A Practical Implementation of Machine Learning Methods for Price Imputation

Lucien MAY [1], Claude LAMBORAY[2], Botir RADJABOV[3], Yu-Lin HUANG[4]

Version 02/05/2024

1 Introduction

Price indices rely on comparing prices of the same products in two comparison periods. However, with dynamic product assortments, a matched model approach that only takes into account products available in the two comparison periods may not be fully satisfactory. In such a case, the hedonic imputation approach can be a possible alternative. The idea is to impute a price for the products that are only available in one of the two comparison periods. An imputed price is typically obtained by estimating a semi-logarithmic hedonic multiple regression in order to explain the relationship between the observed prices and a set of price determining characteristics. In this paper, we are going to explore the use of Machine Learning (ML) for imputing such missing prices in the context of multilateral price index methods.

ML methods can be seen as an alternative to a hedonic regression as both approaches eventually allow estimating a missing price. The main objective of ML methods is to find the best possible prediction for a price that can be learned from the data. While ML methods are nowadays considered in many fields, the application of the ML paradigm in the context of price statistics still raises a number of practical and conceptual questions. On the one hand, there could be a case for considering ML methods given the increasing availability of larger data sets such as scanner data or web scraped data. The ML models are often more flexible than a linear regression. One could argue that a more accurate price imputation will lead to a more accurate price index, and that therefore, predictive accuracy could be preferred over model interpretability. On the other hand, it can remain unclear how to best apply ML methods in the specific context of price indices.

There are already some examples of using ML methods for price statistics. Random forests (Zeng, 2021) or neural networks (Cafarella et al., 2023) have been used to obtain price indices from retail scanner data. Attempts have also been made to apply ML methods on products that undergo rapid technological change, such as laptops (Zafar and Himpens, 2021) or multi-purpose digital devices (Roche et al, 2022). ML can also be considered in the context of residential (Picchetti, 2017) or commercial (Calainho, 2021) property prices.

In section 2, we first present the methodological framework in which we will work. We discuss the methodology both from a price statistics and ML perspective. This methodological framework is then applied on two data sets in section 3. We use an internal web scraped data set on TVs and a public scanner data set on laptops[5](Diewert and Shimizu, 2023). In section 4, we

[1]STATEC, Price Statistics Unit, Contact: prix@statec.etat.lu
[2]STATEC, Head of Methodology
[3]STATEC/GOPA
[4]STATEC/University of Luxembourg
[5]We would like to thank Diewert and Shimizu for making this data set available and for giving us the opportunity to apply our approach on this data set.

propose an operational process that starts with monthly incoming prices data and that ends with a (multilateral) price index that could be published. Finally, our conclusions and some open issues are summarized in section 5.

2 Methodology

2.1 Price Statistics

We denote the price and the quantity of the item i in period t by p_i^t and q_i^t respectively. The expenditure for the item i in period t thus corresponds to $e_i^t = p_i^t q_i^t$. Let N_t be the set of items available in period t. The set of matched items that are available in the two comparison periods t_1 and t_2 is denoted by $M_{t_1,t_2} = N_{t_1} \cap N_{t_2}$. Moreover, let $N_{t_1,t_2} = N_{t_2} \setminus N_{t_1}$ be the set of items available in period t_2 but not in period t_1, and let $D_{t_1,t_2} = N_{t_1} \setminus N_{t_2}$ be the set of items available in period t_1 but not in period t_2.

As a starting point, we calculate a Törnqvist index on the matched items. This means that the aggregate price change is derived only from the set of items that are available in the two comparison periods.

$$I_T^{t_1,t_2} = \prod_{i \in M_{t_1,t_2}} \left(\frac{p_i^{t_2}}{p_i^{t_1}} \right)^{0.5*\left(\frac{e_i^{t_1}}{\sum_{j \in M_{t_1,t_2}} e_j^{t_1}} + \frac{e_i^{t_2}}{\sum_{j \in M_{t_1,t_2}} e_j^{t_2}} \right)} \tag{1}$$

One issue with the matched Törnqvist index is that items that are available in only one of the two comparison periods are ignored. This can lead to biased price indices if, for example, products leave the assortment on a reduced price. We may also miss price changes between essentially identically products that have been subject to a change in package size or weight (relaunches). In order to overcome this limitation, we could estimate a price for an item in the period in which it is not available. We denote an imputed price for item i in period t by $\hat{p}_i^t$. This leads us to the single imputation Törnqvist price index:

$$\begin{aligned} I_{IT}^{t_1,t_2} = & \prod_{i \in M_{t_1,t_2}} \left(\frac{p_i^{t_2}}{p_i^{t_1}} \right)^{0.5*\left(\frac{e_i^{t_1}}{\sum_{j \in N_{t_1}} e_j^{t_1}} + \frac{e_i^{t_2}}{\sum_{j \in N_t} e_j^{t_2}} \right)} \\ & \prod_{i \in D_{t_1,t_2}} \left(\frac{\hat{p}_i^{t_2}}{p_i^{t_1}} \right)^{0.5*\left(\frac{e_i^{t_1}}{\sum_{j \in N_{t_1}} e_j^{t_1}} \right)} \prod_{i \in N_{t_1,t_2}} \left(\frac{p_i^{t_2}}{\hat{p}_i^{t_1}} \right)^{0.5*\left(\frac{e_i^{t_2}}{\sum_{j \in N_{t_2}} e_j^{t_2}} \right)} \end{aligned} \tag{2}$$

In other words, our approach to dynamic item universe is to replace missing prices with imputed prices in order to obtain a full dataset for the two comparison periods. We then apply a standard price index method (in our case, the Törnqvist index) to this augmented dataset. De Haan (2011) investigated the link between the imputation Törnqvist index and the time dummy hedonic index.

Let us now suppose that these imputed prices are available. In practical scanner data applications we use multilateral, rather than bilateral index methods. In a multilateral method, the aggregate price change between two comparison periods is obtained from prices and quantities observed in multiple periods, not only in the two comparison periods. One example is the GEKS or CCDI (Caves et. al., 1982) price index, which uses the bilateral Törnqvist price indices, obtained between any two periods of a given time window T, as building blocks.

In general, the GEKS is based on the matched Törnqvist indices.

$$I^{t_1,t_2} = \prod_{k \in T} \left(I_T^{t_1,k} * I_T^{k,t_2} \right)^{\frac{1}{|T|}} \tag{3}$$

Alternatively, we could use the imputation Törnqvist indices as building blocks of the GEKS index (see Lamboray, 2022). We call this the imputation GEKS or imputation CCDI (de Haan and Daalmans, 2018).

$$I_I^{t_1,t_2} = \prod_{k \in T} \left(I_{IT}^{t_1,k} * I_{IT}^{k,t_2} \right)^{\frac{1}{|T|}} \tag{4}$$

With scanner data, all these indices can be calculated as both prices, quantities and characteristics are available for each item in each period. With web scraped data, weights are typically not available. In that case, we will assume that each item has the same expenditure[6]. Working with equally weighted expenditures leads to the matched Jevons index I_J and the imputation Jevons index I_{IJ}. We can then naturally extend these Jevons indices to a multilateral context, leading to the Jevons and Imputation Jevons variants of the GEKS index.

2.2 Machine Learning

In order to calculate an imputation GEKS index in practice, we need to impute missing prices. We will investigate the use of ML methods to obtain these prices. We work with log prices instead of prices as our target variable in order to be consistent with typical hedonic functions that are applied in price statistics context.

We now consider the following generic function f, which explains log price with help of a set of explanatory variables z.

$$\ln\left(p_i^t\right) = f\left(z_i, t\right) + \epsilon_i^t \quad \forall i \in N_t, \forall t \in [1, .., T] \tag{5}$$

Note that the model is not necessarily based on the economic approach to index number theory. Instead, we follow a data-driven approach, where our goal is to estimate a function f, which estimates log prices across periods as good as possible.

We estimate the model by pooling data from several periods (time window T). This strategy will allow to estimate models on bigger samples. Moreover, each item will have a unique imputed price for a given time window and a given month[7]. There are two types of time windows that must be fixed: the time window (and related rolling or expanding strategy) used in the ML method and the time window (and related rolling or expanding strategy) used in the price index. The time window used for training the models can be equal, or larger, than the time window that is used in the multilateral method. In our practical implementation, we will use identical rolling time windows consisting of 12 months for both ML methods and the multilateral method. The optimal window length from both ML and price statistics perspective deserves to be further investigated, but would go beyond the scope of this initial analysis.

[6]Let us suppose that we fix the expenditure $e = 1$ for each item. As a consequence, an implicit quantity variable will be estimated for each item as follows: $q_i^t = \frac{1}{p_i^t}$.

[7]In some approaches, the imputed price for an item depends on the choice of the two comparison periods (see de Haan and Krsinich (2014)). This means that over a time window, the item can have different imputed prices.

We would like to have a model that not only fits the initial data, but also allows making good predictions on unseen data. A very complex model may perfectly fit known data but will perform poorly on the unseen data. In order to evaluate the performance of a model, we randomly split the data into a training set and a test set. First, the model is estimated using the training set. The estimated model $\hat{f}_{TRAIN}$ is then used to make predictions on the test set. The accuracy of these predictions can be measured using Root Mean Squared Error (RMSE)[8].

$$\text{RMSE} = \sqrt{\sum_{(i,t)\in\text{TEST}} \frac{1}{n_{\text{TEST}}} \left(\ln(p_i^t) - \hat{f}_{\text{TRAIN}}(z_i, t)\right)^2} \tag{6}$$

RMSE is an important measure in ML. The expected test set RMSE (the average over different test sets) can be decomposed into three components (see James et al. (2013)): (i) a squared bias of the learner, (ii) the variance of the learner and (iii) an irreducible error. This is referred to as the bias-variance trade-off. In other words, the optimal (lowest) RMSE corresponds to a good compromise between bias and variance.

Our data consists of price observations of different items for different time periods. When making the random train-test split, we use time period as stratification variable. Suppose that we would like to randomly select 20% of observations for the test set and 80% of observations for the training set. We then require the 20-80 split to hold for each time period. This will make sure that each time period is appropriately represented[9].

We may want to repeat several times the split into a training and test sets. One common strategy is to apply k-fold Cross-Validation. This means that we will split the entire data into k sets of equal size. With $k = 3$, the data is split into 3 sets (S_1, S_2, S_3), each set containing one third of the initial data. First, we consider S_1 to be the test set and S_2-S_3 to be the training set. We then obtain a first RMSE_1. Next, we use S_2 to the test set and S_1 and S_3 to be the training set, leading to RMSE_2. Finally, we use S_3 to be the test set and S_1-S_2 to be the training set, leading to RMSE_3. An estimate for the RMSE is calculated as the average of the RMSE_k ($k = 1, .., 3$). This idea is illustrated in Figure 1. For datasets of our size, it is a common choice to use either 5-fold or 10-fold cross validation.

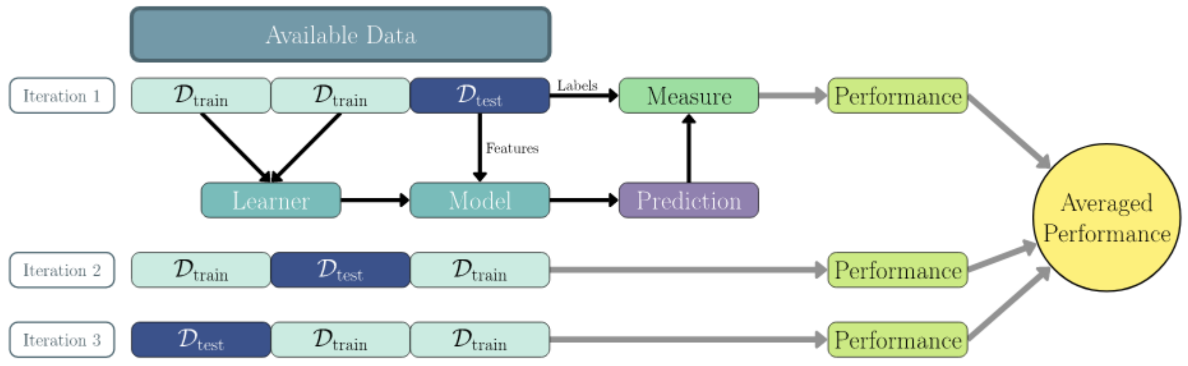
In our analysis, we first consider a standard log linear model as a benchmark model. Let us suppose that z_i is a k-dimensional vector of k price determining characteristics for item i and that d_i^t is a time dummy variable that equals 1, if item i belongs to period t, and 0 otherwise.

$$f_{LIN}\left(z_i, t\right) = \hat{\alpha} + \sum_k z_i^k \hat{\beta}^k + \sum_t d_i^t \hat{\gamma}^t + \epsilon_i^t \tag{7}$$

[8]Note that the exponential of RMSE can be linked to a price relative (observed price versus imputed price).

[9]We also experimented with block sampling to take product dimension into account. This means that each item is treated as a "block": all price observations of a given item are either in the training or in the test set. More research is needed to understand how time and product dimensions can be best incorporated in a re-sampling strategy.

Figure 1: Illustration of 3-fold Cross-Validation (Bischl et al., 2024)


Available Data
Iteration 1
$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{test}}$
Labels
Measure
Performance
Features
Learner
Model
Prediction
Averaged Performance
Iteration 2
$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
Performance
Iteration 3
$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{train}}$
Performance

This model assumes a linear relationship between log-prices and characteristics. From a price statistics perspective, this model is used in the hedonic time dummy index. Note that in our context, we do not use the exponential of estimated parameter γ^t to derive a price index. Instead, we use it to estimate missing prices that are needed for calculation of bilateral Törnqvist indices underlying the CCDI.

We compare the benchmark model f_{LIN} to two common ML methods: Random Forest (Ho 1995, Breiman 2001) and XGBoost (Chen and Guestrin 2016). Random Forest is based on a collection of decision trees. Each tree includes some randomness by selecting a sample of features and observations, in order to reduce the correlation between different trees. XGBoost approach does not grow multiple trees in parallel but grows multiple trees sequentially: the residual error of one tree is the input of the next tree. Compared to the log-linear model, these models tend to be more accurate. Specifically, they are more flexible by taking into account possible non-linearities in the data. Issues such as multicollinearity or heteroscedasticity are less of a problem for ML than for liner regressions. On the other hand, there is a risk that increased flexibility can lead to overfitting. There is a computational cost in implementing these methods as well. Finally, interpretability becomes much more difficult than with a log-linear model.

It is not straightforward how to incorporate time dimension in the models. Time is a critical variable for us as we eventually aim to make price comparisons across "time". In the log-linear model, time is coded through dummy variables, allowing to capture a fixed time effect. We could be tempted to use the same coding in XGBoost and Random Forest. However, the treatment of time dimension may not be straightforward. Consider a time dummy variable for a given period t. This variable splits the data into two parts: the observations from period t, and the observations from all other periods of the time window. The second set is much larger than the first set. Price observations from periods other than t may not be homogeneous. Hence, time dummy variable is less likely to be selected as a splitting variable in a node of a tree in Random Forest. Therefore, we prefer to code time in Random Forest as a single variable that takes values $1, 2, \ldots T$ (referred to as label encoding). In this way, we also capture the order[10] of the time periods (period 1 precedes period 2, which precedes period 3, etc.).

ML methods can also incorporate weights. This is an interesting aspect from a price statistics perspective as we would like to give more weight to observations that have higher economic importance. In a ML setting, the model performs "better" with respect to observations that have a

[10]The treatment of time could be further refined in case of seasonality in the data, by applying cyclical encoding that incorporates seasonality.

higher weight. It may be desirable that the price imputations for well-sold products are "better", because they have a higher weight in the price index. For example in a log-linear model, it is a common practice to use expenditure shares[11] as weights. For Random Forest and XGBoost, we prefer to work with expenditures instead of expenditure shares. In Random Forest, weights are used when sampling the units to be used in a given tree. In other words, price observations with a higher expenditure will appear more often in the trees underlying the Random Forest[12]. In XGBoost, weights have an impact on the way that an observation affects the loss function.

ML methods such as Random Forest or XGBoost[13] require fixing several hyperparameters (HPs). The trained model thus depends on these parameters, which have to be selected in advance. In other words, the final trained model depends on these HPs. In order to evaluate the performance of a specific instance of HPs, we can again use Cross-Validation.

There are different strategies to find the "optimal" HPs. Grid search means that we would try each possible combination of HPs. However, if the search space is large, this strategy can be very time consuming. An alternative would be to use random search where we randomly pick, for example, 10 instances which are then evaluated. There also exist more advanced methods. The idea behind Bayesian optimization (see Section 4.2.3 in Bischl et al. (2023)) is to estimate a relationship between HPs and performance using archive HPs and performance data points. This will help the algorithm to select a new set of HPs that is likely to perform well. Once the performance of these specific HPs has been evaluated, it will be added to the archive and the relationship between HPs and performance is re-estimated.

In practice, we use the following strategy in order to find the optimal HPs:

- In a first step, we apply random search or Bayesian optimization to explore the default search space (see Table 1).
- In a second step, we try to reduce the search space by setting the values of some HPs to a default value.
- In a third step, we then use (local) grid search for a limited number of HPs (for example only 1 or 2 HPs) in a reduced search space (around those values that were shown to lead to good results).

Table 1: Default List of Hyperparameters to be tuned in MLR3 Package

Method	**Hyperparameters**
Random Forest	mtry, num_trees, replace, sample_fraction
XGBoost	alpha, lambda, eta, nrounds, max_depth, subsample, colsample_bylevel, colsample_bytree

We are now ready to introduce the methodology of nested resampling (see Section 4.4 in Bischl et al. (2023) and Bischl et al. (2024)) which aims at correctly estimating accuracy of the method while at the same time optimizing HPs. We need to distinguish the tuning process from the

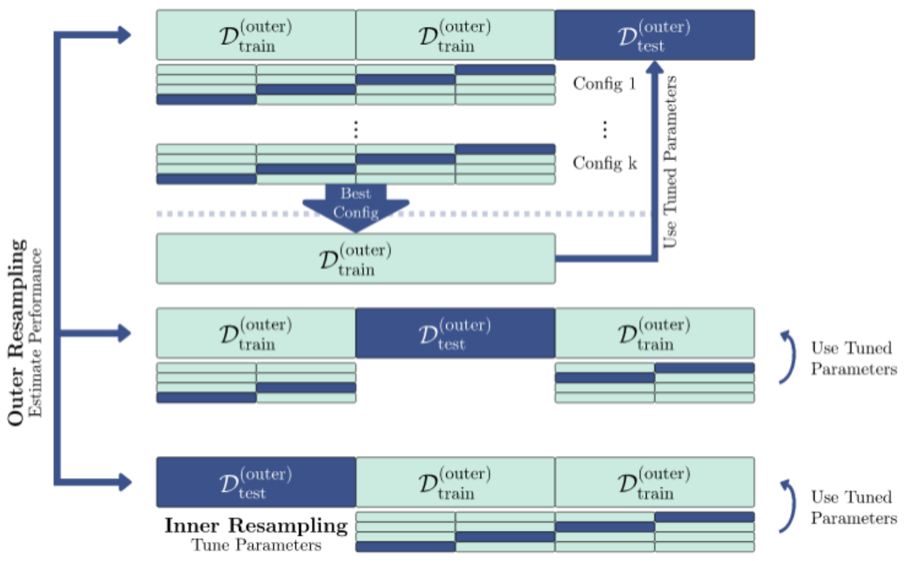
[11]Note that in ML paradigm, expenditure shares must be re-calculated for a given train-test split. In practice, this means specifying a "pipeline" that first calculates expenditure shares and then trains an expenditure share weighted log-linear model. This "pipeline" is then applied to a given training dataset.

[12]The sampling fraction is a hyperparameter of Random Forest. By setting this parameter close or equal to 1, the impact of weights vanishes.

[13]See also `https://xgboost.readthedocs.io/en/stable/parameter.html` for an overview of the parameters of the XGBoost method.

process used to evaluate performance of the model. Under this setup, there are two levels of resampling. The "outer" re-sampling is used to estimate the accuracy of a method. Each loop of the outer re-sampling consists of a specific train-test split. The model obtained from the (outer) training data is evaluated on the (outer) test data. We use several train-test splits in order to avoid the problem of one such split being too specific. For each loop (that is, for each train-test split of the outer training data set), we use a different set of HPs. These HPs are optimized with respect to the "outer" training data only. This means that the "outer" training dataset is again split into an "inner" training and an "inner" test set in order to evaluate a specific instance of HPs. This setup is illustrated in Figure 2, using 3-fold Cross-Validation for outer re-sampling and 4-fold Cross-Validation for inner re-sampling.

Figure 2: Illustration of Nested Resampling (Bischl et al., 2024)

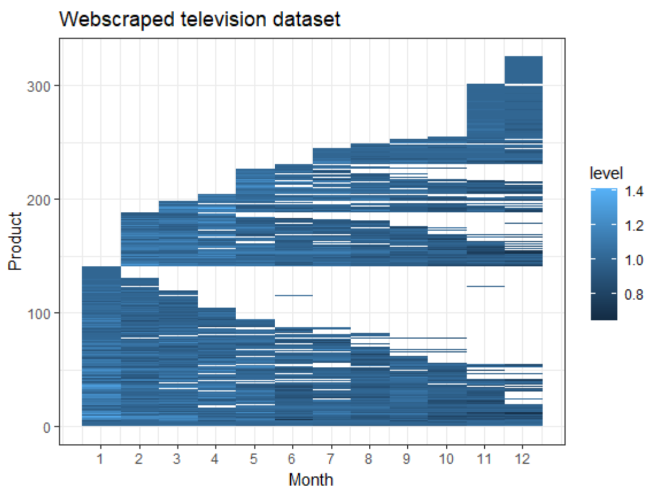


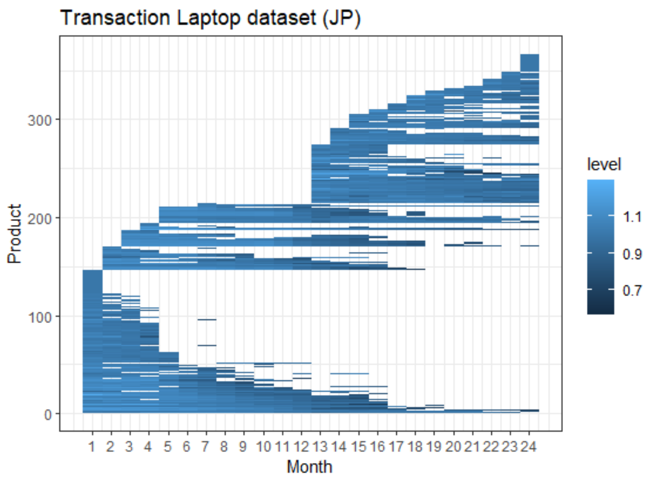
3 Results

ML models can be used for both web-scraped (unweighted) and transaction (weighted) data. We illustrate these methods and the resulting price indices for one internal web-scraped dataset on TVs and a publicly available Japanese transaction dataset on laptops (Diewert and Shimizu, 2023).

Both data sets are characterized by high item churn. After six month, only around 45% of the items are still available on average in both datasets. Moreover, product generally enter the assortment at a high price and leave it at a discount. Figure 3 illustrates both aspects.

Figure 3: Dynamics of the assortments (each line corresponds to a product and the "level" is the relative price of an item compared to its average price over the whole window)





Product characteristics are available for both data sets. Before applying a model, these variables must be transformed appropriately. This treatment is often referred to as "feature engineering" in the ML litterature. The target variable, price, is log transformed as it is a standard approach in hedonic price imputation. In case of laptops dataset, several product characteristics (CLOCK, MEN, SIZE, PIX, CPU) and brands are dummified, just as the time variable. Note that for the ML models (Random Forests, XGBoost) it is strictly speaking not necessary to transform these product characteristics as heteroscedasticity is not an issue, in contrast to linear models. However, brand needs to be dummified as label encoding is generaly not meaningfull[14].

Similar variable transformations are done for the web-scraped television dataset. For ML methods, we additionally keep time as a numeric vector, which is particularly relevant for Random Forests.

In order to rank different ML methods and a standard time dummy hedonic model according to their prediction performance on unseen data, we use nested resampling and the RMSE as a performance measure. We use 5-fold Cross-Validation for the outer and 10-fold Cross-Validation

[14]If the first brand is encoded as 1, the second as 2, etc. there should be an ordinal relationship between the brands, which is usually not given. An alternative would be to do target encoding of the brand variables, which means replacing a brand by its average price in the data set.

for the inner resampling layers. HP tuning is done via a Bayesian optimizer. Training and test data sets are stratified according to the month variable. This is conveniently done using the benchmark() functionality of the mlr3 package.[15] In Annex 1 we illustrate how to use mlr3 packages to implement nested resampling.

Figure 4: Performance (RMSE) for different learners via Nested Resampling (12 months of data)

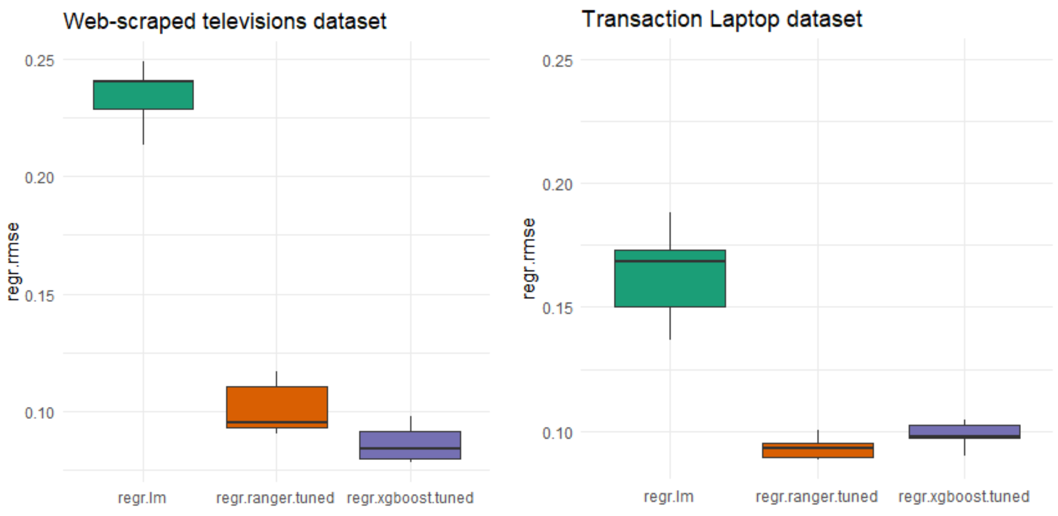


Figure 4 shows the performance of the different learners (Linear Model, Random Forest and XGBoost) for the two data sets. For each learner, the five displayed RMSE correspond to evaluations of a model done on five different outer test sets (see figure 2). Each model is trained on the inner training set and hyperparameter optimization (HPO) is done using 10-fold Cross-Validation (CV).

It is apparent that ML methods perform better on unseen data than linear time dummy hedonic models for both web scraped and transaction data set. For the web scraped dataset, XGBoost performs slightly better than Random Forest, while there is hardly any difference for the laptops data set. It is reasonable to assume that a method, which produces more accurate price predictions for unseen data, should also produce more reliable price imputations for items after they leave or before they enter the market. This result thus suggests that ML methods produce more accurate results for price imputations used for imputation indices, by minimizing estimation variance.

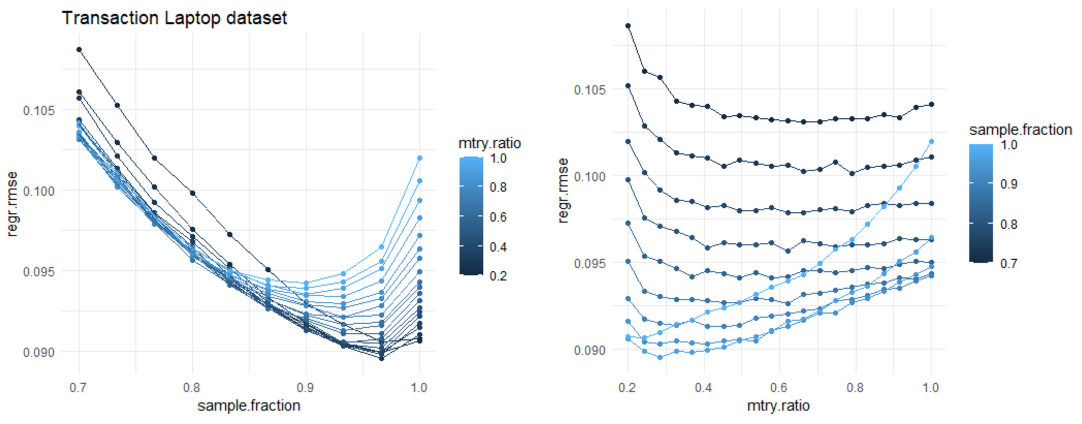
Alternative resampling schemes could also be used for the above analysis, going beyond the use of a simple time-variable stratification. For example, one could require that all observations of a given item are either in the test or training data set, which would mean that when evaluating a model on an outer test, we would evaluate it on completely unseen products in the training set (a strategy known as block resampling). While the latter approach would still show that ML methods perform better than linear models, the differences would be less pronounced. Our view is that block resampling is inappropriate as one never imputes a price of a completely unseen product in practice. However, the simple approach we illustrated here might produce overoptimistic results. The design of relevant resampling schemes for price statistics requires further research as one might overestimate the performance of ML methods by using a too naïve approach.

[15] https://mlr3book.mlr-org.com/chapters/chapter3/evaluation_and_benchmarking.html

In order to use ML for price imputation, it must be trained on the full data set. To do this we must first find out the optimal HPs, which is one of the most challenging parts for any ML method, both in terms of time and expertise required. We first run a Baysean optimizer, using 10-fold CV. This tuning round generally produces acceptable results already. To fine-tune the results even further, we do a local grid search around some of optimal HPs found. In case of Random Forests, a local grid search reduces to a 2-dimensional grid search for the HPs *mtry* and *sample_fraction* (see Table 1). For other HPs, default values are taken. The Bayesian optimizer output indeed allows one to fix the HP *replace*, and the HP *num_trees* must just be large enough to converge[16]. XGBoost has many more parameters to optimize (see Table 1) and for the finetuning we focus on *colsample_bylevel*, *subsample*, *eta* while keeping the values of the other parameters fixed (they were already found using Baysean optimizer). Note that for XGBoost, several different HP combinations may produce very similar results and they are all equally valid HPs for index computation. Generally, similar mean RMSE can be attained for different Random Forests and XGBoost learners.

Figure 5 shows how grid search allows to fine-tune hyperparameters. A minimum is found for *mtry=0.284* (share of randomly selected columns used for a given tress) and *sample_fraction=0.967* (share of line randomly sampled for each tree) (minimal RMSE=0.0895).

Figure 5: Hyperparameter Optimization Surfaces for Random Forest (transaction laptop dataset) (without replacement, number of trees is 1000)



Once optimal HPs have been computed, ML methods and linear model are trained on full dataset using corresponding hyperparameters. The trained model is then used to impute prices needed for bilateral Törnqvist or Jevons indices. We use the same data for training models as for computing multilateral indices (12 months).

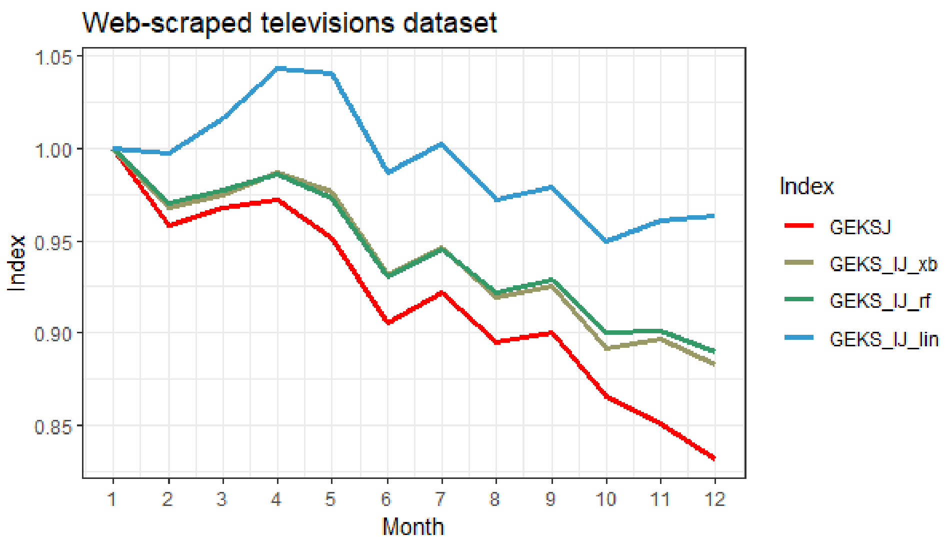
Figures 6 and 7 show several transitive multilateral price indices for the webscraped televison data set and the transaction Laptop data set.

- GEKS-Jevons index, respectively GEKS-Törnqvist index
- GEKS-imputation Jevons indices, respectively GEKS-imputation Törnqvist indices, for several imputation methods (time dummy hedonic model, Random Forest and XGBoost)
- Predicted Share Similarity Linked Index (PS) (Laptop data only, see Diewert and Shimizu (2023))

[16]Increasing the number of trees in a Random Forest doesn't make the model prone to overfitting.

In Annex 2 we illustrate how to use mlr3 packages to implement HP optimization in practice and how to use optimal HPs to impute prices.

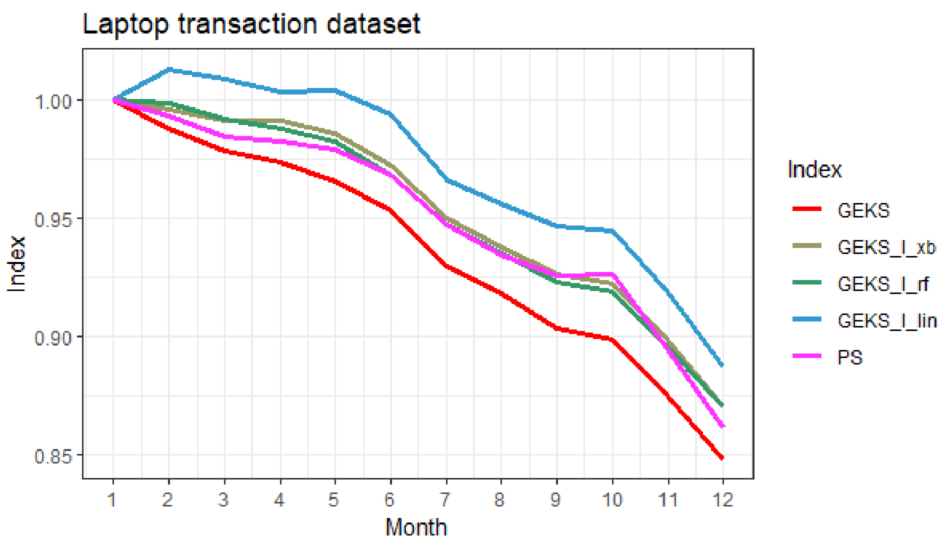
Figure 6: Multilateral Indices for the webscraped televisions data set



The GEKS-Jevons index is downward biased due to the lifecycle effects for this kind of consumer goods. Imputation indices correct for this downward bias by imputing prices once some products leave the sample with a reduced price.

The imputation GEKS index with imputations from a time dummy hedonic model has significantly higher cumulative price changes compared to indices with Random Forest and XGBoost imputations. This result suggests that, even though the fit of the time dummy hedonic model is good (R^2=0.9174), it might overstate the price changes. We have indeed seen that Random Forests and XGboost are able to predict unseen data more precisely and that one should expect imputation to be more reliable.

Figure 7: Multilateral indices for the Laptop transaction data set (OLS is used for the linear imputations based on a time dummy hedonic index)



For the Laptop data set, cumulative price changes for Imputation indices are again higher than the GEKS-Törnqvist Index. There are however hardly any differences between imputation indices using Random Forest and XGBoost imputations. Both indices are, for this 12 month window, close to the Predicted Share Similarity Linked index. Diewert and Shimizu suggested the latter to be the preferred index for this data set (Diewert and Shimizu, 2023).

Figure 8 shows imputed prices for two products to illustrate typical price trends for different imputation methods. For a time dummy hedonic model, almost all imputed prices are equal, with except of a multiplicative time-independent factor. In a time dummy hedonic model, the time dummy variable does indeed not interact with item features, leading to relatively rigid behavior. Even though this allows average residuals per month to be equal to zero, imputed price evolution can deviate significantly in several cases, leading to inappropriate price comparisons (see second graph in figure 8). This leads to a higher index using a linear model in the imputation GEKS index. ML algorithms take into account item features and time in a more flexible manner. This leads to product specific price evolutions. Moreover, price trends of imputed prices tend to better fit to trends of observed prices. Note that ML algorithms are not overfitting and perform better than hedonic linear models on unseen data (see Figure 4).

Figure 8: Imputed prices and real transaction prices for selected products (imputed prices for the linear hedonic model take into account expenditure shares)

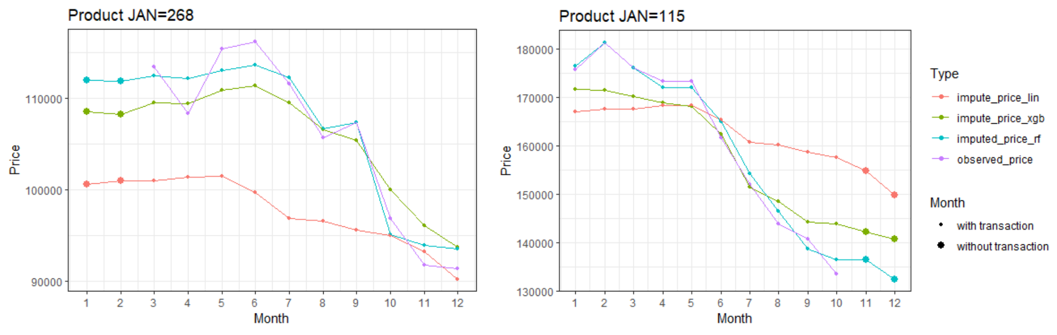
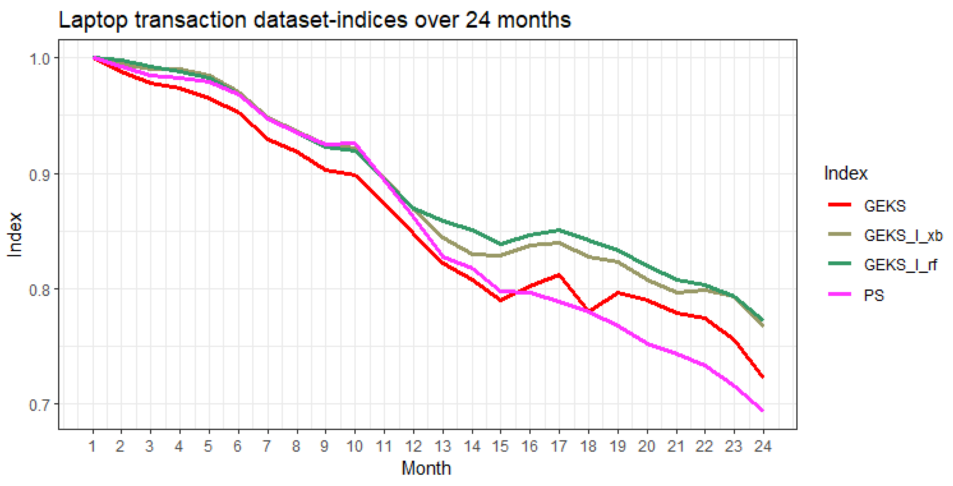


Figure 9: Spliced multilateral GEKS (Imputation) indices for the Laptop transaction data set (12 months window, window splicing) and Predicted Share Similarity index



When looking at longer time periods, (spliced) imputation indices (with Random Forest and XGBoost imputations) and the Predicted Share Similarity Linked Index deviate significantly.

Figure 9 shows that both indices begin to diverge from the month 13 on. This can be related to the assortment, as in month 13 a significant amount of new products appears (see Figure 4), leading to sparse matching between month 13 and any month before. The Predicted Share Similarity Linked index is thus based on a relatively small number of matched products. Imputation indices, on the other hand, take into account additional information coming from the item features in order to overcome this sparse matching between month 13 and previous months.

4 Process

In order to implement the ML based imputation methodology in a CPI production environment, we propose a division of the process into three distinct parts (see Figure 10): data management, machine learning and price statistics. A separate expert can manage each part. A data scientist could be in charge of the ML part, aiming for finding the best possible model. A price statistician could be in charge of the price statistics part, aiming for calculating and validating the price indices. MLFlow[17] is an open source tool that acts as the technical link between the ML and the price statistics environment.

Part 1: Data Source

After obtaining data for a given month t (scanner data or web scraped data), we first apply data cleaning. This may include identifying and removing outliers, or imputing missing product characteristics. We then pool the data for a given month with the data from the previous months, using for exampling a rolling time window strategy. This cleaned and pooled data is then the input of the second (Machine Learning) and third (Price Statistics) part.

Part 2: Machine Learning

The first step is to apply feature engineering by transforming the input data. We then apply CV in order to find the optimal HPs for XGBoost and/or Random Forest. We then train a model using the best HPs on the entire data (one Random Forest model, one XGBoost model). These final models are then stored in MLFlow. As a side product, the HP optimization also provides some accuracy measures[18] that we can store in MLFlow (see Figure 10). We use the R package MLR3 to implement these calculations (see Annex 1 and Annex 2). We also use the R package mlflow for interacting with MLFlow.

Part 3: Price Statistics

The price statistician can now select the model they would like to use for price imputation. Subsequently, this model is used to impute missing prices on the pooled data. This augmented data (observed and imputed prices) can afterward be used to calculate a price index (for example a GEKS index). The resulting indices can then be spliced with previously calculated indices, and integrated into the CPI to obtain higher-level indices. We use the R package IndexNumR to calculate the GEKS indices. We also use the R package MLFlow for interacting with MLFlow.

Integrating the proposed division of the process is straightforward with MLFlow, an open source tool renowned for its robust framework for efficient ML lifecycle management. MLFlow provides centralized storage capabilities for ML models and associated artefacts. It also provides a user-friendly interface for tracking and monitoring model performance (see Figure 11), which

[17]See `https://mlflow.org/`

[18]Strictly speaking, we would need to apply nested resampling to estimate the accuracy. However, for practical reasons, we only do HP optimization in a production environment.

facilitates continuous improvements and decision-making throughout the experimental phase. One distinguish feature of MLFlow is its provision of unique model access for production environment. This includes features such as versioning and staging functionalities, which enables the collaboration and knowledge sharing across teams.

In practice, we interact with MLFLow in the following way each month:

- We store various models (together with their performance and hyperparameters) that were trained in month t based on data from the periods t, t-1, ... t-12
- We select our preferred model which we then register. Each period we thus register a new model
- We access the registered model for period t in order to make the imputations on data from the periods t, t-1, ... t-12

Figure 10: Overview of the process

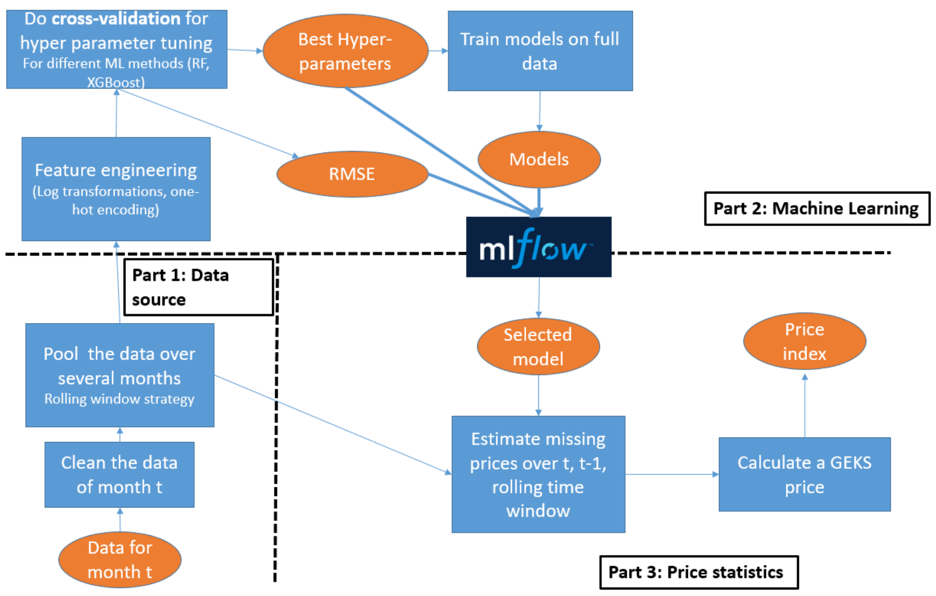
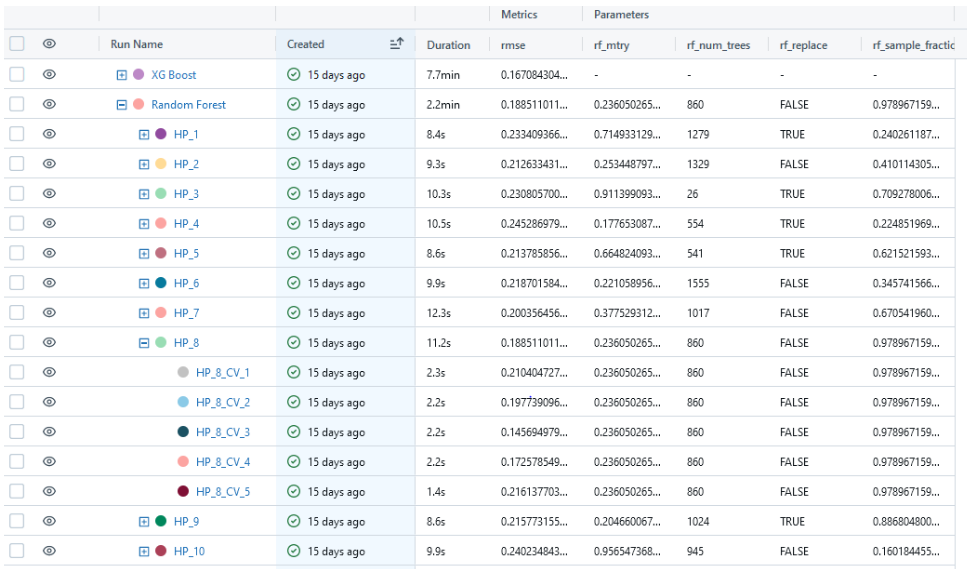

Do cross-validation for hyper parameter tuning
For different ML methods (RF, XGBoost)
Best Hyper-parameters
Train models on full data
Feature engineering
(Log transformations, one-hot encoding)
RMSE
Models
Part 2: Machine Learning
mlflow
Part 1: Data source
Selected model
Price index
Pool the data over several months
Rolling window strategy
Estimate missing prices over t, t-1, rolling time window
Calculate a GEKS price
Clean the data of month t
Data for month t
Part 3: Price statistics

Figure 11: Monitoring model performance in MLFlow



			Metrics	Parameters			
Run Name	Created	Duration	rmse	rf_mtry	rf_num_trees	rf_replace	rf_sample_fractic
XG Boost	15 days ago	7.7min	0.167084304...	-	-	-	-
Random Forest	15 days ago	2.2min	0.188511011...	0.236050265...	860	FALSE	0.978967159...
HP_1	15 days ago	8.4s	0.233409366...	0.714933129...	1279	TRUE	0.240261187...
HP_2	15 days ago	9.3s	0.212633431...	0.253448797...	1329	FALSE	0.410114305...
HP_3	15 days ago	10.3s	0.230805700...	0.911399093...	26	TRUE	0.709278006...
HP_4	15 days ago	10.5s	0.245286979...	0.177653087...	554	TRUE	0.224851969...
HP_5	15 days ago	8.6s	0.213785856...	0.664824093...	541	TRUE	0.621521593...
HP_6	15 days ago	9.9s	0.218701584...	0.221058956...	1555	FALSE	0.345741566...
HP_7	15 days ago	12.3s	0.200356456...	0.377529312...	1017	FALSE	0.670541960...
HP_8	15 days ago	11.2s	0.188511011...	0.236050265...	860	FALSE	0.978967159...
HP_8_CV_1	15 days ago	2.3s	0.210404727...	0.236050265...	860	FALSE	0.978967159...
HP_8_CV_2	15 days ago	2.2s	0.197739096...	0.236050265...	860	FALSE	0.978967159...
HP_8_CV_3	15 days ago	2.2s	0.145694979...	0.236050265...	860	FALSE	0.978967159...
HP_8_CV_4	15 days ago	2.2s	0.172578549...	0.236050265...	860	FALSE	0.978967159...
HP_8_CV_5	15 days ago	1.4s	0.216137703...	0.236050265...	860	FALSE	0.978967159...
HP_9	15 days ago	8.6s	0.215773155...	0.204660067...	1024	TRUE	0.886804800...
HP_10	15 days ago	9.9s	0.240234843...	0.956547368...	945	FALSE	0.160184455...

5 Conclusions and open issues

We have investigated the use of ML methods for making price imputations that can be integrated in price indices for items that are missing.

We first evaluated the performance of ML methods. We have seen that XGBoost and Random Forest have significantly lower RMSE compared to a standard linear regression. We prefer Random Forest over XGBoost as the HPs can be more easily optimized for the former, rather than for the latter. Moreover, the use of weights was crucial to add robustness to the results of XGBoost, while weights tend to play a less important role in Random Forest. The entry cost to the field of machine learning was reduced by relying on existing tools (e.g. MLR3 collection of packages).

The choice of an imputation method also has an impact on the final prices indices. Both Random Forest and XGBoost based price indices are relatively similar, but on our datasets they tend to decrease more than price indices that are based on linear regressions. Finally, we have suggested a process for using ML methods in a production environment. We believe it is important to distinguish the ML part from the price statistics part.

We have identified some open issues that should be further investigated when combining ML and Price Statistics:

- The choice of re-sampling strategy impacts the evaluation of performance of the model and HP optimization. What would be the most appropriate re-sampling strategy in context of prices statistics data that includes time and product dimensions? For example, we may include a stratification (by period or by product) in the resampling design, or we may link together observations of the same period or the same product
- Time plays a crucial role in our context as we are interested in price changes across time. One should further examine how time should be best encoded in ML models in order to properly capture the panel aspect in our data. This becomes increasingly relevant for “longer” time windows

- Feature engineering is often a crucial aspect in ML. We have only done some basic transformations of the available characteristics. There could be some potential to further optimize the models through feature engineering
- While it is common to work with expenditure weights in price statistics, it remains unclear if weight information should also be used when training ML models. The treatment of weights also depends on the specific ML method that is applied
- The ML models aim at imputing individual prices. However, our final objective is to measure aggregated price change. One should better understand the relationship between imputations for an individual price and the corresponding price index. For example, how does the bias-variance trade-off propagates to bilateral and multilateral the price indices?
- We have aligned our re-training strategy to the rolling time window strategy applied in multilateral methods. It still remains to be seen what is the best time window over which ML models should be trained and how should these time windows be adjusted when data of the last period becomes available.

References

[1] Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., and Lindauer, M. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. WIREs Data Mining and Knowledge Discovery, 13(2), e1484. https://doi.org/10.1002/widm.1484.

[2] Bischl, B., Sonabend, R., Kotthoff, L., and Lang, M. (Eds.). (2024). Applied Machine Learning Using mlr3 in R". CRC Press. https://mlr3book.mlr-org.com

[3] Breiman L. (2001). Random Forests. Machine Learning 45, 5–32.

[4] Cafarella, M., Ehrlich, G., Gao, T., Haltiwanger, J. C., Shapiro, M. D., and Zhao, L. (2023). Using machine learning to construct hedonic price indices (No. w31315). National Bureau of Economic Research.

[5] Calainho, F. D., van de Minne, A. M., and Francke, M. K. (2022). A machine learning approach to price indices: Applications in commercial real estate. The Journal of Real Estate Finance and Economics, 1-30.

[6] Caves, D.W., L.R. Christensen and W.E. Diewert (1982), "The Economic Theory of Index Numbers and the Measurement of Input, Output, and Productivity", Econometrica 50, 1393-1414.

[7] Chen, T., and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[8] De Haan J (2004). Hedonic Regression: The Time Dummy Index As a Special Case of the Imputation Törnqvist Index. Paper presented at the 8th Ottawa group meeting, Helsinki, Finland.

[9] De Haan J., Daalmans J. (2019) Scanner Data in the CPI: The Imputation CCDI Index Revisited. Paper presented at the 16th Ottawa group meeting, Rio de Janeiro, Brazil.

[10] De Haan J., Krsinich, F. (2014). Scanner Data and the Treatment of Quality Change in Nonrevisable Price Indexes. Journal of Business and Economic Statistics, 32(3), 341–358.

[11] Diewert E., Shimizu C. (2023). Scanner Data, Product Churn and Quality Adjustment. Paper presented at the UNECE meeting of the Group of Experts on Consumer Price, Geneva.

[12] James G., Witten D., Hastie T., Tibshirani R. (2013). The bias-variance trade-off. In: An introduction to Statistical Learning with Applications in R. Springer Texts in Statistics.

[13] Lamboray C. (2022). Matching, grouping and linking: What impact does the product specification have on a Fisher price index? Paper presented at the 17th Ottawa group meeting, Rome, Italy.

[14] Picchetti, P. (2017). Hedonic residential property price estimation using geospatial data: a machine-learning approach. Paper presented at the Paper presented at the 15th Ottawa group meeting, Eltville, Germany.

[15] Roche, K., Taylor, L., and Keshishbanoosy, R. (2022). Applying advanced techniques to the estimation of Multipurpose Digital Device Price Indices. 17th Meeting of the Ottawa Group.

[16] Switzerland Ho, T. K. (1995). Random decision forests. In Proceedings of 3rd international conference on document analysis and recognition (Vol. 1, pp. 278-282). IEEE.

[17] Zafar, J. D., and Himpens, S. (2019, May). Webscraping Laptop Prices to Estimate Hedonic Models and Extensions to Other Predictive Methods. Paper presented at the 16th meeting of the Ottawa Group on Price Indices, Rio de Janeiro, Brazil.

[18] Zeng, S. (2021, August). Hedonic imputation with tree-based decision approaches. In 36th IARIW Virtual General Conference.

Annex 1 – Nested resampling

```
# Load packages
library(dplyr); library(mlr3); library(mlr3verse); library(mlr3learners); library(mlr3fselect); library(mlr3tuning);
library(mlr3misc); library(xgboost); library(mlbench); library(ranger); library(mlr3mbo)

#######################################################
######### PART 1 - DATA SOURCE ###################################
#######################################################

# Import your data here
# Choose your time window

# The following variables need to be in your data:

# lnP: logarithm of price
# PERIOD: Time periods (1,2,3,etc.)
# E: Expenditure (in case of transaction data only)

# Transform your data (create dummies out of time vector, brand, etc.)

# Dataset which will be used in the Machine Learning part is called mydata

#######################################################
######### PART 2 - MACHINE LEARNING - NESTED RESAMPLING ##############
#######################################################

# Define the cross validation strategies
# inner_cv = rsmp("cv", folds = 10) # inner loop for nested CV
# outer_cv = rsmp("cv", folds = 5) # outer loop for nested CV
inner_cv = rsmp("cv", folds = 5) # inner loop for nested CV
outer_cv = rsmp("cv", folds = 3) # outer loop for nested CV
set.seed(555)

# Define the task, we want to predict log price
mytask= as_task_regr(mydata, target = "lnP")

# Add the time period as a stratification variable when resampling
mytask$col_roles$stratum="PERIOD"

# Alternative resampling strategies could be defined here
# Take account of weights in ML alg. (in case of web-scraped data this can be ignored)

# mytask$col_roles$weight="E"

# We need to exclude the Expenditure variable from the set of features, that is we define a reduced set.
# Drop also the product identifier JAN

mytask$select(names(mydata %>% select(-E, -JAN, -lnP)) )

# Define the tuning strategy for xgboost (inner resampling)
xgb_tuned = AutoTuner$new(learner = lts(lrn("regr.xgboost")),
                        resampling= inner_cv,
                        measure=msr("regr.rmse"),
                        tuner = tnr("mbo"),
                        terminator= trm("evals", n_evals = 20),
                        store_models = TRUE)

# Define the tuning strategy for random forest (inner resampling)
rf_tuned = AutoTuner$new(learner = lts(lrn("regr.ranger")),
                       resampling= inner_cv,
                       measure=msr("regr.rmse"),
                       tuner = tnr("mbo"),
                       terminator= trm("evals", n_evals = 20),
                       store_models = TRUE)


# Define the benchmarking (outer resampling)
bm_design = benchmark_grid(
  tasks = mytask,
  learners = c(lrn("regr.lm"),
               rf_tuned,
               xgb_tuned),
  resamplings = outer_cv
)

# Evaluate performance of models via nested resampling
bmr = benchmark(bm_design, store_models = TRUE)

# Plot the models
autoplot(bmr, measure = msr("regr.mse"), type = "boxplot")
autoplot(bmr, measure = msr("regr.rmse"), type = "boxplot")
autoplot(bmr, measure = msr("bias"), type = "boxplot")
```

Annex 2 – Hyperparameter optimization and model estimation

```
# Load packages
library(dplyr); library(mlr3); library(mlr3verse); library(mlr3learners); library(mlr3fselect); library(mlr3tuning);
library(mlr3misc); library(xgboost); library(mlbench); library(mlr3misc); library(ranger); library(DiceKriging)

##############################################################
######### PART 1 - DATA SOURCE – HPO ################################
##############################################################

# Import your data here
# Choose your time window

# The following variables need to be in your data:

# lnP: logarithm of price
# PERIOD: Time periods (1,2,3,etc.)
# E: Expenditure (in case of transaction data only)

# Transform your data (create dummies out of time vector, brand, etc.)

# Data set which will be used in the ML part is called mydata

##############################################################
######### PART 2 - MACHINE LEARNING – HPO ##########################
##############################################################

# Define the task, we want to predict log price
mytask= as_task_regr(mydata, target = "lnP")

# Add the time period as a stratification variable when resampling
mytask$col_roles$stratum="PERIOD"

# Take account of weights in ML alg. (in case of transaction data only)
mytask$col_roles$weight="E"

# We need to exclude the Expenditure variable from the set of features,that is we define a reduced set.

mytask$select(names(mydata %>% select(-E,-lnP)) )

# Define the Cross -Calidation strategy
cv10 = rsmp("cv", folds = 10)
set.seed(555)

# We search for optimal HPs for Random Forest using mbo instance_rf_mbo = tune(
  tuner = tnr("mbo"),
  task = mytask,
  learner = lts(lrn("regr.ranger")),
  resampling = cv10,
  measure =msr("regr.rmse"),
  term_evals = 20)

# Fine tuning can be done using a grid search
# design = data.table(expand.grid(sample.fraction=seq(0.2, 1, length.out = 10),
#                                 mtry.ratio =seq(0.2, 1, length.out = 20),
#                                 replace = FALSE,
#                                 num.trees=1000))
#
# instance_rf_grid_search = tune(
#   tuner = tnr("design_points", design = design),
#   task = mytask,
#   learner = lts(lrn("regr.ranger")),
#   resampling = cv10,
#   measure =msr("regr.rmse"))


# Now we do the the final model. We run the tuning on the full data
# We use random forest
final_learner_rf = lts(lrn("regr.ranger"))

final_learner_rf$param_set$values = instance_rf_mbo$result_learner_param_vals

final_learner_rf$train(mytask)

##########################################################
######### PART 3 - PRICE STATISTICS ##################################
##########################################################

### Imputation GEKS

# STEP 1: we create an augmented dataframe
# This augmented data set containes features (including features related to time)
# For items whose price is to impute
# We call this dataset items_to_impute

# STEP 2: We estimate the price with Random Forest
predicted_price=final_learner_rf$predict_newdata(items_to_impute)
items_to_impute$imputed_price_rf=exp(predicted_price$response)

# STEP 3: An imputation GEKS can now be computed
```