UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Accuracy comparison between Sparse Autoregressive and XGBoost models for high-dimensional product sales forecasting

---

Author:
Blai RAS

Supervisor:
Dr. Jordi VITRIÀ

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

September 2, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Accuracy comparison between Sparse Autoregressive and XGBoost models for high-dimensional product sales forecasting

by Blai RAS

Predicting future sales is key for any business budgeting and resource allocation. One major concern when trying to build accurate forecasts are the cross-category relationships between some products and the effect that might have on each other's sales. Given today's data abundance, this issue is even more worrying: traditional statistic models can't handle high-dimensional datasets with ten or more products. With the use of popular machine learning and data science tools, we developed a framework that enables the building, training and evaluation of two models and its comparison through a detailed set of forecast metrics[1]. The first model is a modified Vector Autoregressive model (VAR) which takes into account product relationships. The second one is an XGBoost model, which is not specialized into cross-category associations but it's known for its versatility and performance when working with tabular data. After performing a one-month ahead sales forecasting on a huge dataset of multiple product sets, we find that inter-product connections play a huge role in prediction accuracy since the VAR model performed considerably much better than the XGBoost.

[1] Code available on BitBucket

Acknowledgements

In first place, express my deep sense of thanks towards Accenture and my project advisors from there, Sergi Zamora and Riste Gjorgjiev.

Secondly, I would like to acknowledge Universitat de Barcelona for making possible this collaboration, specially to my supervisor Jordi Vitrià, PhD.

Finally, I am extremely thankful to my dad, my family and my mate Aranzazu, who always reached their hands when I needed them.

Contents

Chapter 1

Introduction

A sales forecast is a study that aims to predict future sales of one or multiple products using historical data. An accurate sales forecasting enables companies to perform informed decisions about resource allocation, future benefits and budgeting.

The term "cannibalization" in the marketing background is understood as a loss in sales or revenue from a product of a company because of the launching or promotion of another product from the same company. Cannibalization can be manifested in one to one, one to many, many to one and many to many relationships.

Acknowledging and understanding cannibalization is necessary in every business, despite its catalog size or revenue. For example, a company can deliver discounts to the cannibalized product in order to increase sales. So, even though the term cannibalization has negative connotations, often companies choose to cannibalize their sales for concrete goals.

Also, when we talk about data points (in this case, sales) ordered in time we enter the time series analysis background. The most common aspects that this field has to deal with when predicting new data points are stationary, seasonality and trend effects.

The aim of this project is to determine if also cannibalization relationships heavily affect sales forecasting accuracy.

1.1 Origin of the project

This project is born between the collaboration of the Facultat de Matemàtiques i Informàtica of Universitat de Barcelona and Accenture, an Irish-domiciled consulting firm. Inside their Applied Intelligence unit they've been working on cannibalization-related projects, specially focusing on cannibalization discovering.

After a brief introduction of this works by my project manager and some domain research, I decided to continue their line of investigations with a new approach of the cannibalization problem.

1.2 Proposal

In this project we aim to build two algorithms of sales forecasting and check their accuracies. The first one is an statistical model called Vector autoregression (VAR), and

the second one is the king of Kaggle competitions[1] XGBoost, an ensemble Machine Learning algorithm based on decision-tree decisions.

VARs are multivariate linear time series models designed to capture the joint dynamics of multiple time series. In other words, they are able to understand relationships between multiple variables. They are used a lot in macroeconomics (Hilde, 2000) and finance, for example, when trying to understand the relationship between the Gross Domestic Product and the unemployment of a country alongside other factors.

In contrast, XGBoost (eXtreme Gradient Boosting) is an optimized gradient boosting algorithm that meanwhile it outperforms any other algorithm when working with structured or tabular data, is does not specifically try to understand connections or associations between variables.

In short, we are going to built and tune both algorithms, we are going to train them using a dataset of possible cannibalized sales and finally check their rightness when predicting future sales. This project centres on the study and understanding of VAR models. XGBoost will not be the primarily focus of the project, its role is more about algorithm comparison where he represents the non-autoregressive, machine learning part.

1.3 Roadmap

This thesis is structured as follows:

- **Goals**: a synthesis of what do we want to accomplish with this work.
- **Previous concepts and State of the art**: description of the theoretical basis of the thesis, state of the art and other definitions.
- **Data**: insights about our dataset and the explanation of the data cleaning and pre-processing process.
- **Implementation**: exposition of the process of model creation, training and evaluation.
- **Experiments, Results and Discussion**: the evaluation results alongside the found insights and observations.
- **Conclusions and future work**: assessment of the performed work and how can we improve it in the future.

1.4 Goals

In this master thesis we want to built a framework that (i) allows us to perform a forecast on sales using both algorithms and (ii) allows us to evaluate their predictions numerically and graphically.

In order to do so, the following aspects should be achieved:

- The clear understanding of the procedure of a sales forecast and different evaluating tools.

[1] Kaggle competitions are open Machine Learning challenges were users compete to find the best solution and often win big monetary rewards.

- The clear understanding of the Vector Autoregression methodology.
- The built up of a framework that is able to reproduce our results and show them in a graphical and easy-to-understand way.

During the development of this work, we aim for extending our knowledge in statistics, specially applied to time series analysis; machine learning, specifically in model building, interpretation and comparison; optimization and lineal algebra, in terms of algorithm convergence and loss function configuration and big data, because we will be dealing with high-dimensional datasets.

Chapter 2

Previous concepts and State of the art

In this section it is explained the theory behind Vector Autoregressive models and how is adapted for sales predictions. After that, we introduce a little bit of context talking about other works that have used a similar scientific method. Finally, we will discuss about forecasting evaluation methods. Before we begin, let me introduce a short glossary:

- **Autoregression or Autoregressive**: an statistical model is Autoregressive if it predicts future data points based on past values.
- **Exogenous variable**: an independent variable, with no formulaic relationship at all. Values of exogenous variables are determined outside the model, they are predetermined and they always keep the same value.
- **Endogenous variable**: a variable that can be explained by relationships with other variables. A change in a endogenous variable can be understood as a response to an exogenous or endogenous change.

2.1 Vector Autoregressive models

Let us imagine that we are responsible of a supermarket. In our store, we sell lots of different products. Our goal is to predict how many of this products we are going to sell in any given month. Why is it not recommended to use many AutoRegressive (AR) models, like ARMA or ARIMA models? Because these ones are only able to model one time series at a time, and we know some of our product sales are related.

This is the key factor of Vector Autoregressive (VAR) models: (i) they can model multiple series at a time for longer horizons and without having to assume that some variables are exogenous and (ii) these time series might be related to each other's past.

In our example, sales of soda cans and other soft drinks might not just depend on past values of themselves, but also on past values of chips or other snacks. Bacon is often bought with a basket of eggs (Niraj, Padmanabhan, and Seetharaman, 2008), plastic cups are often bought with plastic dishes and a set of plastic cutlery and liquor consumers often pair their buy with a bag of ice cubes. This kind of interdependencies is what VAR models try to understand.

In a mathematical form, a Vector Autoregressive model with lag p, denoted by VAR(p) is expressed as following:

$$Y_t = \Phi_1 y_{t-1} + \cdots + \Phi_p y_{t-p} + \varepsilon_t\,, \quad t = 1, \ldots, T\,, \tag{2.1}$$

or, what is the same:

$$Y_t = \sum_{i=1}^{p} \Phi_i y_{t-i} + \varepsilon_t\,, \quad t = 1, \ldots, T\,,$$

where $\{y_t\}_{t=1}^{T}$ is vector of size $N \times 1$ of the endogenous variables of our problem, Φ_i is a $N \times N$ matrix of coefficients, p is the period or "lag" and ε_t is often supposed to be a Gaussian white noise, a series of uncorrelated errors from the time series forecast that is normally distributed with mean zero and covariance matrix Σ, i.e. $\varepsilon_t \overset{\text{wn}}{\sim} (\mathbf{0}, \mathbf{\Sigma}_\varepsilon)$.

When we add exogenous variables we have a $VARX_{k,m}(p, s)$ model with a k-dimensional vector series of endogenous variables $\{y_t\}_{t=1}^{T}$ modeled by its own p periods of past values and a exogenous vector series of dimension m $\{x_t\}_{t=1}^{T}$ modeled also with s past periods:

$$Y_t = \sum_{i=1}^{p} \Phi_i y_{t-i} + \sum_{j=1}^{s} B_j x_{t-j} + \varepsilon_t\,, \quad t = 1, \ldots, T\,, \tag{2.2}$$

where $\{\Phi_i \in \mathbb{R}^{k\times k}\}_{i=1}^{p}$ are the coefficient matrices of the endogenous variables, $\{B_j \in \mathbb{R}^{k\times m}\}_{i=1}^{s}$ are the coefficient matrices of the exogenous variables.

A VAR model can be easily written in matrix form. Recalling the previous example about chips and soda sales, if we know their sales for the last $p = 3$ periods, the VAR(3) model can be written as:

$$\begin{bmatrix} SODA_t \\ \text{CHIPS}_t \end{bmatrix} = \Phi_1 \begin{bmatrix} SODA_{t-1} \\ \text{CHIPS}_{t-1} \end{bmatrix} + \Phi_2 \begin{bmatrix} SODA_{t-2} \\ \text{CHIPS}_{t-2} \end{bmatrix} + \Phi_3 \begin{bmatrix} SODA_{t-3} \\ \text{CHIPS}_{t-3} \end{bmatrix} + \varepsilon_t$$

What we are saying here is that the sales of soda cans can be modeled as (i) depending on the sales of soda in the last 3 periods and (ii) depending on the sales of chips of the last 3 periods. The Φ_t are the $k \times k$ matrices containing the coefficients that our model will try to estimate.

2.1.1 Dimensionality problem

Often, when modeling real life problems, the number of variables that we handle, denoted by k, and the lag order, denoted by p, are really high. The Φ_i matrix coefficients needed to estimate per equation (see equation 2.1), $p \times k$, can get close to the amount of observed data that we have. This creates a high-dimensionality problem.

Take for example that our supermarket has 50 products. We observed their daily sales for a full year, so we have approximately $6 \cdot 52 = 300$ rows in our dataset. The

VAR model will try to estimate $k \times p$ parameters for every variable, so the total number of parameters to be estimated is $k^2 \times p$ (there are p matrices of size $k \times k$). If we decide a lag order of 6, the number of times our supermarket opens in a week, we would have to estimate $50^2 \times 6 = 15000$ parameters, quite disproportionate compared to our 300 observations.

The Bias-Variance trade-off

Another reason of why we can't estimate a normal VAR when having more parameters than observations is what is known as the bias-variance trade-off. In any forecast we are interested in minimizing is the Mean Squared Error (MSE), which is defined by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.3}$$

where $\hat{y}$ is our estimated value and y is the observed value. The MSE, however, can be decomposed into two parts. The first one is the variance of our estimations, how sensitive are to small variations, and the second part is the bias, which corresponds to the expected error of these estimations. Sometimes we also find parts of noisiness of the data itself, also called *Irreducible error*

In our VAR model we do not estimate a single parameter, so equation 2.3 is not representative. Instead, we have the matrices Φ_j for $j = 1, \ldots, p$. Overall, then, we could say that we have a matrix Φ which contain all the parameters $[\Phi_1, \Phi_2, \ldots, \Phi_{(p)}]$. If we stack (vectorize) all the columns of Φ above each other and call it θ, our MSE can be represented as following:

$$\begin{aligned} MSE &= \mathbb{E}[||\hat{\theta} - \theta||^2] \\ &= \mathbb{E}[(\hat{\theta} - \theta)'(\hat{\theta} - \theta)] \\ &= \underbrace{\mathbb{E}[(\hat{\theta} - \mathbb{E}\hat{\theta})'(\hat{\theta} - \mathbb{E}\hat{\theta})]}_{Variance} + \underbrace{[(\mathbb{E}\hat{\theta} - \theta)'(\mathbb{E}\hat{\theta} - \theta)]}_{\text{Bias}^2} \end{aligned}$$

We always want the bias to be zero, meaning that, on average, one would obtain the true estimate. As a trade-off, this comes with the cost of increasing the variance of the estimations. If we have lots of parameters to be estimated, this cost is even higher.

In 2.1 we have represented 4 time series with different number of variables lagged from 2 to 16 periods. In standard cases, say 2 to 8 different variables, the MSE is so low that is barely distinguishable. When dealing with a high dimensional case, like 16 variables, things start to get messy: in the last lag, $16^2 \times 16$ parameters must be estimated by the model. If, again, we have 250 observed values, we deal with $4096/250 \approx 16$ parameters per observation!

Why is MSE important for our model? High MSE indicates that its forecasting performance will very likely be poor. We want a strong, stable model with high interpretability. Luckily for us, modern techniques were developed which are able to handle a huge amount of parameters with respect to the number of observations.

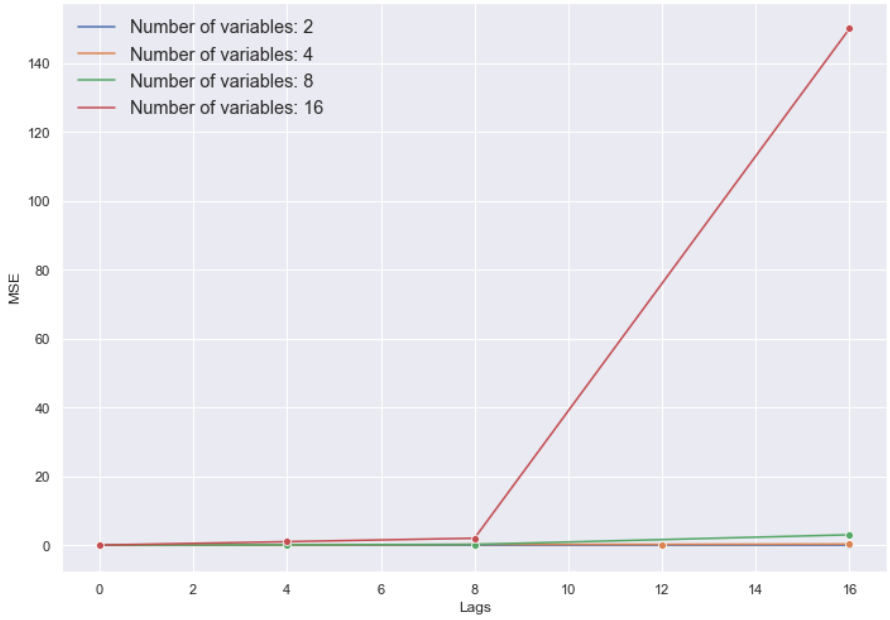

FIGURE 2.1: MSE approximation for 4 time series with different amount of variables

2.2 High-dimensional VAR

In the supermarket example, we saw how multiple products sales can be related. At the same time, some other products might have absolutely zero relationship with themselves. Therefore, we need some kind of regularization, a constrain able to dismiss some parameters and reduce the dimensionality of our model.

The solution of our problem is called *Least Absolute Shrinkage and Selection Operator Regression* or Lasso, a regularization technique that automatically performs feature selection and outputs a sparse model, i.e., with few non-zero coefficients.

Sparsity reduces the freedom of our model structure and how it interprets the data. In our context, we believe that many of the parameters are actually zero, meaning that most variable included in the model have no effect at all on the outcome variable, or in such little effect that they can be ignored.

Methods that enforce sparsity often have lower variance at the cost of a little bit of bias increase (because the sparsity assumption is not exactly true). This is a positive trade-off because we are heavily lowering the MSE compared to a model that tries to achieve an unbiased vision. Thus, sparser models provide better long term predictions.

2.2.1 Lasso Regression

Lasso was firstly introduced by (Tibshirani, 1996a) for linear regression models. Thanks to their work, structured and grouped variations appeared. The most important ones are (Yuan and Lin, 2006) and (**simon2013sparse**).

In these papers they propose a loss function that is a regularized version of least squares but introducing what is called a "L_1" penalty on the coefficients. This penalty allows some of the coefficients to be zero, therefore performing variable selection and outputting a sparser solution.

In our context, the Lasso loss function would be:

$$0.5\|Y - \Phi y\|_2^2 + \lambda\|\Phi\|_1\,, \tag{2.4}$$

where $\|\cdot\|_2$ is the ℓ_2 norm, $\|\cdot\|_1$ the ℓ_1 norm and λ is a scalar penalization parameter that controls the "shrinkage" of the weights. To obtain the optimal penalization, i.e., the hyperparameter that decides how many coefficients will be set to zero, we use cross-validation (CV).

Time series Cross-Validation

Cross-validation is a statistical procedure where we split our data in K-folds in order to perform some sort of test/validation. For example, in Machine Learning we split our testing dataset into several folds, then train our model on all folds except one and finally test the model on this unseen one. We repeat the process for all the folds and at the end we take the average of each metric obtained in every fold. This technique prevents overfitting and it is a more robust way to evaluate a model's performance.

Nonetheless, when dealing with temporal data one has to be careful at the time of splitting into folds, due that we can't train a subset of data with future information.

What we do is use a "rolling" cross-validation. We start with a small subset of data, call it period T_0 to T_1, and we use it for estimating the optimal λ based on minimizing the n-step ahead MSE, using the T_{1+n} periods. The process is visualized in figure 2.2.

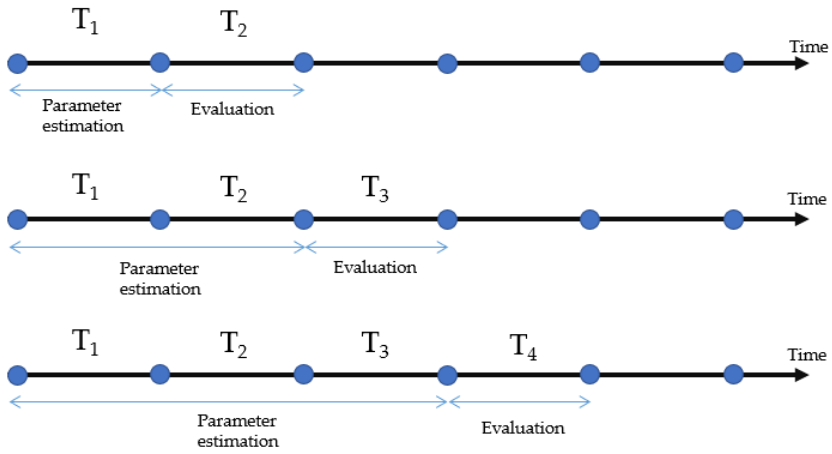

FIGURE 2.2: Rolling cross-validation process using 1-ahead period for testing

2.2.2 Hierarchical sparsity pattern

The idea behind Lasso is great for reducing our dimensionality problem and sacrifice a little bit of bias in order to enhance our variance. Nonetheless, Lasso needs a little bit of real-word context tuning on its implementation.

A conventional Lasso implementation can set the coefficient of the n and $n+2$ lags of my chips time series sales to zero and then set the $n+1$ to non-zero. Does this follow a natural, real-word structure? In other words, does it really make sense that the sales of chips has an effect on, let's say, soda can sales in the n and $n+2$ periods but not in the $n+1$? Probably not, specially if our period is "short" (days or weeks).

To solve this issue, the hierarchical group-lasso ensures that if the coefficient of a lagged variable in one equation is set to zero, then also all coefficients of that variable when lagged n more times are also zero. This behavior helps the interpretation of the inter relationships of our variables.

However, we still can obtain results where the twice lagged chips sales coefficient is set to zero in the equation explaining soda can sales, but the once lagged chip sales coefficient is not. Also, this hierarchical structure does not impose any restrictions across variables or equations.

2.2.3 The lag selection problem

A "lag" is a fixed amount of passing time. In a time series, a lag is essentially a number describing a delay, a shift. When we create a VAR model with lag $p = 2$ we are telling our model to use the information of our time series from 1 and 2 periods ago. If we have quarterly data and set $p = 4$, we allow the mode to use all the information till a year ago.

Selecting the right lag value is crucial for a VAR or a big-dimensional VAR. If we set a high value, i.e., $p \approx N$, being N our number of samples, we won't have enough observations to fit the model. If we set a small value our model might not learn the right relationships and will underfit.

There are several techniques to asses this problem. The most popular one is based upon the Information Criteria:

$$IC(m) = \underbrace{-\frac{2}{T}\log L(m)}_{\text{Lack of Fit}} + \underbrace{\frac{c_T}{T}\dim(m)}_{\text{Complexity Penalty}} \tag{2.5}$$

where m is our VAR model, T the number of total time periods and $dim(m)$ the dimension of the model, that is, how many parameters has. c_T is a value that gives name to each Information Criteria. The most known ones are the following:

- $c_T = \log(T)$, the **Bayesian Information Criterion** (BIC).
- $c_T = 2$, the **Akaike Information Criterion** (AIC).

As we can see in 2.5, the IC is also a trade-off between efficiency (get the smallest forecasting error) and consistency (selecting the right model). There is no IC method that is able to provide the best lag value in terms of both.

If we want to use this IC technique, one can perform the BIC, AIC, Hannan-Quin IC or Schwarz Criterion test on our data and select the most common suggested lag order. The `VARselect` function of the R package `vars` does precisely that, but it has two important drawbacks when dealing with high-dimensional series. The first one is huge execution times, represented in figure 2.3, and secondly, it will always recommend a lag value of 1 when more than 100 variables are passed.

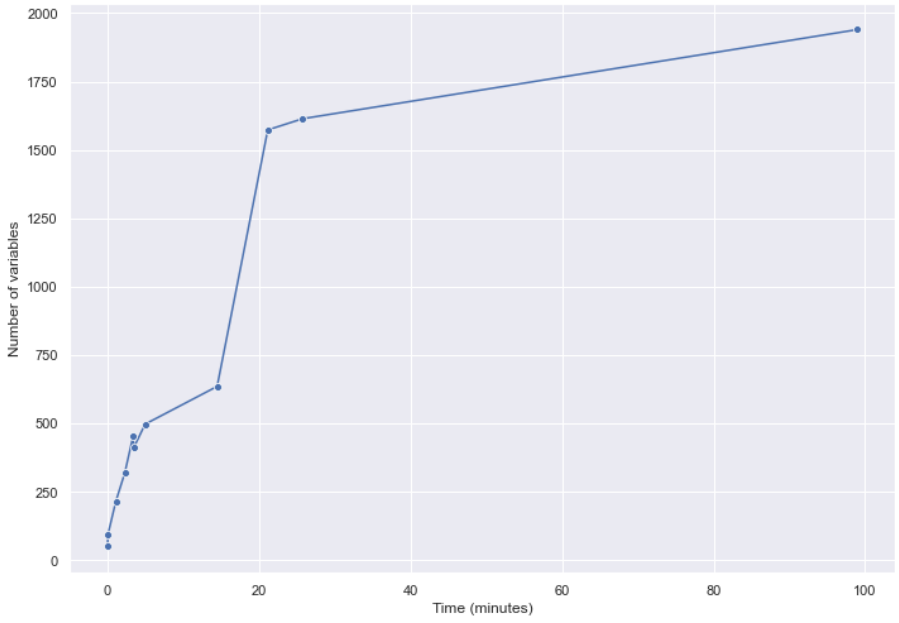

FIGURE 2.3: `VARSelect` execution time for both endogenous and exogenous variables with a max lag order of 52.

There isn't much literature around what lag value to pick when we are dealing with large amount of variables. Some packages, like `BigVAR`, suggest setting the maximal lag order based on the frequency of the data (e.g. 4 for quarterly, 12 for monthly, etc).

In the R language, the package `bigtime` is the only one who performs lag selection through cross-validation, described in section 2.2.1, as well as more hyperparameters.

2.3 XGBoost models

As previously said, this project does not aim to deeply understand the XGBoost logic. Instead, its role is more about for comparison purposes, since it does not purely take into account relationships within variables. Hence, we will explain a global overview of the algorithm in order to grasp the main ideas.

2.3.1 Decision trees

The decision trees paradigm is to split-up the space in various pieces and fit a model for each one of them, similar to the "divide and conquer" philosophy. In each piece

we set the value of a label and all the data falling in that slice of the space will be predicted as such.

A tree is composed by nodes. At each node, we test a value of a feature. These features are not user-defined, they are automatically developed by the algorithm. In figure 2.4 we illustrate a basic classification problem (separating two classes, one represented in blue points and the other one as red) solved by decision trees. The top graphic shows the first feature testing that enables the space partitioning by two: if a value is greater or lower than 3.

Then, we carry on with a second decision tree with another feature, in this case, if a value is greater or lower than 6. Thus, the remaining purple zone shown in the bottom figure contains values bigger than 3 and smaller or equal than 6. We can continue repeating the process till a certain convergence criteria, avoiding overfitting.

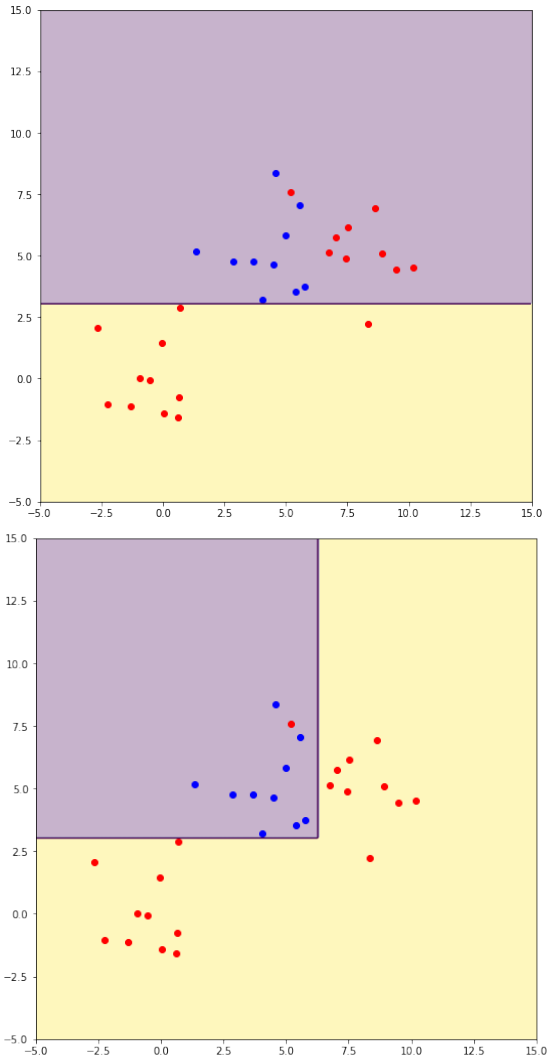

FIGURE 2.4: Graphic illustration of the changes of the classification boundary using decision trees.

2.3.2 Ensemble learning

Ensemble learning is the methodology of training a set of classifiers and then aggregate their results. The combination of different outputs of different models reduces the risk of choosing a poor performing one and enables us to perform a better-informed decision.

There are 3 main types of ensemble models: Bagging, Stacking and Boosting. Our XGBoost model falls into the last category, characterized for horizontally-aggregate each tree in a manner where each one of them "learns" from its predecessor, reducing the errors of the previous tree and updating the residual errors.

At the beginning, the generated trees are weak and barely outperform a random guesser. Nonetheless, some of them contribute with vital information, so we "boost" these useful learners combining one with another. At the end, we get a strong learner with hopefully low bias & variance.

Inside this category, XGBoost is a gradient boosting algorithm, meaning that this combination is performed in an iterative fashion using the Gradient descent optimization algorithm. Below we briefly explain what happens inside an XGBoost algorithm:

- Start with a random prediction. Compute the similarity score for this prediction for the *root* node using 2.6, which gets the sum of all the residuals of each node, squares the sum and then divides it by the number of residuals plus lambda, a L2 regularization parameter used on leaf nodes.

$$similarity = \frac{\text{sum of residuals}^2}{\text{amount of residuals} + \lambda} \tag{2.6}$$

- Compute the relative contribution or "gain" of the corresponding feature, by adding the similarities between the right and left branches and subtracting the root one:

$$Gain = \text{Left similarity} - \text{Right similarity} - \text{Root node similarity}$$

- Prune the tree from the bottom to the top. In order to do that we introduce the parameter gamma, which help us to determine what nodes must be pruned or dropped. If the result from subtracting gamma from the gain is zero or less, the node is eliminated. The higher the gamma, the more regularized our model will be.
- Compute the value of the final tree by evaluating the leftover nodes. Proceed again to compute another prediction using this new tree, and start all over the process. We do not end till we reach a user-defined number of trees or when the residuals of the Gradient Descent algorithm can't no longer be calculated.

The excellence of XGBoost relies on the creative, efficient and robust way of its implementation, specially due to its parallelization procedures. Apart, it offers lots of hyperparameters, which make it easy to adapt and personalize to any context or problem we are modeling.

For a more deep understanding of the XGBoost algorithm we recommend the reading of this introductory guide towards Gradient Boosting and the original paper (Chen and Guestrin, 2016) of the XGBoost release.

2.4 State of the art

In this section we summarize the current knowledge about the studied matter through the analysis of similar or related published work. The goal of the state of the art is to get an overview of what has been done in the field, stating the main differences with the present work.

The first ever concern for high-dimensional time series was (Tibshirani, 1996b), where they propose the use of regularization methods (mainly, Lasso) for forecasting multivariate time series. Following their work, (Yuan and Lin, 2006) presented new structured, grouped variants of Lasso and, years later, (Simon et al., 2013) introduced the sparse-group Lasso technique through ℓ_1 and ℓ_2 penalties.

Despite the high popularity of VAR models, there isn't much work in the past years for creating software able to model sparse, high-dimensional VARs with or without exogenous variables. The R package `glmnet` was the first framework for fitting regularization techniques into linear regression, but is not adapted for high-dimensional cases. (Davis, Zang, and Zheng, 2012) modified part of its code for creating penalized VAR models and added time-dependent constraints.

(Basu and Michailidis, 2015) studied ℓ_1 regularized estimates in two statistical problems in the context of time series, (i) stochastic regression with serially correlated errors and (ii) transition matrix estimation in vector Autoregressive (VAR) models.

(Gelper, Wilms, and Croux, 2016) is the first work that uses sparse VAR models with exogenous variables to perform sales forecast estimations taking in consideration demand effects. They also use Lasso ℓ_1 penalization.

In 2017 (Nicholson, Matteson, and Bien, 2017) presented the package `BigVAR` for R. BigVAR allows the estimation of high-dimensional time series with the possibility of applying structured penalties (Lasso and multiple more) on VAR models with or without exogenous variables.

All presented works base their penalties on standard Lasso procedures. Besides, BigVAR does not handle the lag selection problem stated in 2.2.3. It was the work of (Wilms et al., 2017) that presented for the first time hierarchical grouped Lasso penalization with hyperparametrization of lag values and λ sparsity regulators. In June 2021 they launched `bigtime`, an R package based on their work for sparse estimation of large time series models.

To the best of our knowledge, this master thesis is the first proposal of sales forecasting with exogenous variables comparison between an autoregressive approach and a gradient boosting technique.

2.5 Forecast Evaluation methods

While training a model is a key and dense step in every machine learning project, evaluating its performance and understanding its predictions is an equally important process. With a solid evaluation methodology we can decide to trust or not our

predictions.

In order to evaluate our models accuracy we will split our data into a training set and a test set. The way this data is splitted will be explained in a further section.

- **Training set**: subset of the original dataset used to train our models
- **Test set**: subset of the original dataset that contains unseen data. It will assess the performance of both models.

So, both XGBoost and VAR models are going to learn from the same training set data. Then, they will output a certain number of predictions of sales values for certain future weeks. The way we assess the accuracy of these predictions is through evaluation metrics, which compare the ground truth (our test set values) and the forecast.

2.5.1 Forecast Error (FE)

The simplest way to evaluate a model's performance is looking and the difference between the actual value (a) and the forecast value (f):

$$\text{FE}_i = a_i - f_i$$

The forecast error values are in the same unit as the input data, so, a FE of zero would mean a perfect prediction (no error). The FE is not typically used as an accuracy metric itself, but it's the base of all the following popular evaluation methods.

2.5.2 Mean Absolute Error

The Mean Absolute Error (MAE) is the arithmetic mean of the absolute forecast errors $|FE|_i = |a_i - f_i|$. This metric tells us the average degree of variation between our forecast values and the actual ones:

$$\text{MAE} = \frac{\sum_{t=1}^{N} |a_i - f_i|}{N}$$

Ideally, the smallest the MAE the better. MAE makes sense when it can be compared to something, because a MAE value of, say, 11.39, does not tell us much about how good is the accuracy of our model. In this case, MAE will be used for segment comparison, checking in which segments did we perform better.

2.5.3 Mean Squared Error & Root Mean Squared Error

The Mean Squared Error (MSE) tells us the average of the squared forecast error values. Squaring not only enables all FE values to be positive, but also to heavily punish larger errors rather than small ones.

$$\text{MSE} = \frac{\sum_{i=0}^{N} (a_i - f_i)^2}{N},$$

where N is the number of samples. A really bad prediction means a big FE, which is even more huge once it is squared. Hence, we are increasing the resulting mean, consequently rising the MSE score. Often, the square root of the MSE is taken, obtaining again the original scale of the data and smoothing the error values.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{N} (a_i - f_i)^2}{N}}$$

MAE and RMSE are quite similar and both are ideal for comparison purposes. Nonetheless, the MAE absolute value function is not differentiable at its minimum, while the RMSE one is. As a consequence, RMSE and MSE are more popular as loss function for algorithm convergence when training models.

2.5.4 Absolute Percentage Errors

APE's metrics are scale-independent. They give a solid idea of the relative error because they express the error as a percentage. The most popular one is the percentage version of MAE, the Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{100}{N} \sum_{t=1}^{N} |\frac{a_i - f_i}{a}|\% \tag{2.7}$$

Nonetheless, MAPE has some serious drawbacks:

- Due to the first-differentiation step that will be performed in the pre-processing part, explained in a following chapter, in all of our time series we will have negative values, and, sometimes, zero values as actual values. This operation "changes" meaning of zero values since we are arbitrary shifting the zero point. Now, divisions and ratios do not make sense. Also, MAPE assumes that the unit of measurement of the variable has a meaningful zero value and that all actual and forecast values are non-negative.
- Indeed, if our actual values are zero or close-to-zero, the division can't be done, results in "undefined". Adding a small value to the denominator is not recommended: dividing by very tiny number results in a huge final error.
- MAPE's value bigger than 100% can occur, losing interpretability.
- Finally, even if all of our a_i and f_i values were positive, another drawback of MAPE is that it punishes negative errors (when $a_i < f_i$) heavier than positive errors (when $a_i > f_i$), since the ratio of negative errors can be bigger than one and positive errors can't.
- As MAE, its absolute value function is not differentiable at its minimum.

In order to address some of this issues, we introduce two other types of APE's metrics that are more suitable for our context: MASE and MAAPE.

2.5.5 Mean Absolute Scaled Error

The idea behind MASE is the comparison between our model forecast and a "naive" forecast. Given a time-series $x_1, x_2, \ldots, x_N$, a naive forecast of step one $(s = 1)$ is

$\emptyset, x_1, x_2, \ldots, x_{N-1}$. In other words, we just shift s steps our original time series values and use it as forecast values.

Therefore, MASE is computed as following:

$$\text{MASE} = \frac{MAE_{model}}{MAE_{naive}}$$

The MAE of a naive forecast with step size s is simply:

$$\text{MAE}_{naive} = \frac{\sum_{i=s+1}^{N} |x_i - x_{i-s}|}{N - s}$$

Therefore, a MASE of 0.1 can be interpreted in the sense that our model is 90% more accurate than a naive forecast. A MASE bigger than one means that we are not as accurate as the naive forecast. Being more accurate than a "dummy" method is key for trusting our model and our predictions: it means we built something that really grasps the underlying behaviors of our time series. Also, MASE is immune to the zero-denominator problem of MAPE.

```
mase <- function(actual, predicted, step_size = 1) {

    naive_start <- step_size + 1
    n <- as.numeric(length(actual))
    naive_end <- n - step_size

    sum_errors <- sum(ae(actual, predicted))
    naive_errors <- sum(ae(actual[naive_start:n], actual[1:naive_end]))
    return(sum_errors / (n * naive_errors / naive_end))
}
```

LISTING 2.1: MASE implementation from the `Metrics` package in R. The `ae` function corresponds to the absolute error.

2.5.6 Mean Arctangent Absolute Percentage Error

In 2016 (Kim and Kim, 2016) proposed a new measure called Mean Arctangent Absolute Percentage Error (MAAPE), which sees the slope of the actual and forecast difference as an angle and not as a ratio, like MAPE does.

As we can see in Figure 2.5, the slope can be measured as a ratio of $|A - F|$, which can go from zero to infinity, or as an angle, ranging from 0 to 90º. This solves our zero division problem, but we have to take in consideration the following aspects:

- MAAPE is heavily insensitive against outliers. In table 2.1, we see two wrong observed actuals in Week 2: 100 and 10,000. Even though there's a huge variance between this two wrong values, the MAAPE of the forecast against "Actual 1" and "Actual 2" is barely the same. Thus, MAAPE is not recommended when there can be incorrect or mistaken measurements.
- When our actual value is exactly zero, MAAPE is always $\frac{\pi}{2}$, no matter what our forecast value is.

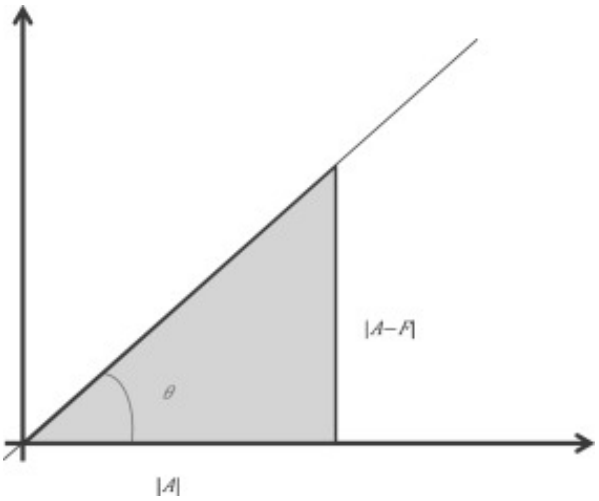

FIGURE 2.5: Conceptual justification of AAPE: AAPE corresponds to the angle θ, while APE corresponds to the slope as a ratio = tan(θ) = $|\frac{A-F}{A}|$, where A and F are the actual and forecast values, respectively. Extracted from (Kim and Kim, 2016).

TABLE 2.1: Illustrating the behavior of MAAPE against outliers with two incorrect actuals.

	Week 1	**Week 2**	**Week 3**	**Week 4**	**MAAPE**
Mistaken Actual 1	1	100	5	1	0.29
Mistaken Actual 2	1	10.000	5	1	0.31
Forecast	1	10	5	1	

2.5.7 R Squared score

The R^2 score or "Coefficient of determination" is also a popular metric that tells us how well the regression predictions approximate the actual values. A R^2 of 0.9 means that the 10% of the variability in the predicted data can be "explained" by the model.

This score relies on the residuals adding up to zero. In a linear regression context, this is guaranteed if we have a constant term (or drift in time series terminology) in the regressor matrix. But, in a Autoregressive context, this "drift" is eliminated by definition. Thus, R^2 is not recommended for time series forecasting.

2.5.8 Round or not to round

When trying to model a real life problem we usually face the issue of rounding or not our sales. In our context, we face this matter two times: when we interpolate our missing data (decimal values appear) and when we predict new sales values (most models output decimal values).

Since we are dealing with home-improvement and workshop "DIY" physical products, there is no such thing as "one hammer and a half" or "two point three drills". Therefore, we must address this concern when performing our evaluation, since our metrics values can change substantially if we previously round our values or not.

Ideally, if we want to check which model is the best, rounding to integer numbers is not much big deal. Nonetheless, in the business world, we do care a lot if we need to add to our stock one or two 8000€ rotating polisher machines, specially if we are a small company. We will test both procedures and explain the results in a following chapter.

2.5.9 Final remarks

As a conclusion, the big drawback of the MSE and RMSE metrics is the interpretability. Finding out that the RMSE of a model is 0.45 is not the same as finding out that the average forecast can be off by a, say, $\pm$15%. Thus, RMSE and MSE should be used for performance comparison and model selection, alongside MAE.

MAE and it's percentage error variants can be useful for understanding the scale of our prediction's errors, but they have to be supervised when we have near zero values or lots of outliers. Also, they can't be used for algorithm training convergence since their absolute value functions are not everywhere differentiable.

Chapter 3

The Dataset

In this section we describe the given dataset and explain some insights that should be reviewed before we start training the algorithms. We also detail about the performed data cleaning and the needed pre-processing for the algorithm input.

3.1 Dimensionality

Accenture gave me access to a private dataset of weekly sales of products for home improvement, gardening, and workshop. The data was already given to me completely anonymized. Each product is identified with a six digit integer, called "SKU". A set of SKU's form what is called a "segment", which is identified by a 5 digit integer.

The data was initially structured in 3 files:

- **"MD" (Master Data)**: should be considered as a dictionary to look up what segment corresponds to each SKU. It also contains the unused fields of Country, Group, Supplier and Brand.
- **Sales**: contains the sales of each SKU for certain weeks. Each sale value is an integer.
- **Promo**: contains the information about certain promotions that happened in a given week. Specifically, we have the discount applied to a specific product, the duration (in days) of the promotion, the channel of the promotion (for example, through e-mail, web or a catalog) and the physical position on a stand of the product during that week (front, inside, etc.).

The are a total of 7823 different SKU's (products) shared all over 40 different segments. We have exactly 765.023 records of sales. We have information about promotions applied to 6955 SKU's, so there are some SKU's without promotions. Specifically, we have a total of 245.979 rows of promotions applied.

Our data was given to us organized by segments. The criteria of the grouping is unknown, but we were advised to train them separately. There are mainly 3 aspects that must be taken in consideration in order to correctly train our algorithms:

- **Amount of SKU's per segment**: as we can see in Figure 3.1, each segment does not have the same number of SKU's.
- **Missing sales**: does each product have the same date span? Does each product have missing sales for a certain week?

- **Stationarity**: stationarity is mandatory because many statistical tests and models rely on it. In our case, is a constrain for our VAR model.

We discuss this issues in the following sections.

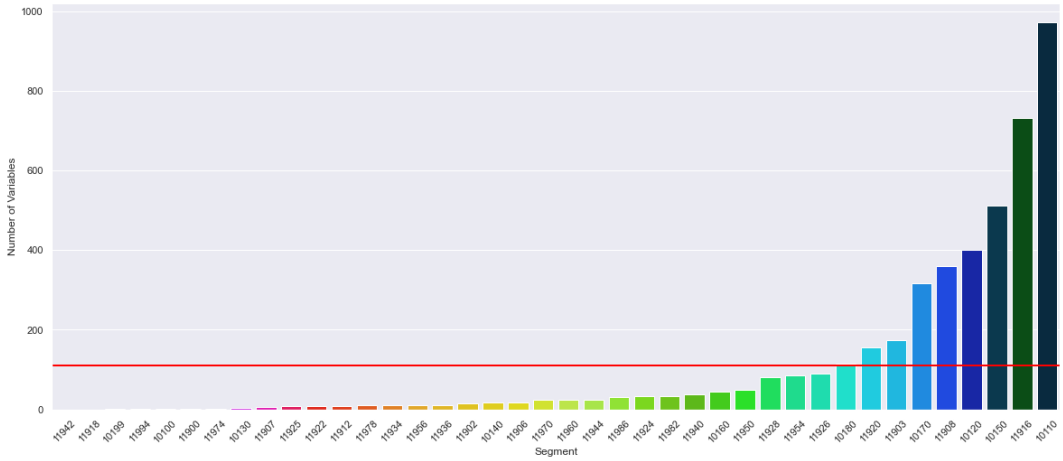

FIGURE 3.1: Distribution of the amount of products per segment. The red line corresponds to the mean of variables per segment: 110.47.

3.2 Data Cleaning

Each segment has a certain number of SKU's with its sales and different segments can have different number of SKU's. We decided that our exogenous variables per SKU will be two: the duration of the promotion, in days, and the discount applied to the SKU price, a positive integer. Therefore, if for example a segment originally has 17 SKU's, we will be adding 34 new exogenous variables. 2 segments were removed because all of its product did not have promotions.

Due to algorithm constraints, for each product we must have a continuous time-series without missing weeks. In other words, each SKU inside a segment will have the same amount of "rows" of sales. In order to do that, we will take the oldest and most recent date and we will build a date span with a weekly frequency, regardless if there is or not a sale for each week in the span. We consider that weeks start on Monday.

This organization opens up certain aspects that must be handled before the algorithm input. The first one is pretty clear: a segment must have at least 2 SKU's in order to test our hypothesis. The second one is more complicated: how do we handle the missing values.

3.2.1 Missing values in sales time-series

There are two reasons of why do we have a missing value on a sales time-series of a certain product, by origin or by its temporal order.

By origin

The first type of missing value is a native one: the original dataset did not have a value for that certain week (an outlier, a negative sale, etc.).

By its temporal order

The second type are those that appear when building the date span, because we are creating "new" weeks inside this span that did not have a sales value.

Regardless of the origin of a missing value, we also classify two other type of missing values given its topology. The first one is a missing value found when we already had a previous registry of a valid sales value. This type of missing values can be interpolated: we estimate the missing sales using a certain range of known past and future data points.

The other type are missing values in the beginning of a time series, i.e., missing values that appear before any kind of valid sales values. If this happens it could be for two reasons: a human or system error —it could not be possible to registry the amount of sales of a product— or simply the product did not exists in the beginning of the time series.

Neither VAR models or XGBoost models support missing values on its data input. Therefore, we are forced to apply a solution to eliminate missing values. Before we apply any kind of cleaning, we decided to design a criteria of when a time series should be cleansed of missing values.

This criteria is the following: if the 30% or more of the values of a time-series are missing values, regardless of its type, this product will be removed from the segment. After applying this constrain, two segments were rejected because they had zero valid products. After the date expansion, we encountered 442539 missing values, a 57.84% of the data points. This caused the elimination of the 43,29% of the products.

After this first cleaning, we have 27 valid segments and 4370 valid SKU's. The total amount of sales rows changed to 675.174. Now we are able to handle the missing values. As mentioned, the missing values "in the middle" of other valid series values are linearly interpolated. Each segment has the same date span (from 31/08/2015 to 06/08/2018) so every segment has exactly 154 rows (weeks) of data.

Linear Interpolation

Interpolation is the process of estimation unknown data points using known samples near those missing values. The simplest method of interpolation is the linear one, where we simply estimate our missing values by using the equation of the straight line passing through our last and next known points:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Where (x_1, y_1) is the latest known point and (x_2, y_2) is the next one. x denotes the index and y the sale value we are estimating.

Ideally, when working with sales prediction, each product should have the same records of sales. We know that this doesn't happen so those missing values that appear because the product did not exist or because of a human error will be replaced by zeros. In the future, a better transformation should be developed in order to tell the algorithm that a product is new.

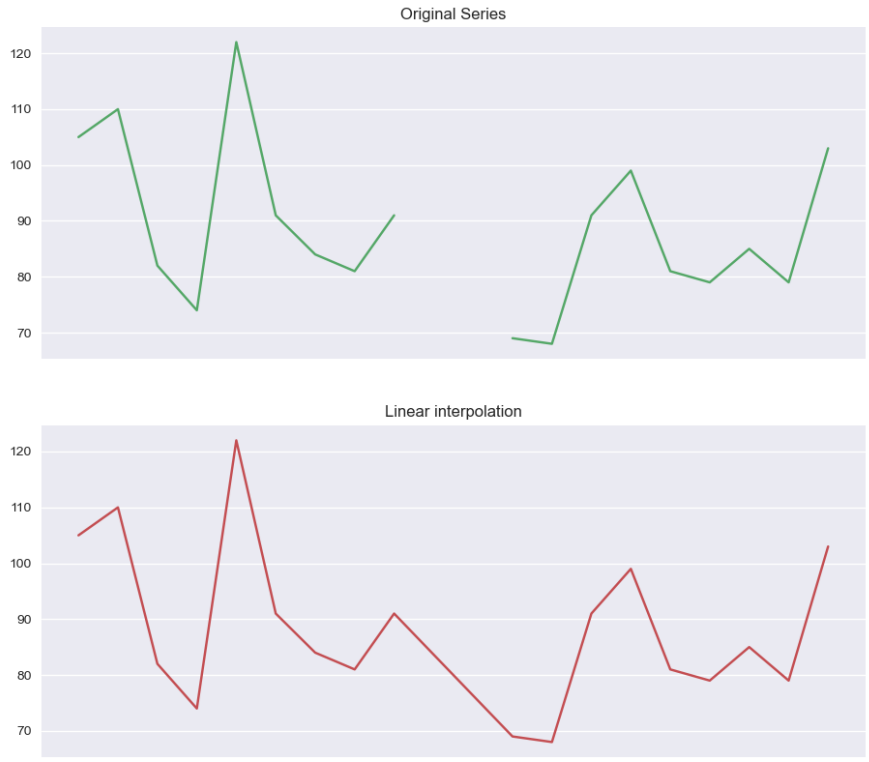

FIGURE 3.2: Linear interpolation of a time series with missing values.

3.3 Data pre-processing

When performing sales forecasting using a model, specially an statistical one, the imputed time series must comply with a series of properties that ensure correct, unbiased predictions. Those characteristics are stationarity, trend and seasonality.

3.3.1 Stationarity

The most important one is stationarity. A stationary time series is one whose data points do not depend on the time at which they were captured. Consequently, a non-stationary time series contains trends or seasonalities. If the process that recorded our sales is non-stationary, we need to transform it.

The first thing we must do is performing an Augmented Dickey–Fuller (ADF) test on each and every product sales in order to decide if we transform our sales values or not. An ADF is a statistical test which can tell us if our process characteristic equation has a root (monomial) equal to 1, also called unit root.

Having a unit root might be problematic because it shows a systematic pattern that is unpredictable, and, for example, we might be dealing with a spurious regression.

The more negative the p-value of the ADF test, the stronger the rejection of the hypothesis that there is a unit root (Kwiatkowski et al., 1992).

There are many ways to transform a time series in order to make it stationary. The simplest of them is performing a first differentiation, i.e., subtracting a t_{i+1} sales value to the t_i sales value. It has been proved that this pre-processing help stabilize the mean of a time series, reducing trend or seasonality patterns. Take in consideration that we lose the first row of sales information using this procedure.

The initial conducted test showed that 27,23% of SKU's sales records did not follow a stationary process. First differentiate a time series that has passed the ADF test does not affect the algorithms performance, so we made the decision of first-differentiating all the time series even if they have passed or not the test. After conducting the transformation, only 3 variables did not pass the test.

It's easy to undo the first differentiation of a time series: we just need to recall the first row of data and perform the cumulative sum of our differentiated points. The `diffinv` function in R, for example, does that for us.

3.3.2 Standardization

The second pre-processing performed is the scale of our time series. The reason behind this transformation is to achieve better performance in both VAR and XGBoost models. It has been proven that standardized data (zero mean and unit standard deviation) helps removing trend and facilitates most algebraic computations of machine learning algorithms.

In order to normalize our data we subtract to each data point by the mean (μ) and divide by the standard deviation (σ) of our time series:

$$\hat{X} = \frac{X - \mu}{\sigma} \tag{3.1}$$

In order to recall the original value, we just have to inverse the process:

$$X = (\hat{X} \cdot \sigma) + \mu \tag{3.2}$$

As recommended by the library `bigtime`, we normalize our endogenous variables (product sales) as well as the exogenous ones (promotion). We can use, for example, the `scale` function in R. As mentioned before, a product might have no applied promotion, therefore it's variables of duration and discount are all zeros. This kind of exogenous variables have zero standard deviation, so we can't apply equation 3.1. In that case, we just don't apply the standardization.

The data is now prepared to be ingested by the algorithm, but, as mentioned in section 2.5, we need to split each segment into Train & Test sets. We decided that both XGBoost and VAR models will forecast a window of 4 periods ahead, or, in other words, we will predict the sales of each product for the 4 following weeks (a month).

FIGURE 3.3: Illustration of the overall pre-process transformation pipeline.

Therefore, each train set has exactly 150 rows of weekly sales and each test set has

4. After the data cleaning, 23 segments remain. It seems like the `bigtime` can't build a VARX model with more than $\approx$ 2.197 endogenous + exogenous variables. Thus, segment 10110 with 971 unique products and segment 11916 with 732 unique products were dropped.

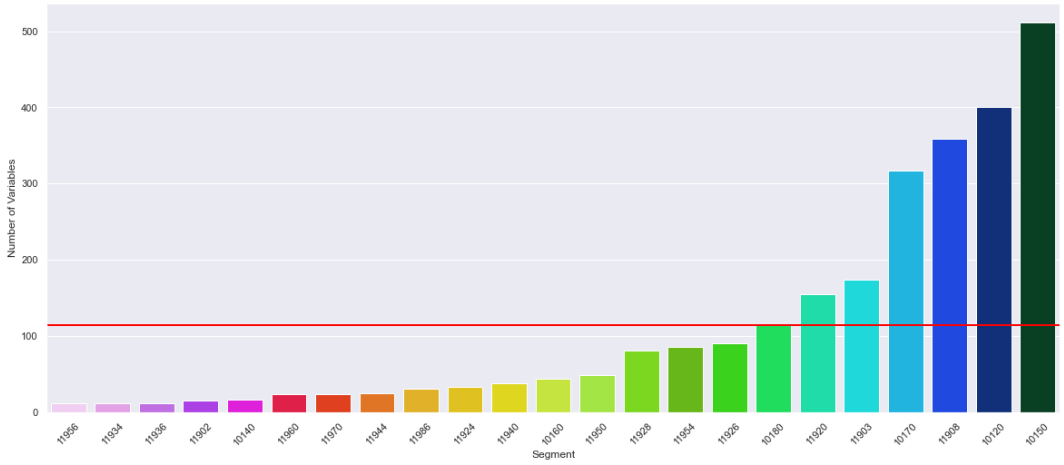

FIGURE 3.4: Distribution of the amount of products per segment after the data cleaning & pre-processing.

The smallest segment has only 11 products, and the biggest one has 511. As we can see in Figure 3.4, the average number of SKU's per segment is 113.73. The median is 44. In Figure 3.1 we can see a piece of a segment with 3 first-differentiated example sales of 3 products alongside its discount and duration variables.

TABLE 3.1: Tabular view of a possible data input after the first-differentiation process on the SKU sales.

Date	**Sales_ 322471**	**Sales_ 322472**	**...**	**Discount_ 322471**	**Discount_ 322472**	**...**	**Duration_ 322472**	**Duration_ 322472**
31/8/2015	19	2	...	30	30	...	7	7
7/09/2015	-16	-1	...	0	0	...	0	0
14/09/2015	24	7	...	15	15	...	7	7
21/09/2015	-6	-24	...	0	0	...	0	0

Chapter 4

Implementation

In this section we describe our dataset of sales, the data cleaning and time series pre-processing done, how did we built both algorithms and the overall architecture of the framework.

4.1 Architecture

This project was coded using mainly the R programming language, as advised by Accenture. Nevertheless, I decided to use Python for certain processes of data cleaning and time series pre-processing. Both Python and R deal flawlessly when managing big data structures and both have strong and efficient open-source packages.

This framework uses mainly six libraries:

- **bigtime**: R package that enables the sparsely estimate of large time series Vector AutoRegressive with Exogenous Variables (VARX) models.
- **xgboost**: optimized and distributed gradient boosting library able to implement machine learning gradient boosting models.
- **data.table**: memory-efficient, fast and concise R package that enhances the R's `data.frame` function.
- **Pandas**: same as `data.table`: provides a high-performance, easy-to-use data structures and analysis tools but for Python.
- **ISOweek**: a framework of useful date-related functions such as format converter to week dates.
- **Metrics**: a collection of machine learning evaluation metrics for measuring regression, classification and ranking performance.

The overall framework of both algorithms is done using Jupyter Notebooks and a R project. A Jupyter Notebook is a useful, easy-to-use and visual web tool that allows to create and share documents with live code, equations, narrative text, etc. The VARX model is develop in one R Notebook whilst the XGBoost model is encapsulated inside an R Project, designed to be opened using RStudio. The overall data pre-processing is explained and coded inside a Python Jupyter Notebook.

4.2 Pipeline of the framework: VARX model

We can divide the algorithm implementation in 4 steps:

- **Data processing**: how is the data read and what operations are needed for the correct algorithm imputation.
- **Model building**: how is the model created and its configuration.
- **Prediction**: how do we compute the predictions of our forecast.
- **Evaluation**: how do we measure the performance of our model and its predictions.

As mentioned, all this procedures are coded in R. We designed a Jupyter Notebook that encapsulates all this steps, called `Training & Evaluation.ipynb`.

4.2.1 Data processing

The algorithm assumes that the input data is cleaned and pre-processed as explained in section 3.2. In order to be read, we saved each segment Train and Test sets in `.csv` files called `<segment_id>Train.csv` and `<segment_id>Test.csv`. Nonetheless, we also saved each original Train and Test sets (before making the data stationary and standardized), needed for the model performance evaluation once the predictions have been re-scaled.

The algorithm reads the file names from a specified directory and extracts the segment id using regex. Then, reads the processed Train and Test files alongside the original Test set. The `bigtime` library needs the endogenous and exogenous variables apart, so we must create this two matrices from the Train file. In order to perform this distinction, we extract the number of endogenous variables of a segment using regex, since we know that all the column's with sales values are named `Sales_<SKU_id>`.

Once we built these two matrices, we standardize the data (including the test set, since we need it for the evaluation step) using the `scale` function in R. We did not perform this step before because we need to know the mean and standard deviation of each SKU, so we can re-scale later on using equation 3.2. There might be multiple products without promotions, so their exogenous variables (columns of zeros) have zero standard deviation and zero mean. Such variables do not need to be standardized, as mentioned in section 3.3.2.

4.2.2 Model building

Even though in early stages of this project the library `BigVAR` was used to create VARX models, we opt to use the library `bigtime` because it has better interpretability and uses the hierarchical sparsity pattern mentioned in section 2.2.2. The function call is the following:

```
sparseVARX(
  Y,
  X,
  p = NULL,
  s = NULL,
  VARXpen = "HLag",
  VARXlPhiseq = NULL,
  VARXPhigran = NULL,
  VARXlBseq = NULL,
  VARXBgran = NULL,
  VARXalpha = 0,
  h = 1,
  cvcut = 0.9,
  eps = 10e-3,
  selection = c("none", "cv", "bic", "aic", "hq"),
  check_std = TRUE
)
```

LISTING 4.1: sparseVARX function documentation and its default values

Where Y is the matrix of endogenous variables, X is the matrix of exogenous variables, p and s are the maximum lag orders for X and Y, *VARXpen* is the penalization structure desired, arguments 7 to 10 are user-specified vectors that in this context we do not use, *VARXalpha* enables us to specify a custom regularization value for the equation 2.4, h is the forecast horizon, *cvcut* is the data proportion used for the model estimation using Cross-Validation, *eps* is the tolerance convergence in the proximal gradient algorithm, *selection* specifies the model selection desired and *check_std* is to enable or disable the warning that checks if whether X or Y are standardized.

We adapt this function call as following:

- p and s are set to NULL, meaning that we let the Cross-Validation estimation compute them.
- *VARXpen*, as mentioned in 2.2.2, is set to *"HLag"* instead of *L1*.
- h will change for each model, from 1 to 4.
- *selection* is set to *cv*, i.e., Cross-Validation.
- *check_std* is set to FALSE because we already ensure that the data is standardized.

The rest of the parameters are set to default.

Our goal is to predict the sales of each product for the next 4 weeks, so our forecast horizon, called h, is 4. In order to be as accurate as possible, we built 4 models, each with a unique value of h from $h = 1, \ldots, 4$. We built an array of dimension 4 that contains on each position each model with its certain forecast horizon. A `bigtime` fitted VARX model class returns multiple arguments, the most important of them are the following:

- k and m: the estimated best lag order for Y and X.
- p and s: the estimated best maximum lag order for Y and X.
- *Phihat* and *Bhat* the section 2.2 matrices of estimated endogenous and exogenous autoregressive coefficients.

- *lambdaPhi* and *lambdaB*: matrix representing the sparsity parameter grid for Y and X.
- *lambdaPhi_opt* and *lambdaB_opt*: optimal sparsity parameter for Y and X.
- *MSFEcv*: Mean Squared Forecast Error (MSE) of the Cross-Validation scores in a two-dimensional sparsity grid.

Given that training 4 models is a long-lasting process, the array is stored in a `.rds` file[1] after the fit of all the models. Thus, if we ever want to use a concrete segment model we can read this file instead of training all over again.

4.3 Prediction

`bigtime` has the function `directforecast(fit, h)` which returns the predictions of all the endogenous variables of the given model "fit" in a given forecast "h". Thus, we iterate our array of models calling successively this function with each value of h, storing the predictions in a $4 \times N$ matrix (being N the number of endogenous variables).

4.4 Evaluation

In this step we do not evaluate the performance of each model separately, we assess the accuracy of the whole prediction matrix created in the last step. The evaluation is done in two times: one without re-scaling and without reverting the stationary process of the predictions, for model selection, and the other one reverting, for business scale.

We decided to compute the following evaluation metrics: RMSE, MAPE, MAAPE, and MedianAPE. Not all are going to be used for model selection criteria; some of them are just for showing the behavior of the algorithms and to get a better understanding of our data.

Some of them are computed using the library `Metrics`, and, by default, when two matrices are passed, the evaluation is done comparing product by product row by row, i.e., its done "weekly". We are also interested in knowing the monthly error, i.e., computing the sum of each column for both train and test sets and then call these metrics with both vectors of dimension $1 \times N$, being N the number of variables.

In our data we have actual sales that are zero or near-zero. Therefore, MAPE and MedianAPE are not computed using `Metrics` since we would get infinite or undefined values. For showing purposes, we use the denominator trick presented in section 2.5.4 and therefore they had to be coded "by hand".

Once we computed the specified weekly and monthly metrics with the "transformed" data, we revert the predictions (i) re-scaling using 3.2 and (ii) performing the cumulative sum to remove the first-differentiation. Now we have the predictions in the original scale, and therefore we call again the same evaluation metrics but using the original test values.

Each value of the mentioned evaluation metric is saved inside a `.csv` file alongside the corresponding segment and the time needed to train the model. At the end, we

[1] files that store a single R object using the base function `saveRDS`.

have a table illustrating the error span of each metric for each segment. This table format facilitates the discovery of hidden insights and conclusions.

Additionally, we developed two functions that enable the evaluation of each saved model, so we do not have to train again a model if we want to try a new metric, for example. Also, they allow the test for rounding or not of the sales predictions before the evaluation part, thanks to the parameter `rounding`.

4.5 Pipeline of the framework: XGBoost model

XGBoost is a relatively young library, it was introduced in 2016 and nowadays we can find it implemented for the most popular programming languages like C++, Java, Python, R, Julia, Perl, Scala... Not only that, it can run on distributed environments like Hadoop and Spark.

We are going to use the R implementation of XGBoost. Specifically, Accenture proposed of using it's own XGBoost implementation, an R project used in multiple internal processes that was adapted by my project advisor. Hence, this code is property of Accenture and we will limit ourselves to briefly explain the overall pipeline and how is it been adapted by this project.

We can divide this explanation in 4 steps:

- **Data input**: how did we adapt our transformed data for the needed format of the algorithm.
- **Model settings**: how did we customize the algorithm to adapt it to our context.
- **Prediction & Data output**: how did we manage the predictions and the post processing of the data, what transformations are necessary in order to evaluate the predictions.
- **Evaluation**: description of how is performed the forecast evaluation.

4.5.1 Data input

In order to correctly evaluate the performance of our VAR and XGBoost model, each model must be trained using the same data. Consequently, the raw files that XGBoost read are the same as the VAR model, explained in section 4.2.1. Nonetheless, the format of the imputed data for our XGBoost model varies a little bit from the VAR model. We designed a `preprocess` function that transforms our data in XX processes:

- Converts a full date into ISOWeek format, e.g., from "2015-08-24" to "2015-35".
- Converts the original dataset to "long" form, also known as "melting" the dataframe, using the `dcast` function in R. Table 4.1 illustrates this transformation, whith "ID" representing the SKU.

4.5.2 Model settings

In order to use the mentioned multiple custom parameters that the XGBoost offers by nature, Accenture's designed a configuration system through an Excel file. This way, the user does not need to modify parts of the code in order to adapt it to a new context.

TABLE 4.1: Long format example after the melting procedure in the XGBoost pre-processing part. Look at the original table 3.1 for comparison purposes.

Date	ID	Sales	Discount	Duration
31/8/2015	322471	19	30	7
7/9/2015	322471	-16	0	0
14/9/2015	322471	24	15	7
...	...	...	...	...
31/8/2015	322472	2	30	7
7/9/2015	322472	-1	0	0
14/9/2015	322472	7	15	7
...	...	...	...	...

In this file we can find more than 40 parameters with multiple options. The most important and the ones modified for our problem are the following:

- **Data location**: the folder path where the algorithm reads the input data.
- **ID_cols**: the name of the column containing the sales variable names. In our case, "SKU" is already renamed to "ID".
- **sales_column**: the name of the column containing the sales values. In our case, "SALES".
- **freq_unit**: the periodicity of our sales data. In our case, "week".
- **h**: forecast horizon. In our case, "4", meaning 4 periods (weeks) ahead.
- **max_depth**: the maximum depth[2] of the generated decision trees.

4.5.3 Prediction & Data output

The modified Accenture's version of XGBoost trains two models: a classification one and a regression one. The potential of both are used in order to predict the sales of each product in each of the 4 future dates. The forecasting is done in an iterative fashion, SKU by SKU and period by period.

By default, the original code outputs a `forecast_<SKU>.csv` file in long format, containing the original sales by SKU mixed with the predictions, alongside unused, useless columns. This format doesn't facilitate the evaluation of our forecast so we decided to modify the original code with the function `postprocess`.

This function reads the default output file and simply extracts the computed predictions into another column called "FORECAST", which is empty for all the other 150 periods. The reformed file is stored again as `forecast_postprocess_<SKU>.csv` and is more suitable and ready for evaluation operations. Table 4.2 illustrates an example of this file.

4.5.4 Evaluation

While the R project of Accenture's XGBoost is a powerful, compact and flexible tool, it does not include evaluation methods. Hence, we coded the evaluation part in a R

[2] the number of decisions or "splits" that the tree can make before computing a prediction.

TABLE 4.2: Forecast post-process example file

ID	DATE	SALES	FORECAST
118803	2015-35	87	
118803	2015-36	83	
...	...	...	...
118803	2018-28	67	
118803	2018-29		70.67077
118803	2018-30		79.95464
118803	2018-31		83.71971
118803	2018-32		95.5824

Jupyter Notebook called `Evaluate XGBoost.ipynb`, more suitable for different tests and it doesn't require the re-training of the models.

This notebook is similar to the `Training & Evaluation.ipynb` notebook of the VARX model. In fact, they share functions such as `find.endogenous.index`, used to count the number of sales variables within a segment, `scaler`, used to scale a time series and return its values alongside the computed standard deviation and mean and all the set of custom evaluation metrics.

We begin creating the evaluation results files and specifying the path containing all the results of each segment forecast. Then, we iterate each file (or segment) performing the following procedure:

- Read the test files of the corresponding segment. Store also the scaled version of them.
- Find out the number of endogenous and exogenous variables of the segment. Standardize accordingly each time serie and exogenous variable.
- Locate the predictions of the XGBoost model. Pass the actual & predicted values to the evaluation function and register the result in the previously created files.
- Undo first the scaling of the predicted values and then the first differentiation. Repeat again the evaluation step with the original actual and the transformed predictions.

At the end we get the same result file as the previous VARX model. This file is again ready and suitable for conclusion extraction and graphic representation of the forecasted time serie. For test purposes, we also coded a small function that performs the evaluation of the model trained with original or "raw" values, i.e., with the series without any kind of pre-processing.

Chapter 5

Experiments, Results and Discussion

In this section we describe the evaluation of our different models. We performed all the model comparison using only the dataset mentioned in Chapter 3. This evaluation is done in two ways: looking at the error in the pre-processed measure (for model selection purposes) and looking at the error once the predictions are re-scaled.

5.1 VAR model evaluation

We will start by presenting the RMSE errors, sorted by smallest Weekly RMSE to biggest.

		Standardized		Re-escalated	
Segment	Variables	Weekly RMSE	Monthly RMSE	Weekly RMSE	Monthly RMSE
11924	33	0.4	0.43	1.21	3.07
10140	16	1.02	1.24	22.37	70.86
11944	25	1.36	2.02	18.11	58.9
11920	155	1.67	2.6	10.47	33.17
11956	11	2.03	3.18	8.11	18.38
10160	44	3.17	1.51	10.53	23.15
11928	81	3.2	5.32	36.99	71.86
10180	114	3.85	5.49	32.81	117.73
11936	11	3.93	4.37	838.74	3118.53
11940	38	3.93	5.13	33.43	100.62
11986	31	4.25	5.63	26.96	64.09
10150	511	4.44	7.87	42.01	159.71
11960	23	5.28	8.5	118.9	377.97
11908	359	5.62	8.21	42.01	89.28
11903	174	6.39	6.67	55.98	164.05
11954	85	6.92	8.53	52.45	89.8
11934	11	12.38	23.58	125.88	292.87
10170	317	13.14	11	87.45	284.43
11970	23	13.69	14.55	118.22	275.33
11950	48	14.93	8.04	100.31	109.54
10120	401	20.48	20	126.19	462.64
11902	15	42.16	61.08	298.87	698.43
11926	90	84.27	159.05	690.08	988.64
Total Sum		258.51	374.00	2898.08	7673.05
Average		11.24	16.26	126.00	333.61
Median		4.44	6.67	42.01	109.54

TABLE 5.1: RMSE errors for the Vector Autoregressive model.

We can observe that there is no correlation between the number of exogenous variables of a segment and their RMSE. As expected, RMSE values are higher when the predictions are re-escalated (specially with outliers of segments 11926 and 11936). Nonetheless, remember that we are using this metric for model comparison.

Let us switch to the MAAPE:

Type	Period	Average	Median
Standardized	Weekly	78.95%	79.3%
	Monthly	77.95%	78.64%
Re-escalated	Weekly	38.69%	39.04%
	Monthly	32%	31.66%

TABLE 5.2: Summarized MAAPE values for the Vector Autoregressive model.

Here, as opposite to the RMSE, the re-escalated predictions are more accurate than the standardized ones. This behavior can be explained due that in the RMSE we are computing the absolut error, while in MAAPE we are talking in terms of relative error, where we are dividing the relative error by the actual value.

In "bussiness scale", the mean of all MAAPE monthly values for each segment are 32% deviated with respect to the actual sales. On the other side, 77.96% of the escalated monthly values are deviated from the actual values. Even though predicting weekly sales of more than 11 different products is nowadays as hard as predicting the stock market, the goal of this project is algorithm comparison, so we will not purely focus on forecast accuracy.

To demonstrate that our VAR model is not as bad as it looks like, below we can present the resulting MASE values, which tell us how good are our model predictions in comparison with a naive forecast (section 2.5.5):

Type	Period	Average	Median
Standardized	Weekly	0.57	0.57
	Monthly	0.62	0.63
Re-escalated	Weekly	0.65	0.57
	Monthly	0.32	0.21

TABLE 5.3: Summarized MASE values for the Vector Autoregressive model.

Where we observe that, on average, our forecast is 37 to 43 times better than a naive forecast for the standardized predictions. For the re-escalated ones we improve a

little bit, achieving till 79 more accuracy than the naive forecast. Let's move on now to the segment comparison part with the MAE metric:

		Standardized		Re-escalated	
Segment	Variables	Weekly MAE	Monthly MAE	Weekly MAE	Monthly MAE
11924	33	0.17	0.27	23.64	88.89
10140	16	0.65	0.89	16.71	51.7
11944	25	0.81	1.33	16.27	62.2
11920	155	0.9	1.42	6.7	14.74
11956	11	1.02	2.07	32.62	115.27
10150	511	1.3	2.56	182.63	400.69
10160	44	1.36	0.99	13.79	47.53
10180	114	1.46	2.53	19.45	72.15
11928	81	1.54	2.68	156.65	455.88
11940	38	1.77	2.42	6.56	20.89
11936	11	2.02	3.05	20.12	51.53
11986	31	2.04	2.59	0.78	2.25
11908	359	2.09	3.67	77.07	196.13
11960	23	2.15	3.94	22.41	50.69
11954	85	2.45	3.89	18.93	52.08
10120	401	2.45	3.67	43.24	162.58
11903	174	2.56	3.16	439.9	1514.65
11950	48	3.5	3.62	4.73	15.05
11970	23	3.77	5.61	24.07	48.83
10170	317	3.93	4.66	36.34	72.36
11934	11	4.69	12.62	12.43	42.63
11926	90	11.82	27.57	13.72	42.47
11902	15	13.05	25.57	60.45	192.09
Total Sum		67.50	120.78	1249.21	3773.28
Average		2.93	5.25	54.31	164.06
Median		2.04	3.05	20.12	52.08

TABLE 5.4: Summarized MAE values for the Vector Autoregressive model.

If we recall from section 2.5.2, MAE is the mean absolute error between our predictions and the actual values. The closer to zero, the better. MAE helps us identifying in which segments did we built the best model. As we can see in Table 5.4, MAE values for the standardized predictions move around ≈2.0, but segments 11926 and 11902 considerably shift the mean due its high MAE values. In consequence, we must be careful when trusting these two segments model predictions.

If we recall Figure 5.1, we can identify a direct correlation between the RMSE and MAE values. As mentioned before, MAE and RMSE are metrics that do not mean much by themselves: they are useful once they are compared with another model results.

Lastly, in Table 5.5 we demonstrate some of the mentioned drawbacks of the MAPE metric applied to the monthly re-escalated predictions. As explained, in MAPE we divide by the actual sale value. Thus, if this number is close to zero, the resulting MAPE value will be extremely big. This behavior is highlighted in red.

Segment	Variables	Monthly MAPE
11924	33	$5.66 \cdot 10^{14}$%
10140	16	14.43%
11944	25	$3.48 \cdot 10^{15}$%
11920	155	24.08%
11956	11	59.76%
10160	44	31%
11928	81	63.97%
10180	114	43.5%
11936	11	20.89%
11940	38	39.16%
11986	31	39.31%
10150	511	46.95%
11960	23	16.79%
11908	359	29.72%
11903	174	41.17%
11954	85	21.41%
11934	11	$2.63 \cdot 10^{16}$%
10170	317	42.35%
11970	23	25.88%
11950	48	55.77%
10120	401	58.82%
11902	15	149.15%
11926	90	36.09%
Total Sum		860.20 *
Average		43.01% *
Median		39.24% *

TABLE 5.5: Monthly re-escalated MAPE values for the Vector Autoregressive model. Highlighted values represent the zero-denominator problem and they are ignored for the calculus of the total sum, average and median.

In addition, if we look at segment 11902 we can observe the interpretability problem. MAPE is a percentage but it can take values higher than 100%. A Mean Absolute Percentage Error of 149.15% is confusing and does not help with the comparison of all the other "standard" MAPE results.

5.2 XGBoost model evaluation

In order to present the results of the XGBoost we will follow the same arrengment as the VAR model, comparing both results and pointing the main differences. Let's start then with the RMSE:

	Standardized		Re-escalated	
Segment Variables	Weekly RMSE	Monthly RMSE	Weekly RMSE	Monthly RMSE
11924	0.71	1.29	6.12	19.68
11956	2.83	4.23	32.78	93.97
11920	3.93	8.48	95.15	318.08
10150	6.54	15.82	144.7	491.16
10160	6.92	12.93	89.5	299.71
11944	7.41	19.01	266.09	998.14
11903	7.85	14.19	356.39	1404.04
10140	8.84	24.21	525.98	1907.31
11928	9.18	21.58	626.36	2276.32
11986	9.2	24.52	311.34	966.25
11960	9.49	24	1054.19	3055.16
10180	10.68	26.32	707.24	2737.98
11908	10.95	24.23	730.04	2520.23
11954	11.87	21.59	487.28	1762.33
11940	13.88	36.82	639.08	2482.81
11950	18.87	29.28	545.54	1664.66
11970	21.56	63.76	3078.58	11426.85
10170	33.85	66.32	3628.78	11226.43
10120	41.44	99.56	2241.68	8779.67
11934	42.72	93.05	9784.67	28387.29
11902	63.07	122.09	25510.41	86613.08
11936	88.72	250.05	57181.25	183334.58
11926	303.98	739.94	12882.15	44043.69
Total Sum	734.49	1743.27	120925.30	396809.42
Average	31.93	75.79	5257.62	17252.58
Median	10.68	24.23	626.36	2276.32

TABLE 5.6: RMSE errors for the XGBoost model.

If we look at the standardized results, the overall sum of errors and the RMSE mean and median are higher with respect to the VAR model errors, illustrated in Table 5.1. Of all these increases, highlight the weekly and monthly standardized mean, which increased (in overall) a 284.12% and 466.11% more respectively.

If we switch to the re-escalated results, the XGBoost fails again when trying to overcome the VAR model accuracy, specially in the re-escalated results.

Type	Period	Average	Median
Standardized	Weekly	96.84%	95.7%
	Monthly	121.19%	122.95%
Re-escalated	Weekly	87.65%	85.05%
	Monthly	94.83%	92.42%

TABLE 5.7: Summarized MAAPE values for the XGBoost model.

As we can see in Table 5.7, the XGBoost predictions are a lot worse when evaluated with the MAAPE metric. In all 4 cases the errors are bigger than in the VAR model, reaching values even bigger than one in the monthly standardized evaluation.

This behavior is also reflected with the MASE metric. As mentioned, a MASE value bigger than 1 means that our forecast is worse than a naive version of a forecast predictions. As we can see in Table 5.8, the overall MASE metrics are much bigger than one.

Type	Period	Average	Median
Standardized	Weekly	2.04	1.24
	Monthly	4.58	2.94
Re-escalated	Weekly	10.7	5.65
	Monthly	5.46	3.45

TABLE 5.8: Summarized MASE values for the XGBoost model.

Finally, we present the results of the MAE metric of our XGBoost model in Table 5.9.

Type	Period	Average	Median
Standardized	Weekly	9.64	4.58
	Monthly	32.16	13.88
Re-escalated	Weekly	2417.30	260.61
	Monthly	9630.70	1031.11

TABLE 5.9: Summarized MAE values for the XGBoost model.

As we can see, again the mean and median for all 4 possible combinations is worse than in the VAR model. Given that our goal is algorithm comparison and not segment comparison, we won't show the MAE error of each segment since they are all worse than the VAR model. Specifically, the lowest increase is a 224.51% from the weekly standardized median.

5.3 Rounding Evaluation

Before jumping to a conclusion we performed the same evaluation process rounding our actual and forecast values to integers, as explained in section 2.5.8. Below we report the following insights:

- In any metric the XGBoost model outperformed the VAR model when rounding the datasets, in both the standardized or re-escalated evaluations.
- All metrics, for both models and both standardized and re-escalated modes, reported barely the same errors after the round evaluation. There's one exception: the standardized MAAPE values of the VAR model, represented in Table

5.10. Again, due to the calculus of relative errors where we divide by the actual value, rounding made our MAAPE errors 40% to 23% lower. We did not observe the same behavior with the XGBoost model.

Evaluation	Period	Average	Median
Without rounding	Weekly	78.95%	79.3%
	Monthly	77.95%	78.64%
Rounding	Weekly	38.14%	39.89%
	Monthly	52.65%	55.63%
Difference	Weekly	40.81%	39.41%
	Monthly	25.30%	23.01%

TABLE 5.10: MAAPE differences on the standardized evaluation with or without rounding

In Table 5.1 we can observe the differences between the XGBoost and VAR models graphically, were we represented the sales of the last 3 months of the product 365955 from the segment 11924. In the last 4 weeks we can see the predictions from the XGBoost and VAR models alongside the ground truth, represented in green.

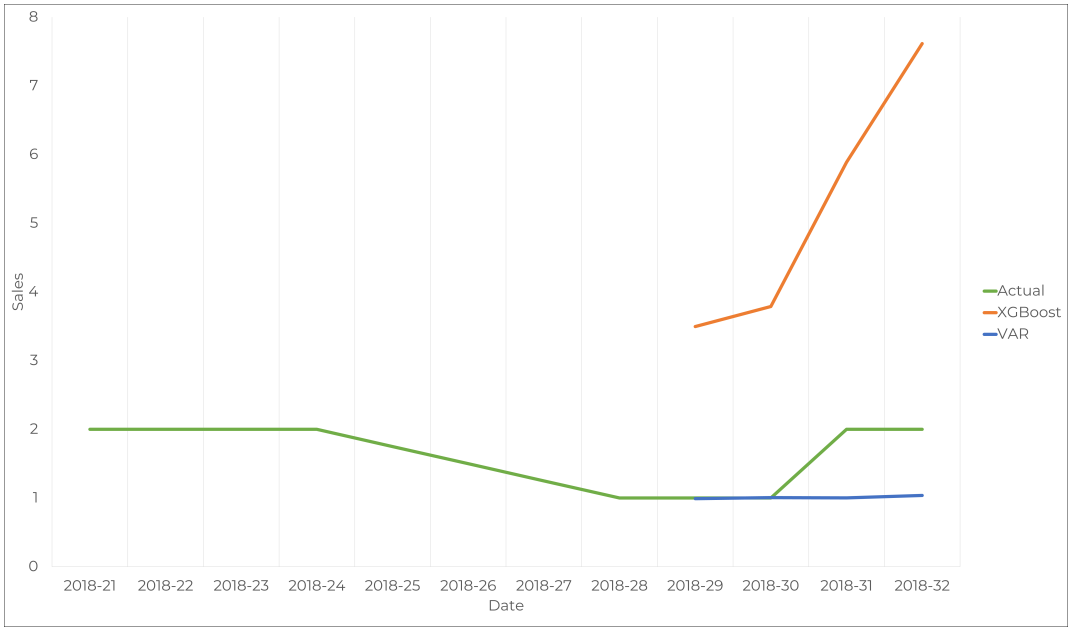

FIGURE 5.1: Graphic demonstration of the poor XGBoost performance in comparison with the VAR model for a concrete product sales prediction.

5.3.1 Execution times

Leaving apart the evaluation results, here we present how much time did each model need for training for each segment for each algorithm. As expected, the VAR model is considerably slower than the XGBoost, because it falls inside the category of a statistical model. In fact, we can see in Table 5.11, we needed 3.43 days or 82.43 hours to train all 23 segments. Here we can see the effectiveness of the parallelization of the XGBoost library, who just needed 3.58 hours to train all the models.

Model	Total minutes	Average mintues
VAR	4946.03	215.04
XGBoost	402.07	17.48

TABLE 5.11: Summary of the execution time needed for training both algorithms.

Chapter 6

Conclusions and future work

This project deals with the forecasting of a high-dimensional dataset of sales records and promotion variables from a huge number of products from a home improvement, gardening, and workshop retailer. The first goal of this project has a global nature and was to prof that the relationships between products and the underlying associations between sales that happen in a defined span of time affect the quality of predictions. In order to do that, our objective was to built two different sales forecasting algorithms and compare its accuracy and performance: a Vector Autoregressive model, specialized in detecting this kind of product connections, and a XGBoost model, which is not. Still, XGBoost has proved to be best performing algorithm when working with structured or tabular data.

In addition, another goal of this work, more specific, was to overcome the challenge of predicting sales of a high number of products (endogenous variables) and taking into account the hypothetical promotions made by the company (exogenous variables).

Accenture gave us a high-dimensional dataset with multiple segments (sets of products) with their weekly sales number and the presence, or not, of promotions (discounts) and their value. We set the target of predicting one-month ahead sales, i.e., 4 weeks of forecast. For this purpose, with the use of popular machine learning and data science tools, we built a framework that allows to perform the overall cycle of model comparison: data cleaning, data pre-processing, model building, model training and model evaluation with the needed data post-processing.

The first step towards this goal was a deep research of time series analysis and its forecast, focusing on evaluation tools suitable for our context (since we wanted a fair comparison of two very different algorithms) and its pre-processing (since our data was not cleaned).

The second step was the profound understanding of Vector Autoregressive models and how can we adapt them to work with huge datasets, since the XGBoost implementation is already capable of handling big data.

Once we had accomplished these two key objectives, we evaluated our models following two different approaches: with the predictions standardized (zero mean and unit standard deviation) and without, also known as in "business scale". For each one of them, we assessed the error in two different ways: weekly error and monthly.

Using the "Root Mean Squared Error (RMSE)" metric, we found that in both ways the Vector Autoregressive model is more accurate than the XGBoost model, showing that relationships between products affect a forecast performance. In order to prove

that our results were valid, we tested more interpretative forecast metrics and we still found that the VAR model got the lead. We also assessed accuracy between segments, as well as performance fluctuations if we round or not the model's sales predictions.

Even though our main goal has been accomplished, some ideas that would lead to clearer conclusions if we keep addressing the problem are:

- Testing more types of time series pre-processes, like logarithmic operations, centering our values (subtracting the mean and not fully standardizing) or performing spline or polynomial interpolations on missing values.
- Addressing differently the problem of a new product appearance in the middle of our date span. That is, building a method that is able to tell the algorithm that some products didn't exist in certain weeks, instead of setting their sales to zero.
- Find a better conversion between pre-processed predictions and "business scale". One approach would be performing the logarithm of our sales alongside the first-differentiation, so our estimators can be considered percentages of sales variations.

Bibliography

Basu, Sumanta and George Michailidis (2015). "Regularized estimation in sparse high-dimensional time series models". In: *The Annals of Statistics* 43.4. ISSN: 0090-5364. DOI: 10.1214/15-aos1315. URL: http://dx.doi.org/10.1214/15-AOS1315.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

Davis, Richard A., Pengfei Zang, and Tian Zheng (2012). *Sparse Vector Autoregressive Modeling*. arXiv: 1207.0520 [stat.AP].

Gelper, Sarah, Ines Wilms, and Christophe Croux (2016). "Identifying Demand Effects in a Large Network of Product Categories". In: *Journal of Retailing* 92.1, pp. 25–39. ISSN: 0022-4359. DOI: https://doi.org/10.1016/j.jretai.2015.05.005. URL: https://www.sciencedirect.com/science/article/pii/S0022435915000536.

Hilde, CB (2000). "VAR Models in Macroeconomic Research". In: *Statistics Norway Research Department*.

Kim, Sungil and Heeyoung Kim (2016). "A new metric of absolute percentage error for intermittent demand forecasts". In: *International Journal of Forecasting* 32.3, pp. 669–679. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2015.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0169207016000121.

Kwiatkowski, Denis et al. (1992). "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" In: *Journal of Econometrics* 54.1, pp. 159–178. ISSN: 0304-4076. DOI: https://doi.org/10.1016/0304-4076(92)90104-Y. URL: https://www.sciencedirect.com/science/article/pii/030440769290104Y.

Nicholson, William, David Matteson, and Jacob Bien (2017). *BigVAR: Tools for Modeling Sparse High-Dimensional Multivariate Time Series*. arXiv: 1702.07094 [stat.CO].

Niraj, Rakesh, V. Padmanabhan, and P. Seetharaman (Mar. 2008). "Research Note—A Cross-Category Model of Households' Incidence and Quantity Decisions". In: *Marketing Science* 27, pp. 225–235. DOI: 10.1287/mksc.1070.0299.

Simon, Noah et al. (2013). "A Sparse-Group Lasso". In: *Journal of Computational and Graphical Statistics* 22.2, pp. 231–245. DOI: 10.1080/10618600.2012.681250. eprint: https://doi.org/10.1080/10618600.2012.681250. URL: https://doi.org/10.1080/10618600.2012.681250.

Tibshirani, Robert (1996a). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178.

– (1996b). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178.

Wilms, Ines et al. (2017). *Interpretable Vector AutoRegressions with Exogenous Time Series*. arXiv: 1711.03623 [stat.ML].

Yuan, Ming and Yi Lin (Feb. 2006). "Model Selection and Estimation in Regression With Grouped Variables". In: *Journal of the Royal Statistical Society Series B* 68, pp. 49–67. DOI: 10.1111/j.1467-9868.2005.00532.x.