MULTIVARIATE PRODUCT DEMAND FORECASTING WITH NEURAL NETWORKS

by

Çağatay Şahin

B.S., Computer Engineering, Bilkent University, 2014

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2019

MULTIVARIATE PRODUCT DEMAND FORECASTING WITH NEURAL NETWORKS

APPROVED BY:

Prof. Taflan Gündem (Thesis Supervisor)	.....
Assoc. Prof. Arzucan Özgür	.....
Assist. Prof. Reyhan Aydoğan	.....

DATE OF APPROVAL: 12.06.2019

ACKNOWLEDGEMENTS

I am grateful to my thesis advisor, Prof. Taflan İmre Gündem for his guidance and many enlightening discussions throughout my thesis work. I have learned a great deal from him.

I would like to thank Assoc. Prof. Arzucan Özgür and Assist. Prof. Reyhan Aydoğan for accepting to participate in my thesis committee.

I would like to thank Migros IT department for sharing the anonymous sales dataset to use in this study.

Finally, I would like to express my deep gratitude to all the members of my family for providing me with extraordinary support and encouragement throughout my education. As the culmination of my academic studies until now, my master's thesis is dedicated to them.

ABSTRACT

MULTIVARIATE PRODUCT DEMAND FORECASTING WITH NEURAL NETWORKS

In retail, there are plenty of use cases that would benefit from predicting the future amount of product sales. Those use cases include cash flow management, campaign execution and inventory planning, all of which are the crucial components for the success of any retail business. Quantitative time series analysis is widely applied for predicting the product demand. It includes a set of well-established methods from statistics and econometrics. However, their capabilities are constrained by certain assumptions and they require careful statistical treatment on the data before the application. Artificial Neural Networks are powerful class of machine learning models which have shown outstanding success on the unstructured data. We proposed five different mathematical formulations to prepare and select hierarchical multivariate time series data to feed into a Long-Short Term Memory network. We referred to the formulations (or "schemes") as (1) *Uni*, (2) *Uni-Store*, (3) *Uni-Product-Pcc*, (4) *Uni-Product-Mi*, (5) *Multi-Store*. Each of them groups the product sales signals in multiple stores according to various association criteria to forecast the sales amounts. In the experiments, the mutual information (3) and correlation (4) based schemes demonstrated poor performance presumably due to the small number of selected products. *Uni* scheme produced the models that resulted in the minimum loss. The *Multi-Store* scheme produced the models that can be trained approximately three times faster than than that of the other schemes. Its average forecast error was not significantly higher than that of the *Uni* scheme.

ÖZET

YAPAY SİNİR AĞLARI İLE ÇOK DEĞİŞKENLİ ÜRÜN TALEP TAHMİNİ

Perakendecilikte, ürün satışlarının gelecekteki miktarını tahmin etmenin fayda sağlayacağı birçok senaryo vardır. Bunlar, her biri perakende sektöründe başarı için çok önemli olan nakit akışı yönetimi, kampanya yürütme ve envanter planlama gibi süreçleri içermektedir. Nicel zaman serisi analizi, ürün talebini tahmin etmek için yaygın olarak uygulanmaktadır. Nicel analizler, istatistik ve ekonometriden gelen bir dizi yerleşik yöntem içermektedir. Ancak, yeterlilikleri belirli varsayımlarla sınırlandırılmıştır ve uygulanmadan önce veriler üzerinde dikkatli şekilde bir takım istatistiksel işlemleri gerektirmektedir. Yapay Sinir Ağları, yapılandırılmamış verilerde olağanüstü başarı gösteren güçlü bir yapay öğrenme modeli sınıfıdır. Bu çalışmada, Uzun-Kısa Süreli Bellek ağına girdi olarak verilecek hiyerarşik çok değişkenli zaman serisi verilerini hazırlamak ve seçmek için beş farklı matematiksel formülasyon önerdik. Formülasyonları (veya "şemalar") sırasıyla; (1) *Uni*, (2) *Uni-Store*, (3) *Uni-Product-Pcc*, (4) *Uni-Product-Mi*, (5) *Multi-Store* olarak adlandırdık. Her biri, satışların miktarını tahmin etmek için çeşitli ilişkilendirme kriterlerine göre birden çok mağazadaki ürünün satış sinyallerini gruplandırmaktadır. Deneylerde, karşılıklı bilgi (3) ve korelasyon (4) temelli şemalar muhtemelen seçilen ürünlerin miktarının az olmasından dolayı düşük performans gösterdi. *Uni* şeması asgari tahmin hatası veren modelleri ortaya çıkardı. *Multi-Store* şeması, diğer şemalardan yaklaşık üç kat daha hızlı eğitilebilen modeller ortaya çıkardı. Bu şemanın ortalama tahmin hatası *Uni* şemasınınkinden önemli ölçüde yüksek çıkmadı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

a_{pst}	Product sales amount
$\boldsymbol{a}_{ps}^{T}$	Signal
e	Euler's number
h	Number of associated products
I(.;.)	Mutual Information
L	Number of lagged observations
M_p	Multivariate LSTM model
M_{ps}	Univariate LSTM model
N	Number of samples
P	Number of products
r_{xy}	Pearson correlation coefficient
$\mathbb{R}$	Set of all real numbers
$\mathbb{R}_{\geq 0}$	Set of all nonnegative real numbers
s_x	Sample standard deviation
S	Number of stores
T	Number of observed timepoints
U	Number of features
V	Number of targets
$\boldsymbol{x}_l^{(n)}$	Input vector to LSTM
$\mathbf{x}^{(n)}$	Input matrix
$\mathcal{X}$	Feature set
$\boldsymbol{y}^{(n)}$	Target vector to LSTM
$\mathcal{Y}$	Target set
$\sigma(.)$	Sigmoid function
$\psi(.)$	Digamma function

LIST OF ACRONYMS/ABBREVIATIONS

ANN	Artificial Neural Network
AR	Autoregressive
ARCH	Autoregressive Conditional Heteroskedasticity
ARIMA	Autoregressive Integrated Moving Average
BPTT	Backpropagation Through Time
CPU	Central Processing Unit
EEG	Electroencephalogram
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
LSTM	Long-Short Term Memory
MI	Mutual Information
MLP	Multilayer Perceptrons
RELU	Rectified Linear Unit
RNN	Recurrent Neural Network
SARIMA	Seasonal Autoregressive Integrated Moving Average
tanh	hyperbolic tangent function

1. INTRODUCTION

Time series analysis is used in a wide range of fields including the climate studies, agriculture, transportation engineering, medicine, stock markets and retail. Predicting seasonal crop production contributes to economic planning and establishing global food security [1]. In transportation engineering, analyzing the historical data lets the engineers respond traffic congestion [2] and lets the commuters choose their daily route. In medicine, tracing EEG signals in seconds is a vital indicator of whether a patient is having a seizure or not [3]. Stock prices change daily, hourly or even within milliseconds depending on the market dynamics [4]. Forecasting the stocks can bring direct monetary return. Demand forecasting is extensively applied in particular use cases of retail business, for example, making purchasing decisions, budget planning, staffing, marketing campaigns and advertising [5]. Moreover, meeting the customer demand on time is crucial for a merchant since failing to do so is equivalent to abandoning certain income from products. Perhaps more severely, it would cause customer dissatisfaction and increase the risk of customer turnover [6]. Thus, one goal of a merchant is to make sure the available collection of products is no less than their respective demand at any time. On the other hand, keeping the products in a stockroom incurs various maintenance costs, staff cost and logistic costs. Products which belong in specific categories such as food are easily perishable and must be sold before their expiration date [7, 8]. Another goal of a merchant is, therefore, to avoid redundant stock of items. In this light of trade-off, the ability to forecast demand for items is valuable for inventory planning as well as managing cash flow.

There are quantitative methods which include a large set of statistical models and machine learning models for demand forecasting. On the other hand, qualitative methods are helpful for the same task when there is lack of available historical data. One popular qualitative method called Delphi Method [9], which is based on expert opinions. Quantitative methods are likely to outperform qualitative methods as the historical data grows. In addition, they are convenient for automation of large systems. Consider the following scenario; a retailer wants to monitor the transaction data of

thousands of products and receive alerts when there is a high probability of stockout for particular item. If the retailer can accomplish this in short periods and with acceptable forecast performance, it gains a significant competitive advantage over the rivals. This task is infeasible for a group of people regardless of their experience, because there are excessive amount of items to perform forecasts. Still, considering the capability of modern computer infrastructure, it is possible to implement a system which would accomplish the task in the previous scenario. Therefore, another advantage of the quantitative approach is that we can programmatically apply elaborate methods to deal with massive datasets and build automated tools for decision making.

2. RELATED WORK

2.1. Statistical Models

Time series data (or "signal") consist of several components. The most prominent components examined in analyses are trend, seasonality, cyclical component, and noise. Trend is a relatively long-term increase or decrease in the time series data. A trend component of a signal can be linear or non-linear. The direction of trend can change throughout the time. Seasonality is the regular fluctuations which occur in fixed and known period of time. The periods can be days, months, years, etc. Cyclical component also represents the repeated fluctuations; but its period is not fixed and on average it is greater than the seasonality period. Noise (or "irregular component") represents random, irregular effects. In Figure 2.1, the time series decomposition for daily sales data is displayed. Between day-40 and day-60 the observed sale amounts (Observed) suddenly increase and decrease. The changes are reflected on the trend in a smoother manner. Retail sales data is likely to show strong weekly patterns. In the seasonality plot, there is a noticeable valley-like structure, which suggests that certain days in a week are preferred over the other days. Residual errors are what is left when subtracting trend and seasonality from the observed data. Note that this decompositon example does not consider the cyclical component.

In order to characterize the statistical properties of time series data, many modeling techniques exist in the literature. A well-known class of statistical techniques for modeling the magnitude of time series is Autoregressive Model (AR). It is a regression model that uses the linear combinations of the previous observations, also known as lagged variables to predict an output variable [10]. Autocorrelation plots are commonly used to depict the dependency between the lagged variables. Figure 2.2 shows an example autocorrelation plot. Each variable is corelated (see Section 3.1.1 for the correlation formula) with the previous observations' lagged values to some degree. The y-axis is the degree of a relationship with an observation in a time series with observations at previous time steps. The AR model assumes that the time series data is

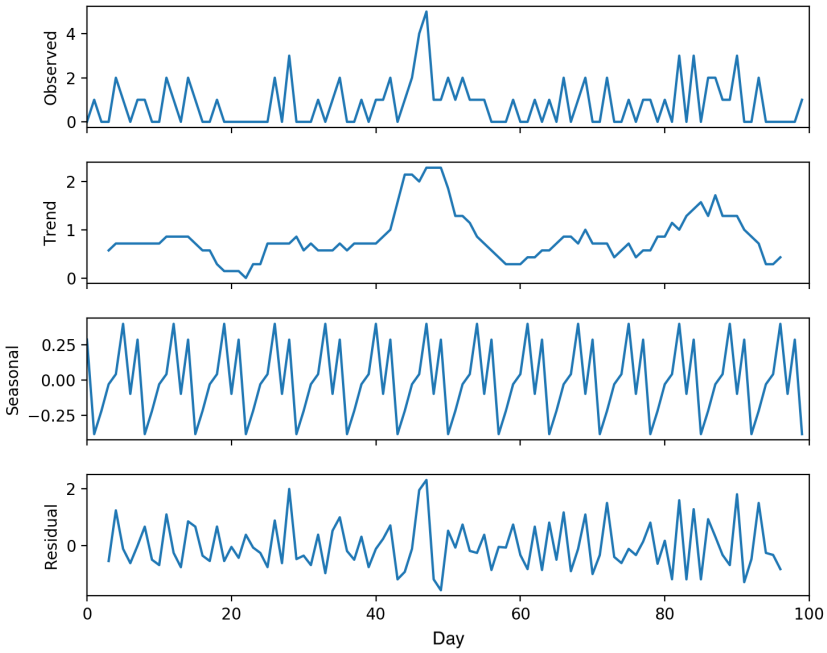


Figure 2.1. Decomposition plot of a product sale amount signal over 100 days

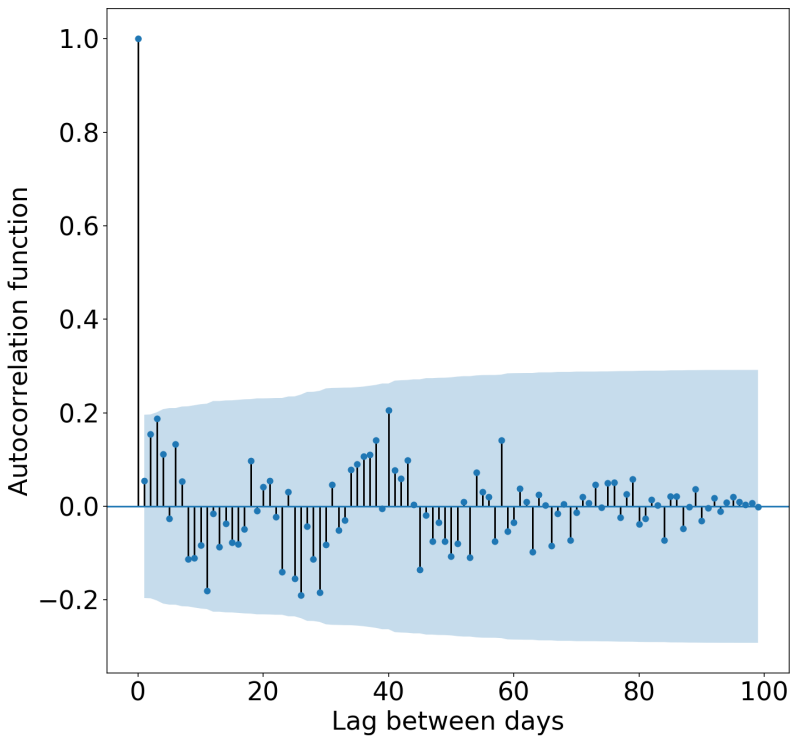
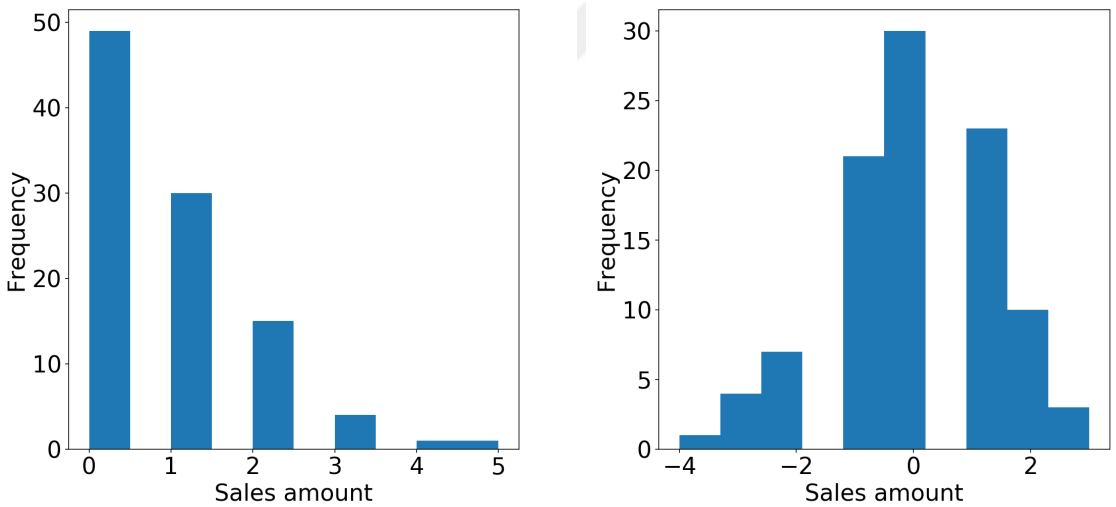


Figure 2.2. Autocorrelation plot of a product sale amount signal over 100 days

stationary. A stationary process has constant mean, variance and autocorrelation over time. Time series with trends or seasonality are non-stationary. Differencing, logging or log-differencing are the example techniques to clean a time series data from trend and seasonality. Log-based techniques are rather successful for the exponentially growing data than the linear data. One property of log-differencing is the symmetry. For example, if one days' log difference is 0.1 and the next days' log difference is -0.1, it makes the value return to initial point. However, without log-differencing 10% increase and 10% decrease does not bring the value to the initial value. Figure 2.3 shows the frequency of the sale amounts of the product in the previous figures. 2.3(a), Figure is the distribution obtained from the original signal which is similar to decaying exponential distribution. In Figure 2.3(b), having differenced the signal, the distribution of the frequencies looks similar to a left-skewed Gaussian distribution.



(a) Original signal (b) Differenced signal

Figure 2.3. Frequency of values in a product sale amount signal over 100 days

The two extensions of the AR model are Autoregressive Integrated Moving Average (or "ARIMA") [5] and Seasonal Autoregressive Integrated Moving Average (or "SARIMA") [2] models. ARIMA introduces initial differencing step to make the time series stationary. Secondly, it dynamically uses the residuals to correct the forecasts. A residual is the difference between the predicted and the observed value of a variable. The assumption is that the residuals in the regression model hold valuable information about the underlying process which generates the data. SARIMA further supports the

seasonal component. In addition, Autoregressive Conditional Heteroskedasticity (or "ARCH") [11] and its generalized variant "GARCH" [12] are used for modeling the volatility which is degree of variation in time series data. It is extensively applied in stock markets because people are especially interested in the returns (i.e. the money made or loss) due to the changes in a stock rather than the value of the stock itself. Overall, the statistical models require lots of empirical observation and pre-processing on the time series data.

2.2. Artificial Neural Networks

Artificial Neural Networks (or "ANNs") allow flexible ways of estimating complex relationships between variables. They have gained popularity in recent years due to their outstanding success on complex tasks on unstructured data, particularly; image recognition, voice recognition and machine translation. They are inspired by the structure of biological neural networks in the brain that "learns" by experience without the need for programming. An ANN consists of the connected group of nodes (or "neurons"), each of which has associated weights and represents a function. In simple terms, a node takes the outputs of its predecessor nodes and outputs the weighted average of them followed by a transformation called activation function. Then the output is fed into the next group of nodes. Together, they can express very complex functions that can simulate the human brain. Indeed, by universal approximation theorem, an ANN can express any function under mild assumptions [13].

The activation functions are used to transform the range of the output value of a node and contribute the expressive power of the ANNs. Some of the most common activation functions are the sigmoid function, hyperbolic tangent function, softmax function and rectifier unit function. A sigmoid function has a typical "s-shaped" curve, which is also called a sigmoid curve. Equation of the sigmoid (σ) function is:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Hyperbolic tangent function is a member of hyperbolic functions which share the properties of the trigonometric functions. Exponential definition of the hyperbolic tangent (tanh) function is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

Softmax is a useful function for multi-label classification problems. It essentially turns the real-valued vectors into probabilities so that their sum will be one. Softmax is defined by the formula:

$$f_i(\vec{\boldsymbol{x}}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_i}}, \text{ where } i = 1, \ldots, J \tag{2.3}$$

Rectified Linear Unit or ("ReLu") is a commonly preferred activation function that has drawn attention upon the study of Hinton et al. [14]. ReLU deals with overfitting by introducing sparsity into the network. The other main advantage of ReLU is the relatively low likelihood of vanishing/exploding gradients while training the network (see Section 2.2.1 for the details). It is also less computationally expensive than sigmoid, tanh and softmax because it involves simpler mathematical operations. The equation of ReLU is:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.4}$$

Preliminary architecture for neural networks are Multilayer Perceptrons (MLPs). They contain sequential layers of variable-sized nodes. Each member of the layer takes the previous layers's inputs and delivers its outputs to the nodes in the next layer. The first layer of MLP is referred to as the input layer and the last layer of MLP is referred to as the output layer. Then, any layer in between input and output layers is called the hidden layer. As the names suggest, the input layer takes the input data and the output layer outputs the target variable(s). In a classification problem, each node i

in the output layer represents the probability that an instance belongs to the class i. Then the node giving the maximum value determines the class of the sample.

In a univariate regression problem, the output layer consists of only one node and the activation function is the identity function $f(x) = x$. In other words, the activation function of the final node is simply omitted to get a real number. As in many other machine learning models, a loss function is defined. The objective is then to minimize the loss function by updating the weights of each node using gradients. The backpropagation algorithm proposed by Hinton [15] solves this problem by making forward and backward passes. There are several methods to optimize the stochastic gradient update process and one of the most effective optimizer is the Adam optimizer [16].

There is a plethora of methods proposed to combine the neural network units in the literature [17,18]. The most popular architectures other than MLPs can be classified in two main groups, these are the Convolutional Neural Networks and Recurrent Neural Networks. Recurrent Neural Networks are used for sequential data such as audio signals and [19] and convolutional neural networks are used for multidimensional data such as images [20]. In the following section, we briefly review the sequential neural network architecture and its extension called the Long-Short Term Memory [21] that we used in our study.

2.2.1. Recurrent Neural Networks

Recurrent Neural Networks (or "RNNs") are powerful model for sequential data. It includes an internal state that acts as a sort of "memory" for the model. Unlike MLPs, the data is fed sequentially into the RNNs step by step; and the output in each step depends on the internal state received from the previous step. In that way, the decision for the current output is affected by the data seen so far. It implicitly creates the dependence assumption between the input variables, so that the order of the input variables matters. For instance, knowing the order of a sentence increases the chance

for predicting the next word. Given a sequence

$$\mathbf{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T) \tag{2.5}$$

RNN returns an output

$$\mathbf{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_T) \tag{2.6}$$

The equation of RNN is:

$$\boldsymbol{h}_t = \begin{cases} \boldsymbol{r} & \text{if } t = 0 \\ \phi(\boldsymbol{W}^{(h)}\boldsymbol{x}_t + \boldsymbol{U}^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(h)}) & \text{if } t > 0 \end{cases} \tag{2.7}$$

$$\boldsymbol{y}_t = \rho(\boldsymbol{U}^{(y)}\boldsymbol{h}_t + \boldsymbol{b}^{(y)}) \tag{2.8}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{e\times 1}$ is the input vector at time t; $\boldsymbol{y}_t \in \mathbb{R}^{f\times 1}$ is the output vector at time t; $\boldsymbol{h}_t \in \mathbb{R}^{d\times 1}$ is the hidden state vector at time t; $\boldsymbol{W}^{(h)} \in \mathbb{R}^{d\times e}$, $\boldsymbol{U}^{(h)} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{U}^{(y)} \in \mathbb{R}^{f\times d}$ are the weight matrices; $\boldsymbol{b}^{(h)} \in \mathbb{R}^{d\times 1}$, $\boldsymbol{b}^{(y)} \in \mathbb{R}^{f\times 1}$ are the bias term vectors; ϕ and ρ can be any activation function applied element-wise. Generally nonlinear activation functions are preferred. $\boldsymbol{r}$ is initialized to any random vector. Nevertheless, it is suggested that configuring $\boldsymbol{r}$ as a learnable parameter similar to the weights would improve the performance of RNNs [22]. They are able to handle variable sized input sequences and inherently deep since the hidden state is propagated forward throughout time. Thus, each step can be imagined as a hidden layer. Indeed, a RNN can simulate any turing machine by having rational number weights [23]. Figure 2.4 shows the RNN diagram.

A slightly modified version of backpropagation algorithm is used to train RNNs. Because of the hidden state, the gradients are determined by the calculations from both the current step and the previous steps. Thus, to calculate the gradient at step

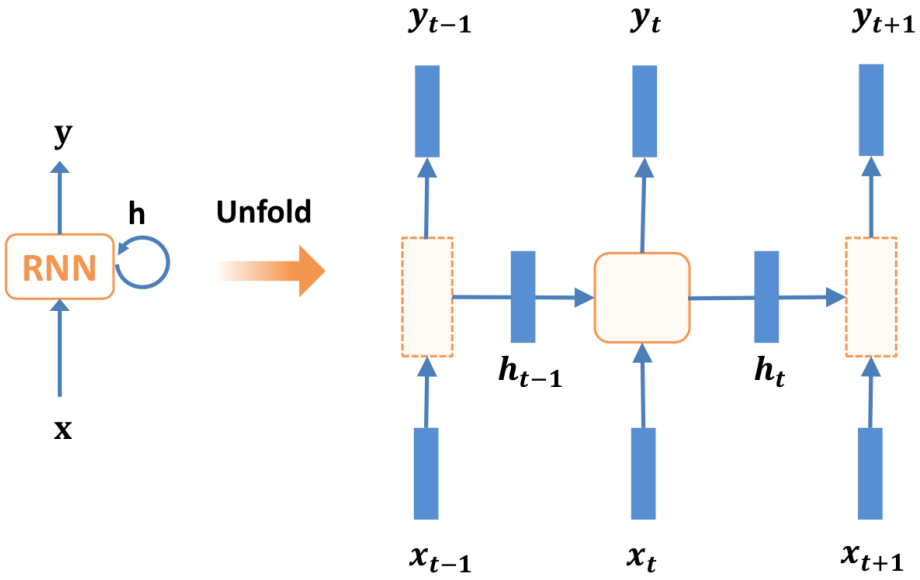


Figure 2.4. RNN diagram, folded and unfolded over time. Vectors are represented by solid rectangles

t, the algorithm backpropagates $t-1$ steps until the first step an sums each gradient. The algorithm is called Backpropagation Through Time (BPTT). Although BPTT is an elegant way to learn the sequential data, there is one significant drawback called exploding/vanishing gradients. Specifically, the gradients of the network's output with respect to the parameters in the early layers gets remarkably large or small. The problem is fundamental with gradient based learning methods including the backpropagation algorithm. It is related to certain activation functions and becomes more problematic as the depth (i.e., number of hidden layers) of the model increase. Therefore, RNNs for long sequences are likely to suffer from this problem.

One remedy for vanishing gradient problem is using rectified linear unit activation function. ReLU introduces sparse regularization effect on the network. The other approach is to use extended structures of RNNs; such as Gated Recurrent Unit [24] and Long-Short Term Memory. In the next section we go over the Long-Short Term Memory and explain how it solves the vanishing/exploding gradient problem.

2.2.2. Long-Short Term Memory

Long-Short Term Memory (or "LSTM") networks are introduced by Hochreiter & Schmidhuber [21]; and now they are widely used in sequence learning applications. Similar to vanilla RNN, LSTM also have a chain-like structure, and each element of the chain is called a cell. A cell is the abstraction of several connected nodes. Also, two cells are connected by not only the hidden state, but also another vector called the "cell state". Specifically, a cell at step t, takes the hidden state and cell state from the cell at step $t-1$, does calculations and then feeds its hidden state and cell state into the cell at step $t+1$. Inside an LSTM cell, there are several gates. A gate is simply a layer of neural network nodes with an activation function. Gates act as regulators of the information flow within the cell. In the original paper, the LSTM cell contained input gate $\boldsymbol{i}_t$, and output gate $\boldsymbol{o}_t$ and memory gate $\boldsymbol{c}_t$. Gers et al. [25] introduced the forget gate $\boldsymbol{f}_t$ into LSTM architecture enabling the cell reset certain part of its own content once it is no longer needed. The equations for this variant of LSTM are:

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}) \\
\boldsymbol{g}_t &= \tanh(\boldsymbol{W}^{(g)}\boldsymbol{x}_t + \boldsymbol{U}^{(g)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(g)}) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \odot \boldsymbol{g}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)
\end{aligned}
\tag{2.9}
$$

where $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$, $\boldsymbol{g}_t$, $\boldsymbol{c}_t$, $\boldsymbol{h}_t$, $\boldsymbol{b}_t \in \mathbb{R}^{d\times 1}$; $\boldsymbol{x}_t \in \mathbb{R}^{e\times 1}$ is the input vector at time t $\boldsymbol{h}_t \in \mathbb{R}^{d\times 1}$ is the hidden state at time t; $\boldsymbol{W}^{(.)} \in \mathbb{R}^{d\times e}$ and $\boldsymbol{U}^{(.)} \in \mathbb{R}^{d\times d}$ are the weight matrices; $\boldsymbol{g}_t$ is the intermediate memory gate, $\boldsymbol{b}^{(.)}$ is the bias vector; tanh and σ are element-wise hyperbolic tangent sigmoid functions, respectively; and $\odot$ is the element-wise multiplication operator. We remark that in some articles, the gate $\boldsymbol{g}_t$ is not given a name and implicitly appear in the formula of $\boldsymbol{c}_t$. The LSTM diagram is in Figure 2.5.

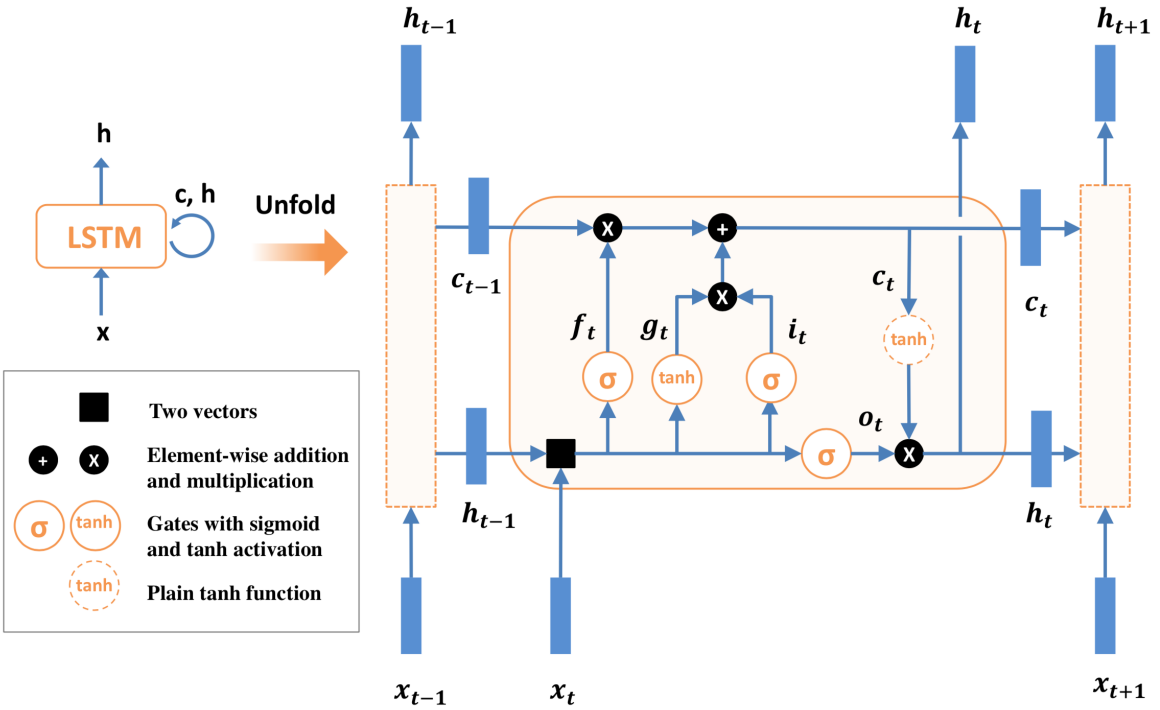


Figure 2.5. LSTM diagram, folded and unfolded over time. Vectors are represented by solid rectangles

Although the equations for an LSTM cell seem tangled, the progress within the network can be imagined as follows: In parallel, $\boldsymbol{g}_t$ computes a new memory about $\boldsymbol{x}_t$ and $\boldsymbol{i}_t$ computes whether $\boldsymbol{x}_t$ matters. After that, they combine outcomes by multiplying them. In the meantime, $\boldsymbol{f}_t$ computes whether the past cell's memory matters to compute current cell. Its outcome is combined with the combination of $\boldsymbol{g}_t$ & $\boldsymbol{i}_t$ by adding this time. The result is $\boldsymbol{c}_t$, and this is delivered to the cell in next step. Besides, one copy of $\boldsymbol{c}_t$ is created to be further used in the cell. Finally, in the meantime, $\boldsymbol{o}_t$ calculates what parts of the memory $\boldsymbol{c}_t$ should be exposed as the output $\boldsymbol{h}_t$ in the following way: tanh is applied to the previously created copy of $\boldsymbol{c}_t$, the result is multiplied by $\boldsymbol{o}_t$. Thus $\boldsymbol{h}_t$ is obtained. Note that $\boldsymbol{h}_t$ is both exposed as the output and fed as the input for the next cell. Therefore, the output and the hidden state refers to the same thing in this context. An acute reader may notice that the terms "gate" and "vector" have been used interchangeably. A gate either refers to the abstractions for specific set of operations in a cell; or the vectors that result from those operations. For example, when we state "$\boldsymbol{i}_t$ is multiplied", it is a vector. But when we state "$\boldsymbol{i}_t$ calculates", "$\boldsymbol{i}_t$" is

actually the set of operations that output the vector $\boldsymbol{i}_t$. In summary, LSTM deals with the vanishing/exploding gradient problem through the gates regulating the information flow. If a piece of information is regarded as important by gates, its corresponding second derivative is sustained in the next states and vice versa.

3. PRELIMINARIES

In the following sections we provide details about the building blocks for our proposed feature selection framework.

3.1. Measures of Association

We use two association measurements in our formulas. The first is the correlation coefficient and the second is Mutual information.

3.1.1. Correlation

The correlation coefficient is a statistical measure that calculates the strength of the relationship between two random variables. Although the term defines any statistical relationship, it commonly refers to a measure for linear association between the two variables. The most familiar one is the Pearson correlation coefficient [26], which is defined as:

$$\begin{aligned} r_{xy} &= \frac{s_{xy}}{s_x s_y} \\ &= \frac{\frac{\sum_{n=1}^{N}(x_n-\bar{x})(y_n-\bar{y})}{N-1}}{\sqrt{\frac{\sum_{n=1}^{N}(x_n-\bar{x})^2}{N-1}}\sqrt{\frac{\sum_{n=1}^{N}(y_n-\bar{y})^2}{N-1}}} \\ &= \frac{\sum_{n=1}^{N}(x_n-\bar{x})(y_n-\bar{y})}{\sqrt{\sum_{n=1}^{N}(x_n-\bar{x})^2}\sqrt{\sum_{n=1}^{N}(y_n-\bar{y})^2}} \end{aligned} \tag{3.1}$$

on a sample data $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$, where s_{xy} is the sample covariance between $\boldsymbol{x}$ and $\boldsymbol{y}$; s_x is the sample standard deviation of $\boldsymbol{x}$ and s_y is the sample standard deviation of $\boldsymbol{y}$.

The range of the correlation coefficient r_{xy} is $[-1, 1]$. This measures the strength and direction of a linear relationship between two variables. The extreme values -1 or 1 imply that there is a perfect linear relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e. all the data points (x_n, y_n) lie on a line. $r_{xy} = 1$ implies that data points are on a line and y_n increases while x_n increases. r_{xy} = -1 implies that y_n decreases while x_n increases. r_{xy} = 0 implies that there is no linear correlation between $\boldsymbol{x}$ and $\boldsymbol{y}$. When either of the standard deviations are zero, the correlation is undefined according to the formula. For instance, the points lying on a horizontal or a vertical line on the xy-plane has undefined correlation coefficient. If the variables are independent, Pearson's correlation coefficient is 0, but the converse is not true since the it only identifies linear dependencies between them.

3.1.2. Mutual Information

The mutual information (MI) is a measure for dependence between two random variables. It is and defined as:

$$I(X;Y) = \sum_{x,y} P(x,y) log \frac{P(x,y)}{P(x)P(y)} \tag{3.2}$$

where X and Y are two random variables; $P(X,Y)$ is the joint distribution of X and Y; $P(X)$ is the marginal distribution of X and $P(Y)$ is the marginal distribution of Y. The range of mutual information is $[0, \infty)$. It is equal to zero if and only if two random variables are independent, and higher I implies higher dependency. Mutual information is heavily studied in information theory and probability theory. Contrary to Pearson's correlation coefficient, it can detect non-linear relationships as well as linear relationships. Nonetheless, neither of them is more powerful than the other, since they describe different traits of the association between two variables. MI quantifies it via joint distribution of the variables whereas the correlation quantifies it via the products of the variables. Mutual information requires the estimation of the probability distributions on sample data. The estimator function we adopted in this study is a non-parametric approach based on entropy estimation from k-nearest

neighbors distances as described in [27]. To estimate $I(\boldsymbol{x};\boldsymbol{y})$ on the sample data

$$\begin{aligned}\boldsymbol{x} &= (x_1, x_2, \ldots, x_N)\\ \boldsymbol{y} &= (y_1, y_2, \ldots, y_N)\end{aligned} \tag{3.3}$$

the algorithm computes a number I_n for each data point (x_n, y_n), based on its nearest-neighbors on the y-axis. Computation of I_n is as follows: First, the kth-closest neighbor to point n on y-axis in the set

$$\mathcal{N} = \{(x_{n'}, y_{n'}) \mid n' \in \{0, 1, \ldots, N\} \wedge x_{n'} = x_n\} \tag{3.4}$$

is chosen. Let the subscript for this point be k. Then, let $d_n = |y_k - y_n|$ and define

$$\mathcal{M} = \{(x_{n'}, y_{n'}) \mid n' \in \{0, 1, \ldots, N\} \wedge |y_{n'} - y_n| \leq d_n\} \tag{3.5}$$

Let $p_n = |\mathcal{N}|$ and $q_n = |\mathcal{M}|$. Then I_n is calculated as:

$$I_n = \psi(N) - \psi(p_n) - \psi(q_n) + \psi(k) \tag{3.6}$$

where $\psi(.)$ is the digamma function [28]. Finally, MI is estimated by taking the average of I_n over all points n:

$$I(\boldsymbol{x};\boldsymbol{y}) \approx \langle I_n \rangle = \psi(N) - \langle \psi(p_n) \rangle - \langle \psi(q_n) \rangle + \psi(k) \tag{3.7}$$

In our implementation we choose k=3.

3.2. Backtesting

In time series forecasting, backtesting is the method for evaluating the out-of-sample performance of a model. Since time series data includes temporal components that depend on each other, shuffling the data as in K-Fold cross test would cause

the loss of information contained in the sequential structure. Backtesting splits the data into chunks without changing the sequence. In each step, increasingly, a model is trained with the concatenation of chunks from the beginning and the residuals are calculated on the next consecutive chunk. Mean of the residuals obtained from those models are considered as the model's test error.

Given timepoints $(x_1, \ldots, x_N)$ and the number of steps $S < N$; at each step $i \in \{1, \ldots, S\}$, the timepoints used for training are $(x_1, \ldots, x_a)$ and the timepoints used for test are $(x_{a+1}, \ldots, x_{a+c})$, where

$$a = i \times c + r \tag{3.8}$$

$$c = \left\lfloor \frac{N}{S+1} \right\rfloor \tag{3.9}$$

$$r = N \bmod (S+1) \tag{3.10}$$

In other words, c is the chunk size and in each time step, the number of training points increase by a factor of c. r is the shift amount which allows conveniently reaching to the last timepoint N at the end of step S. The special case $S = N - 1$ is called walk-forward validation, which provides the most robust evaluation. The drawback of walk-forward validation is the infeasibility to scale for large models.

4. SIGNAL SELECTION FORMULATIONS

Our work draws inspiration from time series forecasting and recurrent neural network models. In crude terms, each model used in this study has an LSTM layer which sequentially takes a set of lagged observations as feature vectors in the recurrent steps. The problem is predicting one-step ahead values of multiple time series of product sales in different stores with minimum error. Initially, we define the following variables:

- P: Number of products.
- S: Number of stores.
- T: Number of observed timepoints.
- L: Number of lagged observations.
- N: Number of samples.
- U: Number of features.
- V: Number of targets, i.e, $V > 1$ is multivariate.
- $p \in \{1, 2, \ldots, P\}$: Product index.
- $s \in \{1, 2, \ldots, S\}$: Store index.
- $t \in \{1, 2, \ldots, T, \ldots\}$: Observed & unobserved timestep index for sales.
- $n \in \{1, 2, \ldots, N\}$: Sample index.
- $l \in \{1, 2, \ldots, L\}$: LSTM step index.
- $a_{pst} \in \mathbb{R}_{\geq 0}$: The amount of product p sold in store s at time t.
- $\boldsymbol{a}_{ps}^{T} \in \mathbb{R}_{\geq 0}^{T}$: $(a_{ps1}, a_{ps2}, \ldots, a_{psT})$, signal (i.e., time series data) for p,s
- $\boldsymbol{x}_{l}^{(n)} \in \mathbb{R}^{U}$: Input vector to step l of LSTM for sample n.
- $\mathbf{x}^{(n)} \in \mathbb{R}^{L \times U}$: $(\boldsymbol{x}_{1}^{(n)}, \boldsymbol{x}_{2}^{(n)}, \ldots, \boldsymbol{x}_{L}^{(n)})$: Input matrix for sample n.
- $\boldsymbol{y}^{(n)} \in \mathbb{R}^{V}$: Target for sample n.

For notational convenience, we assume that the sales amount is zero before a product is listed in a store by a retailer. The number of samples to be created depends on the number of observed timepoints along with the choice of lagged observations, precisely, $N(T, L) = T - L$. A simple idea could be to train a model with all the

available signals rather than a single signal in order to forecast a_{pst} correctly. In this scenario, assuming we don't use any additional features, we would have $P \times S \times L$ features. For the real-world applications generally, $S \times P \gg T$, which implies $P \times S \times L \gg N$. For instance, as of April 2019, the number of products listed on Amazon.com is 119.9 million [29]. Thus, one caveat is that there are inadequate number of samples and a model can easily overfit if one wants to train it with the lagged observations of all signals. We formulated various feature selection and engineering schemes to build $\boldsymbol{x}_l^{(i)}$ and $\boldsymbol{y}^{(i)}$. These are: (1) *Uni*, (2) *Uni-Store*, (3) *Uni-Product-Pcc*, (4) *Uni-Product-Mi*, (5) *Multi-Store*. In each of the schemes, previous L observations of particular set of signals are used as the features for each target variable. In schemes (1)-(4), for each pair (p, s), a separate univariate model M_{ps} whose target variable represents a_{pst} is constructed, thus $V = 1$ and the number of resulting models is $S \times P$ in each of those individual scheme. In (5), for each product p, a separate multivariate model M_p whose target variables represent $\{a_{p1t}, a_{p2t}, \ldots, a_{pSt}\}$ is constructed, thus $V = S$ and the number of resulting models is P. In the rest of the paper, we use respective scheme names to identify models that are trained via the scheme. For example a model that is trained according to "Uni" scheme is called as "Uni-model".

4.1. Uni

For a product p and store s and the model M_{ps}, the element of $\boldsymbol{x}_l^{(n)}$ is:

$$a_{ps\tau} \tag{4.1}$$

and the element of $\boldsymbol{y}^{(n)}$ is:

$$a_{psv} \tag{4.2}$$

where

$$\tau(n, l) = n + l - 1 \tag{4.3}$$

$$\upsilon(n, L) = n + L \tag{4.4}$$

The definition of τ and υ will remain the same in the rest of the paper and their arguments are omitted for brevity. Our choice of $\boldsymbol{x}_l^{(n)}$ and $\boldsymbol{y}^{(n)}$ implies that $U = 1$ and $V = 1$. Essentially, to predict the future value of the signal, we use the previous values of the same signal. Note that this is the same set of features used in the vanilla autoregressive model; only its shape is adjusted for the LSTM structure and there are multiple models, each of which corresponds to a particular signal. This formulation is constructed as the preliminary for next schemes.

4.2. Uni-Store

For a product p and store s and the model M_{ps}, the elements of $\boldsymbol{x}_l^{(n)}$ are comprised by:

$$\mathcal{X} = \{a_{ps'\tau} \mid s' \in \{1, 2, \ldots, S\}\} \tag{4.5}$$

The element of $\boldsymbol{y}^{(n)}$ is:

$$a_{psv} \tag{4.6}$$

This scheme provides the model with the overall sales of product p to predict the product sales of p in a single store. There are many latent factors that affect the number of sales of products independent from a single store; such as television commercial, epidemic, tax increase by government or the factors related to the retailer company itself. Hence, a change in the sales of a product in some stores might point the way to the change (presumably in the same direction) in sales of the product in other stores. We expect the model to capture the structure of any "change" and improve the prediction performance. This time our choice of $\boldsymbol{x}_l^{(n)}$ and $\boldsymbol{y}^{(n)}$ implies that $U = S$ and $V = 1$.

4.3. Uni-Product-Pcc

For a product p and store s and the model M_{ps}, the elements of $\boldsymbol{x}_l^{(n)}$ are comprised by:

$$\mathcal{X} = \left\{ a_{p's\tau} \mid p' \in \underset{P' \subseteq \{1,\ldots,P\}, |P'|=h}{\arg\max} \sum_{\rho \in P'} r(\boldsymbol{a}_{\rho s}^T, \boldsymbol{a}_{ps}^T) \right\} \tag{4.7}$$

The element of $\boldsymbol{y}^{(n)}$ is:

$$a_{psv} \tag{4.8}$$

where r is the sample Pearson correlation coefficient in Eq. 3.1 between variables $\boldsymbol{a}_{\rho s}^T$ and $\boldsymbol{a}_{ps}^T$. In this scheme, the formula contains the time series of different products in the same store. The main motivation is that certain group of products are bought together. Unlike the Market Basket Analysis, we are not interested in explicitly discovering the association rules between products. Besides, we are dealing with temporal data and the association rules might vary through time. Our assumption is that the machine learning model can make use of these relations to improve prediction performance. Nevertheless, using the time series data of all the products in a single store can still lead to overfitting as $L \times P > N$ in practice. The remedy is to limit the number signals. Along with the signal of product p in store s itself, we pick the top h related products sold in s to be used in feature engineering. In this scheme, the the criterion for relatedness is the $F1$ score. Our choice of $\boldsymbol{x}_l^{(n)}$ and $\boldsymbol{y}^{(n)}$ implies that $U = h$ and $V = 1$.

4.4. Uni-Product-Mi

For a product p and store s and the model M_{ps}, the elements of $\boldsymbol{x}_l^{(n)}$ are comprised by:

$$\mathcal{X} = \left\{ a_{p's\tau} \mid p' \in \underset{P' \subseteq \{1,\dots,P\}, |P'|=h}{\arg\max} \sum_{\rho \in P'} I(\boldsymbol{a}_{\rho s}^T, \boldsymbol{a}_{ps}^T) \right\} \tag{4.9}$$

where I is is the Mutual Information score in Eq. 3.7. The element of $\boldsymbol{y}^{(n)}$ is:

$$a_{psv} \tag{4.10}$$

The main idea of this formulation is the same as in *Uni-Product-Pcc*. The only difference is that the relatedness criterion between two signals is the mutual information instead of the Pearson correlation coefficient. Unlike the Pearson correlation coefficient, the mutual information criterion is able to capture non-linear relations between the data. Therefore, applying this scheme into the feature selection pipeline before feeding the features into a neural network might reduce the feature dimension while keeping the informative features that Pearson correlation ignores.

4.5. Multi-Store

For a product p and the model M_p, the elements of $\boldsymbol{x}_l^{(n)}$ are comprised by:

$$\mathcal{X} = \{a_{ps'\tau} \mid s' \in \{1, 2, \dots, S\}\} \tag{4.11}$$

The elements of $\boldsymbol{y}^{(n)}$ are comprised by:

$$\mathcal{Y} = \{a_{ps'v} \mid s' \in \{1, 2, \dots, S\}\} \tag{4.12}$$

In this scheme, similar to *Uni-Store* approach, we used the signals for the sale of product p in every individual store. Here, the target variables are also selected from

the same set of signals, therefore, $U = S$ and $V = S$.

5. EXPERIMENTS AND RESULTS

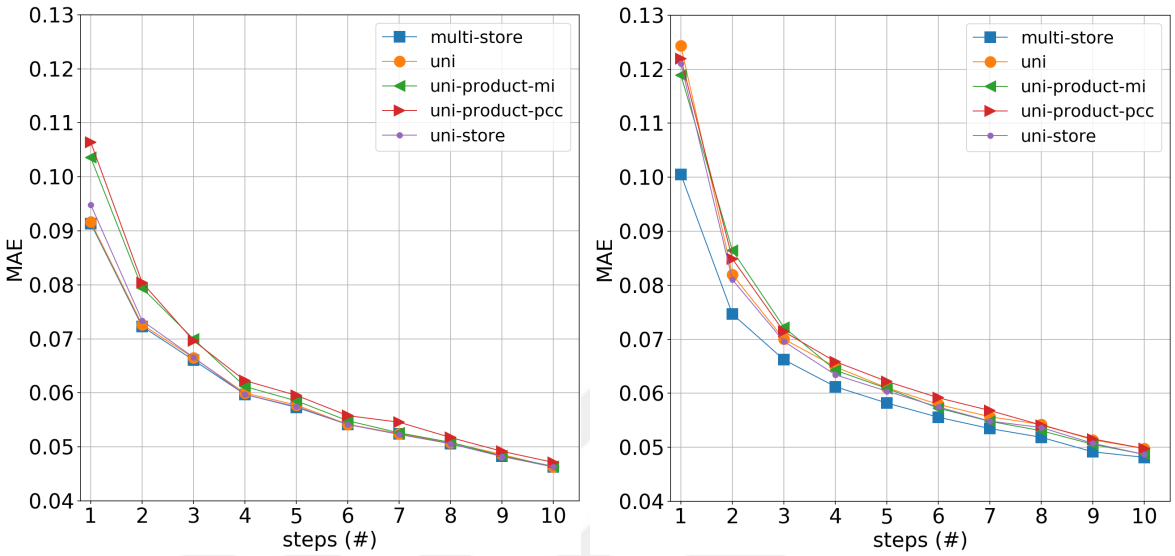
In the experiments, we used Keras Deep Learning library [30] for implementing the neural network models. The model of choice for each experiment is an LSTM model followed by a Dense layer. The activation function of a neuron in the dense layer is simply the identity function to output real-valued numbers for regression. Each LSTM model has latent dimension 10, and trained with Adam optimizer. The loss function is mean absolute error (or "MAE" in the plots). The number epochs is set to 50 with early stopping. This analysis is performed on Google Cloud Platform [31]. The virtual machine is chosen as BASIC scale tier, which has a single worker instance with CPU and no accelerators. This tier is suitable for experimenting with new models using small datasets. On this machine, models are created and removed and the experiment results are saved on the fly.

We applied our proposed methodologies into Migros Online product sales data between years 2014-2019. A trial run in the experiment setup is defined as a complete training and evaluation of models for each store, product, scheme and backtesting step. Specifically, in a single trial, 2500 models are created for each of the *Uni* schemes and 500 models are created for the *Multi-Store* scheme; summing up a total of 10500 models for the single trial. Note that a *Multi-Store* scheme handles the prediction of 5 variables unlike the univariate schemes, therefore, the resulting number of models is 500 instead of 2500. Without assortments (see below), we made two trial runs. Then, with assortments, we made two trial runs. Consequently, 42000 models are trained and evaluated in this experiment. Index of the time series data was originally in unix timestamps; and then we re-sampled it daily, so that the timepoints represent the sales amounts in days. We selected T as $356 \times 5 = 1825$. Due to calendar-day variations, we truncated a few days from the dataset to achieve exactly 1825 timepoints. A random set of 50 products and 5 stores are chosen, that is, $P = 50$ and $S = 5$ and the number of signals is 250. Each sample represents the lagged observations for 9 days, i.e. $L = 9$ and the resulted number of samples is $N = 1816$.

For the sake of robust evaluations of the model performances, we applied 10-step backtesting on each signal. Since unit of the sales data is either in grams or pieces, the magnitude of the signals varied significantly. Generally, a_{pst} differs by three orders of magnitude for a product in grams and product in pieces. This variation would not allow a fair comparison between the model losses, therefore we min-max scaled a_{pst} so that the maximum value of a signal is transformed into 1 whereas the minimum value of the signal is transformed into 0 and the rest of the values are between 0 and 1, $\forall p, s$. In addition, scaling allows speeding up the training process of gradient based algorithms. We both scaled the train and test data according to the minimum and maximum values of the training data in order to avoid "data leakage", that is, presenting any information obtained from the test data to the model for training. Unless otherwise stated, mean values in the experiments include all the respective backtesting steps.

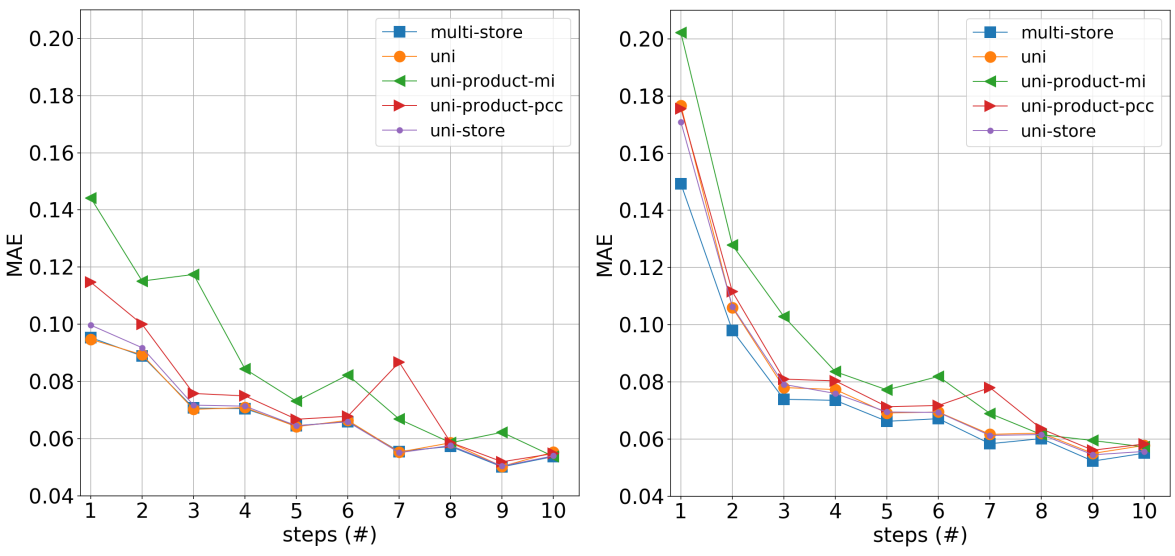
Regarding the formulations in section 5, we considered including particular assortments. The assortments are any kind of extra features that we add on top of the defined features in the schemes. Formally, we prepared a new set of features and took the union of this and $\mathcal{X}$ for each $\boldsymbol{x}_l^{(n)}$. The assorments we selected in this experiment include one-hot encodings of day of the week (7 features), one-hot-encodings of month of the year (12 features), and one-hot-encodings of nine holidays and one no-holiday (10 features), summing up a total of 29 extra feture dimensions. We refer to the trials with assortment features as "assortment-mode" trial and refer to the trials without assortment features as "plain-mode" trial. In the *Uni-Product-Mi* & *Uni-Product-Pcc* models, we selected top 10 signals among 50 signals according to the association criteria, so that $h = 10$.

In Figure 5.1 and 5.2, training and test losses of models decrease as the backtesting steps increase. Since in each step, the models use more timepoints than they use in the previous step, we conclude that the models indeed learn better with more samples. There are two more observations. Firstly, the training losses are below the test losses by a narrow margin. Secondly, the training losses and the test losses converge as the backtesting steps increase. Therefore, the in-sample and out-of-sample errors get desirably close as we use more samples for training.



(a) Plain Mode (b) Assortment Mode

Figure 5.1. Mean training errors as a function of backtesting steps



(a) Plain Mode (b) Assortment Mode

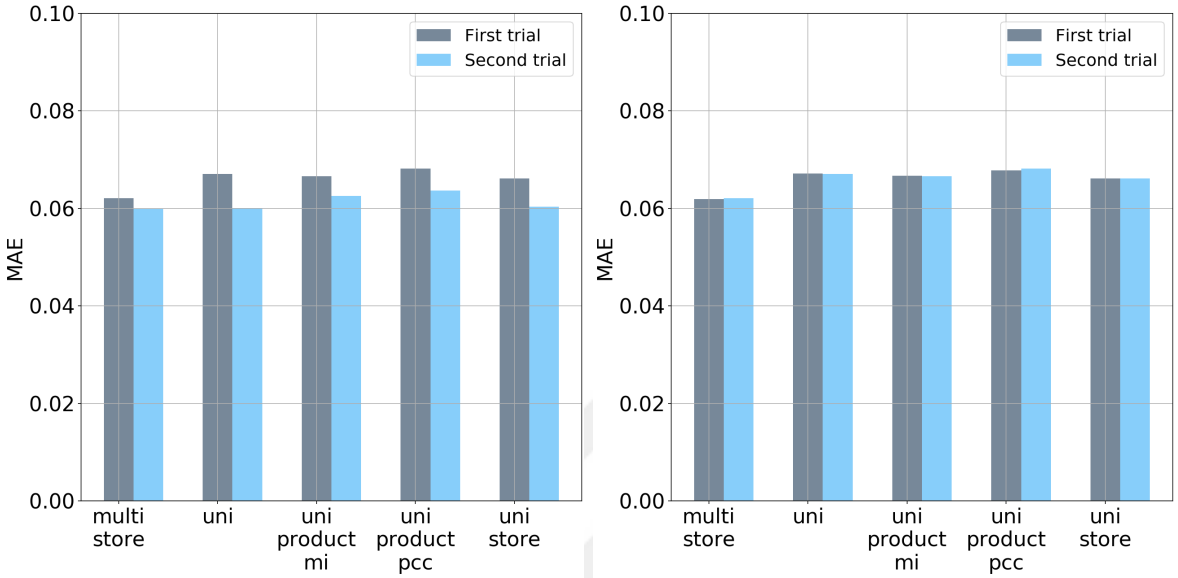
Figure 5.2. Mean test errors as a function of backtesting steps

Figures 5.3 and 5.4 summarize the average train and test errors in two trials & two modes. All of the corresponding results from the first trial and the second trial are very close. Only the *Uni-Product-Mi* differs between trials considerably. In addition, the highest difference between the training and test loss belongs to the *Uni-Product-Mi*. This suggests that the *Uni-Product-Mi* models are more likely to overfit than the other models. It is followed by *Uni-Product-Pcc*, therefore the worst performing schemes on average are the ones using other product signals.

Without exception, every one of the models in the plain-mode has lower training and test losses than their respective models in the assortment-mode. Additionally, the gap between training and test loss is higher in the assortment-mode. This addresses the ovefitting due to the redundancy of the selected assortments.

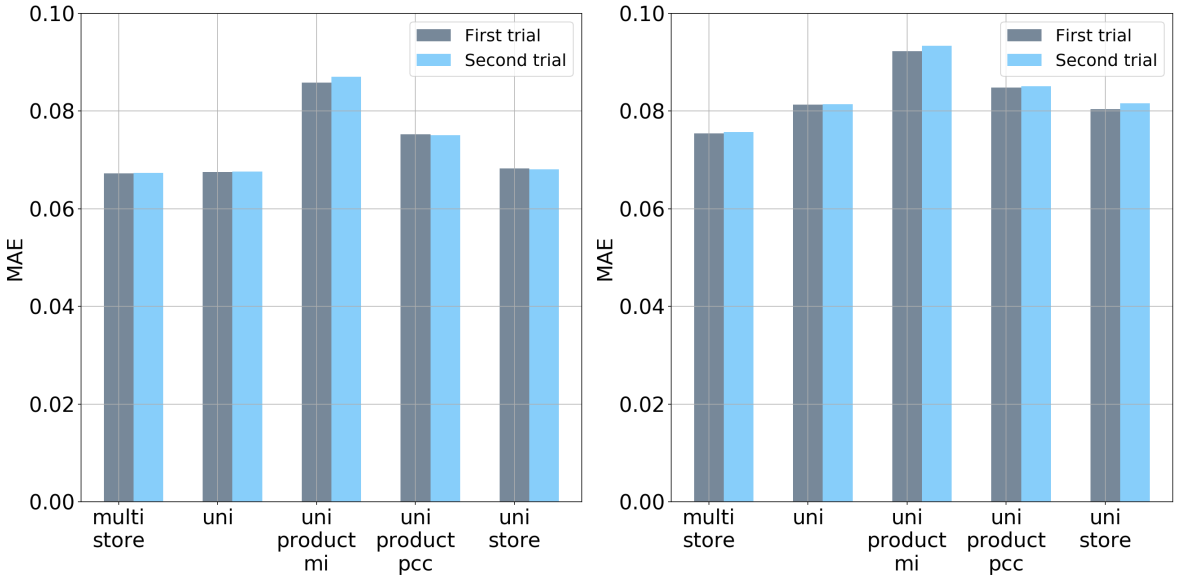
In the assortment mode, the best performing model on average is the *Multi-Store* followed by the *Uni* by a considerable margin. Hence, the *Multi-Store* model is more robust to the overfitting that is caused by the assortments. In the plain mode, the best performing model on average is the *Uni* followed by the *Multi-Store* by a narrow margin. The *Multi-Store* outperforms the *Uni-Store* approach in the two modes. Note that both of the models use the same set of features, only their target variables differ. We conclude that the multivariate model is more capable at capturing the relationships between the signals than the univariate models with the same set of features.

Each plot in Figures 5.5 - 5.7 represents the average values obtained from two trials & two modes. Figure 5.5 shows the average losses of models on all stores. A dot represents the average of the $2 \times 50 \times 10$ values. Interquartile ranges of *Uni-Product-Mi* is noticeably the largest; and that of *Uni-Product-Pcc* is slightly larger than the rest. Figure 5.6 shows the average losses of models on all products. A dot represents the average losses of the $2 \times 5 \times 10$ values. All the interquartile ranges are approximately 0.05. Figure 5.7 shows the averages of models on all signals. Particularly, a dot represents the average of the 2×10 values. All the interquartile ranges are approximately between 0.06 and 0.07.



(a) Plain Mode (b) Assortment Mode

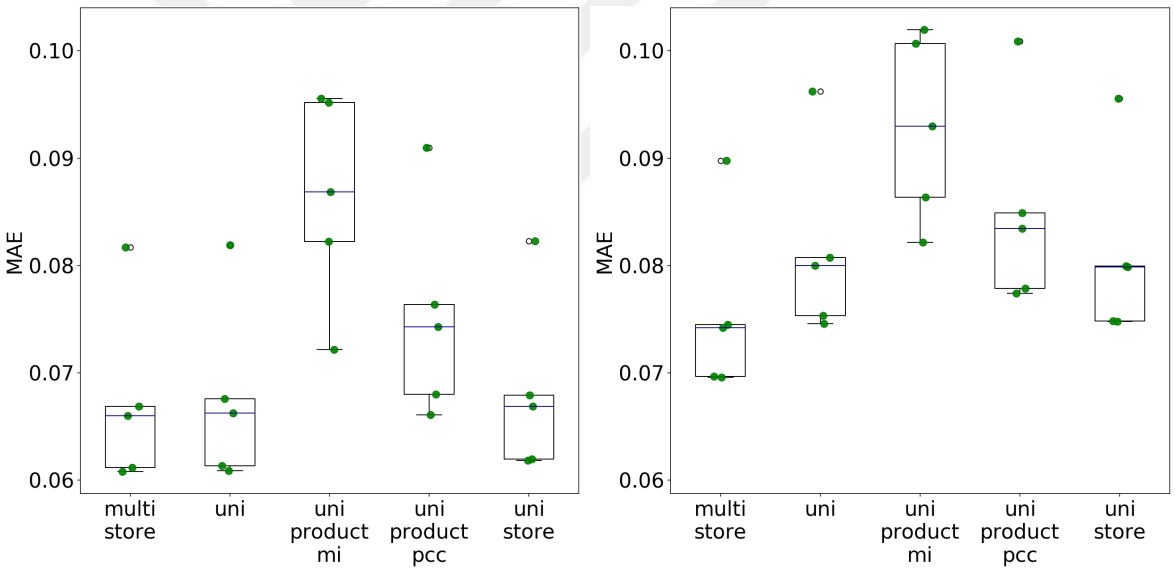
Figure 5.3. Mean training errors



(a) Plain Mode (b) Assortment Mode

Figure 5.4. Mean test errors

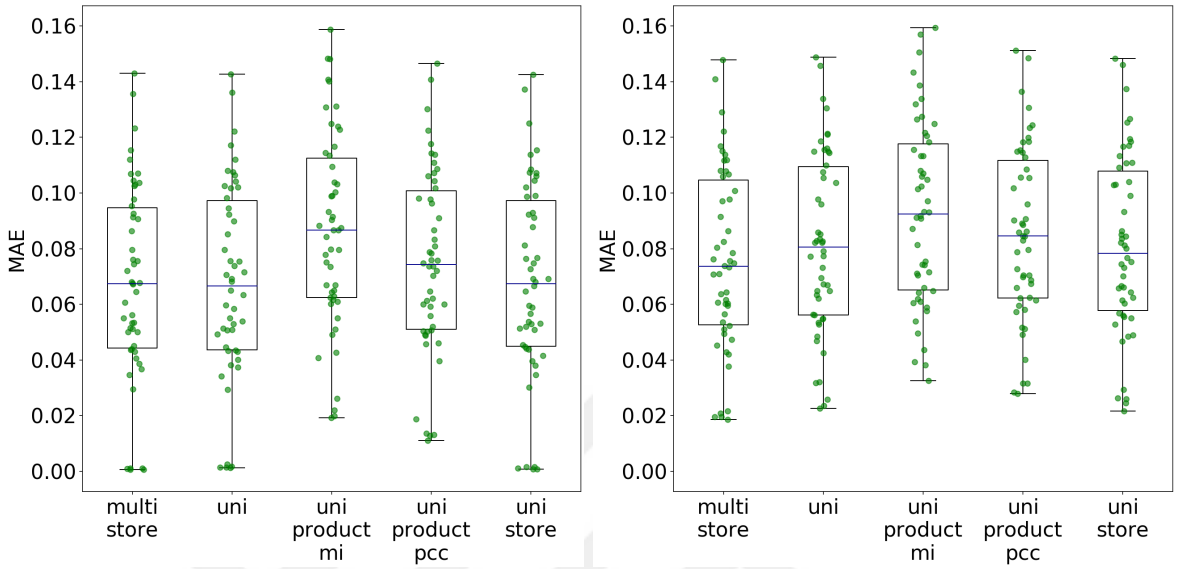
All the median values are close to the mean values in Figure 5.4. In the boxplots, the maxiumum values of all points are similar in the two modes, however, their minimum values are shifted by some amount. It also supports the conclusion drawn from Figure 5.4(b) that the assortments led to overfitting. There are some zero values in the plain-mode. The minimum values are never zero in the assortment-mode. Points with zero errors are obtained from the constant signals, i.e. their values are always the same number regardless of the time. These signals are likely to come from the zero-selling products. The distribution of the points are similar; however the values are shifted by some amount. All the minimum, maximum numbers and quartiles are shifted similar to the mean values from the previous figures. Except for the *Uni-Product-Mi* model, the outliers are close to the maximum values.



(a) Plain Mode (b) Assortment Mode

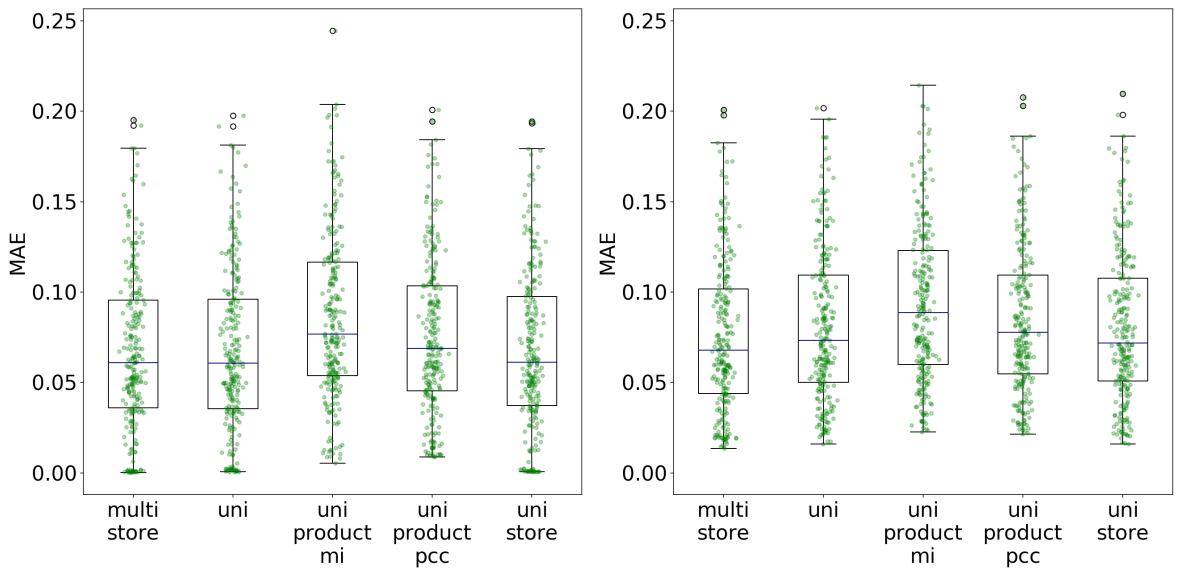
Figure 5.5. Mean test errors on each store

Figure 5.8 demonstrates the percentages of how many of the times each scheme resulted in the best performing model on the signals. For each signal we determined which model gave the minimum test loss. Then we counted those models. This is done for all the trials. The average of two trials are calculated for each modes, respectively. Finally, the numbers are converted into percentages. The results on the pie chart are consistent with the previous figures. *Multi-Store* is the best model on most of the times and on average its loss is also the smallest in the assortment-mode. *Uni* is the best model in most of the times in the plain-mode and on average its loss is also the



(a) Plain Mode (b) Assortment Mode

Figure 5.6. Mean test errors on each product

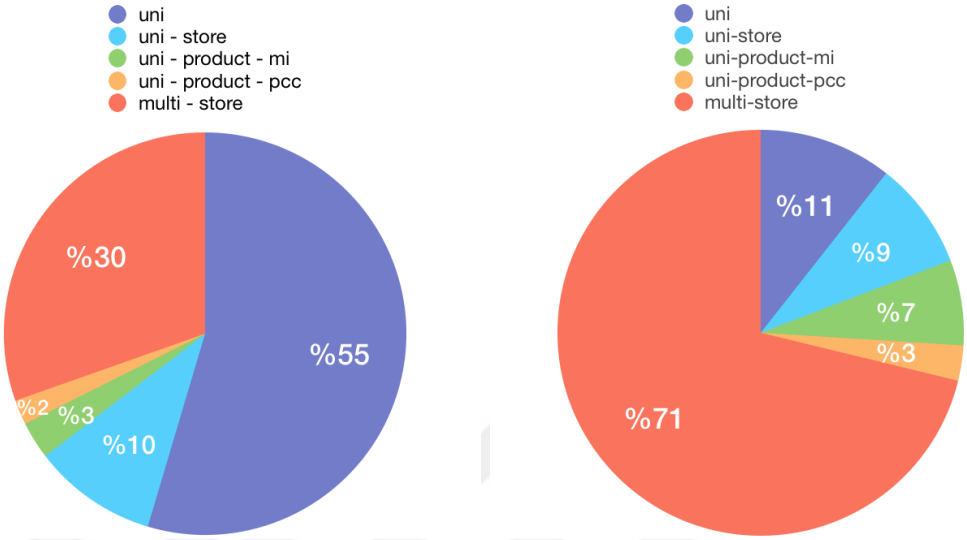


(a) Plain Mode (b) Assortment Mode

Figure 5.7. Mean test errors on each (store, product) pair

smallest in the plain-mode. In addititon, consistent with figure 5.4(b), the percentage of *Multi-Store* is considerably more than the rest of the models in assortment-mode. The percentage of *Uni* is the highest and it is followed by *Multi-Store* in the Plain-Mode. Similar to the mean values, the difference between the percentages of *Uni* model and *Multi-Store* model are close in the plain-mode. The orders of the worst performing three schemes are the same between two modes.

Figure 5.9 shows that the average training speed of the *Multi-Store* scheme is approximately three times faster than that of the other schemes. Instead of creating multiple models and using the same set of features to predict a different target, *Multi-Store* handles it at once with a single model. Although the *Multi-Store* model has more weight parameters to be updated during training than the other models in our experiments, building and training multiple models created more overhead. Consequently, the experiments show that there is a training speed versus prediction performance trade-off between the *Multi-Store* and the *Uni* scheme.



(a) Plain Mode (b) Assortment Mode

Figure 5.8. Percentages of the best performing schemes on individual signals

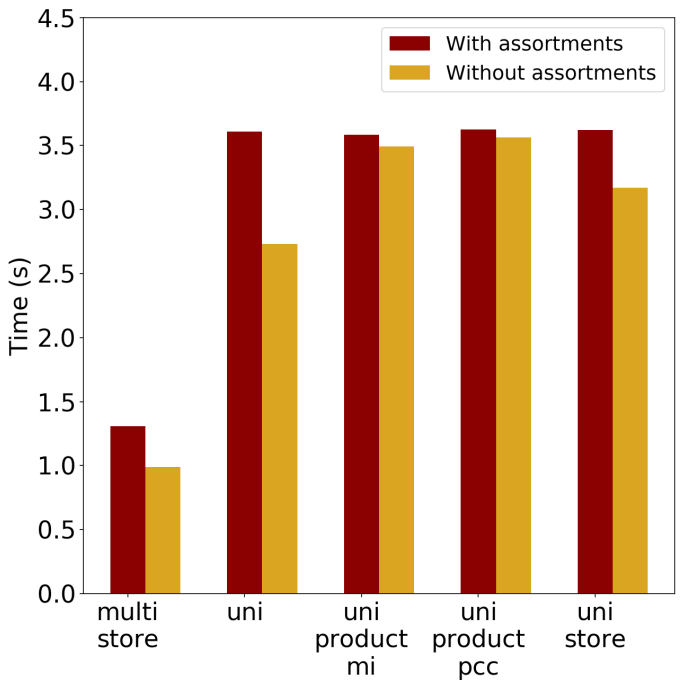


Figure 5.9. Average wall-clock training times of the models in each scheme.

6. CONCLUSION

In this paper, we have proposed five mathematical formulations for hierarchical signals that represent the product sales amounts in multiple stores. The formulas allowed rigorous feature selection and engineering schemes for univariate and multivariate forecasting regarding certain association criteria between the signals. We have applied the formulations on Migros Online sales data and observed that the *Uni* and *Multi-Store* are the two best performing models. Although the latter produced slightly greater test loss than the former, its training speed is approximately three times faster than that of the other models. Instead of training and deploying a new model for predicting each signal; one model for a group of signals by compromising the prediction performance to a certain extent would be preferable in the practical applications. Two possible explanations for the poor performances of *Uni-Product-Pcc* and *Uni-Product-Mi* schemes are the deficiency of samples or excess of h. However, more intriguing cause is the small number product subset. As explained above, we have selected 50 random products, and selected signals among 50 signals of products within a store. However, the entire dataset have thousands of products. Therefore there is a significant probability of finding signals with higher scores outside of the set. In other words, the selected signals according to the association criteria were not informative enough and have led to overfitting. A thorough experiment setup should consider selecting the associated signals among the all products within a store. A more thorough experiment would even select signals from the entire dataset. As a final remark, feature selection and engineering process constitutes the majority of the work in the real-world machine learning problems. We have standardized it for the multivariate demand forecasting on LSTM, thus made the reproducibility of the experiments straightforward. Different experiments can be automated by regarding the number of multivariate features, lagged observations, associated product signals and the store ids as the hyperparameters and the models can be easily optimized.

REFERENCES

1. Iizumi, T., Y. Shin, W. Kim, M. Kim and J. Choi, "Global crop yield forecasting using seasonal climate information from a multi-model ensemble", *Climate Services*, Vol. 11, pp. 13–23, 2018.

2. Williams, B. M., P. K. Durvasula and D. E. Brown, "Urban freeway traffic flow prediction: application of seasonal autoregressive integrated moving average and exponential smoothing models", *Transportation Research Record*, Vol. 1644, No. 1, pp. 132–141, 1998.

3. Claassen, J., S. Mayer, R. Kowalski, R. Emerson and L. Hirsch, "Detection of electrographic seizures with continuous EEG monitoring in critically ill patients", *Neurology*, Vol. 62, No. 10, pp. 1743–1748, 2004.

4. Riordan, R. and A. Storkenmaier, "Latency, liquidity and price discovery", *Journal of Financial Markets*, Vol. 15, No. 4, pp. 416–437, 2012.

5. Ramos, P., N. Santos and R. Rebelo, "Performance of state space and ARIMA models for consumer retail sales forecasting", *Robotics and computer-integrated manufacturing*, Vol. 34, pp. 151–163, 2015.

6. Fitzsimons, G. J., "Consumer response to stockouts", *Journal of consumer research*, Vol. 27, No. 2, pp. 249–266, 2000.

7. Kaplan, R. S. and S. R. Anderson, "Time-driven activity-based costing", *Available at SSRN 485443*, 2003.

8. Varila, M., M. Seppänen and P. Suomala, "Detailed cost modelling: a case study in warehouse logistics", *International Journal of Physical Distribution & Logistics Management*, Vol. 37, No. 3, pp. 184–200, 2007.

9. Rowe, G. and G. Wright, "The Delphi technique as a forecasting tool: issues and analysis", *International journal of forecasting*, Vol. 15, No. 4, pp. 353–375, 1999.

10. Akaike, H., "A new look at the statistical model identification", *Selected Papers of Hirotugu Akaike*, pp. 215–222, Springer, 1974.

11. Engle, R. F., "Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation", *Econometrica: Journal of the Econometric Society*, pp. 987–1007, 1982.

12. Bollerslev, T., "Generalized autoregressive conditional heteroskedasticity", *Journal of econometrics*, Vol. 31, No. 3, pp. 307–327, 1986.

13. Hornik, K., M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", *Neural networks*, Vol. 2, No. 5, pp. 359–366, 1989.

14. Nair, V. and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

15. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors", *Cognitive modeling*, Vol. 5, No. 3, p. 1, 1988.

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

17. Jain, A. K., J. Mao and K. Mohiuddin, "Artificial neural networks: A tutorial", *Computer*, Vol. 29, No. 3, pp. 31–44, 1996.

18. Zaremba, W., I. Sutskever and O. Vinyals, "Recurrent neural network regularization", *arXiv preprint arXiv:1409.2329*, 2014.

19. Graves, A., A.-r. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks", *2013 IEEE international conference on acoustics, speech and*

signal processing, pp. 6645–6649, IEEE, 2013.

20. Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, pp. 1097–1105, 2012.

21. Hochreiter, S. and J. Schmidhuber, "Long short-term memory", *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

22. Le, Q. V., N. Jaitly and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units", *arXiv preprint arXiv:1504.00941*, 2015.

23. Siegelmann, H. T. and E. D. Sontag, "On the computational power of neural nets", *Journal of computer and system sciences*, Vol. 50, No. 1, pp. 132–150, 1995.

24. Chung, J., C. Gulcehre, K. Cho and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", *arXiv preprint arXiv:1412.3555*, 2014.

25. Gers, F., J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM", *Neural computation*, Vol. 12, pp. 2451–71, 10 2000.

26. Pearson, K., "LIII. On lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Vol. 2, No. 11, pp. 559–572, 1901.

27. Ross, B. C., "Mutual information between discrete and continuous data sets", *PloS one*, Vol. 9, No. 2, p. e87357, 2014.

28. Bernardo, J. M., "Psi (digamma) function", *Applied Statistics*, Vol. 25, No. 3, pp. 315–317, 1976.

29. ScrapeHero, *How Many Products Does Amazon Sell? - April 2019*, 2019, `https://www.scrapehero.com/number-of-products-on-amazon-april-2019`,

accessed in May 2019.

30. Chollet, F., *Keras*, 2015, `https://keras.io`, accessed in May 2019.

31. Google, *Cloud ML Engine*, 2019, `https://cloud.google.com/ml-engine`, accessed in May 2019.