POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Price Elasticity Estimation by Causal-inference Methods: Lessons from Real-world Applications

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica


Authors:
**Eknid Mucollari**
**Stefano Minotti**

Student IDs: 952738, 945543
Advisor: Prof. Nicola Gatti
Co-advisor: Jacopo Tagliabue

Academic Year: 2021-22

# Abstract

The elasticity of the demand with respect to the price of a good have always been a key indicator in the field of pricing. In the process of deciding the correct price for a product, its elasticity can be used both as a qualitative metric, informative on how customers react to changes in the price, and as a parameter to be directly incorporated in sophisticated algorithm. The estimation of the elasticity can be seen as a particular problem of causal inference in which we want to estimate the effect that a phenomenon, i.e. the variation in price, has on a dependent variable, i.e. the quantity demanded. This work focuses on the analysis of two causal inference approaches for the purpose of estimating the price elasticity of demand. The first one is a Machine Learning approach based on the Double Machine Learning technique while the second one is a more traditional approach based on Difference-in-Differences. This thesis discuss about their application in the field of elasticity estimation and, thanks to the data of two e-commerce provided by the Canadian company Coveo, provides a practical demonstration of their usage in a real-world scenario.

**Keywords:** price elasticity of demand, causal inference, e-commerce, double machine learning, difference-in-differences

# Abstract in lingua italiana

L'elasticità della domanda rispetto al prezzo di un bene è da sempre un indicatore chiave nel campo del pricing. Nel processo di determinazione del prezzo ottimo per un prodotto, la sua elasticità può essere utilizzata sia come metrica qualitativa, che fornisce informazioni su come i clienti reagiscono alle variazioni del prezzo, sia come parametro da incorporare all'interno di sofisticati algoritmi. La stima dell'elasticità può essere vista come un particolare problema di inferenza causale in cui si vuole stimare l'effetto che un fenomeno, la variazione di prezzo, ha su una variabile dipendente, la quantità richiesta per un prodotto. Questo lavoro si concentra sull'analisi di due approcci di inferenza causale allo scopo di stimare l'elasticità della domanda rispetto al prezzo. Il primo è un approccio di Machine Learning basato sulla tecnica del Double Machine Learning, mentre il secondo è un approccio più tradizionale basato su Difference-in-Differences. Questa tesi discute della loro applicazione nel campo della stima dell'elasticità e, grazie ai dati di due e-commerce forniti dall'azienda canadese Coveo, offre una dimostrazione pratica del loro utilizzo in uno scenario reale.

**Parole chiave:** elasticità della domanda rispetto al prezzo, inferenza causale, e-commerce, double machine learning, difference-in-differences

# Contents

# Introduction

## Research field

Pricing is a basic task that is central in economics worldwide. The study of how choosing the best pricing policy is a long-standing problem in artificial intelligence and machine learning scientific community. The approaches to address this problems come under many flavours, e.g., in non-stationary settings [1], in adversarial settings [2], combining information signaling with pricing [3], and combining advertising with pricing [4]. In our work, we focus on the problem of estimating the *elasticity* of the demand from data. In particular, in economics, we define the price elasticity of demand as the change in demand with respect to a change in price. It provides information on how customers react to changes in the price of goods. This is an essential topic in economics given the various applications it finds in the market. Sellers are interested in discovering about elasticity because this indicator can be used in the field of pricing in order to correctly apply pricing strategies and maximize profits.

If we imagine the demand of a product being regulated by a function, the price elasticity of demand is essentially the derivative of this function with respect to the price. Since the demand function cannot be known *a priori*, it is necessary to rely on algorithms capable of obtaining an estimate of the elasticity that is as reliable as possible. The quality of the estimate depends on the algorithm used, and each algorithm can be used under certain conditions or in the presence of specific scenarios. The algorithms applicable with the purpose of elasticity estimation come from a larger research area that is the field of causal inference.

Causal inference is a research field that has the goal of determining the independent, actual effect of a particular phenomenon that is a component of a larger system. It analyzes the response of a dependent variable related to the realization of that phenomenon. The elasticity estimation can be seen as a particular problem of causal inference in which we have to estimate the independent effect that a change in the price of a good has on the quantity demanded for that good. However, being the price only one of the factors

influencing the demand its effect has to be isolated.

There exists a multitude of algorithms of causal inference, each one working under different assumptions and exploiting different mathematical and statistical theorems. Most of them have a wide history of literature providing proof of works in simulated environments and experimental application on real-world scenarios. We give our contribute to this research area by analyzing the application of two causal inference algorithms in the important field of elasticity estimation for the e-commerce sector. Thanks to the data provided by the Canadian company Coveo, we have been able to analyze the values of elasticity estimated on real-data related to two e-commerce operating in the same market and selling similar products.

This work also presents an in-depth description on how the algorithms under consideration have been applied in the specific case of elasticity estimation. Thanks to this, our approaches can be replicated, both under different assumptions and pre-requisites, in any suitable scenario in which the goal is to estimate the effect of price changes on the demand of certain goods.

## Summary

In this work we begin providing some theoretical basis on the elasticity of the demand with respect to the price. We describe, providing examples from the literature, how it can be used in the process of pricing goods both as a qualitative metric and a parameter for pricing algorithms, and we discuss the problem of its estimation.

We then describe the data provided by Coveo analyzing the differences between the two shops under consideration in terms of volumes of sales and price trends and describe the approach we used to identify data from both shops that refer to the same real-world products. This operation relies on an pre-existent Deep Learning architecture called Deepmatcher [5] that we tuned and trained at our purposes. The results provided by this network are scores that can be interpreted as the probability of two product data entries to refer to the same real-world entity. Being these results related to each pair of product entries we introduced a graph-based approach to exploit the scores of each pair and find clusters in which data represents effectively the same real-world product. In the process we focused on prioritizing quality over quantity thus ensuring our matches to be truthful at the cost of missing a few.

After this preliminary step we move on to the presentation of the two causal inference approaches that we analyzed with the goal of elasticity estimation. The first one is based

on the Double Machine Learning technique originally proposed by Chernozukov et al. [6] while the second one relies on the Difference-in-Differences approach practically realized through two-way fixed effects regressions. For each of the two approaches we provide the theoretical basis on which they relies and describe how they can be applied in the specific case of the elasticity estimation. We describe our practical application and analyze the results obtained. Lastly, we give a final comparison of the two approaches and analyze their strengths and limitations.

Given that a proper way of estimating the accuracy of the effects estimated with causal inference on real-world data does not exist, we analyzed our results based on how we expect the elasticity of the considered products to be. Since the products under consideration are sport goods we expect their demand to increase in a more than proportional way when their price decrease and vice versa, i.e. we expect them to be elastic products. For simplicity, we focused our analysis on 10 products accurately selected between the ones sold by both shops with a relatively high volume of sales and price changes. For these products, the average elasticities obtained are similar between the two approaches and coherent with our expectations. For both Double Machine Learning and Difference-in-Differences, we applied the algorithms with different parameters setups (data aggregation, machine learning models, ...), compared the results, and tried to understand what factors influences the divergences between them.

## Structure

The continuation of this work is organized with the following structure:

- **Price elasticity of demand**. The first chapter provides the theoretical basis behind the price elasticity of demand, from the demand curve and the law of demand to the definition of elasticity. It then introduce the pricing process, describes the factors influencing pricing decisions and provides a short overview of the two main families of pricing algorithms: reinforcement learning and model-based algorithms. It goes on by describing the impact of the elasticity in the pricing process both as decision-making factor and parameter directly incorporated into algorithms. Finally it focuses on its calculation and introduces the two approaches analyzed in this work.
- **Data description**. This chapter is about the data provided by the Canadian company Coveo. It provides a description of the main features of the dataset together with some plots analyzing the prices proposed to the customers by the two shops under consideration and the demand they face.

- **Product matching**. In this chapter we present the approach we use to match common products between the two considered shops. It is based a Deep Learning approach relying on an already known network for entity matching called Deepmatcher. The chapter describes the structure of the network, our specific design choices and the complete training process. Finally, it presents the graph based strategy used to cluster products according to the similarity scores provided by the network.
- **Double Machine Learning**. This chapter focuses on the Double Machine Learning causal inference algorithm. It presents its theoretical basis summarized from its original paper and analyze how the method can be applied in the field of elasticity estimation. Finally, it describes our practical application, present and analyze the results obtained in our real-world scenario.
- **Difference-in-Differences**. This chapter is centered around the other causal inference approach considered in this work, called Difference-in-Differences. As the previous chapter, it provides a theoretical background and describes how, and under which assumptions, this approach can be applied for the estimation of price elasticity of demand. It concludes by describing the application on our data and analyzing the results obtained.
- **Comparison**. In this chapter we provide a brief comparison between the two analyzed approaches. We focus on their advantages and limitations and we try to identify under which condition one approach is be better than the other.
- **Conclusion and future developments**. In the last chapter we summarize our results and discuss about possible future works related to this research.

# 1 | Price elasticity of demand

In this chapter we describe the concept of price elasticity of demand and present its theoretical basis. We then introduce the concept of pricing and dynamic pricing and describe the application of price elasticity at these purposes, bringing also some examples from the literature. After that, we discuss the estimation of elasticity from historical data and introduce the methods we use and describe in the following chapters.

## 1.1. Theoretical definition

In economics the term demand refers to the amount of some good or service consumers are willing and able to purchase at each price. In most of the cases, the demand of a product follows a fundamental principle called *law of demand* that states that the quantity purchased varies inversely with price [7, 8]. The law of demand is represented by a graph called the demand curve, in the graph we have the quantity demanded on the x-axis and price on the y-axis and the curve is downward-sloping according to the law.

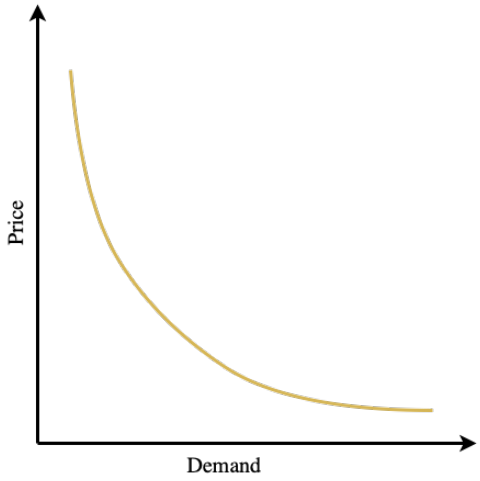


Figure 1.1: Example of demand curve.

This law is generally valid but there are also exceptions such as Giffen and Veblen goods; these are goods for which the demand increases as the price increases [9].

The variation in demand with regards to a change in price is known as the *price elasticity of demand*. It is calculated as the percentage change in quantity demanded divided by the percentage change in price:

$$\epsilon = \frac{\Delta Q}{\Delta P} \times \frac{P}{Q}. \tag{1.1}$$

As the size of the price change gets bigger, the elasticity definition becomes less reliable for two reasons:

- A good's elasticity is not necessarily constant along the demand curve because a 1% change in price has an effect that may depend on the initial price. For example, a change of 1% in price at 1000€ may have a different effect on the demand with respect to the same change at 10€.
- The percentage change between any two values depends on which one is chosen as the starting value and which as the ending value. For example, consider the case in which the price of a good rises from 10€ to 20€ and its demand falls from 100 units to 50. The price increase is 100% and the quantity decline is 50% and thus the elasticity is $-50\%/100\% = -0.5$. If the price falls from 20€ to 10€ and the demand increases from 50 to 100, however, the price elasticity is $100\%/-50\% = -2$ for the same part of the curve.

To overcome these limitations, in our work we consider the case of constant elasticity and we assume that the elasticity of a product is constant between the range of prices at which the product is usually sold. For example if a pair of shoes is usually sold in a range of prices between 50€ and 80€ we consider the elasticity constant for each price in this range. In addition, we can change the elasticity formula in order to minimize the difference between the starting and ending prices and quantities. This formula is called *point elasticity* and it is calculated as:

$$\epsilon = \frac{\mathrm{d}Q}{\mathrm{d}P} \times \frac{P}{Q}, \tag{1.2}$$

where $\frac{\mathrm{d}Q}{\mathrm{d}P}$ is the derivative of the demand with respect to the price. It measures the change in quantity demanded for very small changes in price. However, since it requires to compute the derivative of the quantity demanded with respect to the price, it can be computed only if the demand function is known.

The price elasticity of demand is a fundamental metric that is used by the companies in the process of setting the pricing of their goods. It can be treated just as an indicator that gives an insight on how sensitive are the customers to changes in prices but can also

be incorporated in more sophisticated algorithms to dynamically select the best prices in real-time.

## 1.2. Pricing decisions

Pricing refers to the decision-making process that a business should tackle to establish the value for a product or service. An *efficient price* is a price that is very close to the maximum that customers are willing to pay, it is a price that shifts most of the consumer economic surplus to the producer. The price selected during the process should lay between the *price floor* (the price below which the business ends up in losses) and the *price ceiling* (the price by which the business experiences no demand).

The pricing process must keep into account several factors, we report here the most significant:

- **Costs**. The first aspect to consider is about business costs. This category includes *fixed* costs that a company must pay regardless of sales or production level and variable costs that are the expenses the business incur by producing and delivering its products and services. All these costs should be calculated and kept into account for correctly pricing a product.
- **Competitors**. A company must look at its competitors in order to adapt the price of its goods. If, for example, a competitor sells the same product at a lower price, customers are more likely to buy from it. This is true in general but there may be other aspects to keep into account such as customers loyalty, quality of services and so on. Not only the price of the considered product must be analyzed but also the presence on the market of potential substitute products that may shift customers attention.
- **Laws and regulations**. Some government often applies price regulations in order to protect customers and encourage competitions. One of the main goals of regulations is avoiding the phenomenon of *price fixing* which occurs when companies get together and agree to charge the same prices. It usually involves setting high prices so consumers are obliged to pay high sums regardless of where they purchase a good. Sometimes, government regulations aim at preventing large businesses from selling products below cost to attract customers to the store; this is usually done by setting a minimum selling price.
- **Economy**. The general economy trend must be kept into account in order to adjust prices. When the economy is weak, prices should be lowered accordingly, on

the contrary they can be higher in strong economy periods.

- **Customers**. Customers are maybe the most important aspect to consider when pricing a good. The two main factors to keep into account are the overall number of potential customers and how they react to changes in prices. The second aspect is directly measured by the price elasticity of demand, in fact its estimate is a key point in most of the pricing strategies.

## 1.3. Dynamic pricing

Correctly pricing products is so important that today the world largest companies adopt automated pricing techniques which fall under the name of algorithmic or dynamic pricing. Dynamic pricing is a strategy in which businesses set flexible prices for products or services based on current market status. Companies are able to adjust prices, sometimes in a matter of minutes, based on algorithms that take into account all the factors described in the previous section [10].

Dynamic price is adopted in a lot of different fields. Some well known examples are in the filed of hospitality in which higher prices are charged during the peak season or during special event periods, and companies also adjust the cost of rooms based on the supply and demand at a particular moment [11]. A similar phenomenon happens in the field of transportation [12]. Other examples of dynamic pricing can be found in the retail sector, especially with e-commerce shops in which many businesses adjust their prices automatically based on competitors, market price, seasons, marketing efforts [13–15].

### 1.3.1. Reinforcement learning

A common approach to estimate the optimal price is reinforcement learning (RL) [14, 15]. Reinforcement learning is an area of artificial intelligence that studies decision making; it is about learning the optimal behavior in an environment to obtain maximum reward. Reinforcement learning involves an agent, a set of states, and a set of actions per state. By performing an action, the agent transitions from state to state. Executing an action in a specific state provides the agent with a reward. RL focuses on finding the optimal policy while balancing *exploration* and *exploitation*; it offers a good trade-off between the two phases because the algorithms do not sharply split the phases but keep gradually improving until convergence to optimum. In addition, algorithms based on reinforcement learning usually makes only few — or even none at all — assumptions on the function regulating the demand [16–18].

The main downside of using these algorithms is that they do not rely only on historical data to calculate the best price but instead they require to gather new data after introducing a new pricing policy; the prices must be proposed to real customers in order to measure the reward. This requires some bootstrapping before converging to the optimum and during this period customers may experience prices changing very frequently. This means that if two customers purchase the same product but at even slightly different times, one ends up paying more than the other . This tends to upset customers who had to pay a higher price [19]. In addition these techniques are not easily applicable by small or medium businesses that do not have a large pool of customers on which performing the exploration. With few samples the convergence to the optimum may be very slow thus extending the loss of potential incomes. This may be sustainable for large companies but not worth for smaller businesses.

## Bandit algorithms

A family of algorithms well suited for pricing tasks is the one of bandit algorithms. These algorithms have been studied for decades [20] but their application is increasing in popularity in the last years, especially with the growth of the online retail sector. These algorithms are based on the multi-armed bandit problem and allow to estimate the optimal price for a good in real-time, without making prior assumptions on the demand model [21, 22]. We also mention some works exploiting specific structures of the problem (e.g., monotonicity of the demand) to obtain better performance [23–25], while some attempts focus on merging dynamic pricing algorithms together with clustering techniques to obtain a more accurate segmentation of the customers.

The term "multi-armed bandit" comes from a hypothetical experiment where a person must choose between multiple actions each with an unknown reward; the idea is that selecting an action is like deciding which machine to play between a row of slot machines (sometimes known as "one-armed bandit"). The goal is to determine the best or most profitable outcome by progressively testing different arms.

These kind of algorithms aims at minimizing the regret, i.e. the reward potentially lost during the exploration phase. In fact, before the algorithm converges to the optimum policy, many non-optimal arm are pulled thus providing rewards that are lower than the one that is obtained by pulling the optimal arm. Actually, the regret is the difference between the cumulative reward that would have been obtained by pulling only the optimal arm and the one that is truly obtained.

In the specific case of pricing, these algorithms rely on a set of prices decided in advance.

The prices represent the arms to be pulled in order to select the best one based on a certain reward function (that usually depends on the conversion rate, i.e. the probability that a user will buy a good). This function is initially unknown and will be progressively estimated by the algorithm. At each collected sample, the algorithms update its expected reward for the pulled arm and at each iteration the arm with the highest reward is pulled. At a certain point, the algorithm will converge and start selecting only the optimal price. We also mention that, in realistic settings, the demand curve is not stationary, thus requiring the design of bandit algorithms dealing with changes in the environment, see, e.g., [26].

## Q-learning algorithms

$Q$-learning algorithms [27] are another family of algorithm that is commonly used for dynamic pricing. Given a set of states $S$ and a set of actions $A$, the algorithm is based on a function — the $Q$ function — that calculates the quality of a state–action combination:

$$Q : S \times A \to \mathbb{R}.$$

$Q$ is initialized to an arbitrary value and then, at each time $t$ the algorithm selects an action $a_t$, observes a reward $r_t$, enters a new state $s_{t+1}$, and $Q$ is updated. The estimated $Q$ function is used to select which action to perform at each iteration. For example, using the $\epsilon$-greedy policy we select the action which has the highest $Q$-value with probability $1 - \epsilon$ or any action at random with probability $\epsilon$.

The $Q$-learning approach can be applied to select the optimal pricing policy for a good. An example from the literature can be found in Cheng (2009) [28], in which the authors investigated a $Q$-learning approach which is based on real-time demand learning to solve dynamic pricing problems. In their work, the states of the algorithms are identified by a triple $s_t = ((T - t), I_t, d_t)$, where $(T - t)$ is the time length from decision epoch $t$ to end of sale horizon, $I_t$ is the stock of the product at epoch $t$ and $d_t$ is the current demand state. The set of actions is a fixed set of discounts to be applied to the initial price of the product. This is one of the simplest examples of $Q$-learning application but we also have more sophisticated algorithms that operate in more advanced scenarios. We have multi-agent algorithms that focus on studying the behavior of multiple agents that coexist in a shared environment (e.g. two competitor shops), deep learning based methods that use neural networks to approximate the $Q$ function in very high dimensional state spaces (where calculating $Q$ for each state is not feasible) and many other variations tackling different scenarios and field of application.

### 1.3.2. Model-based algorithms

On the other side we have a family of algorithms which are not based on a trial and error procedure. They are based on classic statistical models or machine learning models trained on historical data. These algorithms usually aim at modeling the demand function by keeping into account many different factors and then suggest the best prices considering the predicted demand and and the business costs (production, advertising, ...). Whenever new data are obtained the models can be calibrated or re-trained in order to account for variations in the demand function over time.

A lot of research have been conducted about these algorithms for price optimization. There exists a multitude of different algorithms each one exploiting different techniques for modeling the demand. A great work of gathering and providing an in-depth overview of the available literature on dynamic pricing and learning have been done by V. den Boer (2014) [29] in which the author reviews the most significant algorithms for dynamic pricing with demand uncertainty.

The advantage of using this dynamic pricing approach is that it guarantees an optimal pricing strategy since the first time it is applied without the need of an exploration phase. This characteristic also makes model based algorithm more suitable for small businesses that does not want to deal with trial and error approaches. The main downside is that the development of these algorithms requires a great knowledge of the environment in which they will run because the demand function must be correctly modeled in order to provide accurate estimations.

## 1.4. Elasticity for pricing

Price elasticity of demand is a key metric that must be kept into account when performing pricing or adopting a dynamic pricing strategy. The elasticity gives us information on the sensitivity of customers to changes in price, i.e. how the demand for a good will change in response to a change in the price of that good. We can identify different types of demand based on the value of elasticity ($\epsilon$):

- For $\epsilon = 0$ the demand is said to be *perfect inelastic*. In particular, changes in the price do not affect the quantity demanded, so raising prices will always cause total revenue to increase. We do not observe this scenario in real-world, it would be the case in which the alternative at buying a good was death.
- For $-1 < \epsilon < 0$ the demand is said to be *relatively inelastic*. In particular, the percentage change in quantity demanded is smaller than that in price, so when the

price is raised, the total revenue increases.

- For $\epsilon = -1$ the demand is said to be *unit elastic*. In particular, the percentage change in quantity demanded is equal to that in price, so changing price will not affect the total revenue.
- For $-\infty < \epsilon < -1$ the demand is said to be *relatively elastic*. In particular, the percentage change in quantity demanded is greater than that in price, so when the price is raised, the total revenue falls, and vice versa.
- For $\epsilon = -\infty$ the demand is said to be *perfectly elastic*. In particular, any increase in the price will cause the quantity demanded to drop to zero, so when the price is raised, the total revenue falls to zero. This is the case of goods that have their value defined by some law or regulation.

When performing traditional pricing, without relying on automated strategies, elasticity is probably the the most informative on how we should change the price of a good. Even though the demand is influenced by many factors, price is usually the main one on which the quantity demanded for a product depends on. The elasticity allows us to estimate this dependency. Theoretically, the closer this metric to zero the higher the chances that increasing the price of a good will bring an overall improvement in the revenues, while the more it is lower (under -1) the more it will be worth reducing the price. This is not always guaranteed since, as said above, other factors may act at influencing the demand.

Consider for instance the case in which the price of a pair of winter gloves have to be set at the end of the cold season. Suppose to know that this product is elastic (i.e. elasticity lower than -1), thus by reducing its current price we should expect a more than proportional increment of the quantity demanded and an increase of the overall profit. Anyway it is very unlikely for this to happen since it must be kept into account that we are approaching the end of the winter season. After this season, gloves start to be bought gradually more seldom, so despite the price reduction their demand will probably decrease in any case. Indeed seasonality of products is one of the factors to be considered when estimating the effect of a price change. However, if we do not expect changes in other factors external from the price that may influence customers behaviour the elasticity can provide reliable information on the pricing strategy to adopt.

Moving to algorithmic pricing, the first way in which price elasticity can be very useful is to estimate the impact of a pricing strategy on the customers. An example from the literature, about the power market, can be found in an article from Ahmad Faruqui and Stephen S. George [30] in which the authors analyze elasticities calculated for different

pricing strategies in order to identify whether the customers shift power usage in response to time-varying price rates. This is interesting because by knowing whether the customers change their power consumption according to different prices, policy makers can understand whether the costs of implementing dynamic pricing strategies are covered by lower supply-side costs. Another interesting study comes from Aparicio (2021) [31]; here the authors analyze the pricing strategies of U.S. online groceries retailers and among many factors they calculate the elasticity of products in order to explain price differentiation.

Elasticity can also be directly incorporated in pricing algorithms and being effectively used to estimate the best price strategy. A first example comes from Fisher (2018) [32]; in this work the authors partnered with a leading Chinese online retailer and studied a competition-based dynamic pricing algorithm. They estimated own price elasticties of products and cross-elasticities (i.e. elasticity of demand with respect to the price products in competitor shops) and incorporate these values in an optimization problem in order to define optimal price responses to competitor price changes. Another interesting approach comes from Kedia (2020) [13], here the authors designed a model to predict the future demand that a product would have if it was sold at a its base price and then, using price elasticity, they calculated multiple demand values obtained by increasing or decreasing this price. In this way they got multiple price-demand pairs among which choosing the price that maximizes the revenue.

## 1.5. Elasticity estimation

In the previous section we gave some insights on how estimating the price elasticity of demand can be useful in the pricing process at different extents. In real-world we do not know the demand function that regulates the quantity demanded for goods and thus the elasticity cannot be exactly calculated using Equation 1.2. It can be estimated essentially in two ways:

- **Randomized experiments**. With this approach a company has to set up an "live" environment in which slightly different price variations are applied at random at different time steps. In this way, the elasticity is estimated by comparing the demand at different prices.
- **Historical data**. This is the most common approach and consists in analyzing the historical data of a product in order to analyze its natural price and demand variations.

The first approach is for sure more precise since it is based on a pricing system im-

plemented ad-hoc at the purpose of estimating price elasticities. It allows to compare different prices in a relatively short period thus mitigating the possible influence of external factor influencing the demand. However, in many situations using historical data can still provide comparable results with significantly less effort since the data to be analyzed have already been collected. The study of this approach is the focus of our work.

One can state that calculating the elasticity is just a matter of finding a price change and calculating the ratio between the change and the variation of the average demand between the periods before and after this change. This is not true because many other factors influence the demand thus introducing bias in this estimate. A simple example is when price changes happen close to sale periods; in these situations the change in demand will be influenced both by the price and by the incoming sales (that may increase the willingness of customers at buying goods). In these situations, in order to have an unbiased elasticity estimation we need to isolate the portion of demand that really depend on price and try to exclude the influence of external factors.

The main goal of our work is analyzing different approaches, based on algorithms of casual inference, to estimate the unbiased elasticity of products from historical sales data. The first one is a Machine Learning approach based on the Double Machine Learning technique from Chernozhukov (2018) [6], applicable in almost any situation, and already proposed in the field of elasticity calculation. The the second one is a more traditional approach based on the Difference-in-Differences algorithm that compares the historical data of a product in two different shops in order to estimate its elasticity. The Difference-in-Differences is a common approach used in the estimation of the effect of a treatment on a population. However, most of its applications consider only binary treatments (e.g. the introduction of a law). Callaway (2021) [33] demonstrates that, under certain assumptions, this method can be extended to the case of continuous treatments and we describe its application in the field of elasticity calculation. We do not limit only at describing the procedures but also provide a practical application in a real-world scenario by calculating price elasticities for products sold by two e-commerce shops whose data have been provided by the Canadian company Coveo.

# 2 | Data description

The technique we will describe in the following chapters have been tested on a real-world scenario thanks to data provided by the Canadian company Coveo. Coveo is a leading company in the field of artificial intelligence and machine larning applied in e-commerce and can collect data coming from a lot of different shops. For our research, Coveo provided us the historical data of two shops operating in the field of sport goods. There exists other similar public datasets [34] but ours, which is private, has the particular feature of being multi-shop. For confidentiality, in this work the two shops will be named Shop A and Shop B.

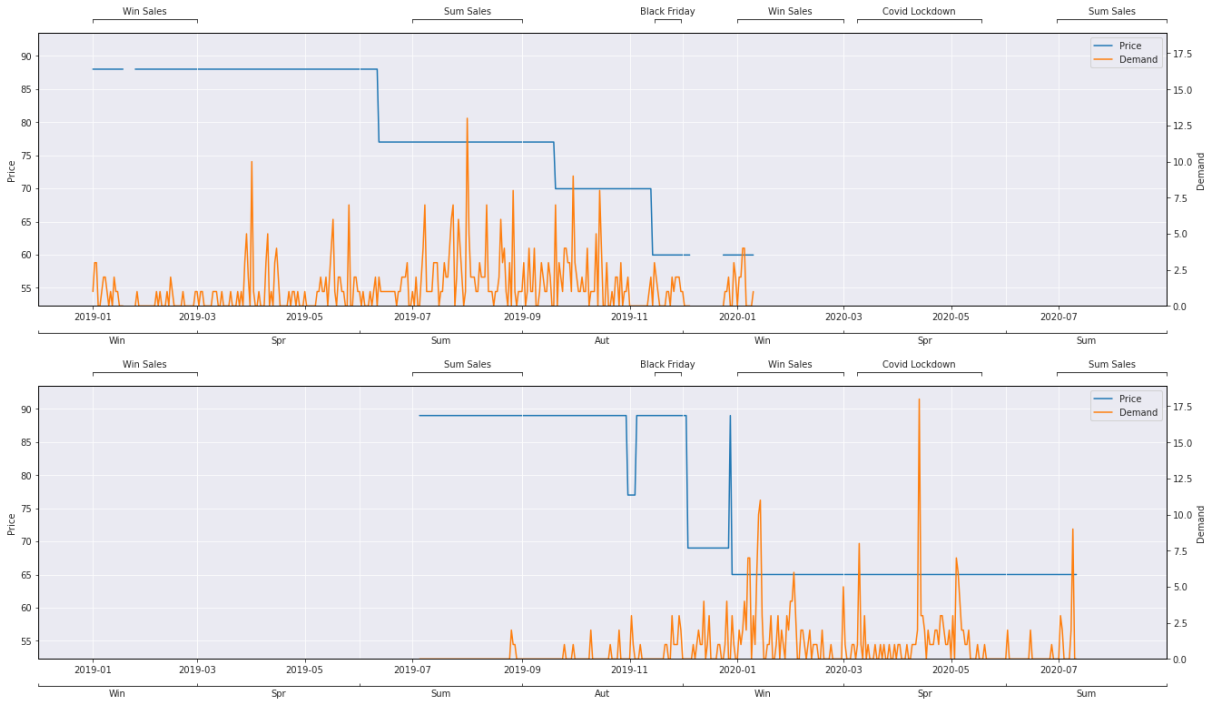


Figure 2.1: Example of sales history for a product in, respectively, Shop A and Shop B. The blue curve represents the price at which the good is sold while the orange curve represents the quantity demanded for each day.

The data have a time span of 19 months, from January of 2019 to July of 2020. For each product and for each day, we have information on the actions performed by the users on that product. In particular, we have the number of visits on the product page, the number of times the product have been added to a user cart and the number of times it have been bought. In addition, we have descriptive data about products such as name, brand, description, categories under which a product is listed on a specific day and other raw metadata that are different between the two shops. In Table 2.1, we report the description of the features of our dataset. With this data we can reconstruct the whole sales history for each product at catalog that can then be analyzed in order to extract information on its elasticity.

| **Feature** | **Description** |
|---|---|
| Date | Date to which the entry refers. |
| Stock Keeping Unit (SKU) | The identification code of the product to which the entry refers. |
| Action type | Indicates whether the row refers to a *detail* (click on product page), *add* (add to cart) or *purchase* action. |
| Count | Number of actions (indicated by *Action type*) measured in that day. |
| Name | Name of the product as displayed on the e-commerce website in that day. |
| Brand | Brand to which the product belong. |
| Categories | Categories under which a product is listed in that day. |
| Description | Description of the product as displayed on the e-commerce website in that day. |
| Price | Price of the product for that day. |
| Raw | Other raw metadata. |

Table 2.1: Original data features.

After some preliminary analysis on the data we found out that most of the products are sold very seldom and that Shop B has an overall volume of sales that is less than half with respect to the volume of Shop A. The average number of unit sold in a day for a product is shown in the following figure. For products that are not sold or sold very few times obviously the estimation will not be reliable (or not possible to be calculated at all).

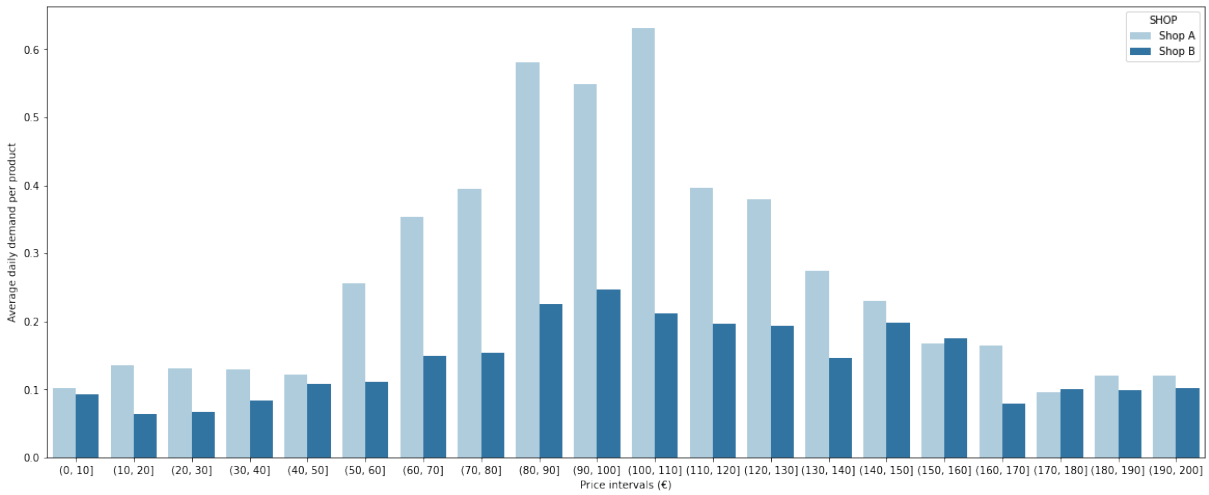


Figure 2.2: Average daily demand for a product.

Another interesting aspect to consider is the number of price changes in the products of the two shops. The fewer prices we can observe for a product the more the elasticity estimation will be uncertain. The distribution of the number of price changes for the products is shown in Figure 2.3. In addition, since we will focus our study on the case of constant elasticities we also have to assume that the elasticity of products is constant among the range of prices at which they are sold. This assumption is acceptable if this range is relatively restricted. In Figure 2.4, we report the distribution of the percentage change of the price when a price change happens. We can notice that most of the price changes bring a reduction of price by a percentage lower than 20%, the general trend for the price of products is decreasing and the gap between the maximum and minimum price for a product, on average, is restricted.

The results we will present in this work are limited to a set of products that we selected between the ones that have been sold a reasonable amount of times in both shops and that undergo a significant amount of price changes in a relatively restricted price interval. In this way, we can at least guarantee that the data on which our techniques are applied are informative on the price elasticity of the considered products.

Before being able to test our approaches for calculating the elasticity of the products we had to apply a deep learning based approach in order to match products to be considered same across the two shops. The procedure will be explained in the following chapter.

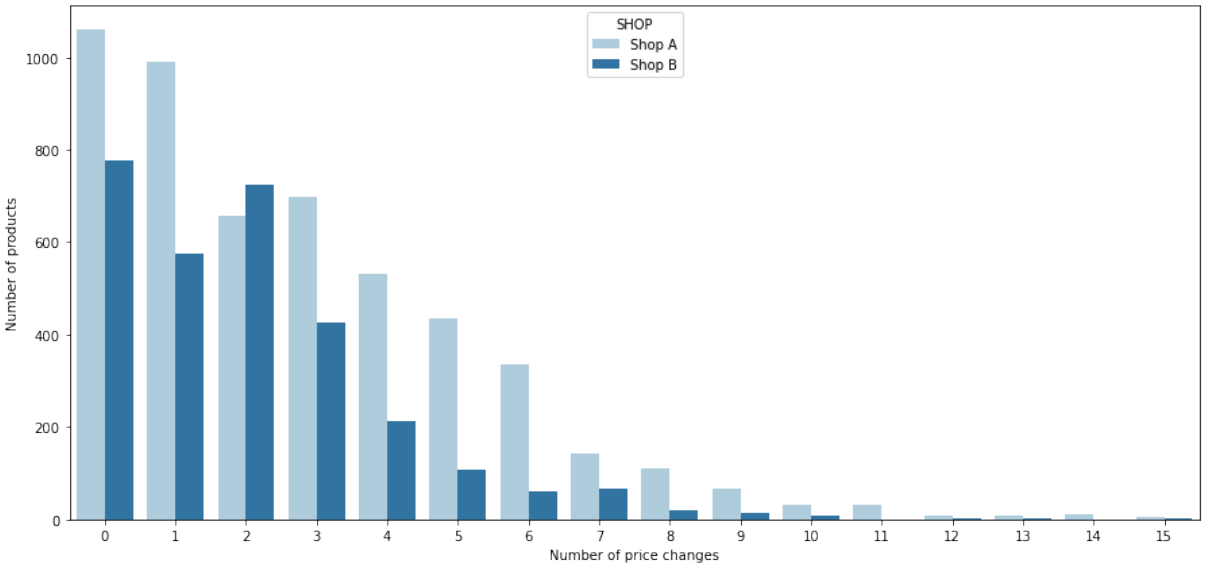


Figure 2.3: Distribution of the number of price changes for a product.

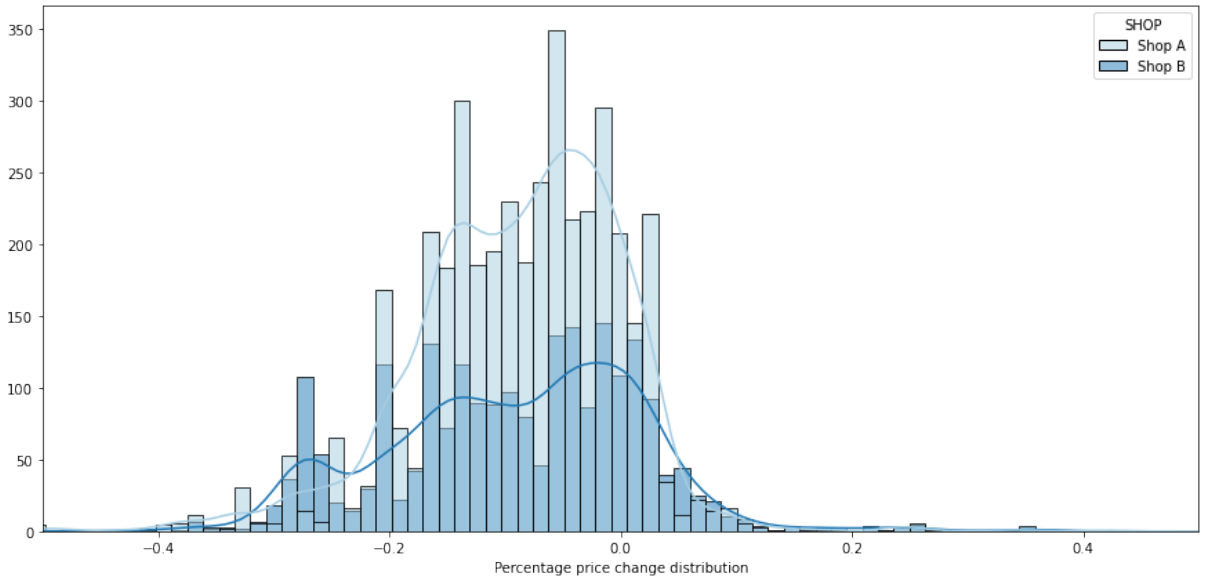


Figure 2.4: Distribution of the average percentage price change.

# 3 | Product matching

In this chapter, we illustrate the procedure used for matching products between the two shops. First, we provide a description of Deepmatcher [5], a pre-existing supervised deep learning solution to find data instances that refer to the same real-world entity. Then we move on to our practical application of the network, starting from the data extraction and preparation of the training set. After that, we explain our training strategy, parameters tuning, final testing and ensembling of the best models. As last thing we show a graph based strategy to automatically cluster products to be considered same based on the scores predicted by the Deepmatcher.

## 3.1. Deepmatcher

Here we describe the structure of the Deepmatcher network, a pre-existing deep learning solution that we used to match products between our two shops. The network is originally presented and described in-depth in [5], here we summarize its design.

### 3.1.1. Input

Before describing the design of the Deepmatcher network we have to specify the structure of its input. Each input point corresponds to a pair of entities ($e_1$, $e_2$), which follow the same attributes schema ($A_1$, ..., $A_N$). Each attribute $A_j$ is assumed to be textual and corresponds to a sequence of words $\mathbf{w}_{e,j}$ for each entity $e$. The length of these sequences is allowed to be different across different entities. To put it better, each input point corresponds to a vector of $N$ entries (one for each attribute $A_j \in \{A_1, ..., A_N\}$) where each entry $j$ corresponds to a pair of word sequences $\mathbf{w}_{e_1,j}$ and $\mathbf{w}_{e_2,j}$.

### 3.1.2. Network modules

The Deepmatcher network is composed by three modules, each one dealing with a specific task. Figure 3.1 shows the high-level architecture of the network that will be described in detail in this section.

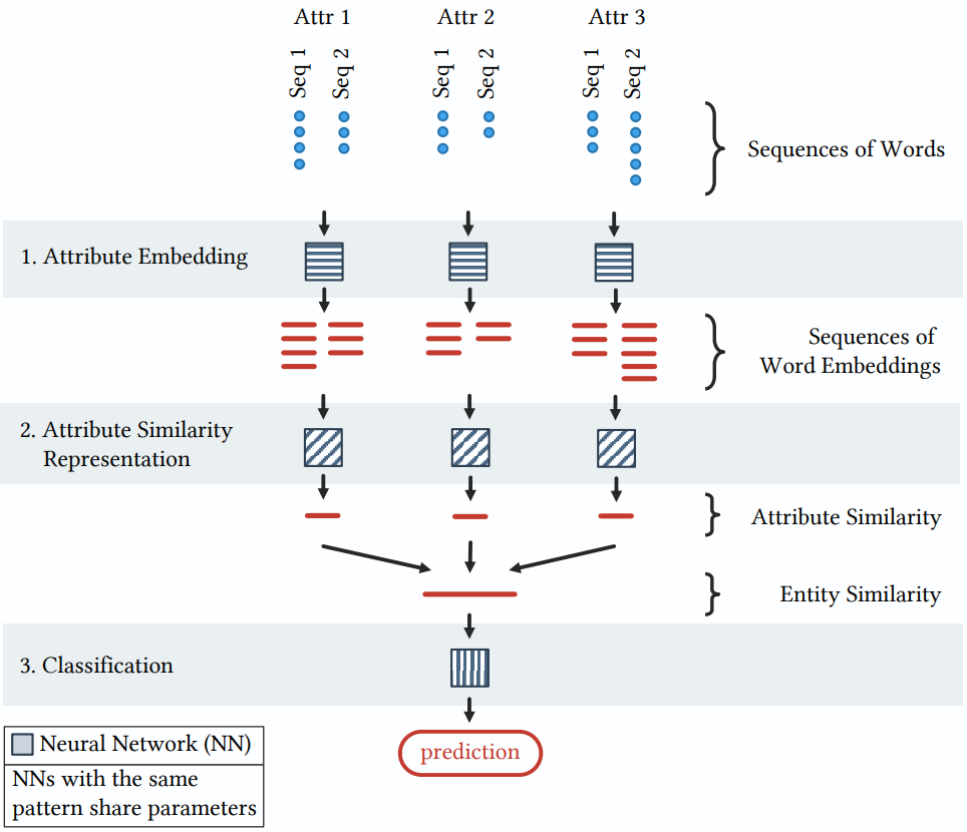


Figure 3.1: Deepmatcher architecture (Mudgal 2018 [5]).

## Attribute embedding module

This module takes sequences of words $\mathbf{w}_{e_1,j}$, $\mathbf{w}_{e_2,j}$ for each attribute $A_j \in \{A_1, ..., A_N\}$ and create two sequences of word embedding vectors $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$. Each element of these vectors correspond to a $d$-dimensional embedding of the corresponding word. If for entity $e_1$ — and same holds for $e_2$ — word sequence $\mathbf{w}_{e_1,j}$ contains $m$ elements then we have $\mathbf{u}_{e_1,j} \in \mathbb{R}^{d \times m}$. The final output of this module is denoted as $\{(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})\}_{j=1}^{N}$ and it's a pair of embeddings $\mathbf{u}_{e_1,j}$, $\mathbf{u}_{e_2,j}$ for the values of each attribute $A_j$ for entities $e_1$ and $e_2$.

## Attribute similarity representation module

This module aims at learning a representation that captures the similarity of two entities given as input. The input of the module are the attribute value embeddings $\{(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})\}_{j=1}^{N}$ while the output is the encoding of the input into a representation that captures the similarity between the attributes of $e_1$ and $e_2$. For each pair of attribute embeddings $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$ — and so for each attribute $A_j$ — the module performs two operations:

1. **Attribute summarization**. In this step the inputs are the two sequences $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$. An operation $H$ is applied to summarize the information in the input sequences. More precisely, let sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ contain $m$ and $k$ elements respectively. An $h$ dimensional summarization model $H$ takes as input sequences $\mathbf{u}_{e_1,j} \in \mathbb{R}^{d \times m}$ and $\mathbf{u}_{e_2,j} \in \mathbb{R}^{d \times k}$ and outputs two summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$. The attribute summarization aggregates information across all tokens in the attribute value embedding. The summarization operation may consider the pair of sequences $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$ jointly.
2. **Attribute comparison**. This step takes as input the summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$ and applies a comparison function $D$ over those summaries to obtain the final representation of the similarity between the attribute values for $e_1$ and $e_2$. We denote that similarity representation by $s_j \in \mathbb{R}^l$ with $s_j = D(s_{e_1,j}, s_{e2,j})$.

The output of the similarity representation module is a set of similarity representation vectors $\{s_1, ..., s_N\}$, one for each attribute $A_1, ..., A_N$.

#### Classifier module

This module takes as input the similarity representations $\{s_1, ..., s_N\}$ and uses those as features for a classifier $M$ that produces a score for the input entities $e_1$ and $e_2$ to refer to the same real-world entity.

### 3.1.3. Design choices

In this section we describe how the modules (and sub-modules) presented above have been implemented during the practical application of the network.

#### Attribute embedding design

For this module we use a fastText [35] word embedding pre-trained on Common Crawl and Wikipedia [36]. It is an embedding that uses character level information to generate the vector representation of words; it learns representations for character $n$-grams and represent words as the sum of their $n$-gram vectors. This differs from other famous embeddings such as word2vec [37] and GloVe [38] that are instead word-based: they represent each word of the vocabulary by a distinct vector ignoring the internal structure of words. Procedurally, word-level embeddings use a lookup table to convert a word into an embedding. Character-level embeddings are more suited for our purpose for two reasons:

- They perform better for morphologically rich languages which include Italian. In these languages each word may have a lot of different inflected forms, most of them used very infrequently and thus occuring rarely (or not at all) in training corpuses, making it difficult for word-based embeddings to learn good word representations.
- They are more robust to out-of-vocabulary words that may occur due to misspellings as they leverage possible substrings of the word to approximate its embedding. This is an advantage since our dataset presents long product descriptions in which typographical errors are widespread.

### Attribute summarization design

We have four different types of attribute summarization. They are the ones presented in the Deepmatcher paper [5] but are also described here for completeness:

- **Smooth Inverse Frequency (SIF)**. This model uses an aggregate function to summarize the attribute values. Specifically the function is a weighted average computed independently for the two input sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$. Given a word $w$ the corresponding embedding is weighted by a weight $f(w) = \alpha/(\alpha + P(w))$ (smooth inverse frequency [39]) where $\alpha$ is a hyperparameter that in our case is set to 0.001 and $P(w)$ the unigram probability of word $w$ computed over all values of this attribute over the entire training dataset. The biggest advantage of this type of summarization is training efficiency since there is no learning involved but its performance relies mostly on the expressive power of the attribute embedding and the classifier used. This model was shown to perform comparably to complex — and much harder to train — models and also in our application it gave results comparable to other more sophisticated summarizers.
- **Recurrent Neural Network (RNN)**. This model aims to learn complex interactions across the tokens in the input sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$. It does not consider yet the two sequences jointly but its a step forward to the SIF summarizer since it'is sequence-aware. The model uses a bidirectional GRU-based RNN [40]. It consists of two RNNs: the forward RNN that processes the input sequence $\mathbf{u}$ in its regular order and produces hidden states $\{f_1, ..., f_t\}$ and the backwards RNN that processes the input sequence in reverse order producing hidden states $\{b_t, ..., b_1\}$. The final attribute summarization representation corresponds to the concatenation of the last two outputs of the bidirectional RNN, i.e., to the concatenation of $f_t$ and $b_1$.
- **Attention**. Here the model takes as input both sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ and uses one as context when summarizing the other. It exploits an attention mechanism

that first learn to compute a soft alignment between the two sequences of words and then perform a word by word comparison. Suppose we want to compute the summarized representation of two embedding sequences $e_1$ and $e_2$. To compute the summarized representation for $u_1$ we give $u_1$ as primary input and $u_2$ as context to the attention model. The process has three steps:

1. *Soft alignment.* Let $K$ and $M$ be the total number of elements in each sequence, respectively $\mathbf{u}_1$ and $\mathbf{u}_2$. For each element $u_1[k]$ in the primary input $\mathbf{u}_1$ a soft-aligned encoding of $u_1[k]$ — denoted by $b_1[k]$ — is computed using all elements in the context sequence $\mathbf{u}_2$. This step exploits a soft alignment matrix $\mathcal{W}$ in which each entry is a weight for a pair of elements $(u_1[k], u_2[m])$. For each pair of entries $u_1[k] \in \mathbf{u}_1$ and $u_2[m] \in \mathbf{u}_2$ a weight $w_{k,m}$ is computed as the soft-max of the dot product over the hidden representations of $u_1[k]$ and $u_2[m]$ obtained by a two layer Highway network [41]. The encoding $b_1[k]$ for each $u_1[k] \in \mathbf{u}_1$ is computed by taking a weighted average over all elements $u_2[m] \in \mathbf{u}_2$ with the weights being the entries of the $k$-th row of $\mathcal{W}$.

2. *Comparison.* Each embedding $u_1[k] \in \mathbf{u}_1$ is compared with its soft-aligned encoding $b_1[k]$. To do so, the embeddings and their soft-aligned encodings are concatenated and then processed through a two layer Highway network with ReLU non-linearities. The comparison representation for each $u_1[k] \in \mathbf{u}_1$ is denoted ad $x_1[k]$.

3. *Aggregation.* Finally, all the comparison representations $x_1[k]$ of all the elements of $\mathbf{u}_1$ are summed and the output is normalized by dividing with $\sqrt{|\mathbf{u}_1|}$. As stated by the authors, this ensures that the variance of the final attribute summarization does not change as a function of the number of words in the attribute.

These steps are obviously replicated with $\mathbf{u}_2$ as primary input and $\mathbf{u}_1$ as context to compute the summarized representation for $\mathbf{u}_2$.

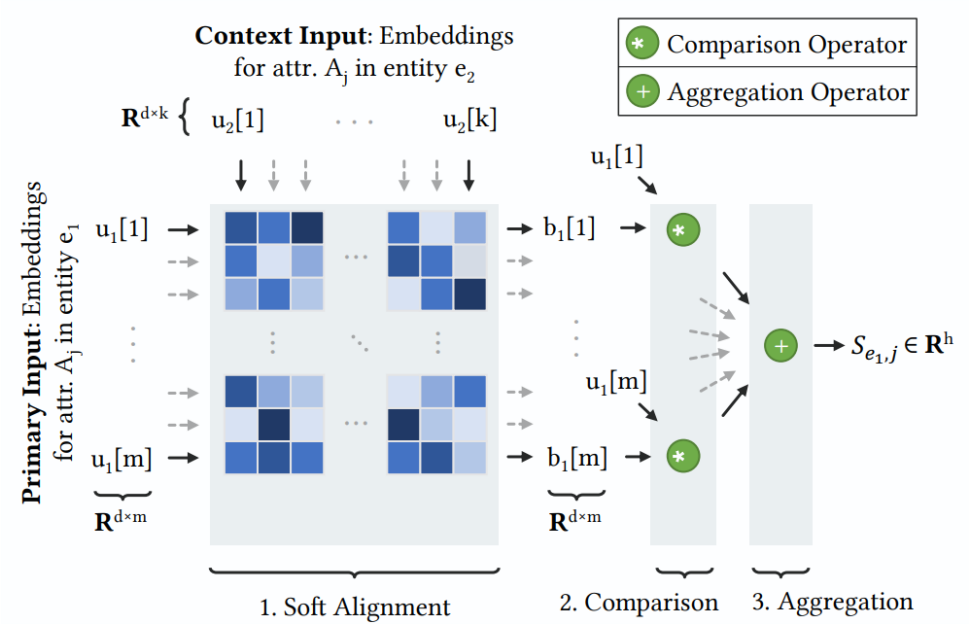


Figure 3.2: Attention attribute summarization module (Mudgal 2018 [5]).

- **Hybrid**. This attribute summarization method is a combination of the sequence-aware (RNN) and sequence alignment (Attention) methods. The mechanism is similar to those of the Attention model but it also utilizes sequence-aware encodings of $\mathbf{u}_1$ and $\mathbf{u}_2$ obtained by a bidirectional RNN. As before we have three steps to compute the summarized representation for $\mathbf{u}_1$:

  1. *Soft alignment.* First a soft alignment matrix $\mathcal{W}$ is generated in the same way as in the Attention model. Then a soft-aligned encoding $b_1[k]$ for each element $u_1[k] \in \mathbf{u}_1$ is constructed by taking a weighted average over an encoding of $\mathbf{u}_2$; this is where this model differs from Attention. The encoding of $\mathbf{u}_2$ is obtained by processing $\mathbf{u}_2$ with a bidirectional RNN and concatenating all the hidden states of the network. The resulting vector of soft-aligned encodings is denoted as $\mathbf{b}_1$.

  2. *Comparison.* In this model the input $\mathbf{u}_1$ is not compared directly with $\mathbf{b}_1$. First an encoding of $\mathbf{u}_1$ — denoted $\mathbf{u}'_1$ — is obtained by processing $\mathbf{u}_1$ in the same RNN used for the Soft alignment step and concatenating the hidden states. Then $\mathbf{u}'_1$ and $\mathbf{b}_1$ are compared in the same way as in the Attention model: they are concatenated and then processed through a two layer Highway network with ReLU non-linearities. The comparison representation for each $u_1[k] \in \mathbf{u}_1$ is denoted ad $x_1[k]$.

  3. *Aggregation.* In this model the elements of the comparison vector $\mathbf{x}_1$ produced

by the previous step are aggregated using a weighted average. First an encoding of $\mathbf{u}_2$ — denoted $g_2$ — is obtained by using a bidirectional RNN — different from the one used in the steps above — and taking its last hidden state. The weight for each element $x_1[k]$ is then obtained by concatenating $x_1[k]$ with $g_2$ and processing it through a two layer ReLU Highway network followed by a soft-max layer. This can be seen as an attention mechanism that identifies the importance of each element $x_1[k]$ given $\mathbf{u}_2$ as context.

As in the Attention model these steps are replicated with $\mathbf{u}_2$ as primary input and $\mathbf{u}_1$ as context to compute the summarized representation for $\mathbf{u}_2$.

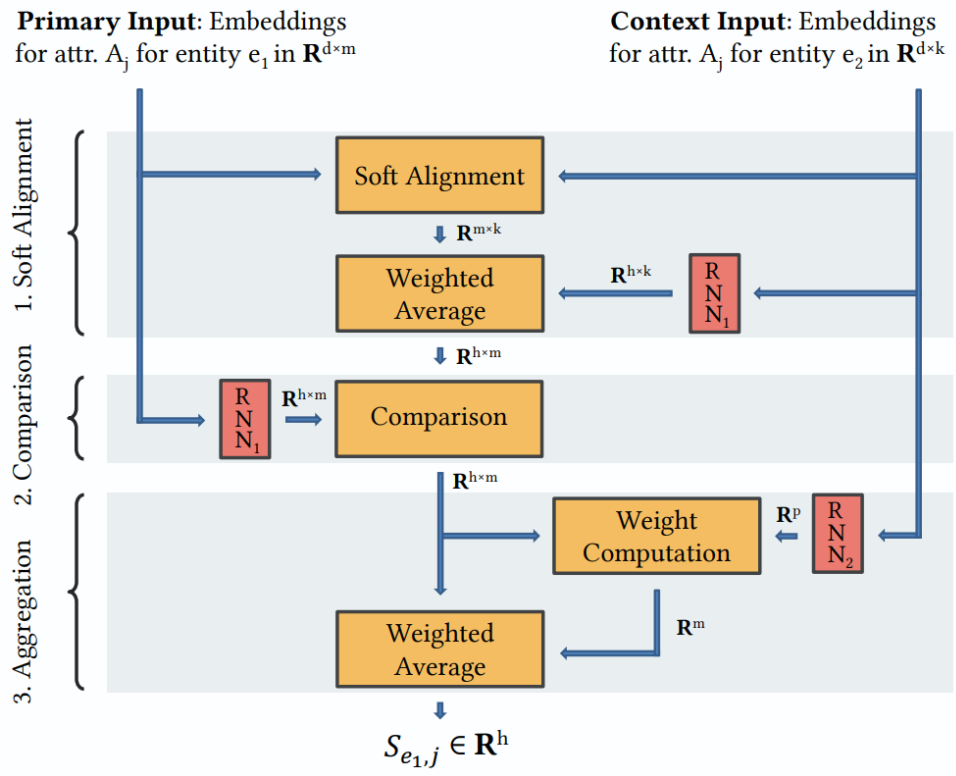


Figure 3.3: Hybrid attribute summarization module (Mudgal 2018 [5]).

## Attribute comparison design

As already stated in the description of the modules, the attribute comparison is a function to be applied to the pairs of summarized attributes to obtain the final similarity representation for each attribute. The comparison operations we use are:

- **Concatenation**. Concatenate the two input vectors.
- **Absolute difference**. Take the absolute difference between two input vectors.

- **Multiplication**. Take the element-wise multiplication between the two input vectors.
- **Concatenation with absolute difference**. Concatenate the two input vectors and then concatenate the resulting vector with the absolute difference between the two input vectors.

### Classifier design

The classifier we use is a Highway network with ReLU non-linearities. The only hyper-parameter here is the number of layers that varies between 2 and 3. The structure of the classifier is very simple because the key for the entity matching is not in this module but in a good similarity representation. We need to have a similarity representation that really captures the similarities between the attributes of the two entities so that they may be exploited by the classifier.

## 3.2. Training procedure

### 3.2.1. Data extraction

From the raw data provided by Coveo we extracted a subset of product entries to be used for our purpose. The first activity we did was extracting the list of the brands in common between the two shops and filtering out all the products not belonging to those brands. Obviously two product entries belonging to different brands cannot refer to the same real-world product. The product entries extracted from the raw data have the following features:

- *Shop*: Indicates whether the product entry belongs to Shop A or B.
- *Name*: Name of the product as shown on the e-commerce website.
- *Description*: Description of the product limited at 128 words without stop words.
- *Categories*: Categories under which the product have been listed in the periods in which it appears on the e-commerce catalog. Since a lot of categories are too specific or are present because a product is temporarily included in a promotion (e.g. if a certain product is on sale during a week it may be placed under the category "Big Sales") we analyzed by hand all the categories to keep only the most useful for describing the products.
- *Price*: Average price of the product in the periods in which it appears on the e-

commerce catalog.

At this point we have a dataset containing all the product entries — for the common brands — of the two shops.

Recalling from the description of the input of Deepmatcher, each input point corresponds to a pair of entities $(e_1, e_2)$, which follow the same attributes schema $(A_1, ..., A_N)$, so we take the Cartesian product between the entries of Shop A and the ones of Shop B for each brand ($\sim$ 300000 pairs). In the final dataset each entry represents a pair of products and the feature set is in the form $\{left\_Name, left\_Description, ..., right\_Name, right\_Description, ...\}$ where the $left$ (or $l$ for simplicity in Table 3.1) prefix means that the feature refers to the entity of Shop A while the $right$ prefix refers to Shop B.

| **l_Name** | **l_Description** | **l_Categories** | **l_Price** | ... | **Label** |
|---|---|---|---|---|---|
| Air Max 90 | Le Nike Air Max 90 sono... | scarpe, casual | 90.00 | ... | 1 |
| scarpe air max bambino | Le Nike Air Max 90 sono sneakers... | scarpe, abbigliamento | 60.00 | ... | 0 |
| ... | ... | ... | ... | ... | ... |

Table 3.1: Deepmatcher sample dataset.

### 3.2.2. Filtering and labeling

To train the Deepmatcher models we first need to generate a training set by labeling a random subset of product pairs by hand. The probability of pulling at random a pair with two matched products (positive label) is extremely low since most of the pairs refers to different products, so we filtered the dataset in order to reduce the number of negative samples. The filter is based on a similarity ratio between product names; the procedure to calculate the similarity, relying on the Python *difflib* package, is the following:

1. Filter the names keeping only letters and numbers, trim and put them in lowercase.
2. Reorder the words composing the names in alphabetical order.
3. Find the lengths of the longest disjoint substrings that match between the two names and sum them.
4. Calling $M$ the sum calculated in the step above and $T$ the sum of the lengths of the two names, calculate the similarity as $2 \cdot \frac{M}{T}$.

A pair of products is kept if the similarity between their names is above 0.5. Then, with the new filtered dataset we randomly extracted a subset of samples and labelled them.

### 3.2.3. Training and models selection

In this section we explain our training process in which we trained Deepmatcher models and selected the best combinations of the module designs described in Section 3.1.3. We present the evaluation metric and optimization algorithm used and describe the three phases in which the training is organized.

#### Evaluation metric

The performances of the models are measured with the *precision metric*. It is calculated as the ratio between the number of positive samples (matches) correctly classified to the total number of samples classified as positive (either correctly or incorrectly). The precision measures the model's accuracy in classifying a sample as positive, the formula is:

$$Precision = \frac{TP}{TP + FP}, \tag{3.1}$$

where $TP$ is the number of true positives, i.e. the number of positive samples correctly classified and $FP$ is the number of false positives, i.e. the number of negative samples wrongly classified as positives. Since our goal is finding the products that really matches between the two shops we do not care about missing some positive samples but we do not want to misclassify a negative sample as positive.

#### Optimization algorithm

The optimization algorithm used is Adam. It is a popular algorithm in the field of deep learning and it's an extension to stochastic gradient descent, that uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. As stated in its original paper, it is computationally efficient, has little memory requirements and is well suited for problems that are large in terms of data and/or parameters [42].

#### Early stopping

In each phase of our training plan we reserved a set of samples for validation. The validation set is used to evaluate the performances of the models on unseen samples not used for training. By doing this we could have an unbiased estimate of models

performances and we could also perform early stopping.

Early stopping is a regularization technique aimed at reducing overfitting; it consists in stopping the training of the network if the performance on the validation set does not improve for an arbitrarily large number of training epochs. When training a large network, there will be a point during training when the model will stop generalizing and start learning the statistical noise in the training dataset thus losing generalization capabilities. The goal of early stopping is to prevent the network from being trained for too long after this point.

## Hyperparameters

In the following table we provide a brief description of the hyperparameters we tested during the training phase. Most of them are the ones already presented in depth in Section 3.1.3 but a brief description is reported here for completeness.

| **Parameter** | **Description** | **Values** |
|---|---|---|
| Batch size | Number of training samples utilized in one training iteration (epoch). | 16, 32 |
| Attribute summarizer | Network module that takes in two word embedding sequences and summarizes the information in them to produce two summary vectors as output. | SIF, Attention, Hybrid |
| Attribute comparator | Network module that takes as input the two summary vectors and applies a comparison function over those summaries to obtain the final similarity representation of the two attribute values. | Concatenation, Absolute difference, Multiplication, Concatenation with absolute difference |
| Attribute condense factor | The factor by which to condense each attribute similarity vector. The purpose of condensing is to reduce the number of parameters in the classifier module. The operation is performed through a fully connected layer without non-linearities before the classifier module. E.g. If the condense factor is set to 3 and the attribute similarity vector size is 300, then each attribute similarity vector is transformed to a 100 dimensional vector. | 1, 2, 3 |

| | | |
|---|---|---|
| Classifier number of layers | Number of layers in the Classifier Highway network. | 2, 3 |
| Hidden size | Hidden size of the networks used in the Attribute summarizer and Classifier. | 256, 512 |

Table 3.2: Deepmatcher training hyperparameters.

## Training phases

We organized the training and models evaluation in three phases:

- **Baseline**. In the first phase we trained a large number of models to estimate their baseline performances. We obtained 96 different models by combining the parameters reported in the previous section. The models are trained on a training set containing 3510 samples and validated using a set of 877 samples.
- **Cross-validation**. In the second batch of trainings we selected the models that obtained a precision score above 0.7 (39 models) in the baseline evaluation and re-trained them with a 5 fold cross-validation in order to have a more accurate estimation of their performance. The dataset is the same as the step before, containing 4387 samples. The procedure is the following:
  1. Shuffle the dataset randomly.
  2. Split the dataset into 5 folds.
  3. For each model and for each fold:
     (a) Train the model on the samples that do not belong the the fold.
     (b) Evaluate the model on the samples that belong to the fold.
  4. Calculate the final precision of each model as its average over the 5 folds.

  By using cross-validation we have the opportunity to train each model on multiple train-test splits without the need of labelling other samples. This is a very computational expensive procedure but it gives us a better indication of how well the models will perform on unseen data (i.e. the rest of products to be matched).
- **Final training**. At this point we selected the top 10 models based on their cross-validation score ($0.75 \leq$ precision $\leq 0.85$) and trained them using all the samples (4387) used in the steps before.

### 3.2.4. Ensemble

After we trained the top models, we built an ensemble using them. It is a weighted voting ensemble [43] based on cross-validated performances. It does not operate on the discrete predictions but on the continuous scores that are the output of the Deepmatcher network and calculates the final score as the weighted average of the scores of each model. The weight for each model is its precision score obtained through cross-validation in the second phase of training presented in the previous section. The formula for calculating the average is the following:

$$d_{i,j} = \frac{\sum_{m \in M} p_m \cdot s_{m,i,j}}{\sum_{m \in M} p_m}, \tag{3.2}$$

where $M$ is the set of models used for the ensemble, $p_m$ is the cross-validation precision of model $m$ and $s_{m,i,j}$ is the predicted score of model $m$ on pair $(e_i, e_j)$

## 3.3. Clustering

Using the ensemble presented in the previous section we predicted a matching score for all the pairs of products in the dataset and grouped together products to be considered same using a graph based approach.

First of all we calculated an equality score for each pair of products by taking a weighted average between the score of the Deepmatcher and the similarity between the two product names calculated as in Section 3.2.2; we weight the Deepmatcher score to be two times more important than the names similarity. This is done in order to mitigate possible prediction errors of the network. Given two products $p_i$ and $p_j$, their similarity is:

$$s_{i,j} = \frac{2d_{i,j} + n_{i,j}}{3}, \tag{3.3}$$

where $d_{i,j}$ is the Deepmatcher equality score between the two products and $n_{i,j}$ their names similarity. We built a graph considering each product as a node and connecting each pair of products for which we have an equality score with an edge whose weight is the score. The graph is bipartite since we do not have an equality score for two products belonging to the same shop.

**Definition 1** (Bipartite graph). *A bipartite graph, also called a bigraph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent (i.e. connected by an edge) [44].*

To connect products within the same shop we added an edge between the pairs of products that are connected to at least one common product belonging to the other shop. The weight of the edge is the average between the weights connecting the two nodes with common nodes. Calling $N(p)$ the set of neighbours for a generic product $p$, the weight of the edge connecting two products $p_i$ and $p_j$ belonging to the same shop is:

$$s_{i,j} = \frac{\sum_{p_k \in N(p_i) \cap N(p_j)} (s_{i,k} + s_{j,k})}{2\,|N(p_i) \cap N(p_j)|}. \tag{3.4}$$

Figure 3.4 shows an example of this process. Note that the graph is still not complete since the products that we considered a priori different are not connected, it is even disconnected.

**Definition 2** (Complete graph)**.** *A complete graph is a graph in which each pair of graph vertices is connected by an edge [45].*

**Definition 3** (Disconnected graph)**.** *A graph $G$ is said to be disconnected if it is not connected, i.e., if there exist two nodes in $G$ such that no path in $G$ has those nodes as endpoints [46].*

A proof that the graph is disconnected is the fact that products belonging two different brands are considered a priori different thus never connected; at least each brand generates an isolated subgraph.

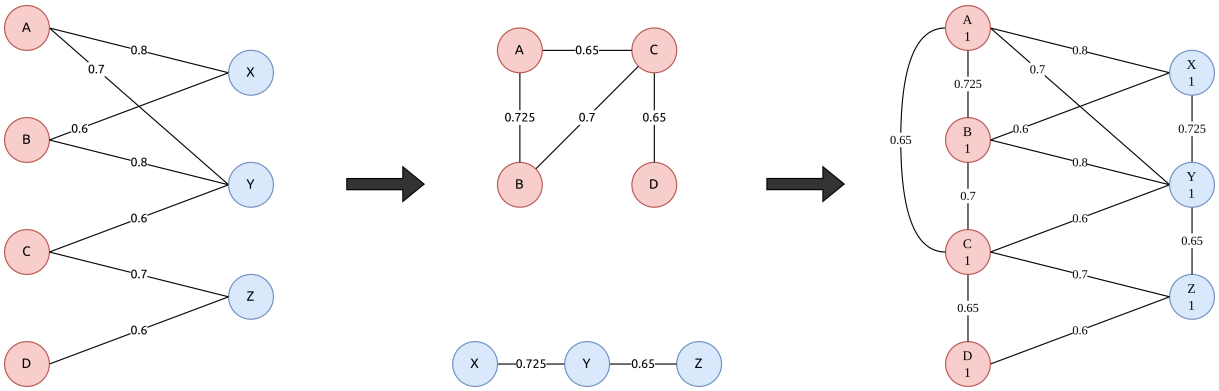


Figure 3.4: Example showing the process used to add edges between nodes belonging to the same shop.

For each subgraph we merged nodes — products to be considered equal — with the following prodcedure:

1. Initialize each node with a value of 1, the value of each node is the equality score for the products merged into that node. Initially it is set to 1 since each node represent a single product.

2. For each pair of connected nodes calculate the score to be assigned to the node generated by their possible merge. Given two nodes $p_i$ and $p_j$ whose scores are respectively $s_i$ and $s_j$, their merged score is:

$$s = s_{i,j} \cdot \frac{s_i + s_j}{2}, \tag{3.5}$$

   where $s_{i,j}$ is the weight of the edge between $p_i$ and $p_j$.

3. If there is not any merge with score greater than 0.5 stop, otherwise merge the two nodes with the higher score and repeat from step 2. The edges connected to the two nodes are also merged and the scores recalculated; considering a generic node $p_k$ connected to at least one of the two merged nodes $p_i$ and $p_j$, the score of the new edge connecting the aggregated node to $p_k$ is:

$$s = \begin{cases} \frac{s_{i,k}}{2}, & \text{if } p_j \notin N(p_k) \text{ ($p_j$ is not connected to $p_k$)}, \\ \frac{s_{j,k}}{2}, & \text{if } p_i \notin N(p_k) \text{ ($p_i$ is not connected to $p_k$)}, \\ \frac{s_{i,k}+s_{j,k}}{2}, & \text{otherwise.} \end{cases} \tag{3.6}$$

From here on we will refer to the a group of products as *product group* or *real-world product* while we will refer to a single product inside a group as just *product* or *SKU* (i.e. *Stock Keeping Unit*, the unique identifier of a product inside a shop).

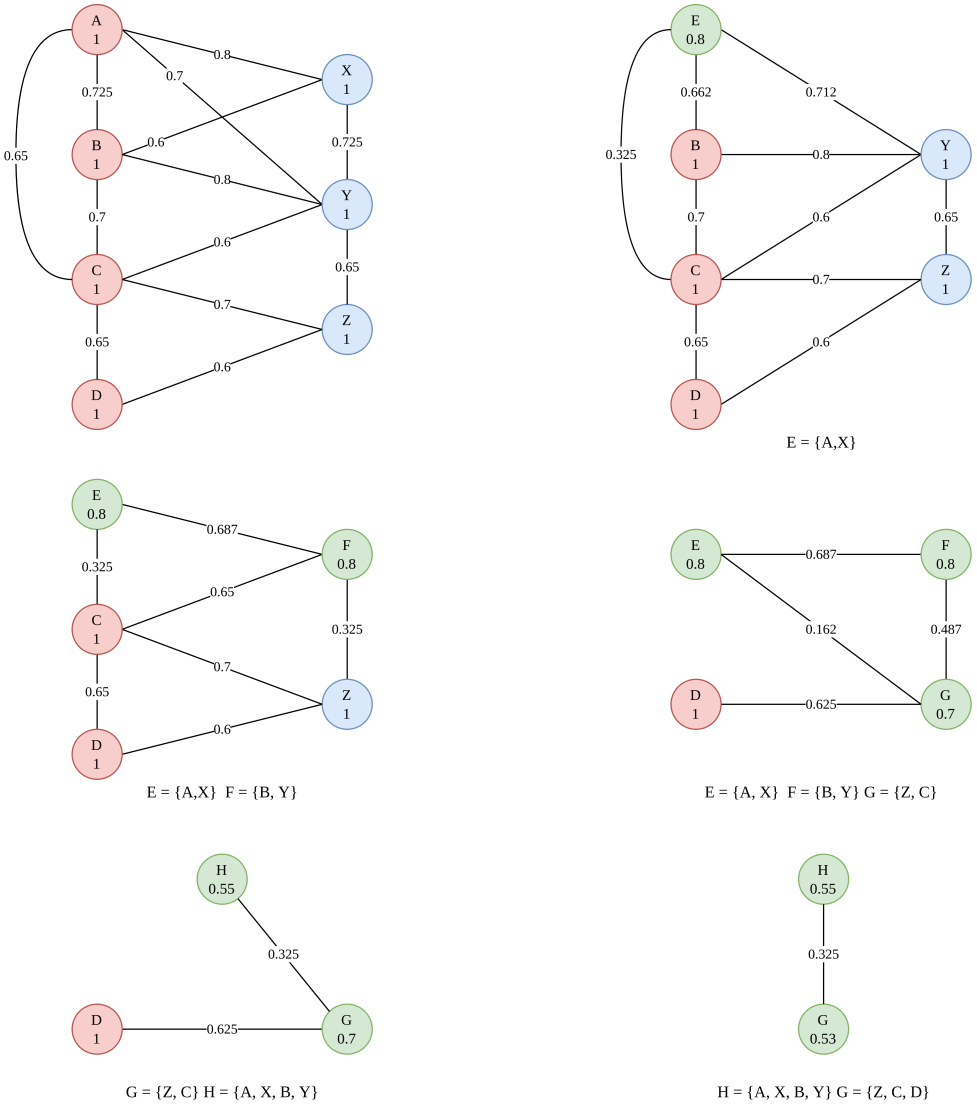


Figure 3.5: Example showing the process used to merge nodes representing the same products.

# 4 | Double Machine Learning

In this chapter, we present a Machine Learning based approach to calculate the price elasticitiy of the products keeping into account the possible confounders that we can extract from our data. The approach is based on the procedure called Double Machine Learning (DML) originally presented by Chernozukov et al. [6].

## 4.1. Theoretical background

Double Machine Learning (DML) is a method for estimating treatment effects keeping into account potential confounders that are observed. We define as confounder every factor that simultaneously had a direct effect on the treatment decision in the collected data and the observed outcome. DML exploits machine learning techniques because they are well suited in the case in which confounders are too many for classical statistical approaches to be applicable or when their effect on the treatment and outcome cannot be properly modeled by parametric functions. In addition, DML aims at delivering point estimators that have a $\sqrt{N}$ rate of convergence for $N$ observations (root n-consistency).

**Definition 4** (Consistent estimator)**.** *An estimator $T_n$ of parameter $\theta$ is said to be consistent, if it converges in probability to the true value of the parameter:*

$$\lim_{n\to\infty} Pr(|T_n - \theta| > \epsilon) = 0. \tag{4.1}$$

**Definition 5** ($n^\delta$-consistent estimator)**.** *An estimator is said to be $n^\delta$-consistent if its variance is $O(1/n^{2\delta})$.*

### 4.1.1. Partially linear regression model

The model we will use to explain DML, that is also the one we used to model our scenario, is a partially linear regression model. In a partially linear regression model we assume that the data are generated by the following model:

$$Y = D\theta_0 + g_0(X) + \zeta, \qquad \mathbb{E}[\zeta|D, X] = 0,$$
$$D = m_0(X) + V, \qquad \mathbb{E}[V|X] = 0,$$

where $Y$ is the outcome variable and $D$ is the treated variable. The vector $X = (X_1, \ldots, X_p)$ is the vector of confounding variables, and $\zeta$ and $V$ are stochastic errors. More intuitively, the relation between the variables can be represented through a directed acyclic graph.

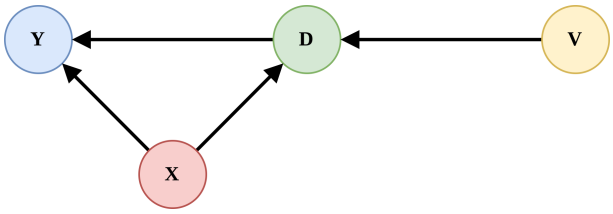


Figure 4.1: Partially linear model DAG.

This set-up allows both $Y$ and $D$ to be non-linear with respect to the set of confounders $X$. However, this partially linear model assumes that the effect of $D$ on $Y$ is additive and linear.

### 4.1.2. Naive estimator

A naive approach to estimate $\theta_0$ using ML methods would be constructing a ML estimator $\hat{Y} = D\hat{\theta}_0 + \hat{g}_0(X)$ for learning the regression function $D\theta_0 + g_0(X)$. This can be done, for example, by splitting the samples into two parts: a main part of size $n$ and an auxiliary one of size $N - n$. The auxiliary set can be used to estimate $\hat{g}_0(X)$ from $Y$ and $X$ and then $\hat{g}_0(X)$ can be used to estimate $\hat{\theta}_0$ with a standard regression (note that $Y$ is linear on $\theta_0$ and $\hat{g}_0(X)$) on the main set of samples.

The estimator $\hat{\theta}_0$ however will generally have a slower than $1/\sqrt{n}$ rate of convergence:

$$|\sqrt{n}(\hat{\theta}_0 - \theta_0)| \xrightarrow{p} \infty. \tag{4.2}$$

This happens because the estimator is biased. In Chernozhukov (2018) [6], the authors provide an example of this bias by running a simulation experiment where they learned $g_0$ using a random forest. They also underline that $g_0$ has been set to a smooth function with few parameters, that should in principle be well approximated by random forests. The resulting distribution of $\hat{\theta}_0 - \theta_0$ obtained by the simulations is reported in Figure 4.2. The estimator is biased, shifted to the right with respect to the true value $\theta_0$, and its distribution is substantively different from a normal approximation (shown by the

red curve). The main reason of this phenomena is that $g_0(X) \neq \mathbb{E}[Y|X]$. Thus, it is not possible in general to get a good estimate of $g_0(X)$ by regressing $Y$ on $Z$, and this obviously affects the estimate of $\theta_0$.

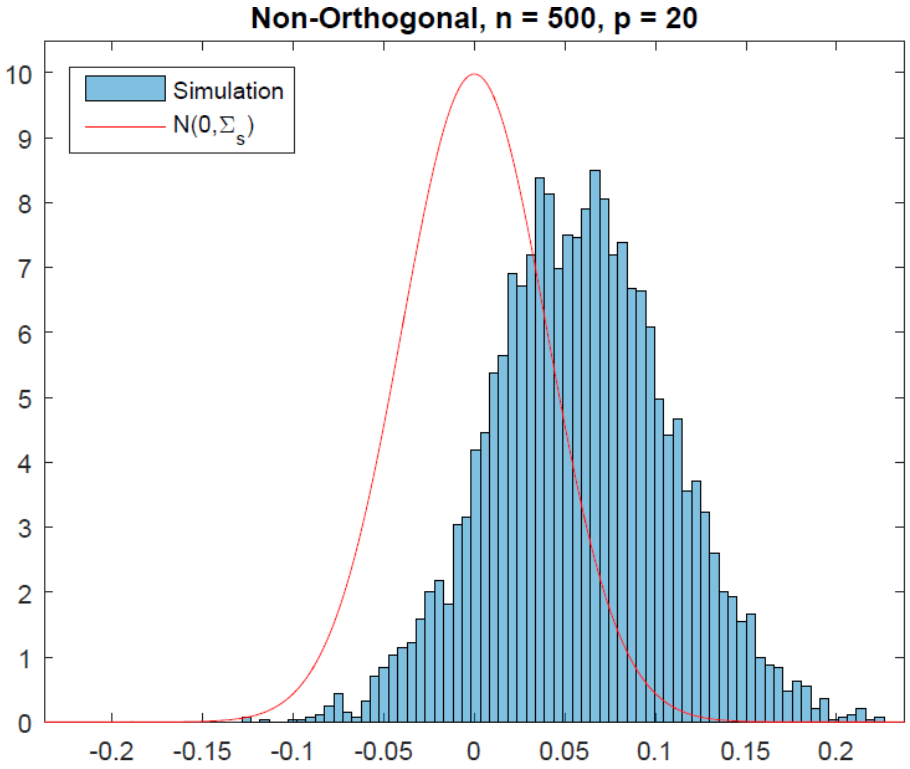


Figure 4.2: Distribution of $\hat{\theta}_0 - \theta_0$ using a naive estimator (Chernozhukov 2018 [6]).

The bias introduced by this type of naive estimation — and in general when doing causal inference using machine learning methods — can be originated essentially by two sources: regularization and overfitting. Double Machine Learning tries to correct regularization bias by means of orthogonalization and overfitting bias by means of cross-fitting.

### 4.1.3. Orthogonalization

To overcome regulatization bias DML uses orthogonalization. In the case of the partially linear regression model presented above, instead of fitting a single machine learning model we fit two models:

- The first one is $\hat{m}_0$, an estimator for $m_0$ obtained by regressing $D$ on $X$. It can be seen as the conditional mean of $D$ given $X$.
- The second one is $\hat{l}_0$, the conditional mean of $Y$ given $X$.

According to the Frisch–Waugh–Lovell theorem [47, 48], the partially linear regression model can be rewritten in the following residualized form:

$$\begin{aligned} W &= V\theta_0 + \zeta, & \mathbb{E}[\zeta|D, X] &= 0, \\ W &= Y - l_0(X), & l_0(X) &= \mathbb{E}[Y|X], \\ V &= D - m_0(X), & m_0(X) &= \mathbb{E}[D|X]. \end{aligned}$$

We can obtain the orthogonalized regressors $\hat{V} = D - \hat{m}_0(X)$ and $\hat{W} = Y - \hat{l}_0(X)$ and perform a linear regression of $\hat{W}$ on $\hat{V}$ to obtain an estimate of $\theta_0$ (call it $\check{\theta}_0$). By approximately removing the effect of confounding on $D$ and $Y$ by subtracting an estimate of $g_0$ and $l_0$, $\check{\theta}_0$ removes the effect of regularization bias. This process is called partialling out or orthogonalization. In Chernozhukov (2018) [6], the authors shows the behaviour of the orthogonal, DML estimator, $\check{\theta}_0$, in the partially linear model. The experiment uses simulated data that are exactly the same as those of the naive estimation experiment; the distribution of $\check{\theta}_0 - \theta_0$ is reported in Figure 4.3.

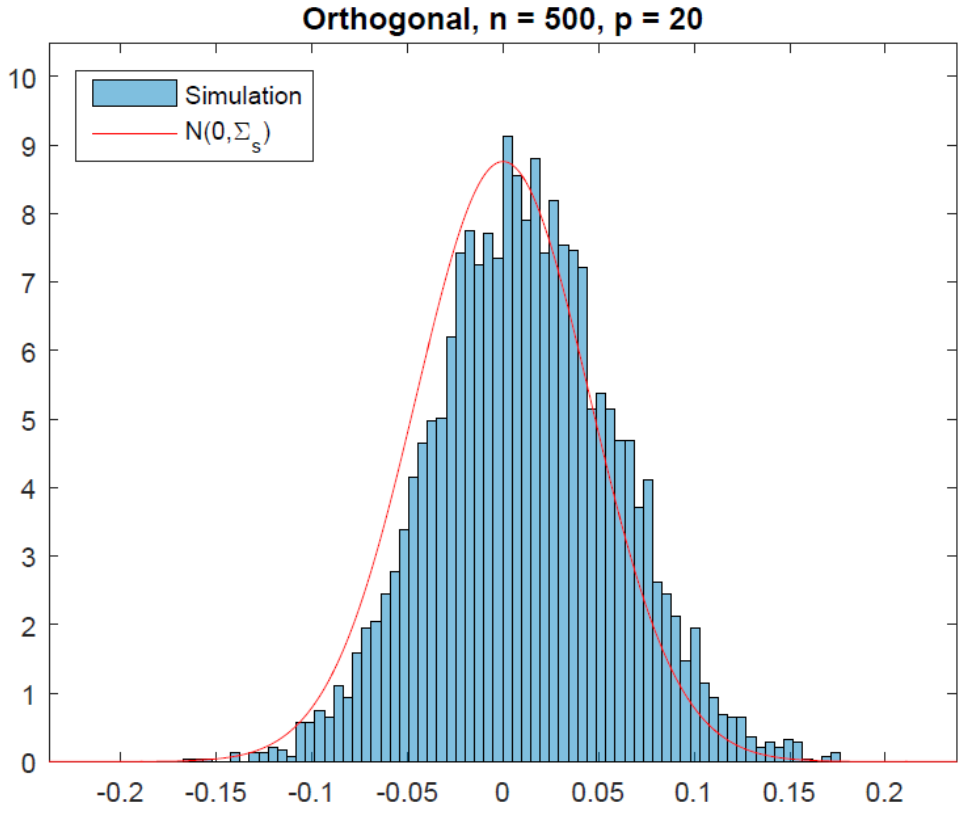


Figure 4.3: Distribution of $\check{\theta}_0 - \theta_0$ using orthogonalization (Chernozhukov 2018 [6]).

In the general case, to estimate $\theta_0$ we use a method-of-moments estimator based upon the empirical analog of the moment condition:

$$\mathbb{E}[\phi(W; \theta_0, \eta_0)] = 0, \tag{4.3}$$

where $W = (Y, D, X)$, $\eta_0$ is the nuisance parameter (e.g. $g_0$ in the partially linear regression model) and $\phi$ is the score function of our model, i.e. the gradient of the log-likelihood function with respect to the parameter vector. To eliminate the bias introduced by the usage of ML techniques in the estimation of $\eta_0$ we want the score function to be robust to changes to the parameter values.

In the case of the partially linear model, the score function is $\phi(W; \theta, g) = (Y - \theta D - g(X))D$. This score function is sensitive to biased estimation of $g$ and this is proved by the fact that the the pathwise (Gateaux) derivative operator with respect to $g$ does not vanish when evaluated at the true parameter values:

$$\partial_g \mathbb{E}[\phi(W; \theta_0, g)]|_{g=g_0} \neq 0. \tag{4.4}$$

By applying the partialling out process we obtain the score function $\phi(W; \theta, \eta) = (Y - l(X) - \theta(D - m(X)))(D - m(X))$ where $\eta = (l, m)$. For this function instead holds:

$$\partial_\eta \mathbb{E}[\phi(W; \theta_0, \eta_0)]|_{\eta=\eta_0} = 0. \tag{4.5}$$

This property is called Neyman orthogonality. Intuitively, the Neyman orthogonality means that the moment conditions used to identify $\theta_0$ are locally insensitive to the value of the nuisance parameter. This result allows to use noisy estimates of these parameters such as the ones obtained by ML techniques because using a Neyman-orthogonal score eliminates the first order biases introduced by these estimators.

### 4.1.4. Cross-fitting

A first approach to remove the overfitting bias from our estimation would be a simple sample-splitting. This technique consists in randomly splitting the dataset into two sets. On the auxiliary set we fit Machine Learning models to estimate the nuisance functions while on the main set we use these models to estimate $\theta_0$. This approach reduces the overfitting bias but has a drawback: the estimator of the parameter of interest only makes use of a portion of the samples, which can result in a loss of efficiency as we are not using all the available data.

It is for this reason that Chernozukov et al. extended this procedure to the concept of cross-fitting. Instead of using the auxiliary set exclusively for training the models of the nuisance functions and the main set for estimating $\theta_0$, we can swap the role of the two sets to obtain a second version of the estimator. The two estimators will be approximately independent and by averaging them we can regain full efficiency.

In Figure 4.4, it is reported an example of how the bias resulting from overfitting in the estimation of nuisance functions can cause the estimator $\check{\theta}_0$ to be biased. In the left panel there is the difference $\check{\theta}_0 - \theta_0$ when $\check{\theta}_0$ is estimated in the partially linear model with nuisance parameters estimated with overfitting using the full sample; in the right panel there is the difference $\check{\theta}_0 - \theta_0$ when $\check{\theta}_0$ is estimated in the partially linear model with nuisance parameters estimated using sample-splitting. It is clear that the use of sample-splitting eliminated the bias induced by overfitting.

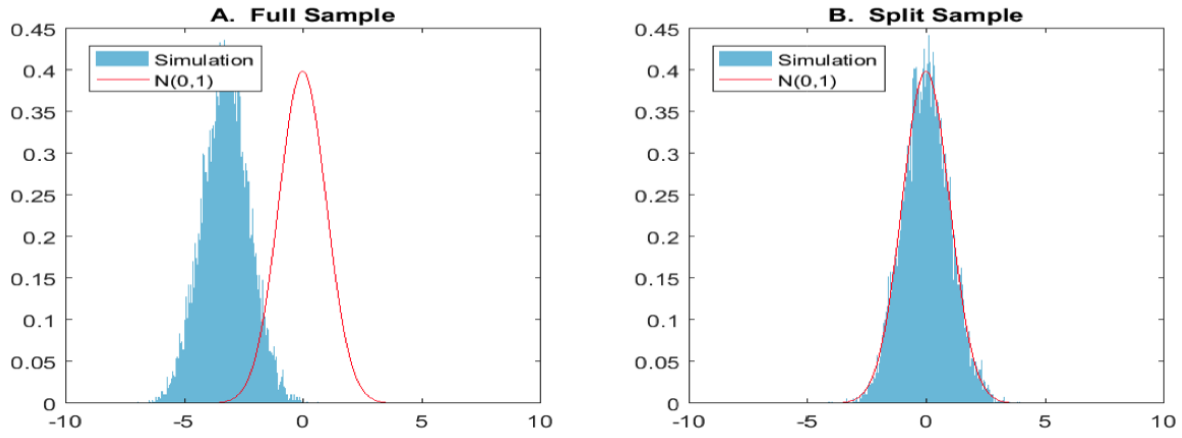


Figure 4.4: Distribution of $\check{\theta}_0 - \theta_0$ with and without cross-fitting (Chernozhukov 2018 [6]).

The procedure of cross-fitting described above can be extended to a k-Fold version. The authors presented two variants, named DML1 and DML2, stating that the second one behaves well in most of the cases and especially in the case of small samples.

## DML1

The DML1 procedure is the following:

1. Take a K-fold random partition $(I_k)_{k=1}^K$ of observation indices $[N] = \{1, \dots, N\}$ such that the size of each fold $I_k$ is $n = N/K$. Also, for each $k \in [K] = \{1, \dots, K\}$, define its complement $I_k^c := \{1, \dots, N\} \setminus I_k$.

2. For each $k \in [K]$, construct a ML estimator

$$\hat{\eta}_{0,k} = \hat{\eta}_0((W_i)_{i \in I_k^c}) \tag{4.6}$$

of the nuisance parameter $\eta_0$.

3. For each $k \in [K]$, construct the estimator $\check{\theta}_{0,k}$ as the solution of the following equation:

$$\mathbb{E}_{n,k}[\phi(W; \check{\theta}_{0,k}, \hat{\eta}_{0,k})] = 0, \tag{4.7}$$

where $\phi$ is the Neyman orthogonal score, and $\mathbb{E}_{n,k}$ is the empirical expectation over the $k^{th}$ fold of the data, $\mathbb{E}_{n,k}[\phi(W)] = \frac{1}{n} \sum_{i \in I_k} \phi(W_i)$.

4. Aggregate the estimators:

$$\tilde{\theta}_0 = \frac{1}{K} \sum_{k=1}^{K} \check{\theta}_{0,k}. \tag{4.8}$$

To simplify, this algorithm consists in estimating the $\check{\theta}_{0,k}$ for each fold by solving the moment condition and then averaging the values of the parameter obtained for each fold.

## DML2

The DML2 procedure is the following:

1. Take a K-fold random partition $(I_k)_{k=1}^K$ of observation indices $[N] = \{1, \ldots, N\}$ such that the size of each fold $I_k$ is $n = N/K$. Also, for each $k \in [K] = \{1, \ldots, K\}$, define its complement $I_k^c := \{1, \ldots, N\} \setminus I_k$.

2. For each $k \in [K]$, construct a ML estimator

$$\hat{\eta}_{0,k} = \hat{\eta}_0((W_i)_{i \in I_k^c}) \tag{4.9}$$

of the nuisance parameter $\eta_0$.

3. Construct the estimator $\tilde{\theta}_0$ as the solution to

$$\frac{1}{K} \sum_{k=1}^{K} \mathbb{E}_{n,k}[\phi(W; \tilde{\theta}_{0,k}, \hat{\eta}_{0,k})] = 0, \tag{4.10}$$

where $\phi$ is the Neyman orthogonal score, and $\mathbb{E}_{n,k}$ is the empirical expectation over the $k^{th}$ fold of the data, $\mathbb{E}_{n,k}[\phi(W)] = \frac{1}{n} \sum_{i \in I_k} \phi(W_i)$.

This algorithm differs from the previous one since it does not estimate the $\check{\theta}_{0,k}$ for each fold but instead it averages the moment conditions of the folds and solves the equation at the end to find $\tilde{\theta}_0$.

## 4.2. DML for elasticity estimation

In the case of elasticity estimation, the Double Machine Learning approach can be used to estimate the treatment effect of the price of a product (or a set of products) on its demand. In this section we will introduce the Log-Log model to represent the relation between price and demand and then extend it in order to introduce the influence of the confounding variables. This final model can be used to estimate the price elasticity of demand using DML.

### 4.2.1. Log-Log model

In our scenario we assume that the treatment (price) and treated (demand) variables are related through the generic relation:

$$Q = \alpha P^{\beta} \zeta, \tag{4.11}$$

where $\alpha$ and $\beta$ are two generic constant whose values can capture different behaviours of the relation and $\zeta$ is a multiplicative noise. Recalling from Section 1.1, to calculate the elasticity we have to take the derivative of $Q$ with respect to $P$ and multiply it by $P/Q$:

$$\frac{dQ}{dP}\frac{P}{Q} = \alpha\beta P^{(\beta-1)}\zeta\frac{P}{\alpha P^{\beta}\zeta} = \beta. \tag{4.12}$$

The equality shows that the value $\beta$ represents the price elasticity of demand. Since we want to estimate this parameter using the DML approach, we need a partially linear model in which $\beta$ is one of the regression coefficients. We can obtain it by logarithmically transforming the variables of the non-linear model:

$$\log Q = \log(\alpha P^{\beta}\zeta) = \log\alpha + \beta\log P + \log\zeta. \tag{4.13}$$

If we call $\log\alpha$ as $\alpha$ and $\log\zeta$ as $\zeta$ we can write the final Log-Log model as:

$$\log Q = \alpha + \beta\log P + \zeta. \tag{4.14}$$

Using the logarithm of the variables makes the non-linear relation representable through a linear model.

### 4.2.2. Elasticity model for DML

The Log-Log model shown in the previous section can be used as the partially linear regression model presented in Section 4.1.1 in which $Y$ and $D$ are the logarithmic transformations of demand and price, and $\theta_0$ represents the elasticity $\beta$.

We have a discrepancy between the two models: in the Log-Log model we have the additive term $\alpha$ that is a constant while in the partially linear regression model we have the nuisance function $g_0(X)$. In order to make this discrepancy irrelevant we have to introduce the additional assumption that $\mathbb{E}[X|P] = 0$, i.e. the confounding variables must be independent from the price. In this way the derivative of $g_0$ ($\alpha$) with respect to the price will be 1 and so it will not influence the derivative of $Q$ with respect to $P$ leaving the elasticity unaltered.

The model on which applying the DML is:

$$\begin{aligned} \log Q = \theta_0 \log P + g_0(X) + \zeta, \qquad & \mathbb{E}[\zeta|P, X] = 0, \mathbb{E}[X|P] = 0, \\ \log P = m_0(X) + V, \qquad & \mathbb{E}[V|X] = 0, \end{aligned}$$

where $P$ and $Q$ are price and demand, the vector $X = (X_1, \ldots, X_p)$ is the vector of confounding variables, $\zeta$ and $V$ are stochastic errors and $\theta_0$ is the elasticity of the demand with respect to the price. Transforming it in the residualized form we have:

$$\begin{aligned} W = V\theta_0 + \zeta, \qquad & \mathbb{E}[\zeta|P, X] = 0, \mathbb{E}[X|P] = 0, \\ W = \log Q - l_0(X), \qquad & l_0(X) = \mathbb{E}[\log Q|X], \\ V = \log P - m_0(X), \qquad & m_0(X) = \mathbb{E}[\log P|X]. \end{aligned}$$

With first stage estimators based on Machine Learning we can estimate $l_0(X)$ and $m_0(x)$, obtain the residuals $W$ and $V$ and then regress $W$ on $V$ to obtain the coefficient $\theta_0$ which is an estimation of the price elasticity of demand.

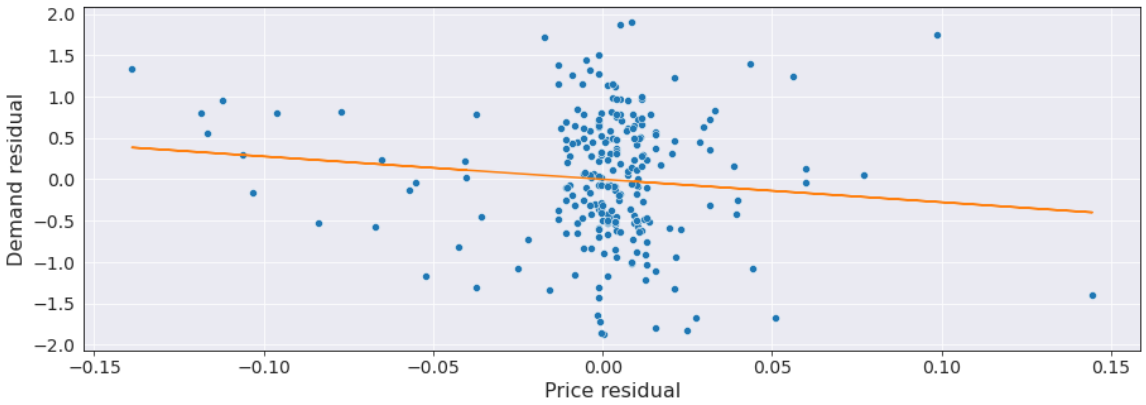


Figure 4.5: Example distribution of the residuals of price and demand ($W$ and $V$). The orange line represents the regression $W = V\theta_0 + \zeta$.

## 4.3. Method deployment

### 4.3.1. Dataset generation

As shown in Chapter 3 we have grouped together product entries to be considered the same real-world product. For every product group in which we have at least one product (SKU) for each of the two shops we extracted the historical data of the products in the group. Starting from this data we generated two different variants of the dataset to be used as input for the DML, one considers each entry as a single day while the other aggregates the data weekly (Monday to Sunday). The features of the two dataset are described in Table 4.1.

| **Feature** | **Disaggregated** | **Aggregated** |
|---|---|---|
| *Stock Keeping Unit (SKU)* | The identification code of the product to which the entry refers. | The identification code of the product to which the entry refers. |
| *Group ID* | The identification code of the group (i.e. the real-world product) to which the product belongs. | The identification code of the group (i.e. the real-world product) to which the product belongs. |
| *Shop* | Indicates whether the product belongs to Shop A or B. | Indicates whether the product belongs to Shop A or B. |
| *Price* | The price of the product that day. | The average price of the product that week. |

| | | |
|---|---|---|
| *Demand* | The quantity demanded for the product that day. | The total quantity demanded for the product that week. |
| *Promo* | Indicates whether that day the product is listed under a category considered as promotional. | Indicates whether in the majority of the days of that week the product is listed under a category considered as promotional. |
| *Month* | The month to which that day belongs to. | The month to which that week belongs to. |
| *Season* | The meteorological season of that day. | The meteorological season of that week. |
| *Covid lockdown* | Indicates whether the lockdown due to COVID-19 was in effect that day. | Indicates whether the lockdown due to COVID-19 was in effect that week. |
| *Lagged price* | Average price of the product in the previous 7 days. | Price of the product in the previous week. |
| *Lagged demand* | Average daily demand of the product in the previous 7 days. | Demand of the product in the previous week. |
| *Lagged details* | Average daily number of clicks on the product page in the previous 7 days. | Number of clicks on the product page in the previous week. |
| *Lagged conversion rate* | Conversion rate of the product in the previous 7 days. | Conversion rate of the product in the previous week. |
| *Lagged shop details* | Average daily number of clicks across the whole shop (to which the product belongs) in the previous 7 days. | Daily number of clicks across the whole shop (to which the product belongs) in the previous week. |

Table 4.1: Double Machine Learning dataset features.

### 4.3.2. First stage estimators

To estimate, using the confounding variables, the nuisance functions $l_0$ and $g_0$ of the model shown in Section 4.2.2 we tried different ML estimators.

## Lasso regressor

The lasso regressor is a linear model that is trained with an L1 regularizer as prior. It aims at maximizing the objective function:

$$\frac{1}{2 \cdot n\_samples} \|y - X\mathbf{w}\|_2^2 + \alpha \|\mathbf{w}\|_1, \tag{4.15}$$

where $X$ is the vector of independent variables (i.e. the confounding variables in our case) and $\mathbf{w}$ is the vector of coefficients assigned to each variable. The hyperparameter $\alpha$ regulates the strength of the L1 regularization.

The main limitation of the model is the assumption of linearity between the dependent variable and the independent variables that usually does not hold in real-life scenarios. The interesting aspect is that Lasso performs both variable selection and regularization because it forces certain coefficients to zero, excluding them from impacting prediction. While applying this model we have to consider that being a linear model it does not support categorical variables (e.g. season, month, ...) and so they need to be transformed into one-hot encoding.

## Random forest regressor

A random forest is an ensemble estimator that fits a number of regression trees on various sub-samples of the dataset and provides as final result the average above the predicted values of each tree. This training algorithm is based on the general technique of bagging:

1. For $n = 1, ..., N$:
   (a) Sample, with replacement, $K$ data points from the training set.
   (b) Train a regression tree $f_n$ for each subset $n$.
2. After training, for a new data point $x$, the prediction can be made by averaging the predictions from all the individual regressions:

$$f(x) = \frac{1}{N} \sum_{n=1}^{N} f_b(x). \tag{4.16}$$

In addition random forest extends this concept to features: it uses a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. If one or a few features are very strong predictors for the dependent variable, these features will be selected in many of the sub-trees. In our experiments we will use

the version of the algorithm provided by the Python package *scikit-learn* [49].

The main advantages of this technique are that it can handle linear and non-linear relationships, it balances the bias-variance trade-off well by means of bagging and also implicitly performs feature selection. On the other side the model is not easily interpretable and may be computationally expensive to train.

## Gradient boosting

Gradient boosting gives a prediction model in the form of an ensemble of weak prediction models, which in our case are decision trees. The procedure, as reported on the original paper [50], is the following:

1. Initialize the model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma). \tag{4.17}$$

2. For each step $m = 1$ to $M$:

   (a) Compute pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n, \tag{4.18}$$

   where $L$ is the loss function that in our case is the squared error.

   (b) Fit a tree learner $h_m(x)$ using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$. The tree partitions the input space into $J_m$ disjoint regions $R_{1m}, \ldots, R_{J_m m}$ and predicts a constant value in each region. The output of $h_m(x)$ for input $x$ can be written as the sum:

$$h_m(x) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(x), \tag{4.19}$$

   where $b_{jm}$ is the value predicted in the region $R_{jm}$.

   (c) Compute multiplier $\gamma_m$ by solving:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma). \tag{4.20}$$

(d) Update the model:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(x). \tag{4.21}$$

3. Output $F_M(x)$.

The gradient boosting technique is generally more accurate than any other ML algorithm. It is faster to train than random forest regressors especially on large datasets but it is still an ensemble of tree based learners and so it is computationally expensive.

In our experiments we will use two different implementations of the gradient boosting algorithm. The simplest one, that is very close to the algorithm described above, is the one provided provided by the Python package *scikit-learn* [49]. Its main limitation is the lack of support for categorical features that instead have to be transformed into one-hot encoding. To overcome this limitation we tested another implementation of the gradient boosting algorithm originally developed by Microsoft, called LightGBM [51]. It differs by most other implementation because it does not grow a tree level-wise — row by row — but instead it grows trees leaf-wise. LightGMB adopt an highly optimized alorithm which yields great advantages on both efficiency and memory consumption and also support categorical features without requiring any prior transformation.

### 4.3.3. Training procedure

The training procedure is based on the two cross-fitting approaches that we presented theoretically in Section 4.1.4. The two approaches aim at solving the moment condition equation leveraging the cross-fitting approach to estimate its parameters. We calculated the elasticity for each product group (i.e. real-life product) with different setups obtained by combining these four factors:

1. **Input dataset**. For the input dataset we have the two variants described in Section 4.3.1. The difference is the granularity of the data that can be daily or weekly.
2. **First stage estimator**. The first stage estimator is selected between one of the algorithms described in the previous section. The practical implementations are:
   - *sklearn.linear_ model.Lasso* from *scikit-learn*
   - *sklearn.ensemble.RandomForestRegressor* from *scikit-learn*
   - *sklearn.ensemble.GradientBoostingRegressor* from *scikit-learn*

- *lightgbm.LGBMRegressor* from *lightgbm*

3. **Training procedure**. The training procedure is one between DML1 and DML2, their application in our specific case will be described in the following sections.
4. **Shop aggregation**. The elasticity of a product can be calculated per shop, i.e. calculating two values of elasticity (one for each shop), or considering the data from both shops jointly but considering the shop as feature. In the second case we obtain a single elasticity value for each product.

## Score function

Before going on with the practical description of DML1 and DML2 we should talk about the score function used in the moment condition. Recalling from Section 4.1.3 the score function for a partially linear regression model (as the one we use to model our data) is:

$$\phi(W;\theta,\eta) = (Y - l(X) - \theta(D - m(X)))(D - m(X)), \tag{4.22}$$

where in our specific case $X$ and $Y$ are the logarithmic transformations of respectively Price and Demand and $\theta$ the elasticity (see Section 4.2.2). This score function is linear in $\theta$ and can be rewritten as the linear combination of two terms [52]:

$$\phi(W;\theta,\eta) = \phi_a(W;\eta)\theta + \phi_b(W;\eta), \tag{4.23}$$

where $\phi_a(W;\eta)$ and $\phi_b(W;\eta)$ are:

$$\phi_a(W;\eta) = -(D - m(X))(D - m(X)), \tag{4.24}$$
$$\phi_b(W;\eta) = (Y - l(X))(D - m(X)). \tag{4.25}$$

The decomposition of the score functions will come handy to describe the practical application of DML1 and DML2.

## DML1

The practical application of the DML1 procedure is the following:

1. Take a K-fold random partition $(I_k)_{k=1}^K$ of observation indices $[N] = \{1,\dots,N\}$ such that the size of each fold $I_k$ is $n = N/K$. Also, for each $k \in [K] = \{1,\dots,K\}$, define its complement $I_k^c := \{1,\dots,N\} \setminus I_k$.
2. Select one of the ML estimators presented in Section 4.3.2 and train an instance for

the two nuisance functions for each $k \in [K]$:

$$\hat{m}_{0,k} = \hat{m}_0((D_i, X_i)_{i \in I_k^c}), \tag{4.26}$$
$$\hat{l}_{0,k} = \hat{l}_0((Y_i, X_i)_{i \in I_k^c}). \tag{4.27}$$

We have that $\hat{\eta}_{0,k} = (\hat{m}_{0,k}, \hat{l}_{0,k})$.

3. For each $k \in [K]$ construct the estimator $\check{\theta}_{0,k}$ as:

$$\check{\theta}_{0,k} = -\frac{\sum_{i \in I_k} \phi_b(W_i; \hat{\eta}_{0,k}(W_i))}{\sum_{i \in I_k} \phi_a(W_i; \hat{\eta}_{0,k}(W_i))}, \tag{4.28}$$

where $\hat{\eta}_{0,k}(W_i)$ are the values predicted for sample $i$ by the estimators of the nuisance functions.

4. Aggregate the estimators:

$$\tilde{\theta}_0 = \frac{1}{K} \sum_{k=1}^{K} \check{\theta}_{0,k}. \tag{4.29}$$

## DML2

The practical application of the DML2 procedure is the following:

1. Take a K-fold random partition $(I_k)_{k=1}^K$ of observation indices $[N] = \{1, \ldots, N\}$ such that the size of each fold $I_k$ is $n = N/K$. Also, for each $k \in [K] = \{1, \ldots, K\}$, define its complement $I_k^c := \{1, \ldots, N\} \setminus I_k$.

2. Select one of the ML estimators presented in Section 4.3.2 and train an instance for the two nuisance functions for each $k \in [K]$:

$$\hat{m}_{0,k} = \hat{m}_0((D_i, X_i)_{i \in I_k^c}), \tag{4.30}$$
$$\hat{l}_{0,k} = \hat{l}_0((Y_i, X_i)_{i \in I_k^c}). \tag{4.31}$$

We have that $\hat{\eta}_{0,k} = (\hat{m}_{0,k}, \hat{l}_{0,k})$.

3. Calling $J_k$ the size of the fold $I_k$, construct the estimator $\tilde{\theta}_0$ as:

$$\check{\theta}_{0,k} = -\frac{\sum_{k \in [K]} \frac{1}{J_k} \sum_{i \in I_k} \phi_b(W_i; \hat{\eta}_{0,k}(W_i))}{\sum_{k \in [K]} \frac{1}{J_k} \sum_{i \in I_k} \phi_a(W_i; \hat{\eta}_{0,k}(W_i))}, \tag{4.32}$$

where $\hat{\eta}_{0,k}(W_i)$ are the values predicted for sample $i$ by the estimators of the nuisance functions.

### Parameters tuning

The final aspect to consider is the tuning of the parameters for the first stage estimators. For each estimator we have a grid of parameters from which to select the best set. The process of tuning is done separately from the training procedures explained before and is based on a cross-validation approach. The procedure is applied to the same data on which we calculate the elasticity and is the following:

1. Take a K-fold random partition $(I_k)_{k=1}^K$ of observation indices $[N] = \{1, \dots, N\}$ such that the size of each fold $I_k$ is $n = N/K$. Also, for each $k \in [K] = \{1, \dots, K\}$, define its complement $I_k^c := \{1, \dots, N\} \setminus I_k$. If we perform tuning, the procedure for calculating elasticity should generate folds different from the ones generated during tuning; this can be obtained by shuffling the data before each step.

2. Select the ML estimator to tune and generate all the possible combinations of parameters $(P_j)_{j=1}^J$ to be tested.

3. For each parameters set $P_j$ and for each fold $I_k$ train an instance of the estimator on $I_k^c$ and evaluate it on $I_k$ using the $R^2$ score, we call $s_{jk}$ the score of a parameter set on a fold.

4. For each parameters set $P_j$ calculate the final score as the average score obtained on the folds:
$$s_j = \frac{1}{K} \sum_{k=1}^{K} s_k. \tag{4.33}$$

5. Select the parameters set with the best score $s_j$.

## 4.4. Experimental results

As already anticipated, we focused our analysis on a restricted set of products. These products have been sold a reasonable amount of times throughout their lifespan into the shop catalogs and also have undergo some changes in their price. We recall that these products are, in reality, a group of products containing each one some SKUs from both shops that we identified as referred to the same real-world entity. We now provide a general review of the outcome of our experiments. For the numeric results of each single test run, refer to the Appendix.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | | | | | | | | | | | |
| Daily | elasticity | -1.22 | -2.49 | -3.37 | -4.95 | -1.85 | -3.81 | -0.90 | -2.11 | -4.10 | -2.45 |
| | std. err. | 0.59 | 0.83 | 0.74 | 0.80 | 0.59 | 0.60 | 0.92 | 0.52 | 0.78 | 0.78 |
| Weekly | elasticity | -3.66 | -6.93 | -5.73 | -5.70 | -5.06 | -6.08 | -3.05 | -1.43 | -6.70 | -3.46 |
| | std. err. | 1.31 | 1.89 | 2.12 | 2.07 | 1.49 | 1.53 | 1.85 | 1.15 | 2.13 | 2.30 |
| First stage estimator | | | | | | | | | | | |
| Lasso | elasticity | -1.84 | -4.29 | -3.75 | -3.86 | -2.76 | -5.28 | -1.59 | -0.88 | -3.98 | -3.93 |
| | std. err. | 0.62 | 0.99 | 0.91 | 0.93 | 0.71 | 0.85 | 0.98 | 0.77 | 0.77 | 1.04 |
| RF | elasticity | -2.49 | -6.00 | -4.94 | -5.90 | -3.78 | -5.18 | -2.21 | -2.08 | -5.41 | -3.07 |
| | std. err. | 1.05 | 1.54 | 1.63 | 1.6 | 1.13 | 1.14 | 1.53 | 0.85 | 1.55 | 1.62 |
| GBR | elasticity | -2.85 | -4.68 | -5.05 | -5.83 | -3.98 | -4.84 | -2.29 | -2.11 | -6.49 | -3.24 |
| | std. err. | 1.10 | 1.55 | 1.69 | 1.63 | 1.19 | 1.20 | 1.61 | 0.88 | 1.76 | 1.81 |
| LGBM | elasticity | -2.57 | -3.88 | -4.47 | -5.70 | -3.31 | -4.49 | -1.81 | -2.01 | -5.72 | -1.57 |
| | std. err. | 1.04 | 1.37 | 1.51 | 1.59 | 1.14 | 1.07 | 1.40 | 0.84 | 1.73 | 1.69 |
| Shop aggregation | | | | | | | | | | | |
| Aggregated | elasticity | -2.47 | -4.92 | -4.79 | -4.99 | -2.93 | -4.87 | -2.10 | -1.19 | -4.82 | -3.01 |
| | std. err. | 0.60 | 1.06 | 1.05 | 1.05 | 0.75 | 0.77 | 1.08 | 0.59 | 1.02 | 1.19 |
| Shop A | elasticity | -2.75 | -4.42 | -5.02 | -7.60 | -5.40 | -5.12 | -0.81 | -3.71 | -7.72 | -5.85 |
| | std. err. | 1.52 | 1.64 | 1.78 | 1.78 | 1.58 | 1.43 | 1.76 | 1.25 | 1.89 | 1.88 |
| Shop B | elasticity | -2.10 | -4.80 | -3.85 | -3.38 | -2.04 | -4.85 | -3.02 | -0.41 | -3.66 | -0.01 |
| | std. err. | 0.74 | 1.39 | 1.47 | 1.49 | 0.81 | 1.00 | 1.31 | 0.66 | 1.45 | 1.55 |

Table 4.2: Average Double Machine Learning results with different training factors. In the *Dataset* row are reported the average elasticity values in the cases of daily and weekly datasets. In the *First stage estimator* row are reported the average values estimated with the different first stage estimators. In the *Shop aggregation* row are reported the average elasticities estimated individually for the two shops and in the aggregated case.

We can start by saying that the results are coherent to what we expect to be the elasticity of the products under consideration (i.e. sport goods). The estimated elasticities are always negative and in many cases lower than $-1$, the average value across all products and experiments is $-3.75$. This means that the products are elastic: when their price increases, their demand lowers in a more than proportional way and vice versa. This also makes sense since sport products are not essential goods thus people may avoid buying them when their price rises and buy them more when their price drops.

Between the different runs, obtained by combining the four factors described in the previous section, the results are coherent but the resulting coefficients are sometimes slightly

different. We will try to explain and motivate the main differences between each factor that characterize the runs. The average results obtained on each product for each training factor is reported in Table 4.2.

The first one we consider is the input dataset for which we have two variations, one with daily granularity and one with weekly granularity. In most of the cases the daily dataset provides results that are less elastic (closer to zero) than the ones calculated using the weekly dataset, the average difference between the two is 2.05. This difference is for sure highly related to the data but it may be also due to the fact that in the weekly dataset, being the data aggregated, we have samples with higher values of demand (since it is the sum of the demand of the days in a week). In general, the two approaches have advantages and disadvantages. Using the daily dataset we have more samples (on average 2490 samples per product, $\sim 7$ times the number of samples w.r.t weekly) to train the first stage estimators but we also introduce more bias since the samples, that comes from a time series, are unavoidably correlated. On the other hand, by using weekly aggregated data we reduce this correlation but we also reduce the number of samples that can be used to train the first stage estimators.

The second factor characterizing different runs is the first stage estimator used to generate the residuals. For the for stage estimator we use four algorithms: *Lasso*, *RandomForestRegressor*, *GradientBoostingRegressor* and *LGBMRegressor*. In many cases the results obtained with *RandomForestRegressor*, *GradientBoostingRegressor*, and *LGBMRegressor* are very similar and coherent with the assumption of elastic products, while the results obtained with *Lasso* are sometimes slightly different but still elastic. This is reasonable since *Lasso* is a linear model that can only assume a liner relationship between the confounding variables and the dependent variables (price or demand). On the other hand the other three algorithms are tree based ensembles that can also capture non-linear dependencies between the data but they are also more prone to overfitting. The average elasticities obtained by the four algorithms are respectively $-4.11$, $-4.14$, $-3.55$ and $-3.22$.

We will not analyze the results of DML1 and DML2 since they obtained almost the same values. We will provide results only for DML2 because, as stated by the original authors of DML, it gives more reliable results in almost any situation and also works better in few samples scenarios, like ours.

The final aspect considered is shop aggregation. We estimated elasticities both for single shops and considering the data of the two shops together. Shop B has more inelastic values with respect to Shop A. We consider the results on Shop A more reliable since it has more data and an higher volume of sales. The low volume of sales of Shop B may

probably negatively influence the accuracy of first stage estimation. If a product is always sold few times, an estimator that always predict low sales can be considered good even if it does not capture the real dependency on the confounding variables. For Shop A and Shop B the average elasticities are respectively $-4.84$ and $-2.81$, while in the aggregated case the average estimation is $-3.61$.

In any case, the reliability of the Double Machine Learning approach depends on the number of samples we can observe and even more on the quality of the confounding variables that we can include in the first stage estimators models. In our experiments we tried to extract from the data a lot of possible factors that may influence price and demand, however some important aspects cannot be observed. We believe that the most important is for sure the advertising on channels external to the shops (e.g. social advertising) that may significantly increase sales and that we are unaware of. We also cannot know the factors that influences pricing decisions, we know that in the two shops they are taken by humans but we obviously do not know which aspects they considered in their decision process.

# 5 | Difference-in-Differences

In this chapter, we present a strategy to estimate the price elasticity by comparing the demand of a product that is sold in the same period across two different shops, in those situations in which it undergoes a price change in one of the shops and not in the other. The models we will describe are based on a statistical technique called Difference-in-Differences (DD), a technique used in the quantitative social sciences since 1855 by John Snow in his analysis of cholera transmission [53].

## 5.1. Theoretical background

Difference-in-Differences is a technique that uses panel data to estimate the effect of a treatment on an outcome variable. With panel data, sometimes referred to as longitudinal data, we define a set of observational data collected at a regular frequency across a group of individuals. To be used for the DD, the data need to contain two periods and two groups of individuals. The first period is where there is no treatment, and the second is where some individuals are exposed to the treatment. The two groups of individuals are the control group, which had never been exposed to the treatment, and the treatment group, which underwent the treatment in the second period.

The DD methodology consists of taking two differences between group means. The first difference is the difference in the average of the outcome variable between the two periods for each of the groups. The second difference is the difference between the differences calculated for the two groups in the first stage (from here derives the definition Difference-in-Differences). The result measures the causal effect of the treatment.

### 5.1.1. The $2 \times 2$ case

The simplest case in which DD is applied, called $2 \times 2$ DD [54], is that in which there are only two groups, the control group and the treatment group, and only two periods that we will define as *Pre*, the period before the treatment, and *Post*, the period after the treatment. We can estimate the ATT (average treatment effect on the treated), i.e. the

average treatment effect on the outcome of the treated group, as:

$$\widehat{\beta} = (\overline{y}^{Post}_{treat} - \overline{y}^{Pre}_{treat}) - (\overline{y}^{Post}_{control} - \overline{y}^{Pre}_{control}), \tag{5.1}$$

where $\widehat{\beta}$ is the ATT, and $\overline{y}$ is the sample mean of the outcome variable for a group (*treat* or *control*) in a time period (*Pre* or *Post*).

To understand better how the DD works, we can transform Equation 5.1 from a difference of sample means to a conditional expectation of outcomes. We define $y_{i,t}(d)$ as the outcome that unit $i$ would experience in period $t \in \{0, 1\}$ if it had received the treatment $d \in \{0, 1\}$. The variable $t = 0$ represents the period before the treatment and $t = 1$ is the period after the treatment while the variable $d = 0$ means that the treatment is absent while $d = 1$ indicates that the treatment is active. The expected average outcome of all units in period $t$ that have received treatment $a$, if they had received treatment $b$, is equal to $\mathbb{E}[Y_t(b)|D = a]$. We can rewrite Equation 5.1 as:

$$\beta = \mathbb{E}[Y_1(1) - Y_0(0)|D = 1] - \mathbb{E}[Y_0(0) - Y_0(0)|D = 0]. \tag{5.2}$$

By adding and subtracting $\mathbb{E}[Y_1(0)|D = 1]$, that is the expected outcome in *Post* period of the treated group if it had not been treated, we have:

$$\begin{aligned} \beta &= \mathbb{E}[Y_1(1) - Y_0(0)|D = 1] - \mathbb{E}[Y_1(0) - Y_0(0)|D = 0] \pm \\ &\quad \mathbb{E}[Y_1(0)|D = 1] \\ &= \mathbb{E}[Y_1(1) - Y_1(0)|D = 1] + \\ &\quad \mathbb{E}[Y_1(0) - Y_0(0)|D = 1] - \mathbb{E}[Y_1(0) - Y_0(0)|D = 0]. \end{aligned} \tag{5.3}$$

We obtain that the estimation is given by the term $\mathbb{E}[Y_1(1) - Y_0(0)|D = 1]$, that is the ATT, while the second term is a difference that should go to zero if the trend of the two groups, if neither was treated, is the same. In order for this difference to go to zero we have to introduce an assumption called parallel trend assumption. This is a critical point in the application of DD because it requires that in the absence of treatment, the difference between the treated and control group is constant over time; otherwise, the estimation will be contaminated by selection bias. However, there is no statistical test for this assumption since it's impossible to get the needed observations $\mathbb{E}[Y_1(0)|D = 1]$. We can verify that the trends are parallel at least before the treatment, but this does not guarantee that they are also parallel after it.

**Assumption 1** (Parallel trends)**.** *The mean change in outcomes over time in all units if they were not exposed to treatment is the same as the mean change in outcomes over time for all units that did not experience the treatment.*

$$\mathbb{E}[Y_1(0) - Y_0(0)|D = 1] = \mathbb{E}[Y_1(0) - Y_0(0)|D = 0] \tag{5.4}$$

Taking up Equation 5.1 we can rewrite the $2 \times 2$ DD as:

| **Group** | **Time** | **Expected outcome** | **D$_1$=Post-Pre** | **D$_{1,Treat}$-D$_{1,Control}$** |
|---|---|---|---|---|
| *Control* | *Pre* | $\alpha$ | | |
| | *Post* | $\alpha + \lambda$ | $\lambda$ | |
| *Treat* | *Pre* | $\alpha + \gamma$ | | |
| | *Post* | $\alpha + \gamma + \lambda + \beta$ | $\lambda + \beta$ | $\beta$ |

Table 5.1: $2 \times 2$ Difference-in-Differences.

In the table, $\alpha$ is a fixed effect, $\lambda$ is the effect given by the post period, $\gamma$ is the effect of being in the treated group, and $\beta$ is the effect of the treatment. Figure 5.1 shows the $2 \times 2$ regression in an optimal scenario and in a scenario in which the parallel trends assumption does not hold.

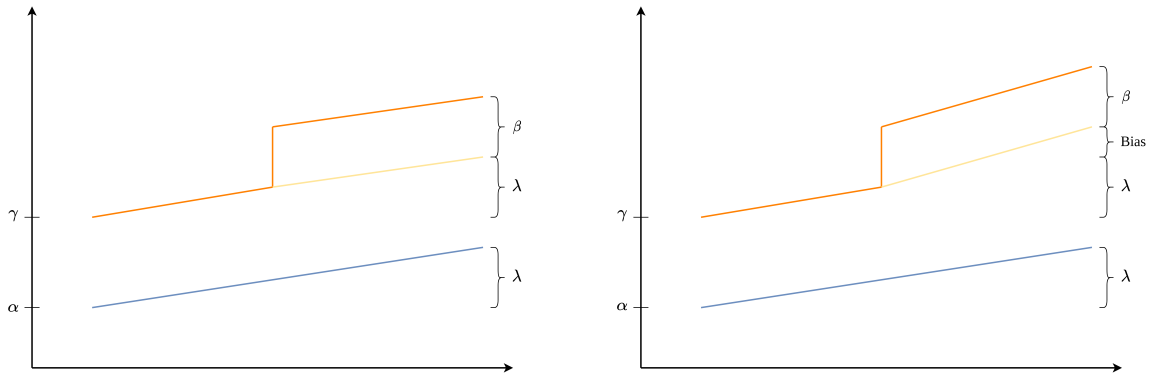


Figure 5.1: $2 \times 2$ Difference-in-Differences regression diagram in optimal scenario (left) and in the case of parallel trends assumption violated (right).

The ATT can be estimated with the following regression:

$$y_{i,t} = \alpha + \gamma TREAT_i + \lambda POST_t + \beta(TREAT_i \times POST_t) + \epsilon_{it}, \tag{5.5}$$

where $y_{i,t}$ is the value of the outcome of entity $i$ in time $t$, $TREAT_i$ is a dummy variable equal to 1 if the sample is from the treated group or 0 if it is from the control group, $POST_t$ indicates whether the sample is from a period after the treatment, and $\epsilon_{i,t}$ is a random noise.

### 5.1.2. Two-way fixed effects

The regression introduced in Equation 5.5 is too simple for most of the cases because it works only if all the units in the treated group receive the treatment at the same time. In most situations the treated units receive the treatment at different points in time. For this reason we introduce a slightly different regression:

$$y_{i,t} = \alpha_i + \alpha_t + \beta D_{i,t} + \epsilon_{it}. \tag{5.6}$$

This model is called two-way fixed effects (TWFE) regression because it includes both time fixed effects $\alpha_t$ and unit fixed effects $\alpha_i$. $D_{i,t}$ is a dummy variable equal to 1 if the sample is exposed to the treatment at time $t$, and $\beta$ is the treatment effect.
Goodman-Bacon demonstrated that TWFE is a weighted average of all possible $2 \times 2$ DD estimators that compare timing groups between each other [54].

**Theorem 5.1** (Difference-in-Differences decomposition Theorem)**.** *Assume that the data contain $k = 1, ..., K$ timing groups of units ordered by the time when they receive a binary treatment, $k \in (1, T]$. There may be one timing group, $U$, that includes units that never receive treatment. The OLS tha estimate $\widehat{\beta}$ in a two-way fixed-effects regression (5.6) is a weighted average of all possible $2 \times 2$ DD estimators.*

$$\widehat{\beta} = \sum_{k \neq U} s_{k,U} \cdot \widehat{\beta}^{2\times2}_{k,U} + \sum_{k \neq U} \sum_{l > k} \left[ s^k_{k,l} \cdot \widehat{\beta}^{2\times2,k}_{k,l} + s^l_{k,l} \cdot \widehat{\beta}^{2\times2,l}_{k,l} \right], \tag{5.7}$$

*in which the $2 \times 2$ DD estimators are:*

$$\widehat{\beta}^{2\times2}_{k,U} = (\overline{y}^{Post(k)}_k - \overline{y}^{Pre(k)}_k) - (\overline{y}^{Post(k)}_U - \overline{y}^{Pre(k)}_U), \tag{5.8}$$

$$\widehat{\beta}^{2\times2,k}_{k,l} = (\overline{y}^{Mid(k,l)}_k - \overline{y}^{Pre(k)}_k) - (\overline{y}^{Mid(k,l)}_l - \overline{y}^{Pre(k)}_l), \tag{5.9}$$

$$\widehat{\beta}^{2\times2,l}_{k,l} = (\overline{y}^{Post(l)}_l - \overline{y}^{Mid(k,l)}_l) - (\overline{y}^{Post(l)}_k - \overline{y}^{Mid(k,l)}_k), \tag{5.10}$$

*and the weights are:*

$$s_{k,U} = \frac{(n_k + n_U)^2 n_{k,U}(1 - n_{k,U})\overline{D}_k(1 - \overline{D}_k)}{\widehat{V}^D}, \tag{5.11}$$

$$s^k_{k,l} = \frac{((n_k + n_l)(1 - \overline{D}_l))^2 n_{k,l}(1 - n_{k,l})\frac{\overline{D}_k - \overline{D}_l}{1-\overline{D}_l}\frac{1-\overline{D}_k}{1-\overline{D}_l}}{\widehat{V}^D}, \tag{5.12}$$

$$s^l_{k,l} = \frac{((n_k + n_l)\overline{D}_k)^2 n_{k,l}(1 - n_{k,l})\frac{\overline{D}_l}{\overline{D}_k}\frac{\overline{D}_k - \overline{D}_l}{1-\overline{D}_k}}{\widehat{V}^D}, \tag{5.13}$$

*where $k$ represents a group treated at time $k$, $l$ represents a group treated at time $l$, $U$ is the untreated group, $n$ represent the number of samples for each group, and $D$ represents the amount of time for which a group is treated (e.g. $\overline{D}_k = 0.8$ means that group $k$ spent 80% of its time being treated).*
*At last, we have:*

$$\sum_{k \neq U} s_{k,U} + \sum_{k \neq U} \sum_{l > k} (s^k_{k,l} + s^l_{k,l}) = 1. \tag{5.14}$$

As stated by this theorem, TWFE allows us to estimate casual effect with DD strategy even when we have to compare groups that receive the treatment at different instants. Since the TWFE is a combination of $2 \times 2$ DD the Assumption 1 needs to be still valid.

### 5.1.3. Two-way fixed effects with a continuous treatment

Until this point, groups may or may not be exposed to treatment, which means that the treatment variable is binary. In many real cases the treatment the groups are subjected to can have different intensities. As explained in Callaway 2021 [33], TWFE regression works well also with a continuous or discrete non-binary dosage of treatment. However, for the estimate to have no selection bias, we need to consider further assumptions.

Before talking about the assumptions, we need to extend some notation previously introduced in Section 5.1.1. There are multiple time periods $t \in \{1, ..., T\}$, $d$ is no more a binary variable, it can be continue or discrete, and we define $y_{i,t}(g, d)$ the potential outcome of the unit $i$ over period $t$ which receives the dose $d$ at time $g \in \{2, ..., T + 1\}$. We call the group of units that receive the treatment at a certain time $g$ as group $g$.

**Assumption 2** (Strong prallel trends)**.** *For all doses $d$, and treatment timing $g$, the average change in outcomes over time across all units if they had been assigned that amount of dose is the same as the average change in outcomes over time for all units that*

*experienced that dose.*

$$\mathbb{E}[Y_t(g,d) - Y_{t-1}(0)|G = g] = \mathbb{E}[Y_t(g,d) - Y_{t-1}(0)|G = g, D = d]. \tag{5.15}$$

Assumption 2 is a stronger arrangement of Assumption 1 because, for all doses $d$, if a group $g$ would have received dose $d$ — whatever dose it really has — it should have the same trend as the group that has received dose $d$ at the time $g$.

**Assumption 3** (Staggered adoption)**.** *Units become treated once with dose $d$ and they remain treated with dose $d$ in all subsequent periods.*

$$y_{i,t}(k,d) = y_{i,t}(0) \text{ for each } t < k \text{ and } y_{i,t}(k,d) = y_{i,t}(d) \text{ for each } t \geq k. \tag{5.16}$$

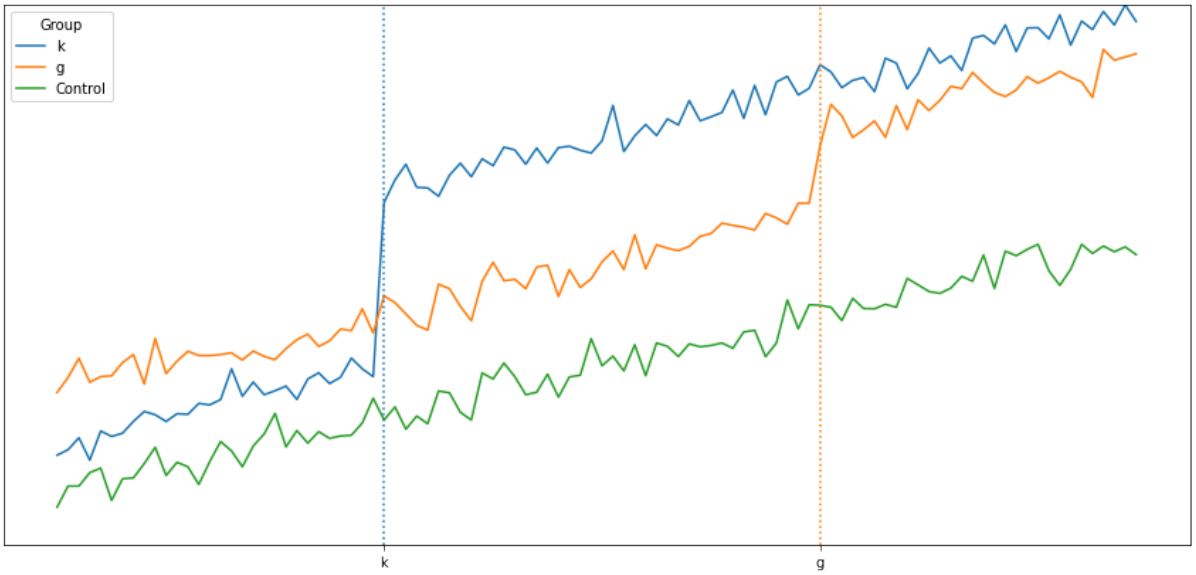


Figure 5.2: Example of staggered timing and variation in the dose.

We need to add and explain more notations to understand things better:

- $ATT(g,t,d|g,k)$: is the average effect of dose $d$, for timing group $g$, in time period $t$, among units in group $g$ that experienced dose $k$.

$$ATT(g,t,d|g,k) = \mathbb{E}[Y_t(g,d) - Y_t(0)|G = g, D = k]. \tag{5.17}$$

- $ATE(g,t,d)$ (average treatment effect): is the average effect of dose $d$ among all

units in timing group $g$, not just those that experienced dose $d$ (not all units in the population though), in time period $t$.

$$ATE(g,t,d) = \mathbb{E}[Y_t(g,D) - Y_t(0)|G=g]. \tag{5.18}$$

- $ACR(g,t,d)$ (average causal response): is the effect of a marginal change in the dose $d$ on the outcomes of timing group $g$ in period $t$.

$$ACR(g,t,d) = \frac{\partial \mathbb{E}[Y_t(g,d)|G=g, D=d]}{\partial d}. \tag{5.19}$$

Under Assumption 2 we consider $ATT(g,t,d|g,k) = ATT(g,t,d|g,d))$ for each $k$, so we have also that:

$$ATE(g,t,d) = ATT(g,t,d). \tag{5.20}$$

**Assumption 4** (No treatment effect dynamics)**.** *For all $g \in G$ and $t \geq g$ (i.e. post-treatment periods for group $g$), $ACR(g,t,d)$ and $ATE(g,t,d)$ do not vary with $t$*

Assumption 4 states that the average effect of dose $d$, for timing group $g$ does not vary over time after the dose have been applied. Otherwise, as you can see in Figure 5.4, the effect strongly depends on the selected period.

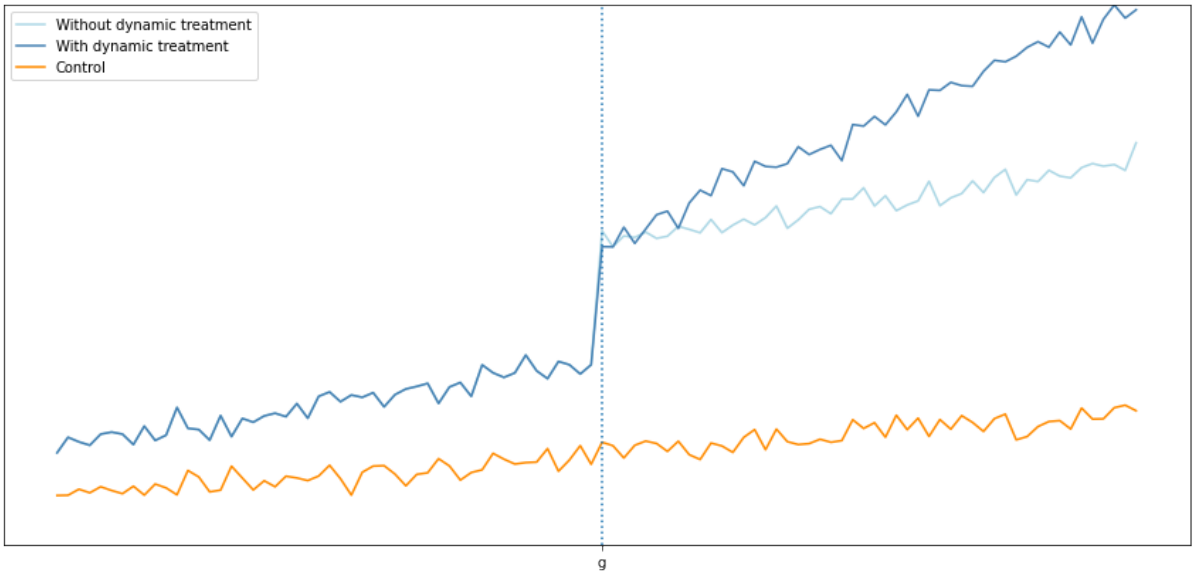


Figure 5.3: Example of dynamic treatment effects.

**Assumption 5** (Homogeneous causal response across groups)**.** *For all $g \in G$ with $t \geq g$ and $k \in G$ with $t \geq k$, $ACR(g,t,d) = ACR(k,t,d)$ and $ATE(g,t,d) = ATE(k,t,d)$.*

Assumption 5 imposes that, for a fixed time period, causal responses to the treatment are constant across timing groups. If two groups underwent the same treatment at different times, their causal response have to be the same.

**Assumption 6** (Homogeneous causal response across dose)**.** *For all $g \in G$ with $t \geq g$, $ACR(g,t,d)$ not vary across $d$, and, in addition, $ATE(g,t,d)/d = ACR(g,t,d)$.*

Assumption 6 imposes that, within timing group and time period, the causal response to more dose is constant across different values of the dose. The dose and its causal effect are linearly correlated.

Finally, under all the assumptions announced so far, Callaway (2021) [33] states that by a TWFE regression like the one presented in Equation 5.6, where $D_{i,t}$ is the amount of dose that unit $i$ experiences in time period $t$, the estimated coefficient $\beta$ is:

$$\beta^{TWFE} = ACR^*, \tag{5.21}$$

where $ACR^*$ is the aggregate overall causal response across all timing groups that participate in the treatment in any period across the distribution of the dose.

## 5.2. DD for elasticity estimation

In the case of price elasticity estimation, the Difference-in-Differences strategy can be used to estimate the effect of a price change in a group of products by comparing their demand with the demand of the same group of products in another environment where they maintain a constant price. In particular, having the sales data of two e-commerce in the same period, we devised to apply the TWFE model by taking into account the time windows in which a product is in the catalog of both stores and, in one of the two, the price changes during the period considered while in the other it remains constant. In this section, we show how DD can be applied with panel data obtained from different e-commerce and provide some considerations on the necessary assumptions that make it possible to get a reliable estimate.

### 5.2.1. Logarithmic transformation

Generally, DD is designed to estimate the absolute value of the effect caused by a treatment. However, in the case of elasticity, we need to calculate the percentage change in demand caused by a percentage change in price.

Consider the simple $2 \times 2$ DD in which our outcome is the logarithmic transformation of the demand, we get:

$$\begin{aligned} ATT &= \mathbb{E}\left[\log(Y_1(1)) - \log(Y_1(0))|D=1\right] \\ &= \mathbb{E}\left[\log\left(\frac{Y_1(1)}{Y_1(0)}\right)\middle|D=1\right] \\ &= \mathbb{E}\left[\log\left(\frac{Y_1(1)-Y_1(0)}{Y_1(0)}+1\right)\middle|D=1\right] \\ &\sim \mathbb{E}\left[\frac{Y_1(1)-Y_1(0)}{Y_1(0)}\middle|D=1\right]. \end{aligned} \tag{5.22}$$

This means that by applying the DD procedure on logarithmic outcome, we obtain its percentage change caused by the treatment.

As seen in Section 5.1.3 by operating TWFE with continuous treatment and multiple periods, the coefficient $\beta$ that we estimate is equal to:

$$\beta = ACR^* = \frac{ATE^*}{d}, \tag{5.23}$$

where $ATE^*$ is the aggregate overall treatment effect across all timing groups that participate in the treatment in any period across the distribution of the dose.

If we use the percentage change in price as the treatment dose, calculated as:

$$\begin{aligned} d &= \log(Price_1) - \log(Price_0) \\ &= \log\left(\frac{Price_1}{Price_0}\right) \\ &= \log\left(\frac{Price_1 - Price_0}{Price_0}+1\right) \\ &\sim \frac{Price_1 - Price_0}{Price_0}, \end{aligned} \tag{5.24}$$

and if the outcome is the logarithm of the demand, then $ATE$ is the percentage change of the demand caused by price variation $d$. This means that $\beta$ is the price elasticity.

### 5.2.2. Assumptions

As seen in the previous section, in order for TWFE to not being contaminated by selection bias, we need to discuss about the assumptions reported in the previous section.

Assumption 2 which, as said, includes Assumption 1 is about parallel trends. As already mentioned, there is no way to make sure that this assumption is valid on real data. What we can do is an analysis of the environments from which the data are collected and the behavior of the data in such environments, in order to understand if it makes sense to consider this assumption valid. In our specific case, the data compared are collected from two similar e-commerce operating in the same market, in the same historical period, and in the same country. It is, therefore, logical to assume that in the absence of price changes, the variation in time of the demand for a product is the same between the two shops. In practice, however, the difference in popularity between them and the different starting price at which a product is sold in the shops could, in the long term, lead to different sales trends for the same product. Since the time windows used to estimate the elasticity values are short, we can state that it is reasonable to approximately consider Assumption 1 valid.

Assumption 3 requires that once a product experiences a price change, it holds that price for the rest of the time. In the time frame in which the data available to us have been collected, a product experiences multiple price changes. However, this is not a problem as the considered time windows used for obtaining an estimate are cut as soon as a product encounters a second price variation. So, for each time window considered, we are sure that once a product changes its price, it keeps it for the rest of the analyzed period.

Assumption 4 requires the treatment effect to be non-dynamic. This means that, as the price changes, there may be an increase or decrease in demand, but the trend should remain the same. In reality, as already mentioned before, it is unlikely that in the long term, a product with different prices or in different stores will maintain the same trends. However, in the considered short time windows and within the limits of the price range at which a product is usually sold, it is legitimate to approximately consider this assumption valid.

Assumptions 5 and 6 are valid by the definition of constant elasticity. For Assumption 5, we have that the response of the demand to a price change does not depend on the period in which the price change is applied, and for Assumption 6, we have that the percentage change in demand is linearly dependent on the percentage change in price.

### 5.2.3. Model

The TWFE regressions can be applied to estimate the elasticity in each period in which a product changes its price in a shop and keeps it constant in the other. It is based on the model:

$$\log Q_{it} = \alpha_i + \gamma_t + \beta T_{it} + \delta X_{it} + u_{it}, \tag{5.25}$$

where $i$ indexes entities, i.e. products (SKU), and $t$ indexes time. The variables $\alpha$ and $\gamma$ are entity and time fixed effects, $\beta$ is the elasticity, $X_{it}$ is the vector of covariate (confounding) variables, $\delta$ is the vector of weights assigned to each variable and $u$ is an error term. Before the price change, $T_{it}$ is 0, while after the change, it is equal to the percentage variation. The outcome is obtained by removing the fixed effects by residualizing from the dependent and independent variables their unit-specific and time-specific means and performing the regression on the residualized variables. An important assumption of the TWFE is that the error term $u_{it}$ is not correlated with the covariates $X_{it}$ not only at time $t$ but at any time step. Therefore, not only the non-correlation between $u_{it}$ and $X_{it}$, but also between $u_{it}$ and $x_{i1}, \ldots, x_{iT}$ is assumed. For this reason, lagged dependent variables or lagged variables depending from the dependent variable — the demand in our case — are ruled out. As proved by Barros (2020) [55], violating this assumption can introduce significant bias in the estimation of the elasticity coefficient $\beta$.

## 5.3. Method deployment

In this section, we show the different types of data aggregation that we used, the data manipulation required to correctly apply TWFE, and finally, we describe the different training procedures used to estimate the price elasticity.

### 5.3.1. Dataset generation

As in the case of DML, for every product group in which we have at least one product for each of the two shops, we extracted the historical data of the products in the group. Starting from this data we generated two different variants of the dataset to be used as input for the TWFE. In one of them, we aggregated the data of all products (SKU) from a store that are considered the same real-world product, while in the other we treated each SKU separately.

As mentioned in Section 5.2, to apply a TWFE regression, we cannot use the whole history of data for each product but we need to extract only short and reasonable time windows. The factors considered when extracting valid periods for a product were:

- The presence of the product in the catalog of both shops for each day of the period.
- The presence of a percentage price change of minimum 1% in at least one of the shops.
- The presence of at least 4 days before and after the price change.
- Limit of at most 10 days between the start of the period and the first price change.
- Limit of at most 10 days between the end of the period and the last price change.
- For both shops the product must have been sold at least once.
- If the product changes its price in Shop A then it must keep its price constant in Shop B, or vice versa.

For each period and for each store, we mark each aggregate product or single SKU — depending on the type of dataset used — as belonging to the control group if it underwent a price change, or to the treated group if it does not.

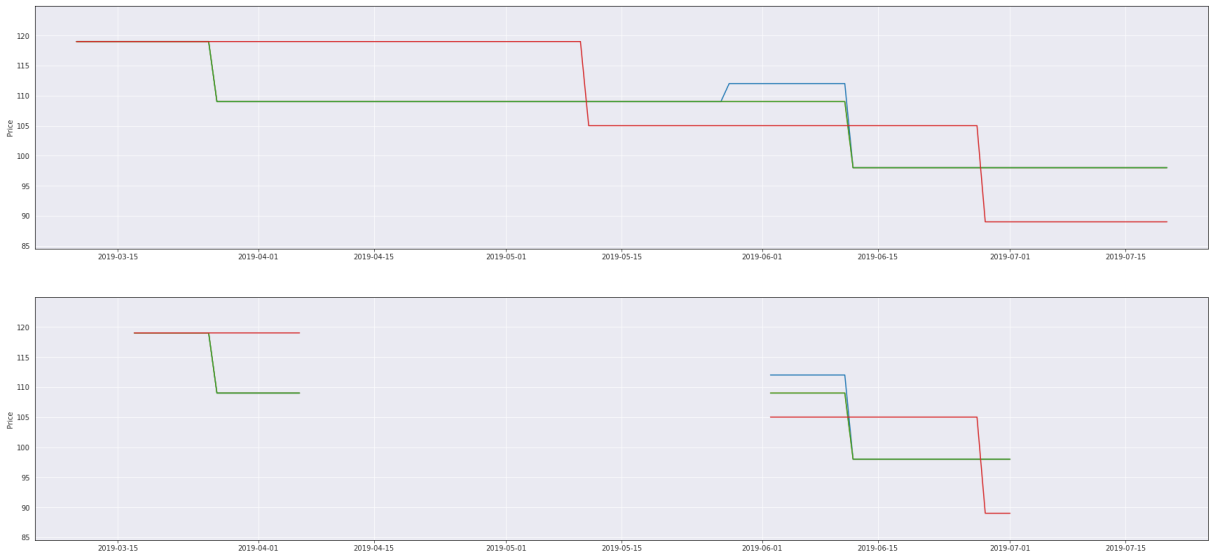


Figure 5.4: Example of periods extraction. The first graph represents the price history for 3 SKUs belonging to the same product group while in the second graph we have the valid periods extracted.

In the end, for each product we have a dataset containing a list of periods that can be used independently to estimate the elasticity with a TWFE regression. Each period also has a set of confounding variables, similar to the ones of DML. The features of the dataset are:

| Feature | Disaggregated | Aggregated |
|---|---|---|
| *Product ID (entity fixed effect)* | The identification code of the product used in the catalog of the respective shop (SKU) | The Shop ID |
| *Date (time fixed effect)* | Day in which the sample is observed | Day in which the sample is observed |
| *Details* | Number of clicks on the product page that day | Sum of the number of clicks on the product page that day for all the products of the group |
| *Promo* | Indicates whether that day the product is listed under a category considered as promotional | Indicates whether that day the majority of the products in the group are listed under a category considered as promotional |
| *Lagged price* | Average price of the product (SKU) in the previous 7 days. | Average price between all the products in the product group in the previous 7 days. |
| *Lagged details* | Average daily number of clicks on the product page in the previous 7 days. | Average daily number of clicks between all the products in the product group in the previous 7 days. |

Table 5.2: Two-way fixed effects dataset features.

There are significantly less features with respect to the ones described in Section 4.3.1. Some lagged features depending on the demand (such as the demand of previous days) are excluded in this regression since, as explained in previous section, TWFE assumes the error term to be independent from the covariates at any time step. Other features such as sale periods or seasons are absorbed by the fixed effects since they are fixed for a day or a product.

### 5.3.2. Training procedure

The training procedure consists in the application of the strategy described theoretically in Section 5.1.3 and based on the model of Section 5.2.3. We calculated the elasticity for each product group and for each period with three different setups:

- In the first setup we used the disaggregated dataset and trained a TWFE model for each period and for each pair of treated and control SKUs, obtaining an elasticity estimate for each possible comparison. If, in a given period, a product has several SKUs in a shop and some of them change their price while some other keep their price always constant, then we can have treated and control groups also within the same shop and compare them to have additional elasticity estimates. However, the amount of possible intra-shop comparisons are very few since most of the SKUs belonging to the same shop have similar behaviours and usually change price in the same time points.
- In the second setup we used again the disaggregated dataset but instead we trained, for each period and for each product, a single TWFE model using the data for all the SKUs in the product group. Each SKU is treated as an entity in the fixed effects. In this scenario we obtained an aggregated estimate for each product in each period.
- In the third setup we used the aggregated dataset thus we had TWFE models performing a single comparison for each product group and for each period. Recall that for each period all the data of the SKUs belonging to a product group for a shop are aggregated as a single time series.

The final elasticity for a product is calculated as the mean of the coefficients calculated for that product. We tested these three procedures with two regression models, one including the covariates while the other without. In our application we used Python to perform the TWFE regressions and in particular the *PanelOLS* class from *linearmodels* library.

## 5.4. Experimental results

As for the Double Machine Learning we focused our analysis on a restricted set of products, that is the same used for DML. In this section we provide a review of the outcomes of our experiments.

Recalling from the previous section, we calculated the value of elasticity for all the products in various periods and then averaged the results to get the final elasticity for a product. Also in the Difference-in-Differences approach the most of the results are co-

herent to what we expect to be the elasticities of the product under consideration. The elasticities — exception made for one product — are negative and elastic (lower than $-1$) and the global average elasticity obtained is $-3.81$. In Table 5.3 we report the average elasticities obtained in each scenario.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Without covariates | | | | | | | | | | | |
| Disagg 1 | elasticity | 0.51 | -6.63 | -5.39 | -7.01 | -3.75 | -6.29 | -5.00 | -2.58 | -3.07 | -2.58 |
| | std. err. | 0.50 | 0.78 | 0.74 | 1.22 | 0.57 | 0.76 | 0.85 | 1.05 | 1.13 | 1.27 |
| Disagg 2 | elasticity | 0.37 | -6.35 | -5.71 | -7.01 | -3.29 | -5.43 | -4.76 | -2.93 | -3.60 | -2.20 |
| | std. err. | 0.86 | 0.98 | 1.09 | 1.44 | 0.83 | 1.04 | 0.94 | 1.52 | 1.68 | 1.65 |
| Agg | elasticity | 0.44 | -8.01 | -4.96 | -7.99 | -0.93 | -1.67 | -0.73 | -3.56 | -2.74 | -3.4 |
| | std. err. | 1.07 | 0.97 | 1.37 | 1.76 | 1.00 | 1.11 | 1.00 | 1.54 | 1.39 | 1.85 |
| With covariates | | | | | | | | | | | |
| Disagg 1 | elasticity | 0.84 | -6.50 | -5.46 | -6.19 | -3.23 | -5.36 | -4.5 | -1.93 | -4.73 | -4.53 |
| | std. err. | 1.09 | 1.24 | 1.42 | 1.31 | 0.73 | 0.94 | 1.31 | 1.85 | 1.33 | 1.67 |
| Disagg 2 | elasticity | -0.77 | -6.12 | -7.58 | -6.18 | -3.38 | -5.76 | -4.92 | -2.63 | -4.09 | -2.39 |
| | std. err. | 1.60 | 1.67 | 2.15 | 2.65 | 1.47 | 1.55 | 1.69 | 3.00 | 2.72 | 3.64 |
| Agg | elasticity | 0.70 | -9.27 | -2.78 | -4.42 | -0.55 | -1.52 | -2.68 | -5.33 | -3.03 | 2.14 |
| | std. err. | 4.84 | 2.02 | 3.04 | 3.47 | 1.94 | 2.30 | 3.10 | 2.43 | 4.14 | 5.06 |

Table 5.3: Average Difference-in-Difference elasticities for each aggregation, with and without covariates. In the *Disagg 1* case we train a model for each period by using the data from all the SKUs while in the *Disagg 2* case we train a model for each period and for each combination of treated-untreated SKUs. The *Agg* case is the case in which the data of all the SKUs are aggregated into a single series.

We can notice that for a product, the three coefficients calculated in the three different aggregation scenarios are sometimes very different. In particular the last scenario, in which the data of the SKUs are aggregated and considered as a single product, seems to provide significantly different results with respect to the other two approaches that are more similar. The disaggregated approaches provides average elasticities of $-4.24$ and $-4.17$ while in the aggregated case we have an average coefficient of $-3.01$. We believe the first two approaches to be more reliable since by keeping the data disaggregated we have much more periods and SKUs that are compared during the estimation process. Indeed the standard errors in the aggregated case are very high, showing that the measure is uncertain.

In addition, we estimated the coefficients by excluding the covariate variables from the regression. Also this case the coefficients obtained are negative and elastic (exception made for one product) and more coherent between the three scenarios of data aggregation. These results are slightly different than the ones obtained including covariates and the average difference is 1.02. This is probably because a portion of the demand is captured by these extra independent variables. Including them may bring advantages because they may capture additional factors influencing the demand but with TWFE the model only assumes a linear relation between these variables and the dependent one and this may lead to a bad modeling of the dependence if it is actually non-linear.

With TWFE we can also have an estimation of the elasticity for each period in which the regression is applied and thus obtain different elasticities for each product across different time spans. In the scenario of constant elasticity with respect to both price and time, we expect this coefficient to be constant across different periods. In our estimates this does not happen in the majority of the cases and this can be due, in principle, to the constant assumption being violated. In a real-case indeed it is possible for the elasticity to be influenced by time-dependent factors. For instance, in a scenario like ours that is about sport goods which usually are replaced by new versions each year, the demand might decrease in time and the products might progressively become more inelastic. Also factors such as seasons and sale periods can influence not only the demand but also the way in which customers react to price changes. Finally, also the linearity assumption of the model and the quality of the covariates can introduce variance in the estimates across different periods. By averaging across a large number of periods, however, we obtain an average estimation that can be considered more or less reliable depending on the number of periods considered, that in our case is on average 7 for each product. Figure 5.5 shows the trend of the elasticity across different periods for two of the products under consideration.

To conclude, almost all the elasticities estimated can be considered reliable values. In general, due to the linearity assumption of the model, this method is good if covariate variables are few or can be absorbed by time and entity fixed effects, or obviously if the dependence with their dependent variable (demand) is linear. Finally, all the assumptions described in previous sections must be ensured in order for the method to be applicable.

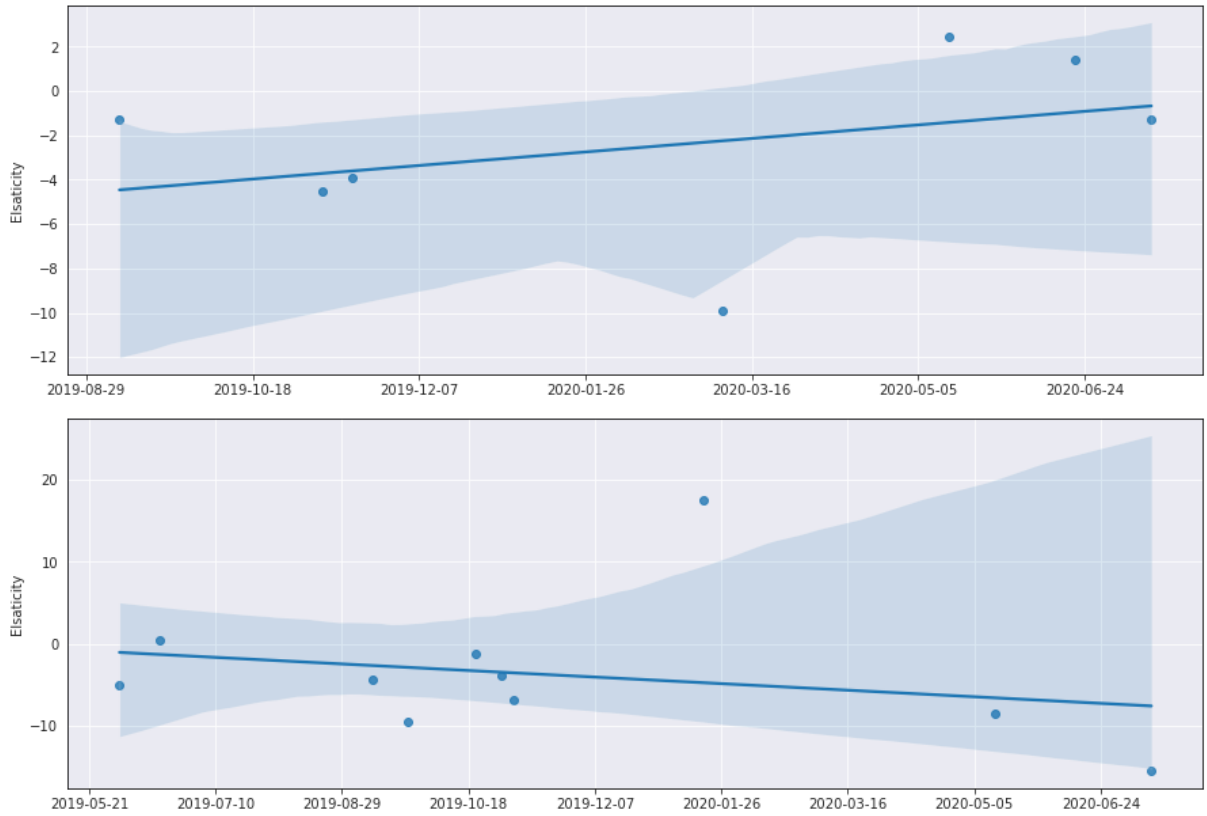


Figure 5.5: Trend of the elasticity across different time periods for products *P5* and *P7*.

# 6 | Comparison of the methods

After providing the results for both the approaches we now recall the similarities and differences between them and give a description of their main limitations. Based on our tests they both provide reliable elasticity estimates, according to our assumption of elastic goods. On average they give similar results on all the products, except for *P1* in which DD gives an unrealistic coefficient of 0.36. The average difference between the two approaches is 1.45. In Table 6.1 we report the average coefficients obtained for the two approaches on the 10 products under test.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DML | elasticity | -2.44 | -4.71 | -4.55 | -5.32 | -3.46 | -4.95 | -1.98 | -1.77 | -5.40 | -2.95 |
| | std. err. | 0.95 | 1.36 | 1.43 | 1.44 | 1.04 | 1.06 | 1.38 | 0.84 | 1.45 | 1.54 |
| DD | elasticity | 0.35 | -7.15 | -5.31 | -6.47 | -2.52 | -4.34 | -3.76 | -3.16 | -3.54 | -2.16 |
| | std. err. | 1.66 | 1.28 | 1.64 | 1.98 | 1.09 | 1.28 | 1.48 | 1.90 | 2.07 | 2.52 |

Table 6.1: Average elasticities estimated with Double Machine Learning and Difference-in-Differences.

Both methods can be valid alternatives to calculate the price elasticity of demand, however each one has advantages and disadvantages. The main difference between the two is that DML can be applied in any general scenario, while the DD approach with TWFE regression requires panel data, i.e. multi-dimensional data involving measurements over time for different entities. The DML is a very good model when there is the possibility to measure the covariates which can influence price and demand because, being based on ML techniques, it can learn non-liner dependencies with the dependent variables. However, when some confounding variables cannot be observed its estimates become less reliable since its performances are very correlated with the ability of the first stage estimators to incorporate the dependency on these variables. Alternatively, the DD approach with TWFE regression, can mitigate the absence of some covariates through the time and entity fixed effects but it has the limitation of assuming a liner relationship between the

demand and the covariates not absorbed by the fixed effects.

The DD approach requires some assumption to be respected in the environment from with the data are gathered. In the case of elasticity estimations, most of the assumptions can be approximately considered valid by comparing data in short periods and coming from similar shops operating within similar contexts, but any specific situation have to be analyzed. For example, comparing sales collected from *Amazon* with those collected from a small e-commerce would be wrong because the assumptions, and in particular the parallel trend, are surely violated even in short time periods.

Talking about the amount of data required by the two techniques, we can say that, obviously, the more are the available data the more their estimations are reliable. However, DML requires to train ML models that requires a quantity of samples that increases based on the number of training features. On the other hand, TWFE can estimate the elasticity with just a period of observations in which we observe a price change, but a good estimation is obtained by averaging on more periods and with many entities involved.

In terms of computational efforts the two approaches are different. DML can be very computational expensive based on the amount of data available, on number of folds in which the data are split during cross-fitting and on the ML algorithm used as first stage estimator. Using a simple linear regression might be very fast while training a complex ensemble like Random Forest might require a significant amount of time. Furthermore, the correct choice of the hyperparameters for the ML first stage estimators is crucial to reach a good final result, thus a preliminary tuning might be needed. TWFE, instead, being based on a linear regression has a computational complexity that is only based on the amount of data used for training, does not have any hyperparameter to tune and it is usually very light compared to DML. It may require a preliminary effort because it is necessary to extract from the data valid periods on which applying the regression.

Both the approaches works under the assumption of constant elasticity and they estimate a unique coefficient based on the input data. Through DD, however, by calculating an elasticity coefficient in different periods we can have information on how it varies along time. If the amount of data allows it, also DML can be applied separately to multiple time intervals to obtain the same result.

To conclude, DML is a better method in the general case since it can approximate also non-linear dependencies on the confounding variables and works without requiring particular assumptions on the data. However, DD can be a valid alternative in the case of unsatisfactory covariates and if it is possible to rely on panel data gathered from different environments.

# 7 | Conclusions and future developments

In this chapter, we provide a summary of the results of our approaches, applied on the real-world data provided by Coveo. Then, we suggest some topics for future research, which could expand and bring improvements to the work done so far.

## 7.1. Summary of the results

In this work, we focused on analyzing and adapting two causal inference algorithms at the purpose of estimating the price elasticity of several products from a real-world dataset containing the data of two shops operating in the same market.

The preliminary goal of this thesis was finding data instances across the two shops which refer to the same real-world product. We accurately described the process we used to tune and train at our purpose a pre-existing Deep Learning architecture (Deepmatcher) and provided the graph based approach we studied in order to cluster products data based on the predictions of the network. In the process we focused on generating only truthful cluster at the expense of the total number of clusters generated. However, we managed to obtain a significant amount of groupings that, after a careful handmade analysis, revealed to be reliable. From the analysis of sales and price changes of these product groups we extracted the 10 candidates on which we focused in the rest of the work. We managed to extract the groups containing products from both shops with the highest number of purchases and price changes, balanced across the two stores.

In the continuation of the work we studied and presented two causal inference approaches that we applied at the purpose of estimating the price elasticity of demand. For each of the two, we in-depth described how it can be applied in the specific case of elasticity estimation and presented the application process we developed in our scenario. The two approaches, on average, provided reliable elasticity estimates for the 10 product under analysis. The average estimated values are in a range that goes from $-1.77$ to $-7.15$

(with only an unrealistic exception of 0.36) and they can be considered reasonable under the assumption of elastic goods. For the kind of products under consideration, i.e. sport goods, we expect the sales to increase in a more than proportional way when price drops and vice versa; and this should reflect in an elasticity lower than $-1$.

The reliability of our results can be questioned mainly for two reasons, both related to the data at our disposal. The first one is about the lack of some confounding variables that can influence the demand and of which we are not aware, the most relevant being advertising campaigns conducted on channels external to the shops. The second factor is about the number of price changes of our products, which are limited even if, in our experiments, we selected a restricted set of products with a number of price variations above the average of the two shops. To obtain more accurate estimates it is necessary to adopt exploratory policies which slightly variate prices in time or for each single customer in order to acquire observations for a larger set of prices.

During the application of the two approaches we tested various setups and parameters. The Double Machine Learning have been tested with different data aggregation in term of time and shops and different first stage estimators and estimation procedures (*DML1* and *DML2*). The Difference-in-Differences approach have been tested with different covariates and different comparison strategies between the treated and control groups. For these different setups we provided the results and tried to analyze the factors influencing the possible divergences between them.

Finally, we compared the two approaches in term of results obtained and provide some insights on when one may be better that the other. Double Machine Learning can be used in any general scenario and support non-linear dependencies but its more computational expensive and requires to have high quality confounding variables in order to train good first stage estimators. Difference-in-Difference is less computational expensive and can absorb some confounding effects through time and entity differences but it is limited to linear relationships and requires panel data to be applied.

## 7.2. Future works

Future works related to this thesis should be aimed at testing the real reliability of the two approaches presented. This can be done by experimenting the estimation of the elasticity through randomized experiments in a "live" environment. By slightly varying the price proposed to customers in short time period it should be possible to have a series of different but close price-demand observations from which estimate an unbiased value of elasticity. However, this requires the shops to implement an ad-hoc pricing strategy

that could be hard to apply in practice.

Additionally, another way in which the reliability of these approaches can be proved is the analysis of their stability. This can only be done by estimating elasticity values in different years in order to check if the estimates are relatively stable. However, this requires to be tested on products for which the elasticity is little influenced over the years. An example of these goods can be some famous shoes models like the *Nike Air Force 1*, which are not seasonal but are being constantly sold by decades. This was not feasible in our case since the products considered have been sold for at most one year and overall the time span of our dataset is limited.

Another aspect that future works could deal with is the assumption of constant elasticity. In our thesis we based the estimates on the assumption that the elasticity is constant in time and between the range of prices at which the products are sold. This is obviously an approximation because in real world, the elasticity of a good changes especially in relation of time periods. During sales periods, for example, not only the overall demand is influenced but also the willingness of single customers of buying, i.e. the elasticity. Future works can focus on analyzing approaches which can deal with variable elasticity or studying ways of applying the methods presented in order to estimate different elasticities in time.

Finally, it would be interesting to see these approaches applied on shops having higher volume of sales and charging the customers with more dynamic prices. We think that these, together with the lack of some important confounders such as advertising period, are the main factors which can question the reliability of the elasticity values obtained. However, another multi-shop real dataset like the one provided to us by Coveo would be hard to obtain; especially if one is looking for bigger shops.

# Bibliography

[1] E. Italia, A. Nuara, F. Trovo, M. Restelli, N. Gatti, and E. Dellavalle, "Internet advertising for non-stationary environments," in *International Workshop on Agent-Mediated Electronic Commerce*, pp. 1–15, 2017.

[2] G. Romano, G. Tartaglia, A. Marchesi, and N. Gatti, "Online posted pricing with unknown time-discounted valuations," in *AAAI*, pp. 5682–5689, 2021.

[3] M. Castiglioni, G. Romano, A. Marchesi, and N. Gatti, "Signaling in posted price auctions," in *AAAI*, 2022.

[4] M. Castiglioni, D. Ferraioli, N. Gatti, A. Marchesi, and G. Romano, "Efficiency of ad auctions with price displaying," in *AAAI*, 2022.

[5] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 19–34, 2018.

[6] V. Chernozhukov, D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, and J. Robins, "Double/debiased machine learning for treatment and structural parameters," 2018.

[7] W. Hildenbrand, "On the" law of demand"," *Econometrica: Journal of the Econometric Society*, pp. 997–1019, 1983.

[8] W. Härdle, W. Hildenbrand, and M. Jerison, "Empirical evidence on the law of demand," *Econometrica: Journal of the Econometric Society*, pp. 1525–1549, 1991.

[9] L. S. Bagwell and B. D. Bernheim, "Veblen effects in a theory of conspicuous consumption," *The American economic review*, pp. 349–373, 1996.

[10] P. Cohen, R. Hahn, J. Hall, S. Levitt, and R. Metcalfe, "Using big data to estimate consumer surplus: The case of uber," tech. rep., National Bureau of Economic Research, 2016.

[11] G. Abrate, G. Fraquelli, and G. Viglia, "Dynamic pricing strategies: Evidence from european hotels," *International Journal of Hospitality Management*, vol. 31, no. 1, pp. 160–168, 2012.

[12] R. P. McAfee and V. Te Velde, "Dynamic pricing in the airline industry," *Handbook on economics and information systems*, vol. 1, pp. 527–67, 2006.

[13] S. Kedia, S. Jain, and A. Sharma, "Price optimization in fashion e-commerce," *arXiv preprint arXiv:2007.05216*, 2020.

[14] J. Liu, Y. Zhang, X. Wang, Y. Deng, X. Wu, and M. Xie, "Dynamic pricing on e-commerce platform with deep reinforcement learning," 2018.

[15] R. M. Weiss and A. K. Mehrotra, "Online dynamic pricing: Efficiency, equity and the future of e-commerce," *Va. JL & Tech.*, vol. 6, p. 1, 2001.

[16] O. Besbes and A. Zeevi, "Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms," *Operations Research*, vol. 57, no. 6, pp. 1407–1420, 2009.

[17] N. B. Keskin and A. Zeevi, "Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies," *Operations research*, vol. 62, no. 5, pp. 1142–1167, 2014.

[18] O. Besbes and A. Zeevi, "On the (surprising) sufficiency of linear models for dynamic pricing with demand learning," *Management Science*, vol. 61, no. 4, pp. 723–739, 2015.

[19] N. Ayadi, C. Paraschiv, and X. Rousset, "Online dynamic pricing and consumer-perceived ethicality: Synthesis and future research," *Recherche et Applications en Marketing (English Edition)*, vol. 32, no. 3, pp. 49–70, 2017.

[20] M. Rothschild, "A two-armed bandit theory of market pricing," *Journal of Economic Theory*, vol. 9, no. 2, pp. 185–202, 1974.

[21] F. Trovo, S. Paladino, M. Restelli, and N. Gatti, "Multi-armed bandit for pricing," in *12th European Workshop on Reinforcement Learning*, pp. 1–9, 2015.

[22] F. Trovò, S. Paladino, M. Restelli, and N. Gatti, "Improving multi-armed bandit algorithms in online pricing settings," *Int. J. Approx. Reason.*, vol. 98, pp. 196–235, 2018.

[23] S. Paladino, F. Trovò, M. Restelli, and N. Gatti, "Unimodal thompson sampling for graph-structured arms," in *AAAI*, pp. 2457–2463, 2017.

[24] J. W. Mueller, V. Syrgkanis, and M. Taddy, "Low-rank bandit methods for high-dimensional dynamic pricing," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[25] M. Mussi, G. Genalti, F. Trovó, Francesco, A. Nuara, N. Gatti, and M. Restelli, "Pricing the long tail by explainable product aggregation and monotonic bandits," in *SIGKDD*, 2022.

[26] F. Trovò, M. Restelli, and N. Gatti, "Sliding-window thompson sampling for non-stationary settings," *J. Artif. Intell. Res.*, vol. 68, pp. 311–364, 2020.

[27] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[28] Y. Cheng, "Real time demand learning-based q-learning approach for dynamic pricing in e-retailing setting," in *2009 International Symposium on Information Engineering and Electronic Commerce*, pp. 594–598, IEEE, 2009.

[29] A. V. Den Boer, "Dynamic pricing and learning: historical origins, current research, and new directions," *Surveys in operations research and management science*, vol. 20, no. 1, pp. 1–18, 2015.

[30] A. Faruqui and S. S. George, "The value of dynamic pricing in mass markets," *The Electricity Journal*, vol. 15, no. 6, pp. 45–55, 2002.

[31] D. Aparicio, Z. Metzman, and R. Rigobon, "The pricing strategies of online grocery retailers," tech. rep., National Bureau of Economic Research, 2021.

[32] M. Fisher, S. Gallino, and J. Li, "Competition-based dynamic pricing in online retailing: A methodology validated with field experiments," *Management science*, vol. 64, no. 6, pp. 2496–2514, 2018.

[33] B. Callaway, A. Goodman-Bacon, and P. H. Sant'Anna, "Difference-in-differences with a continuous treatment," *arXiv preprint arXiv:2107.02637*, 2021.

[34] J. Tagliabue, C. Greco, J.-F. Roy, B. Yu, P. J. Chia, F. Bianchi, and G. Cassani, "Sigir 2021 e-commerce workshop data challenge," *arXiv preprint arXiv:2104.09423*, 2021.

[35] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[36] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vec-

tors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[37] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[38] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[39] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *International conference on learning representations*, 2017.

[40] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[41] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[43] J. Large, J. Lines, and A. Bagnall, "A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates," *Data mining and knowledge discovery*, vol. 33, no. 6, pp. 1674–1709, 2019.

[44] E. W. Weisstein, "Bipartite graph," *https://mathworld.wolfram.com/*, 2002.

[45] E. W. Weisstein, "Complete graph," *https://mathworld.wolfram.com/*, 2001.

[46] E. W. Weisstein, "Disconnected graph," *https://mathworld.wolfram.com/*, 2002.

[47] R. Frisch and F. V. Waugh, "Partial time regressions as compared with individual trends," *Econometrica: Journal of the Econometric Society*, pp. 387–401, 1933.

[48] M. C. Lovell, "Seasonal adjustment of economic time series and multiple regression analysis," *Journal of the American Statistical Association*, vol. 58, no. 304, pp. 993–1010, 1963.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[50] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[51] 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. 30, 2017.

[52] P. Bach, V. Chernozhukov, M. S. Kurz, and M. Spindler, "Doubleml–an object-oriented implementation of double machine learning in r," *arXiv preprint arXiv:2103.09603*, 2021.

[53] J. Snow, *On the mode of communication of cholera*. John Churchill, 1855.

[54] A. Goodman-Bacon, "Difference-in-differences with variation in treatment timing," *Journal of Econometrics*, vol. 225, no. 2, pp. 254–277, 2021.

[55] L. A. Barros, D. R. Bergmann, F. H. Castro, and A. D. M. d. Silveira, "Endogeneity in panel data regressions: methodological guidance for corporate finance researchers," *Revista brasileira de gestão de negócios*, vol. 22, pp. 437–461, 2020.

# Appendix

## Tools

The tools used for the practical implementation of the product matching and the causal inference algorithms are the following:

- **Snowflake**. *Snowflake* is the data warehouse on which the data were hosted. We queried it through its Python library during the data extraction process of our experiments.
- **Metaflow**. *Metaflow* is the data science framework developed by *Netflix*, we used it to build the three main pipelines of our work, for product matching, DML and DD.
- **AWS Batch**. *AWS Batch* is the AWS service used to run batch computing workloads. Coveo gave us the opportunity to use their AWS Batch GPUs in order to train Deepmatcher models.
- **Comet.ml**. *Comet.ml* is a Machine Learning experimentation platform which we used to track and compare our Deepmatcher training experiments.
- **Deepmatcher package**. We relied on the Python package built using *PyTorch* by the creators of the Deepmatcher architecture to provide a customizable implementation of the network. We extended it in order to introduce a different evaluation metric (precision) and to allow it to run on AWS Batch.
- **Pandas**. *Pandas* is the python library we used to load raw data and generate our datasets.
- **scikit-learn**, **lightgbm** and **linearmodels**. These are the libraries from which we took the implementation of the Machine Learning models used. For the DML approach we used *sklearn.linear_model.Lasso*, *sklearn.ensemble.RandomForestRegressor*, *sklearn.ensemble.GradientBoostingRegressor* and *lightgbm.LGBMRegressor* while for the DD approach we used *linearmodels.panel.model.PanelOLS*.

## Deepmatcher models

The following table reports the hyperparameters and the precision scores of the top 10 models obtained by training Deepmatcher networks. These models are the ones used for the final embedding.

| **Batch size** | **Summarizer** | **Comparator** | **Condense factor** | **Classifier layers** | **Hidden size** | **Precision** |
|---|---|---|---|---|---|---|
| 32 | Hybrid | Mult | 1 | 2 | 512 | 0.85135 |
| 16 | Hybrid | Mult | 1 | 3 | 256 | 0.84821 |
| 32 | SIF | Abs diff | 1 | 3 | 512 | 0.84615 |
| 32 | Hybrid | Concat with abs diff | 3 | 2 | 512 | 0.81105 |
| 32 | Attention | Concat with abs diff | 3 | 2 | 256 | 0.79661 |
| 32 | SIF | Abs diff | 1 | 2 | 512 | 0.78604 |
| 32 | Hybrid | Mult | 2 | 3 | 512 | 0.78389 |
| 16 | SIF | Abs diff | 1 | 3 | 512 | 0.77192 |
| 32 | Hybrid | Mult | 1 | 2 | 256 | 0.76543 |
| 16 | SIF | Abs diff | 1 | 2 | 512 | 0.75527 |

Table A1: Deepmatcher models.

# Products

The following table reports descriptive data about the 10 product groups considered in our experiments. The data are aggregated between all the SKUs belonging to each group. In the table $Q$ represent the quantity demanded and $P$ represent the price in euros.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Shop A | | | | | | | | | | | |
| Q | mean | 0.89 | 1.89 | 1.47 | 4.09 | 1.34 | 4.96 | 1.18 | 0.75 | 2.41 | 2.92 |
| | std | 1.93 | 3.57 | 2.65 | 7.21 | 2.43 | 8.22 | 1.79 | 1.45 | 4.19 | 5.31 |
| | min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | max | 19 | 52 | 30 | 65 | 22 | 74 | 15 | 11 | 71 | 50 |
| P | mean | 91.20 | 101.96 | 102.07 | 101.70 | 119.86 | 125.01 | 78.92 | 87.91 | 101.89 | 120.9 |
| | std | 10.82 | 13.02 | 12.18 | 13.07 | 23.79 | 20.23 | 9.88 | 12.65 | 9.75 | 17.33 |
| | min | 76.00 | 69.95 | 79.95 | 79.95 | 85.00 | 85.00 | 55.00 | 59.95 | 84.00 | 99.95 |
| | max | 111.00 | 119.00 | 119.00 | 119.00 | 170.00 | 170.00 | 88.00 | 105.00 | 112.00 | 144.00 |
| Samples | | 906 | 1383 | 1890 | 2278 | 1480 | 1707 | 834 | 883 | 1920 | 2410 |
| Shop B | | | | | | | | | | | |
| Q | mean | 0.23 | 1.69 | 0.77 | 2.09 | 0.45 | 1.50 | 0.76 | 0.21 | 0.80 | 1.22 |
| | std | 0.92 | 3.46 | 1.51 | 3.14 | 1.08 | 3.41 | 1.68 | 0.61 | 1.69 | 2.72 |
| | min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | max | 24 | 26 | 12 | 28 | 12 | 33 | 18 | 5 | 16 | 26 |
| P | mean | 89.12 | 104.63 | 108.10 | 111.34 | 134.63 | 139.96 | 76.8 | 87.79 | 106.50 | 127.87 |
| | std | 16.30 | 17.32 | 14.95 | 12.47 | 22.14 | 20.38 | 10.38 | 13.37 | 15.15 | 17.82 |
| | min | 59.00 | 79.00 | 79.00 | 79.00 | 99.00 | 99.00 | 65.00 | 74.00 | 79.00 | 99.00 |
| | max | 105.00 | 133.00 | 133.00 | 133.00 | 170.00 | 170.00 | 89.00 | 114.00 | 119.00 | 171.00 |
| Samples | | 1804 | 435 | 504 | 514 | 1059 | 890 | 967 | 1301 | 1140 | 906 |

Table A2: Descriptive data about products considered in the experiments.

# Double Machine Learning results

The following tables report the results of the Double Machine Learning approach with weekly and daily datasets and with and without shop aggregation, using the four first stage estimators presented.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Shop A | | | | | | | | | | | |
| Lasso | elasticity | -1.03 | -1.82 | -2.09 | -3.53 | -0.10 | -3.65 | -0.45 | -2.27 | -3.27 | -2.75 |
| | std. err. | 0.51 | 0.62 | 0.51 | 0.48 | 0.54 | 0.64 | 0.65 | 0.68 | 0.45 | 0.63 |
| RF | elasticity | -1.78 | -3.16 | -5.13 | -9.72 | -3.39 | -6.13 | -0.47 | -5.05 | -9.23 | -3.51 |
| | std. err. | 1.15 | 1.33 | 1.14 | 1.24 | 1.11 | 1.11 | 1.36 | 1.00 | 1.23 | 1.33 |
| GBR | elasticity | -1.70 | -0.57 | -4.60 | -9.92 | -4.16 | -5.46 | -0.42 | -4.97 | -7.94 | -3.56 |
| | std. err. | 1.25 | 1.46 | 1.22 | 1.29 | 1.17 | 1.11 | 1.35 | 1.02 | 1.36 | 1.43 |
| LGBM | elasticity | -2.64 | -1.60 | -5.56 | -9.00 | -3.96 | -6.17 | 0.54 | -5.03 | -7.00 | -2.76 |
| | std. err. | 1.26 | 1.23 | 1.14 | 1.17 | 1.24 | 0.99 | 1.30 | 1.01 | 1.34 | 1.44 |
| Shop B | | | | | | | | | | | |
| Lasso | elasticity | -0.83 | -3.77 | -2.71 | -2.80 | -1.45 | -2.91 | -1.36 | -0.64 | -2.87 | -3.28 |
| | std. err. | 0.25 | 0.64 | 0.49 | 0.53 | 0.27 | 0.30 | 0.59 | 0.28 | 0.45 | 0.50 |
| RF | elasticity | -1.08 | -2.69 | -3.01 | -2.73 | -1.31 | -2.54 | -1.07 | -0.69 | -2.06 | -1.49 |
| | std. err. | 0.44 | 0.75 | 0.71 | 0.88 | 0.43 | 0.44 | 0.92 | 0.31 | 0.77 | 0.65 |
| GBR | elasticity | -0.96 | -2.52 | -2.42 | -2.73 | -1.16 | -2.32 | -1.69 | -0.83 | -1.21 | -1.47 |
| | std. err. | 0.46 | 0.75 | 0.73 | 0.98 | 0.49 | 0.52 | 1.00 | 0.32 | 0.79 | 0.69 |
| LGBM | elasticity | -0.93 | -2.65 | -2.19 | -3.25 | -1.12 | -2.60 | -1.72 | -0.62 | -2.52 | -0.79 |
| | std. err. | 0.45 | 0.77 | 0.74 | 0.91 | 0.42 | 0.51 | 0.92 | 0.32 | 0.77 | 0.52 |

Table A3: Double Machine Learning results with weekly dataset without shop aggregation.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Lasso | elasticity | -1.00 | -2.56 | -3.06 | -3.41 | -0.87 | -2.94 | -0.75 | -1.08 | -3.36 | -3.06 |
| | std. err. | 0.24 | 0.45 | 0.37 | 0.36 | 0.27 | 0.30 | 0.45 | 0.29 | 0.34 | 0.41 |
| RF | elasticity | -0.84 | -2.76 | -4.61 | -4.57 | -1.74 | -3.65 | -1.8 | -1.32 | -3.97 | -2.22 |
| | std. err. | 0.36 | 0.71 | 0.63 | 0.59 | 0.39 | 0.38 | 0.83 | 0.34 | 0.60 | 0.57 |
| GB | elasticity | -0.88 | -2.80 | -2.45 | -3.77 | -1.34 | -3.56 | -1.14 | -1.47 | -2.84 | -2.23 |
| | std. err. | 0.39 | 0.69 | 0.59 | 0.62 | 0.41 | 0.45 | 0.85 | 0.34 | 0.64 | 0.60 |
| LGBM | elasticity | -0.97 | -3.02 | -2.62 | -3.95 | -1.63 | -3.82 | -0.47 | -1.37 | -2.94 | -2.28 |
| | std. err. | 0.37 | 0.63 | 0.62 | 0.57 | 0.39 | 0.42 | 0.79 | 0.33 | 0.61 | 0.59 |

Table A4: Double Machine Learning results with daily dataset and aggregated shops.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Shop A | | | | | | |
| Lasso | elasticity | -2.21 | -4.00 | -3.25 | -3.90 | -5.16 | -7.78 | -1.62 | -1.24 | -2.97 | -5.62 |
| | std. err. | 1.35 | 1.40 | 1.40 | 1.41 | 1.43 | 1.72 | 1.59 | 1.57 | 1.35 | 1.74 |
| RF | elasticity | -3.37 | -11.52 | -4.37 | -7.36 | -8.83 | -5.37 | -1.14 | -3.84 | -7.82 | -10.23 |
| | std. err. | 2.29 | 2.54 | 3.40 | 2.86 | 2.31 | 2.05 | 2.66 | 1.55 | 2.91 | 2.53 |
| GB | elasticity | -5.62 | -10.16 | -8.57 | -8.66 | -10.02 | -4.63 | -1.76 | -3.96 | -13.12 | -11.35 |
| | std. err. | 2.38 | 2.74 | 3.04 | 2.77 | 2.37 | 2.00 | 3.03 | 1.67 | 3.29 | 3.06 |
| LGBM | elasticity | -3.64 | -2.50 | -6.61 | -8.70 | -7.56 | -1.74 | -1.15 | -3.32 | -10.43 | -6.99 |
| | std. err. | 1.98 | 1.80 | 2.42 | 3.04 | 2.43 | 1.80 | 2.11 | 1.53 | 3.15 | 2.90 |
| | | | | | Shop B | | | | | | |
| Lasso | elasticity | -3.09 | -8.27 | -5.88 | -4.26 | -4.43 | -7.13 | -3.36 | -0.15 | -5.50 | -4.04 |
| | std. err. | 0.71 | 1.68 | 1.59 | 1.67 | 0.93 | 1.16 | 1.45 | 0.97 | 1.12 | 1.70 |
| RF | elasticity | -3.39 | -7.01 | -4.51 | -3.93 | -2.36 | -8.21 | -4.74 | -0.23 | -3.97 | 3.18 |
| | std. err. | 1.19 | 2.23 | 2.25 | 2.34 | 1.33 | 1.66 | 1.93 | 1.05 | 2.25 | 2.72 |
| GB | elasticity | -3.59 | -4.86 | -5.58 | -3.84 | -2.78 | -5.50 | -4.94 | 0.28 | -5.90 | 3.66 |
| | std. err. | 1.15 | 2.08 | 2.70 | 2.36 | 1.40 | 1.78 | 1.88 | 1.02 | 2.64 | 2.98 |
| LGBM | elasticity | -2.92 | -6.67 | -4.49 | -3.51 | -1.70 | -7.57 | -5.28 | -0.38 | -5.24 | 4.19 |
| | std. err. | 1.27 | 2.25 | 2.52 | 2.21 | 1.21 | 1.61 | 1.74 | 1.03 | 2.80 | 2.67 |

Table A5: Double Machine Learning results with weekly dataset without shop aggregation.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Lasso | elasticity | -2.91 | -5.33 | -5.52 | -5.24 | -4.54 | -7.27 | -2.01 | 0.09 | -5.91 | -4.85 |
| | std. err. | 0.63 | 1.17 | 1.08 | 1.12 | 0.82 | 1.0 | 1.15 | 0.84 | 0.93 | 1.25 |
| RF | elasticity | -4.49 | -8.88 | -8.0 | -7.1 | -5.07 | -5.18 | -4.05 | -1.36 | -5.42 | -4.13 |
| | std. err. | 0.89 | 1.7 | 1.67 | 1.71 | 1.23 | 1.18 | 1.45 | 0.87 | 1.53 | 1.91 |
| GB | elasticity | -4.34 | -7.19 | -6.71 | -6.09 | -4.39 | -7.56 | -3.78 | -1.72 | -7.9 | -4.5 |
| | std. err. | 0.99 | 1.6 | 1.84 | 1.75 | 1.33 | 1.33 | 1.58 | 0.92 | 1.86 | 2.13 |
| LGBM | elasticity | -4.34 | -6.81 | -5.33 | -5.78 | -3.9 | -5.01 | -2.77 | -1.31 | -6.21 | -0.8 |
| | std. err. | 0.94 | 1.52 | 1.59 | 1.65 | 1.14 | 1.07 | 1.57 | 0.83 | 1.69 | 2.05 |

Table A6: Double Machine Learning results with weekly dataset and aggregated shops.

## Difference-in-Differences results

The following tables report the results of the Difference-in-Differences two-way fixed effects approach trained with and without covariates. In the *Disagg 1* case we train a model for each period by using the data from all the SKUs while in the *Disagg 2* case we train a model for each period and for each combination of treated-untreated SKUs. The *Agg* case is the case in which the data of all the SKUs are aggregated into a single series.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Disagg 1 | elasticity | 0.51 | -6.63 | -5.39 | -7.01 | -3.75 | -6.29 | -5.00 | -2.58 | -3.07 | -2.58 |
| | std. err. | 0.50 | 0.78 | 0.74 | 1.22 | 0.57 | 0.76 | 0.85 | 1.05 | 1.13 | 1.27 |
| Disagg 2 | elasticity | 0.37 | -6.35 | -5.71 | -7.01 | -3.29 | -5.43 | -4.76 | -2.93 | -3.60 | -2.20 |
| | std. err. | 0.86 | 0.98 | 1.09 | 1.44 | 0.83 | 1.04 | 0.94 | 1.52 | 1.68 | 1.65 |
| Agg | elasticity | 0.44 | -8.01 | -4.96 | -7.99 | -0.93 | -1.67 | -0.73 | -3.56 | -2.74 | -3.4 |
| | std. err. | 1.07 | 0.97 | 1.37 | 1.76 | 1.00 | 1.11 | 1.00 | 1.54 | 1.39 | 1.85 |

Table A7: Difference-in-Differences results without covariates.

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Disagg 1 | elasticity | 0.84 | -6.50 | -5.46 | -6.19 | -3.23 | -5.36 | -4.5 | -1.93 | -4.73 | -4.53 |
| | std. err. | 1.09 | 1.24 | 1.42 | 1.31 | 0.73 | 0.94 | 1.31 | 1.85 | 1.33 | 1.67 |
| Disagg 2 | elasticity | -0.77 | -6.12 | -7.58 | -6.18 | -3.38 | -5.76 | -4.92 | -2.63 | -4.09 | -2.39 |
| | std. err. | 1.60 | 1.67 | 2.15 | 2.65 | 1.47 | 1.55 | 1.69 | 3.00 | 2.72 | 3.64 |
| Agg | elasticity | 0.70 | -9.27 | -2.78 | -4.42 | -0.55 | -1.52 | -2.68 | -5.33 | -3.03 | 2.14 |
| | std. err. | 4.84 | 2.02 | 3.04 | 3.47 | 1.94 | 2.30 | 3.10 | 2.43 | 4.14 | 5.06 |

Table A8: Difference-in-Differences results with covariates.

# List of Figures

# List of Tables