well as the advantages and disadvantages of each strategy. In addition, it emphasizes the need of studying several approaches and the possible benefits of integrating them for more accurate and complete results.

Appendix A. Dataset code and graph

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(10)

x = np.linspace(1, 200, 200)
y = np.linspace(30, 100, 100)
oney = np.ones(100) * 30
yconc = np.concatenate([oney, y])
ycontroll = np.ones(200) * 20
noise = np.random.uniform(-5, 5, size=200)
yfinal = np.add(yconc, noise)
ycontrfin = np.add(ycontroll, noise)

cm = 1 / 2.54
fig3, ax3 = plt.subplots(figsize=(80 * cm, 40 * cm))
```

```
ax3.plot(yfinal, linewidth=2.5, color='red', label='Treated␣
    group')
ax3.plot(ycontrfin, linewidth=2.5, color='blue', label='
    Control␣group')
plt.legend(fontsize=30, loc="best")
ax3.set_xlabel("200-period", fontsize=17)
ax3.set_ylabel("Value", fontsize=17)

ax3.grid()

fig3.savefig('fig1.png')
plt.show()
```

Generating and plotting random data for experimental and control groups.

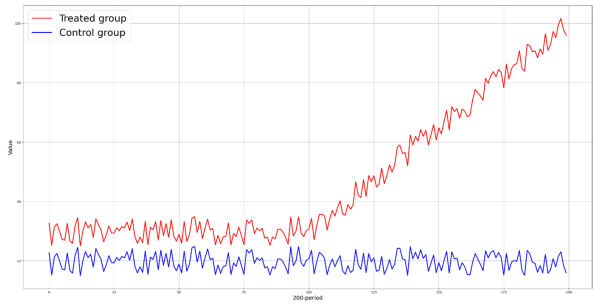

FIGURE 10. Plot of treated and control group time series data with random noise.

Appendix B. Bidirectional LSTM Model code

```
def bi_lstm_iteration(treatment_period, seed, data):
    pre_period = [0, treatment_period]
    post_period = [treatment_period + 1, 199]

    list_steps = [8, 9, 10, 11, 12, 13, 14, 15, 16]
    best_epoch_value = []
    best_epoch_cicle = []
    list_val_mae = []
    lista_result_fin = []

    for step in list_steps:
        seed_values = seed
```

```
os.environ['PYTHONHASHSEED'] = str(seed_values)
random.seed(seed_values)
tf.random.set_seed(seed_values)

PaAn = pre_period[1]
parametro_inizio = pre_period[0]

raw_seq = data[0:treatment_period]
n_steps = step

X, y = split_sequence(raw_seq, n_steps)
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

callback = tf.keras.callbacks.EarlyStopping(monitor='
    val_loss', restore_best_weights=True, patience=40)

model = Sequential()
model.add(Bidirectional(LSTM(50, activation='LeakyReLU
    '), input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='Adam', loss=custom_loss,
    metrics='mae')

history = model.fit(X, y, epochs=100, validation_split
    =0.33, callbacks=[callback], verbose=0)

list_val_loss = history.history['val_loss']
list_val_loss_mae = history.history['val_mae']

xmin = min(list_val_loss)
yindex = list_val_loss.index(xmin)
val_mae_val = list_val_loss_mae[yindex]
list_val_mae.append(val_mae_val)
best_epoch_cicle.append(yindex)
best_epoch_value.append(xmin)

os.environ['PYTHONHASHSEED'] = str(seed_values)
random.seed(seed_values)
tf.random.set_seed(seed_values)
```

```
        model1 = Sequential()
        model1.add(Bidirectional(LSTM(50, activation='
            LeakyReLU'), input_shape=(n_steps, n_features)))
        model1.add(Dense(1))
        model1.compile(optimizer='Adam', loss=custom_loss,
            metrics='mae')

        history1 = model1.fit(X, y, epochs=yindex+1,
            validation_split=0.33, verbose=0)

        yhat = model1.predict(X, verbose=1)

        prediction = []
        current_batch = np.array(yhat[:PaAn])[-n_steps:]
        current_batch = current_batch.reshape(1, n_steps,
            n_features)

        for i in range(10):
            current_pred = model1.predict(current_batch,
                verbose=0)[0]
            prediction.append(current_pred)
            current_batch = np.append(current_batch[:, 1:, :],
                [[current_pred]], axis=1)

        observed = listadd[PaAn:PaAn+10]

        resultfin = np.mean(np.array(observed) - np.array(
            prediction))
        listaresultfin.append(abs(resultfin))

    return best_epoch_value, best_epoch_cicle, list_val_mae,
        listaresultfin
```

Bidirectional LSTM Model with custom Loss and MAE Evaluation for time Series prediction.

References

Brodersen, K. H., Gallusser, F., Koehler, J., Remy, N., and Scott, S. L. (2015). "Inferring causal impact using Bayesian structural time-series models". *The Annals of Applied Statistics* **9**(1), 247–274. DOI: 10.1214/14-AOAS788.

Caliendo, M. and Kopeinig, S. (2008). "Some practical guidance for the implementation of propensity score matching". *Journal of Economic Surveys* **22**(1), 31–72. DOI: 10.1111/j.1467-6419.2007.00527.x.

Cervantes, P. A. M. and Rambaud, S. C. (2020). "An empirical approach to the "Trump Effect" on US financial markets with causal-impact Bayesian analysis". *Heliyon* **6**(8), e04760. DOI: 10.1016/j.heliyon.2020.e04760.

Cui, Z., Ke, R., Pu, Z., and Wang, Y. (2018). "Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction". arXiv: 1801.02143 [cs.LG]. URL: https://arxiv.org/abs/1801.02143.

Dolton, P., Bondibene, C. R., and Wadsworth, J. (2010). "The UK national minimum wage in retrospect". *Fiscal Studies* **31**(4), 509–534. URL: https://EconPapers.repec.org/RePEc:ifs:fistud:v:31:y:2010:i::p:509-534.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. URL: http://www.deeplearningbook.org.

Hochreiter, S. and Schmidhuber, J. (1997). "Long short-term memory". *Neural Computation* **9**(8), 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

Hu, J., Wang, J., and Ma, K. (2015). "A hybrid technique for short-term wind speed prediction". *Energy* **81**, 563–574. DOI: 10.1016/j.energy.2014.12.074.

Leeflang, P. S. H., Bijmolt, T. H. A., Doorn, J. van, Hanssens, D. M., Heerde, H. J. van, Verhoef, P. C., and Wieringa, J. E. (2009). "Creating lift versus building the base: current trends in marketing dynamics". *International Journal of Research in Marketing* **26**(1), 13–20. DOI: 10.1016/j.ijresmar.2008.06.006.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Vol. 25. San Francisco, CA, USA: Determination Press. URL: http://neuralnetworksanddeeplearning.com/.

Wing, C., Simon, K., and Bello-Gomez, R. A. (2018). "Designing difference in difference studies: best practices for public health policy research". *Annual Review of Public Health* **39**(1), 453–469. DOI: 10.1146/annurev-publhealth-040617-013507.

Zhang, S., Bi, K., and Qiu, T. (2019). "Bidirectional recurrent neural network-based chemical process fault diagnosis". *Industrial & Engineering Chemistry Research* **59**(2), 824–834. DOI: 10.1021/acs.iecr.9b05885.

[a] Università degli Studi Mediterranea di Reggio Calabria,
Dipartimento di Giurisprudenza, Economia e Scienze Umane,
Reggio Calabria, Italy

[b] Università Bocconi,
Dipartimento di Management e Tecnologia, ICRIOS,
Milano, Italy

* To whom correspondence should be addressed | email: massimiliano.ferrara@unirc.it

Communicated 24 November 2022; manuscript received 2 February 2023; published online 1 July 2023