



FEDERAL UNIVERSITY OF AMAZONAS - UFAM
INSTITUTE OF COMPUTING - ICOMP
POSTGRADUATE PROGRAM IN INFORMATICS - PPGI

**Predicting purchasing intention through a single stage siamese deep
learning models**

Kevin Kimiya Takano

Manaus - AM
2022

Kevin Kimiya Takano

**Predicting purchasing intention through a single stage siamese deep
learning models**

Dissertation submitted to evaluation, as a partial requirement, to obtain the title of Master of Informatics in the Postgraduate Program in Informatics, Institute of Computing.

Supervisor

André Luiz da Costa Carvalho

Federal University of Amazonas - UFAM

Institute of Computing - IComp

Manaus - AM

2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

T136p Takano, Kevin Kimiya
Predicting purchasing intention through a single stage siamese
deep learning models / Kevin Kimiya Takano . 2022
100 f.: il. color; 31 cm.

Orientador: André Luiz da Costa Carvalho
Dissertação (Mestrado em Informática) - Universidade Federal do
Amazonas.

1. Purchase predicting.. 2. Customer behavior. 3. Siamese-
networks. 4. Triplet-loss. I. Carvalho, André Luiz da Costa. II.
Universidade Federal do Amazonas III. Título




FOLHA DE APROVAÇÃO

"Predicting purchasing intention through a single stage siamese deep learning models"

KEVIN KIMYA TAKANO

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. André Luiz da Costa Carvalho - PRESIDENTE

Profa. Eulanda Miranda dos Santos - MEMBRO INTERNO 

Prof. Raoni Simões Ferreira - MEMBRO EXTERNO 

Dr. Márcio Palheta Piedade - MEMBRO EXTERNO 

Manaus, 06 de Julho de 2022

For my family, my father, my mother and my wife.

Computer science is no more about computers than astronomy is about
telescopes.

Edsger W. Dijkstra

Predicting purchasing intention through a single stage siamese deep learning models

Author: Kevin Kimiya Takano

Supervisor: André Luiz da Costa Carvalho

Abstract

Understanding consumer buying behavior in the context of e-commerce is a recent trend at large retail stores. It can be very attractive for retail companies to know which users will buy in their market and what products they will buy. Through the study of online user behavior, models can be created to improve marketing personalizing, and build digital products.

Through the historical data of user events, such as clicked items, it is possible to use them as resources to forecast purchases. Despite how valuable is this data, it is not so simple to create machine learning models using them. A very small number of user sessions are buyers of items, and, in general, models have greater learning difficulties with unbalanced classes. Moreover, there are a large number of products in a store, making the problem even more complex.

Previous works in the literature show that it is better to solve the problem in two stages, i.e., using two models: one model to predict which customers will be buying items and another to predict which products will be purchased among these consumers. Solving problems in two stages makes the problem simpler since it divides the model's complexity. However, creating two models the second model does not use information from non-buyer-sessions to solve the item classification. Furthermore, if the first model fails to classify a session as a buyer-session, the second model may have its results negatively impacted. Therefore, for this work, our objective is to develop a model that solves the problem with just a single model, a *single-stage model*. We deployed Siamese neural networks to extract features to deal with imbalances.

In our single-stage framework, we had several contributions. First, is the creation of a new loss function, the quartet-loss, which optimizes the parameters differently from the triplet-loss. Second, is the development of two different strategies for modeling user click sessions. Third, is the creation of metrics that evaluate the results of e-commerce models in online sessions. And finally, we developed machine learning methods using this project framework that reached the state-of-the-art for this problem.

Keywords: session-based modeling, predicting purchase intention, customer behavior, siamese-networks, triplet-loss

List of Figures

Figure 1 – Sessions data illustration for customers. On the left we have consumer click views on items and on the right we have their item's ids purchases. Our goal is to forecast purchases through views. (Source: Own author)	17
Figure 2 – Machine Learning workflow. (Source: Own author)	23
Figure 3 – Simple Neural Network	24
Figure 4 – Transfer Learning diagram workflow for cats and dogs recognition. (Source: Own author)	27
Figure 5 – Sequential inputs directly in a RNN cells. (Source: Own author)	27
Figure 6 – LSTM and GRU representation with internal equations elements emphasized. (Source: Own author)	29
Figure 7 – Encoder and decoder model example from portuguese to english. (Source: Own author)	30
Figure 8 – Graphical illustration of the proposed model by Bahdanau to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) . Image taken from [Bahdanau et al., 2014]	30
Figure 9 – A common convolutional neural network architecture. In general, CNN has three components, the convolutional layer, the pooling layer and the fully connected layer. (Source: Own author)	31
Figure 10 – Example of convolution operation for an image of dimension 6×5 to a kernel of dimension 3×3 . (Source: Own author)	32
Figure 11 – Visualization of causal convolution. (Source: Own author)	33
Figure 12 – TCN residual block and dilated causal convolutions with dilation factors $d = 1, 2$ and 4 with filter size of $k = 3$. (Source: Own author)	35
Figure 13 – Traditional Siamese Network, proposed by Bromley and LeCun [Bromley et al., 1993]. (Source: Das et al. [2016])	36
Figure 14 – Methodology of the systematic review. (Source: Own author)	41

Figure 15 – Problem Class × Number of Publications. (Source: Own author)	43
Figure 16 – Overview of the classification process for the single-stage model. (Source: Own author)	51
Figure 17 – Triplet-loss vs Quartet-loss. In triplet-loss we are separating only one negative while in quartet-loss we are separating two negatives. (Source: Adapted from Schroff et al. [2015a])	57
Figure 18 – MLP classifier architecture. (Source: Own author)	58
Figure 19 – Siamese network general architecture. BaseNetwork generates the latent representation for each input, and loss is calculated to update weights. The grey drawing represent the represents the fourth BaseNetwork and Input for quartet-loss. (Source: Own author)	59
Figure 20 – SequentialFeaturesNetwork architecture. (Source: Own author)	61
Figure 21 – AggregatedFeaturesNetwork architecture. (Source: Own author)	62
Figure 22 – Session types distribution and Items purchasing distribution. (Source: Own author)	65
Figure 23 – Cumulative session’s length. (Source: Own author)	65
Figure 24 – Session sizes vs buying rate. (Source: Own author)	66
Figure 25 – Time features vs Buying rates. Weekdays are the days of the week, such as Monday, Tuesday, Wednesday. Weeks of the year refers to each of the 52 weeks in each year. (Source: Own author)	66
Figure 26 – dwelltime (min) vs buying rate. (Source: Own author)	67
Figure 27 – Feature item_clk_rank vs buying rate. (Source: Own author)	68
Figure 28 – Feature item_purc_rank vs buying rate. (Source: Own author)	68
Figure 29 – Categories vs Buying rate. (Source: Own author)	69
Figure 30 – Proposed models naming diagram.	74
Figure 31 – Baseline naming diagram.	75
Figure 32 – Recall metrics for best models that were outstanding in $S_{i_{bn}}$ and $S_{i_{bb}}$. (Source: Own author)	81
Figure 33 – Confusion matrices for the presented models. The lighter the cell color, the higher the score. (Source: Own author)	82

Figure 34 – Recall by session lengths for baseline models. (Source: Own author)	84
Figure 35 – Recall by session lengths for proposed models which uses aggregated-features. (Source: Own author)	85
Figure 36 – Recall by session lengths for proposed models which uses sequential-features. (Source: Own author)	86
Figure 37 – Embeddings visualizations for TCNQuartet and TCNTriplet for small and large-sessions. (Source: Own author)	90
Figure 38 – Embeddings visualizations for AGQuartet and AGTriplet for small and large-sessions. (Source: Own author)	91
Figure 39 – Embeddings visualizations for proposed models with K-means. (Source: Own author)	92

List of Tables

Table 1 – Confusion matrix for a binary classification problem.	39
Table 2 – Inclusion criteria	41
Table 3 – Exclusion criteria	42
Table 4 – Example of a session described sequentially by sequential-features for section 4.3.1.1. We developed all features except for item_prob.	53
Table 5 – Features engineered for the model for the section	55
Table 6 – FeaturesEncodingLayer details. The FeaturesEncondingLayer is the concatenation of multiples encodings networks.	60
Table 7 – Baselines configurations. SCCE: sparse-categorical-cross-entropy, BCE: binary-cross-entropy. RUS, random under-sampling. 1st: first-model, 2nd: second-model, Both: for both first and second stages. SIFN: AggregatedFeaturesNetwork, SFN: SequentialFea- turesNetwork.	71
Table 8 – Model proposals parameters for siamese networks. The "na"stands for "not applicable"	75
Table 9 – F1-score results for our baseline and proposed models. The first four models in the first part of the divider are our baselines.	79
Table 10 – Precision and recall results for our baseline and proposed mo- dels. The first four models in the first part of the divider are our baselines.	80
Table 11 – Results for MRSNC for presented models. The higher the metric value, the better the result.	87
Table 12 – Results for MSNC for presented models. The lower the metric value, the better the result.	87
Table 13 – Results for MRSNC and MSNC for non-buyer-sessions. For MRSNC the lower, the better the result. For MSNC the higher, the better the result.	88
Table 14 – Results for Silhouette Coefficient for presented models.	92

List of abbreviations and acronyms

I_a Anchor input

I_p Positive input

B_s Set of items bought

c_i click in item

I_s Set of items clicked

s Customer session

S_b Session which have at least one buy event

S_{nb} Session which that do not end in a transaction

aggregated-features Aggregated features

AggregatedFeaturesNetwork Deep neural network for aggregated-features

AGQuartet Deep neural network with handcrafted features and quartet-loss

AGQuartet-CW Deep neural network with handcrafted features and triplet-loss
with class-weights

AGTriplet Deep neural network with handcrafted features and triplet-loss

AGTriplet-CW Deep neural network with handcrafted features and quartet-loss
with class-weights

category Item category feature

day Day feature

dweltime Time between clicks feature

GRU Gated Recurrent Unit

hour Hour feature

I_n Negative input

I_n^1 First-negative input

I_n^2 Second-negative input

item_clk_rank Item clicks rank feature

item_id Item identifier feature

item_prob Item probability of purchasing

item_purc_rank Item purchase quantity rank feature

item_session_prob Item probability of purchasing by sessions

LSTM Long Short Term Networks

MLPs Multi-layer perceptrons

month Month feature

MRSNC Mean of reciprocal smallest number of clicks

MSNC Mean of smallest number of clicks

RNNs Recurrent neural network

RomovNN-S Single-stage neural Network with Romov features

RomovNN-T TWO-stage neural Network with Romov features

RSNC

sequential-features Sequential features

SequentialFeaturesNetwork Deep neural network for sequential-features

Si_{bb} Buyer session and buyer item

Si_{bn} Buyer session and non-buyer item

Si_{nn} Non-buyer item-session

Siamese-Quartet Siamese quartet approach

Siamese-Triplet Siamese triplet approach

SNC Smallest number of clicks

TCNBaseline-S Single-stage TCN neural network with sequential features

TCNBaseline-T Two-stage TCN neural network with sequential features

TCNQuartet TCN Sequential quartet-loss model

TCNQuartet-CW TCN Sequential quartet-loss model with class-weights

TCNTriplet TCN Sequential triplet-loss model

TCNTriplet-CW TCN Sequential triplet-loss model with class-weights

weekday Weekday feature

Contents

1	INTRODUCTION	16
1.1	Problem context	17
1.2	Motivation	18
1.3	Research objective	19
1.4	Approach	19
1.5	Contributions	21
1.6	Outline	21
2	FUNDAMENTALS	22
2.1	Machine learning	22
2.2	Artificial neural networks and deep Learning	24
2.2.1	Training a neural network	25
2.2.2	Transfer learning	26
2.3	Recurrent neural networks	26
2.4	Sequence to Sequence and Attention mechanisms	29
2.5	Convolution Neural Networks	31
2.6	Temporal Convolution Networks	32
2.7	Siamese Networks	34
2.7.1	Triplet loss	36
2.8	Principal Component Analysis	37
2.9	Silhouette Score	37
2.10	Evaluation metrics	38
2.10.1	Confusion matrix, precision, recall, f1-score	38
2.10.2	Mean Reciprocal Rank and Mean Rank	40
3	RELATED WORK AND LITERATURE REVIEW	41
3.1	Systematic literature review	41
3.2	Research results	42
3.3	Research analysis	44
3.3.1	Buyer classification (P1)	44

3.3.2	Feature prediction (P2)	44
3.3.3	Purchase intention prediction (P3)	45
3.3.4	Next-item recommendation (P4)	46
4	PREDICTING PURCHASE INTENTION THROUGH SINGLE-STAGE SIAMESE DEEP LEARNING MODELS	48
4.1	Formal problem statement	49
4.2	The single-stage classes	50
4.3	Single-stage siamese networks approaches	51
4.3.1	Input features	52
4.3.1.1	sequential-features inputs	53
4.3.1.2	aggregated-features inputs	54
4.3.2	Loss function	56
4.3.3	Classification	57
4.4	Base network architectures	58
4.4.1	SequentialFeaturesNetwork	59
4.4.2	AggregatedFeaturesNetwork	62
5	METHODOLOGY AND EXPERIMENTS	64
5.1	Dataset	64
5.2	Methodology	69
5.2.1	Model naming	73
5.2.2	Hyperparameter configuration	75
5.2.3	Model evaluation	76
5.2.3.1	Simulating an online evaluation	76
5.3	Experimental results	78
5.3.1	Overall results	78
5.3.2	Recall metric by sessions lengths	83
5.3.3	Simulated online evaluation	87
5.3.4	Embeddings visualization	89
5.4	Final considerations	93
6	CONCLUSIONS AND FUTURE WORK	94
6.1	Conclusions	94

6.2	Future works and Limitations	95
	Bibliography	96

1 Introduction

Year after year since its inception, e-commerce's share of the total human shopping interactions keeps increasing. By 2023, online sales by retail stores are expected to accumulate 6.17 trillion dollars. Latin America was responsible for 85 billion dollars in 2021, a 25% growth compared to 2020 when it reached 68 billion dollars. The estimate for the United States is that it will reach 875 billion dollars by the end of 2022. China is currently the leader in sales, accounting for 52.1% of sales worldwide, reaching a total of about 2 trillion dollars in 2021. ¹

This market represents great potential for e-commerce activities and encourages large companies to create new retention strategies to create personalized campaigns and increase profit. One important approach to do so is leveraging user-provided data, such as their profile, previous purchases, and browsing behavior. Even when the user is not logged, user data can be saved using unique keys created by cell phone browsers, desktop programs, or the web platform itself. In this way, e-commerce stores can make use of models on the data to make real-time predictions to increase purchases or create personalized campaigns to engage consumers.

The deployment of such techniques enabled merchants to significantly grow sales profits by knowing which customer niche they are dealing with and how marketing and advertising techniques can be applied. According to Phillips [Phillips et al., 2005] merchants can improve profit by between 2% and 11% depending on the contributing variable (e.g., price management, the variable cost, etc.), by adjusting the product placement according to each consumer niche. In the fluid and highly competitive world of online retailing, these margins are significant, and understanding a user's shopping intent can positively influence the final profit [Sheil et al., 2018]. Moreover, most of the research in this area is done on not publicly disclosed datasets or with a lack of details regarding method-specific implementations, making it even harder for smaller retailers to compete with bigger merchants.

¹ Total Retail Sales. <<https://www.emarketer.com/>>

1.1 Problem context

All click data for a certain online user within a given time limit is called a *session*. When a session has at least one item purchased, we call it a *buyer-session* and when there is none, we call it a *non-buyer-session*. Figure 1 exemplifies a customer session.

The problem of predicting purchases given a session can be difficult. One of the adversities is the fact that the data is heavily unbalanced. We can exemplify this with the RecSys dataset [Recsys, 2015], a reference dataset for open data research on shopping behavior, where only 5.5% of users are buyers. Also, in the Retail Rocket dataset [Rocket, 2017] buyers are only 0.7%. A further problem is that user sessions can be too short (i.e., less than 3 clicks) or too long (i.e., more than 200 clicks). This situation can lead a classifier to misidentify temporal patterns. Moreover, several factors, as showed by Toth [Toth et al., 2017], can influence user purchase intent. These factors include concerns about costs, perceived decision difficulty, and selection conflicts due to many similar choices.

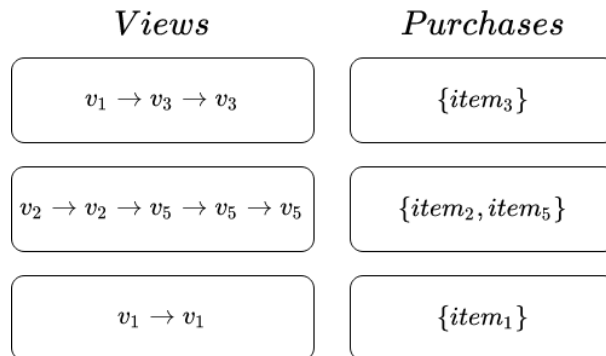


Figure 1 – Sessions data illustration for customers. On the left we have consumer click views on items and on the right we have their item’s ids purchases. Our goal is to forecast purchases through views. (Source: Own author)

In this dissertation, we intend to solve the purchase prediction problem by identifying the sessions that end in a buy and the items purchased within a session using historical click product views. Naturally, much of the literature break this problem down into two questions to facilitate the implementation of the algorithms: 1) Which sessions-clicks will have at least one purchased item

(buyer classification problem) 2) Within the buyer sessions, which items will be purchased (item prediction problem).

In the literature, most of the researchers solved the problem of predicting purchases based on views using two models, one for each specific question. When this approach is used, we call it a **two-stage model**, one for finding buyer sessions and another to find which item was bought. Although theoretically harder, We believe it is also possible to solve this problem using a single model. We call this approach a **single-stage model**. Each approach contains its pros and cons, as we will see in the next sections.

In 2015, Romov [Romov et al., 2015] solved the RecSys Challenge with a two-stage approach. Romov showed that this problem could be solved by classical machine learning algorithms using extensive feature engineering in two stages. In this work, differently from Romov, we intend to solve the problem using deep neural networks with only a single-stage model.

1.2 Motivation

Romov [Romov et al., 2015] won the RecSys 2015 contest using the gradient boosting algorithm and feature engineering through the two-stage algorithm. When machine learning models are deployed in two-stages to solve a problem, [the second model does not use information from non-buyer-sessions to forecast the item, which can also lead to losses of important information.]. Moreover, when using two-stage models, if the first-stage model fails, so will the second-stage model.

Solving the problem with a single-stage model through deep learning should treat these problems since we will model both questions as a single problem and will use all available information [(i.e non-buying sessions and non-purchased items)].

However, solving this problem in a single stage is not an easy task. A single-stage model prediction will contain most of the labels with an empty set (no purchase) since about 95% of customers are non-buyers. This imbalance makes single-stage models generally worse in precision than two-stage models, as Chen [Chen et al., 2016] reported.

The problem of using two stages or not also happens in sentiment analysis: A sentence may first be classified as subjective or not and then the sentence polarity is predicted as positive or negative. Recent work such as Devlin [Devlin et al., 2018], and Sun [Sun et al., 2019a] have shown that sentiment analysis can be performed in a single stage through the BERT deep attention neural network.

Most datasets do not contain data completely clean and well-tracked. A poorly done extraction job can cause a shortage of data labels or other features. Besides that, other reasons also favor the imbalance. In some cases, the very nature of the problem contains a distorted distribution, as in the case of fraud and disease detection [Johnson et al., 2019]. Moreover, unbalanced datasets always favor the sensitivity of the model: That is, learning models tend to be more accurate in the classes that appear at most, as studies show that unequal datasets worsen algorithm models [Eggermont et al., 2004]. In this work, we show in the unbalanced data of e-commerce that single-stage models can improve classification even in an uneven number of labels.

1.3 Research objective

Our general objective in this work is to demonstrate that single-stage is competitive to two-stage approach for the problem to identify buyer-sessions and purchased items. The specific objectives are:

1. Explore ways to represent problem classes and input for the single-stage approach.
2. Minimize the effect of the imbalance problem.
3. Generate session representation (or embeddings) that easily differentiate buyer-sessions and non-buyer-sessions as well as which item is bought.
4. Evaluate whether the single-stage approach is viable to the problem.

1.4 Approach

Although previously proposed models achieved satisfactory results, we believe that it is possible to improve those models. To achieve our research goals,

we explore the following topics: how to represent the problem, how to represent a session, and how to represent relationships between sessions.

How to represent the problem: In 2015, several competitors submitted a solution for RecSys competition. Most of the solutions, including the winner, Romov [Romov et al., 2015], solved the problem with two models. The first model contained only two classes, one that described which session was a buyer and the other that the session was a non-buyer. Likewise, the second model also contained two classes, one that described whether the selected item was purchased and another that described whether the item was not purchased. For our single-stage approach, we developed three classes to represent the problem. The first, Si_{nn} , refers to items that have not been purchased. The second, represented by Si_{bn} , refers to items that are in buying sessions but were not bought. Finally, a Si_{bb} class represents items that have been purchased.

How to represent a session: Previous works have shown that to solve the purchase prediction problem, two types of input representations are commonly created. A representation that describes the characteristics of clicks over time, which we call sequential-features. And another representation that is not temporal, that is, it characterizes the clicks observing the entire session and also, when needed, features from a specific item in the session. We call this representation as aggregated-features. In this work, we showed how we can implement these two representations and how to implement neural network models that can read these inputs.

How to represent relationships between sessions: Session relationship representations must be thought in order to address the problem of imbalance. A common way to solve the problem of class imbalance is through deep metric learning, in which a network tries to represent the input in latent space. Through these learned representations and embeddings, the minority classes tend to be further away from the majority classes due to the application of the loss function. In our context, we need the purchase classes, Si_{bn} and Si_{bb} , which will be formally described in chapter 4, to be far from the non-purchase class. In this work, we deployed the triplet-loss [Schroff et al., 2015b] for the purchase prediction problem. However, in our case, there is the presence of two very

distinct negative classes regarding items that were not bought: The ones from a "buying session" and the ones from a "non-buying" session. These two classes are treated in the same way in triplet-loss. To tackle this scenario, we developed a variation of the triplet-loss with one extra negative example: the quartet-loss. Using quartet-loss, we can consider the classes independently and also control how much the network pushes the different classes through weights.

1.5 Contributions

According to our systematic review, no single-stage model proposed was as efficient as the other two-stage models previously proposed for this problem. Our main contribution is showing how to implement a single-stage model using siamese networks to extract features for both session buyer and item classification. [We use the Siamese network because the generated representation separates the classes in the latent space so that the problem of unbalance is minimized]. Besides, we have other contributions, such as:

1. Implementation of a new loss function, quartet-loss, which handles two possible negative classes of purchases separately.
2. Achieving excellent results that overcame the state-of-the-art method by Romov [Romov et al., 2015] using single-stage models.
3. The design of Siamese network models for sequential-features and aggregated-features.

1.6 Outline

This dissertation is organized in the following structure. The next Chapter 2, contains a list of our proposal's theoretical concepts. Chapter 3 presents details of related work returned through a systematic review. In Chapter 4 we present our solutions to the purchase intention problem. Chapter 5 describes our results. Finally, Chapter 6 shows our considerations and future works.

2 Fundamentals

In this chapter, we present the fundamental concepts of this work. Section 2.1 covers machine learning concepts. Section 2.2 details artificial neural networks and deep learning theory. Section 2.3 discusses recurrent neural networks. Section 2.4 ends with the attention mechanism. Section 2.5 covers convolution neural networks. Section 2.6 discusses temporal convolution network. Section 2.7 presents the definition of siamese network and triplet-loss. Section 2.8 covers about principal component analysis. Section 2.9 introduce Silhouette score. And finally, Section 2.10 discuss some evaluation metrics: confusion matrices, precision, recall and f1-score.

2.1 Machine learning

Machine Learning can be summarized as a set of techniques in which, through selected data, we intend to make a model understand or learn patterns in the data [Palmer et al., 2011]. Mitchell defines it [Mitchell, 1997] more formally as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ". A good example of a machine learning application is the implementation of recommendation systems for movies and series as is done at Netflix. In 2006, Netflix launched a competition 'The Netflix Prize', to predict user preferences and improve the accuracy of movie recommendation models by at least 10% [Bennett et al., 2007].

In general, the machine learning tasks can be divided into two phases [Hertzmann and Fleet, 2010]: The training phase and the test phase. The data must be broken into two sets before this next phase: the training data and the test data. The data splitting is designed to simulate model performance at the production level. The training data represents all existing collected data that the model should use to learn a pattern. The test data represents the future or the actual forecast at the production level.

As an example, suppose we want to recognize handwriting data from

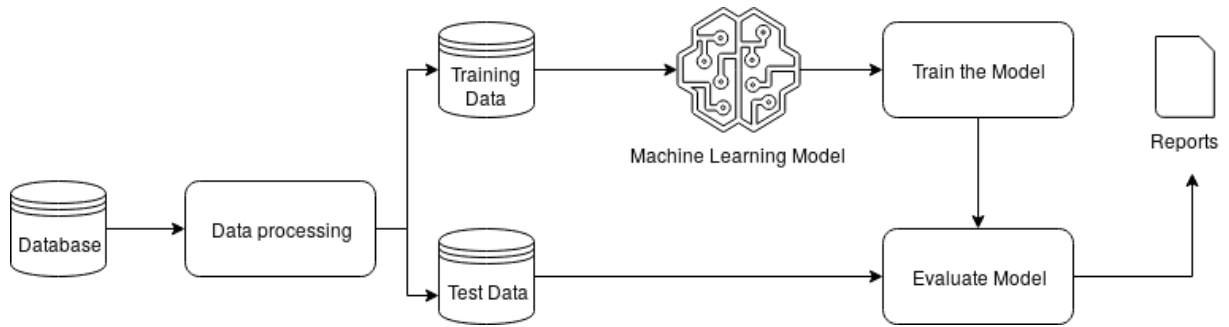


Figure 2 – Machine Learning workflow. (Source: Own author)

28×28 pixel images [Lecun et al., 1998]. Our purpose is to build a machine learning algorithm that takes the image as input and produces the identity of each digit $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. A machine learning approach can be adopted in which a large set of N digits $\{x_1, \dots, x_N\}$, the training the data, is used to tune the parameters of an adaptive model. The categories of the digits in the training set are known in advance, typically by inspecting them individually and hand-labeling them. We can express the category of a digit using a target vector t , which represents the corresponding digit's identity. The result of running the machine learning algorithm model takes new digit images x , the test data, as input and generates an output vector y , encoded in the same way as the target vectors from the training data [Bishop, 2006].

A very important step in machine learning is data pre-processing. The data needs to be preprocessed so that the model understands as well as possible. For example, the digit recognition problem must scale the digits and move them to a position so that there is not so much variability. The data must be preprocessed for both training and test data sets. Once the data is properly processed and divided into test and training, the model can be trained on training data and evaluated in the test data set. Once the test results are achieved, we can report through various evaluation metrics present in the literature. Figure 2 shows a machine learning model workflow. [Bishop, 2006].

In the field of machine learning, we classify in three types for the majority of tasks. The first most common is the **Supervised Learning** which the training data is labeled with the correct answers. The two most common types of supervised learning are classification (where the outputs are discrete labels) and regression (where the outputs are real-values numbers). The second most

common machine learning type is the **Unsupervised Learning** which are given a collection of unlabeled data where we want to analyze and discover some patterns within. The most important examples are dimension reduction and clustering. Furthermore, the last most common is Reinforcement Learning, which is currently growing and being mixed with other learning types. **Reinforcement Learning** is learning in which the agent seeks to learn the optimal set of actions following a policy based on past actions [[Hertzmann and Fleet, 2010](#)].

2.2 Artificial neural networks and deep Learning

Artificial Neural Networks are machine learning models that are based on a biological model of the human brain and its neurons. Therefore, we want to find a mathematical approach to process information similarly to biological systems. However, today we know direct biological realism would impose entirely unnecessary constraints [[Bishop, 2006](#)].

An artificial neural network, or *perceptron*, is a supervised model that was based on the biological neuron, initially used to solve binary classification problems. So what a perceptron tries to solve is a nonlinear regression problem. A nonlinear function of the input variables is fitted by optimizing the weights of a nonlinear function $y(W, X)$. Where W is the vector to be computed, X is the vector of known input variables, and y is the unknown output [[Dean, 2014](#)]. Figure 3 is a diagram for a simple neural network.

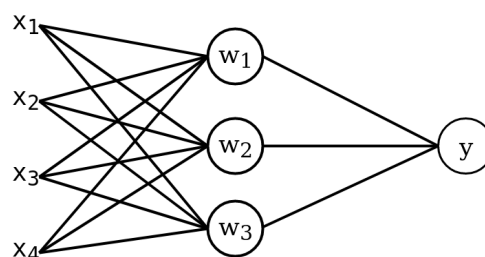


Figure 3 – Simple Neural Network

However, the most commonly used neural network type contains three or more layers of multiple perceptrons; input, one or more hidden, and output layers. These feed-forward networks use squashing activation functions for the neurons in the hidden layer. Recently neural networks of five or more hidden

layers are useful for feature detection in images and natural language processing. Networks like these, generalized to include multiple hidden layers, are referred to as *multilayer perceptrons* (MLPs). However, the term *neural network* has been applied to various other architectures, such as autoencoders, convolutional neural networks, radial basis function networks, self-organization maps, etc. Recently, the term *deep learning* has become widely used to describe machine learning methods that can learn more abstract concepts from multiple hidden layers and achieve optimal results in several areas [Dean, 2014].

2.2.1 Training a neural network

To train a neural network it is necessary to call a method or function to compute the weights that minimize a highly nonlinear objective function for a set of training examples. The objective function is not the network output. It represents the network error: the real difference between what the network should be outputting and what the networks is predicting, overall training examples. The computation of optimizing a minimum value of a nonlinear differentiable function is usually accomplished using some variation of the gradient descent method. For neural networks, the gradient is the vector of partial derivatives of the objective function concerning each weight. Each partial derivative value in this vector represents the amount the objective function changes for a small change in weight [Dean, 2014].

This mechanism to modify each hidden neural weight is called the *back-propagation* error. At each step of the training network, two distinct stages occur: in the first stage, as mentioned, the derivatives of the error function for the weights must be evaluated, and errors are propagated backward through the network. In the second stage, the derivatives are then used to compute the adjustments to be made to the weights [Bishop, 2006].

The gradient descent optimizer method uses the idea that we can use a randomly chosen initial set of weights and slowly move in the direction of the negative gradient of the objective function (in the steepest downhill direction) you can eventually end up at the bottom (global optima), or at least at a place where the gradient is a zero (local optima). The reason the function has to be

differentiable is that gradient exists, which it does for multi-layer perceptrons using standard activation functions. The next lines show a default version of the gradient descent algorithm [Dean, 2014]:

Gradient descent algorithm

$$W_{n+1} = W_n - \lambda \nabla f(W_n)$$

where

W_n = current vector of weights

λ = learning rate

$\nabla f(W_n)$ = objective function to be minimized

2.2.2 Transfer learning

When it is impossible to collect a huge amount of labeled data to feed a deep network model, transfer learning could be a solution. First, we train a base network containing a large set of data, and then we repurpose the learned features or *transfer* them to a second target network to be trained on the target dataset. This process works if the features are general, meaning suitable to both base and target tasks, instead of specific to the base task. Recent studies have taken advantage of this fact to obtain state-of-the-art results when transferring from higher layers ([Donahue et al., 2014], [Zeiler et al., 2014], [Sermanet et al., 2013]) [Yosinski et al., 2014]. Figure 4 shows the workflow for transfer learning to cats and dogs classifying.

2.3 Recurrent neural networks

Recurrent neural networks or RNNs are a family of neural networks for processing sequential data. A recurrent neural network is a neural network that is specialized for processing a sequence of values $[x(1), \dots, x(n)]$. The broad idea is that RNNs are neural networks in which there is some dynamic change over time. Recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialization. Recurrent networks share parameters differently. Each member of the output is a function of the previous members of the output. Each member of the output is produced using the same

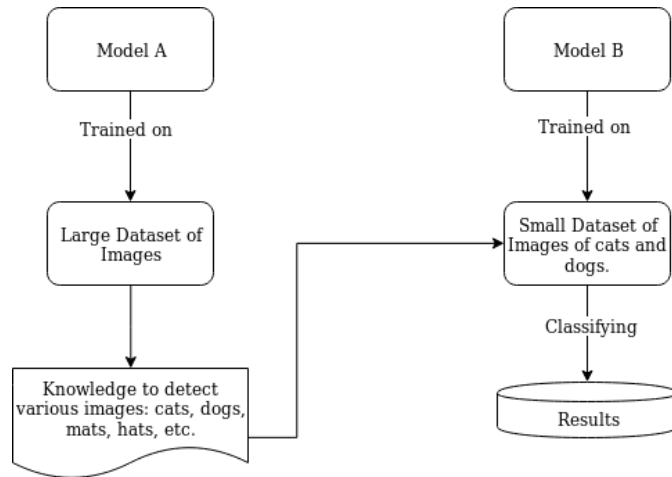


Figure 4 – Transfer Learning diagram workflow for cats and dogs recognition. (Source: Own author)

update rule applied to the previous outputs. In effect, an RNN is a type of neural network that has an internal loop [Goodfellow et al., 2016]. Figure 5 shows a single cell of an RNN.

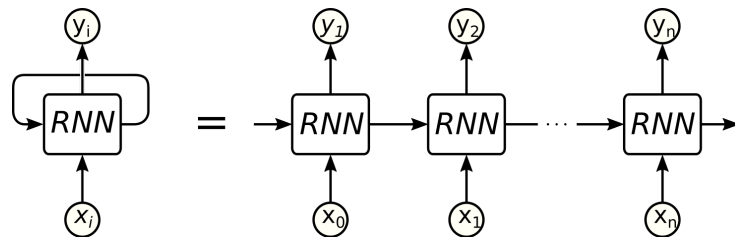


Figure 5 – Sequential inputs directly in a RNN cells. (Source: Own author)

Even though RNNs are quite powerful, they suffer from a vanishing gradient problem that hinders them from using long-term information. They are suitable for storing memory with low numbers of past iterations, but many instances do not provide satisfactory results. New types of recurrent neural networks were developed; one of the most important is the Long Short Term Networks LSTM and the Gated Unit Recurrent GRU.

An LSTM unit comprises a memory unit, an input gate, an output gate, and a forget gate. The memory unit is responsible for remembering the values over the temporal sequence. The gating mechanism regularizes the values of the hidden weights that go through the intern connections of the LSTM. The expression long short-term refers to the fact that LSTM is a model for short-term memory that can last for a long time. An LSTM is well-suited to classify, process

and predict time series better than traditional RNNs [[Hochreiter and Schmidhuber, 1997](#)]. Consider that each LSTM cell compute a hidden state s_{t-1} as previous input and x_t as instance input to predict the next hidden state s_t . The next lines show the calculation for a hidden state s_t :

$$i = \sigma(x_t U^i + s_{t-1} W^i) \quad (2.1)$$

$$f = \sigma(x_t U^f + s_{t-1} W^f) \quad (2.2)$$

$$o = \sigma(x_t U^o + s_{t-1} W^o) \quad (2.3)$$

$$g = \tanh(x_t U^g + s_{t-1} W^g) \quad (2.4)$$

$$c_t = c_{t-1} \odot f + g \odot i \quad (2.5)$$

$$s_t = \tanh(c_t) \odot o \quad (2.6)$$

LSTM hidden state calculation.

Despite the success of LSTMs, new researchers wondered if all the complexity of LSTMs was needed. One of the exemplary architectures widely used until 2019 is the gated recurrent units, a simplified version of LSTMs. The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit [[Goodfellow et al., 2016](#)]. Empirical results showed that it is not a cleaner, better neural network, but GRU is faster to train [[Chung et al., 2014](#)]. A GRU has two gates, a reset gate r , and an update gate z . Intuitively, the reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory to keep around. Next, we show the GRU equations:

$$z = \sigma(x_t U^z + s_{t-1} W^z) \quad (2.7)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r) \quad (2.8)$$

$$h = \tanh(x_t U^h + (s_{t-1} \odot r) W^h) \quad (2.9)$$

$$c_t = c_{t-1} \odot f + g \odot i \quad (2.10)$$

$$s_t = (1 - z) \odot h + z \odot s_{t-1} \quad (2.11)$$

GRU hidden state calculation.

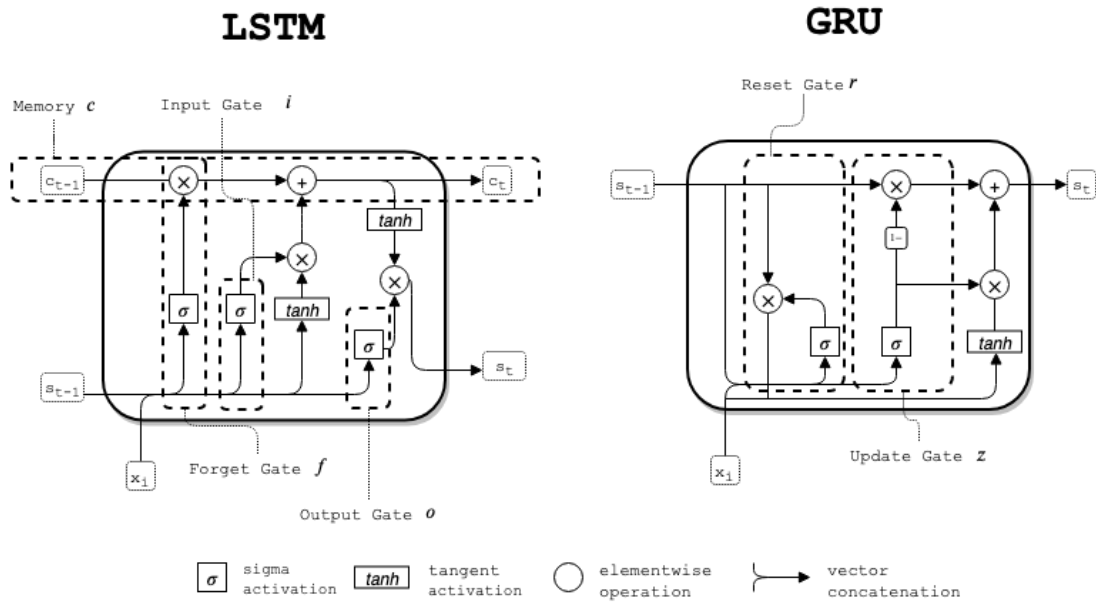


Figure 6 – LSTM and GRU representation with internal equations elements emphasized. (Source: Own author)

Figure 6 demonstrates the LSTM and GRU basic unit cells for the presented equations in this section.

2.4 Sequence to Sequence and Attention mechanisms

We know that the sequence to sequence learning model has produced excellent results in natural language processing [Sutskever et al., 2014]. It aims to transform an input sequence to a new one, and both sequences can be of arbitrary lengths. We call this kind of problem a sequence to sequence learning or seq to seq problem. Examples of transformation in seq to seq problems include machine translation between multiple languages in either text or audio, question-answer dialog generation, or even parsing sentences into grammar trees.

To create a sequence to sequence learning model it is expected to use an encoding and decoding architecture. An encoder processes the input sequence and compresses the information into a context vector (or sentence embedding or thought vector) of a fixed length. This representation is expected to be a good summary of the meaning of the whole source sequence. The decoder part is initialized with the context vector to emit the transformed output. It is common to use the last state of the encoder network as the initial decoder state. Both the encoder and decoder are recurrent neural networks. Figure 7 exemplifies it.

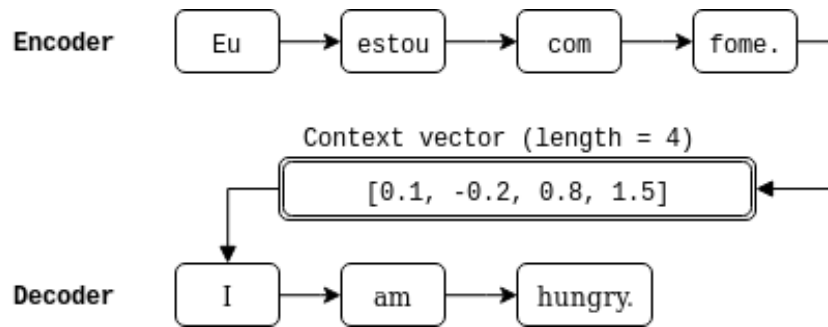


Figure 7 – Encoder and decoder model example from portuguese to english. (Source: Own author)

Bahdanau [Bahdanau et al., 2014] solves a critical problem in the encoding and decoding traditional model. The sequence to sequence model is incapable of remembering very long sentences. It often forgets the first part once it completes processing the whole input. Bahdanau proposes an attention mechanism that does not just create a single context vector of the encoder’s last hidden state but creates a shortcut between the context vector and the entire source input. While the context vector has access to the entire input sequence, we do not need to worry about forgetting. The alignment between the source and target is learned and controlled by the context vector [Weng, 2018]. Figure 8 shows the attention mechanism in recurrent neural networks.

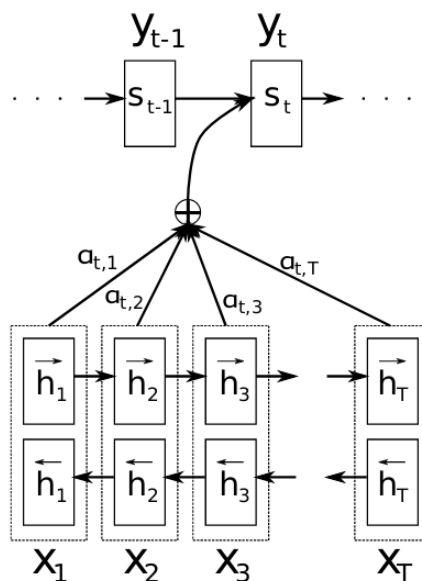


Figure 8 – Graphical illustration of the proposed model by Bahdanau to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) . Image taken from [Bahdanau et al., 2014]

Besides the commented model, several other attention mechanism models were also proposed. We can mention location-base, general and dot-production [Luong et al., 2015].

2.5 Convolution Neural Networks

Convolutional neural networks (CNN) are a type of artificial neural network widely applied in the areas of image classification, object detection, image segmentation, and other tasks. A convolutional neural network is composed of an input layer, hidden layer, and output layer with shared-weights architecture. Hidden layers are composed of layers that perform a convolution operation. In a convolution operation, filters (or kernels) are applied in a sliding way on the input, producing an intermediate layer of responses called a feature map that can be used by the next layer. After generating the convolution layer, a new set of layers can be connected, such as pooling layers and a fully connected layer. The figure 9 shows an example of an architecture to solve a classification of cats and dogs. In this example, we can see that the convolution layer applies a filter (or kernel) to the input and throws its output to the pooling layer. This process occurs one more time until it passes to a fully neural network for the final classification. [Lecun et al., 1998]

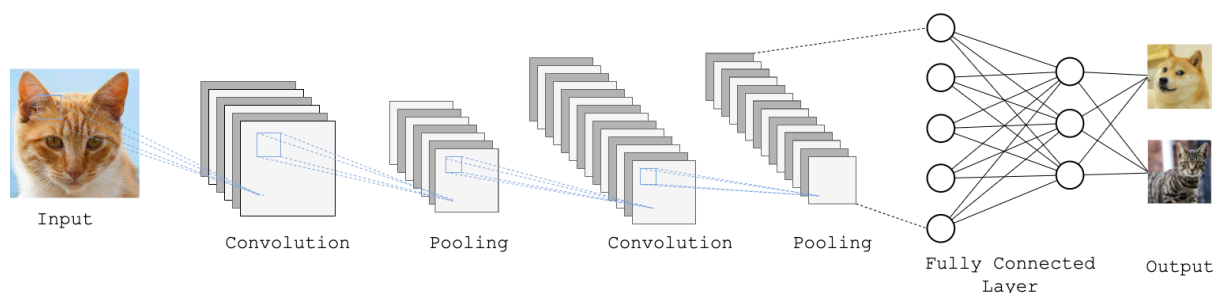


Figure 9 – A common convolutional neural network architecture. In general, CNN has three components, the convolutional layer, the pooling layer and the fully connected layer. (Source: Own author)

The convolution layer is composed of three components: the input, the filter, and the feature map. Filters move in a sliding window (which can vary in size) to find the important features in the input. In an image of a cat, for example, filters can pick up important features such as ears, teeth, and fur. This process is called convolution. The filter is implemented by a two-dimensional matrix with

weights that is applied to an area of the input and then a dot product is calculated between the pixels and the filter weights. The output of the dot product is then fed into an output array. Afterward, the filter is then continuously applied in strides until it passes through the entire input. The final generated output is called a feature map. The Figure 10 exemplifies the convolution operation for an image of dimension 6×5 to a kernel of dimension 3×3 .

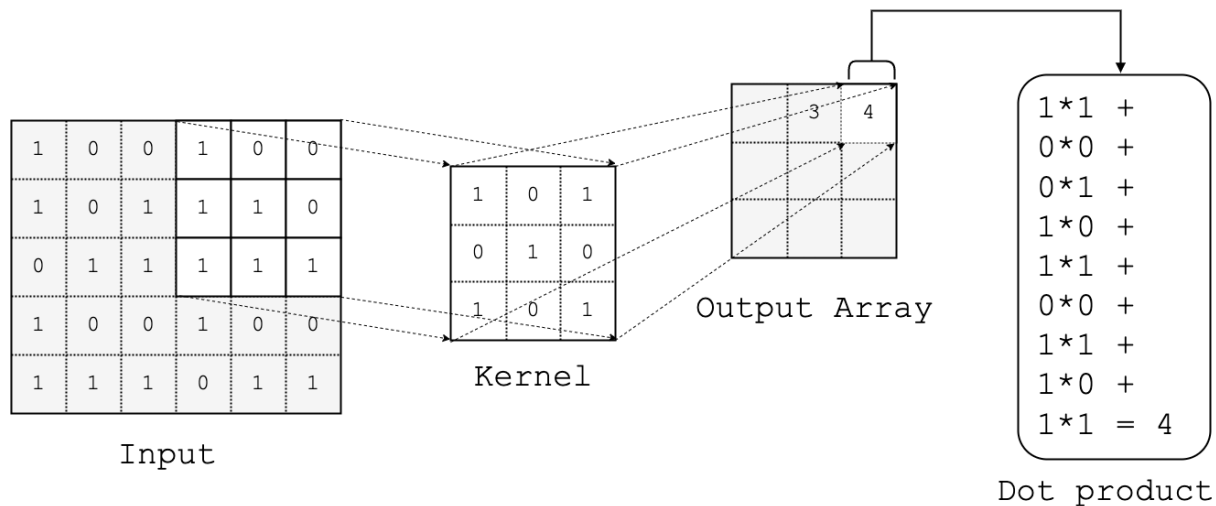


Figure 10 – Example of convolution operation for an image of dimension 6×5 to a kernel of dimension 3×3 . (Source: Own author)

After the convolutional layer generates the feature map, they are passed to a Pooling layer. Pooling layers are similar to convolutional layers, they also pass a filter on the entire input, although the filters do not have any weights. The difference is that the kernel applies an aggregation function to the input. The two most applied functions are the max (which we call max pooling) and average (which we call average pooling) functions. After the convolution and pooling operations, the feature maps are passed to a fully connected layer for the final classification to take place.

2.6 Temporal Convolution Networks

Temporal Convolutions are Networks that have two properties:

1. They contain a 1-dimensional fully connected network to keep the network output the same length as an input sequence.

2. The outputs are only influenced by information of the present and past inputs in each layer by using causal convolutions.

A causal convolution is a type of convolution created for sequential data in which the model does not take away the guarantee of the temporality of the data. That is, the prediction $(x_{t+1}|x_1, \dots, x_t)$ emitted by the model at timestep t cannot depend on any of the future timesteps $x_{t+1}, x_{t+2}, \dots, x_T$. Figure 11 shows a visualization of a causal convolution. You can see that the data doesn't get information from data that is in the future like a common filter does.

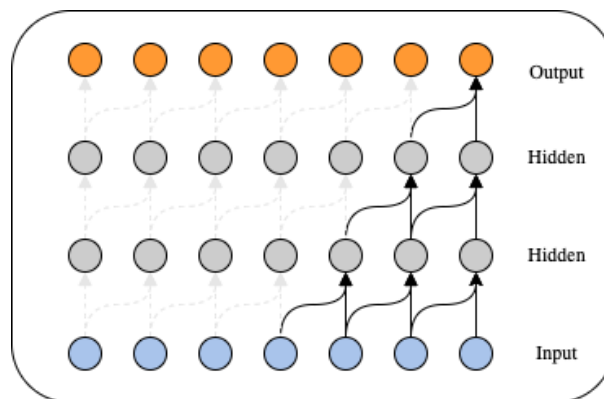


Figure 11 – Visualization of causal convolution. (Source: Own author)

Although recurrent neural networks work well for several sequential modeling tasks, convolutional neural networks also proved to be efficient for this task. Shaojie Bai et al. [Bai et al., 2018] showed that temporal convolutional networks (TCN) can substantially outperform LSTMs and GRUs while demonstrating longer effective memory. Furthermore, using a convolutional network instead of a recurrent one can lead to performance improvements as it allows for parallel computation of outputs.

Simple causal convolution still has the problem of traditional convolutional neural networks, and the modeling length of time is limited by the size of the convolution kernel. In this case, if you want to learn longer dependencies between data, you need to stack many layers linearly. To solve this problem, TCN uses one-dimensional dilated convolution.

The difference between dilated convolution and traditional convolution is that it allows the input of convolution to have interval sampling. In other words, a dilated convolution is a convolution where the filter is applied over an area

larger than its length by skipping input values with a certain step. [van den Oord et al., 2016].

For an input sequence $x \in \mathbb{R}^T$ and a filter $h: \{0, \dots, k-1\} \rightarrow \mathbb{R}$, the dilated convolution operation on element s of the sequence is defined as:

$$H(x) = (x * {}_d h)(s) = \sum_{i=1}^{k-1} f(i) \cdot x_{s-d \cdot i} \quad (2.12)$$

Dilated convolution operation formula.

where $d = 2^v$ is the dilation factor, with v the level of the network, and k is the filter size. The term $s - d \cdot i$ accounts for the direction of the past. Dilation is equivalent to introducing a fixed step between every two adjacent filter taps [Richter, 2020]. Choosing a larger filter size k or increasing the expansion factor d can obtain a larger network receptive field. Similar to the common dilated convolutions, d increases exponentially with the depth of the network layers, which allows the network to use a larger effective history while receiving each input.

Another common method to further increase the network is to connect several TCN residual blocks. The Residual Blocks have two layers of dilated with rectified linear units (ReLU) as non-linearities. Also, Weight normalization [Salimans et al., 2016] is applied to the convolutional filters and a spatial dropout [Srivastava et al., 2014] is added after each dilated convolution for regularization, meaning that at each training step a whole channel is filled with zeros. Figure 12 shows a diagram of a TCN residual blocks with dilated convolutions.

2.7 Siamese Networks

Siamese networks were developed in 1990 by Bromley and LeCun to solve signature verification through an image matching [Bromley et al., 1993] problem. The term Siamese evolved from a concept of physically conjoined twins connected. Analogously, Siamese Neural Networks make use of two or more identical sub-networks (e.g same parameters and layers) to quantify the similarity between the instances. The sub-networks can be common neural

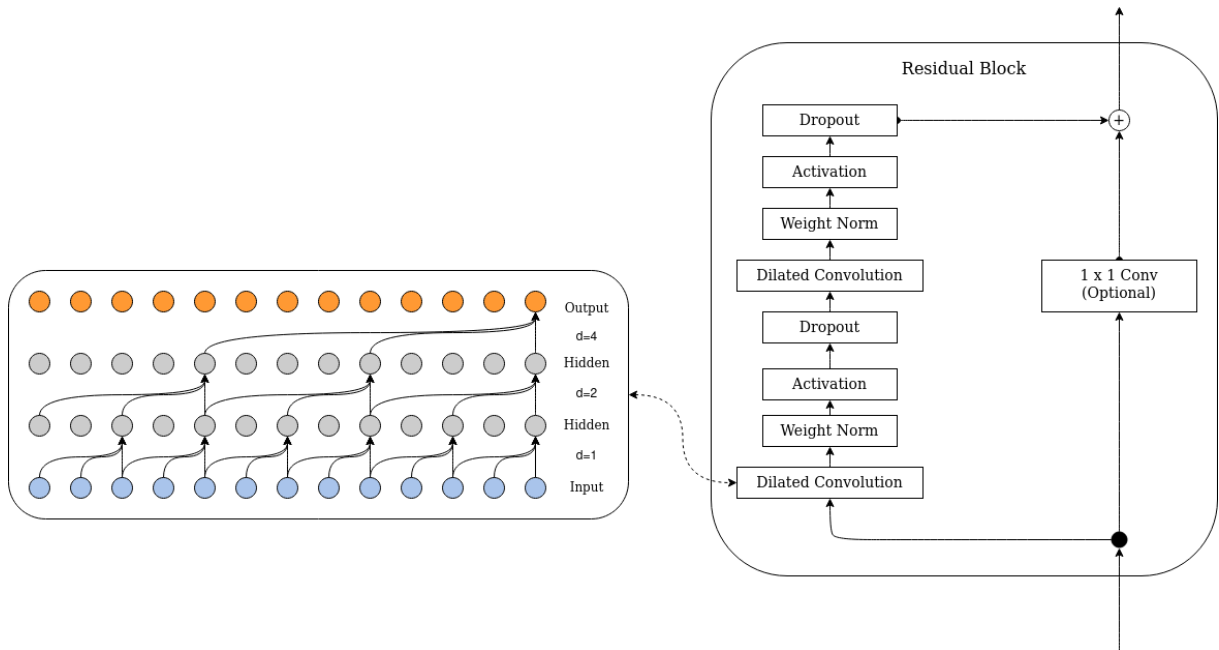


Figure 12 – TCN residual block and dilated causal convolutions with dilation factors $d = 1, 2$ and 4 with filter size of $k = 3$. (Source: Own author)

networks such as convolutional neural networks, recurrent neural networks, multi-layer perceptrons, etc.

The traditional Siamese network proposed by Bromley and LeCun receives the inputs, processes them through the backpropagation algorithm, and generates the outputs for the sub-network pair as predictions. These outputs are compared through some similarity function, usually by cosine or Euclidean distance. You can observe a diagram in the Figure 13. Through this distance, the neural network can identify whether the classes are different or the same. In the case of cosine distance, the network considers it as a different class if the distance is between $[-1, 0]$ or similar if the distance is between the range $[0, +1]$.

A Siamese neural network, in addition to comparing objects, also develops a latent representation for them, usually with lower dimensionality. For this reason, it is possible to reuse the generated representation for other tasks. An example of the use of embeddings in the latent space is the solution to the clothing matching problem by Yuan et [Yuan et al., 2018]. The clothing matching problem tries to find the clothing combinations for people, such as finding shoes of one color for pants of the same color. Yuan et al. solved the task through Siamese networks with visual and categorical features from a contrastiveT loss. The model

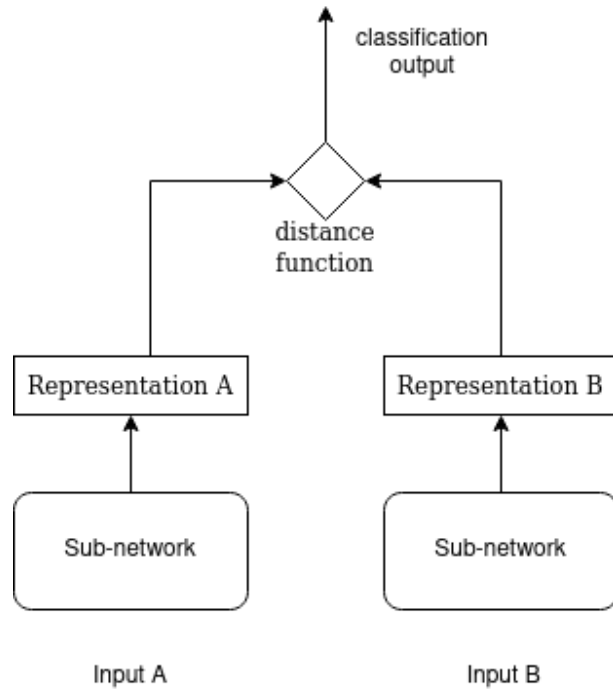


Figure 13 – Traditional Siamese Network, proposed by Bromley and LeCun [Bromley et al., 1993]. (Source: Das et al. [2016])

achieved good results for the recommendation of the Weighted Nearest Neighbor algorithm in the embeddings generated by the Siamese Network.

2.7.1 Triplet loss

A variation of the traditional Siamese network, they are networks that contain three sub-networks, called a triplet network. This network is trained using three inputs, commonly named anchor, positive, and negative. The triplet network is optimized by a cost function called triplet-loss [Ha and Blanz, 2021]. The main idea of triplet loss is to bring instances of the same class close to each other and move instances of different classes away. Hence, the triplet-loss minimizes the distance between the anchor input and the positive input, while trying to maximize the distance between the anchor input and the negative input. Therefore, we can describe the function as follows:

$$L(I_a, I_p, I_n) = \max(\text{dist}(I_a, I_p) - \text{dist}(I_a, I_n) + \text{margin}, 0) \quad (2.13)$$

Triplet-loss formula.

where I_a , I_p and I_n represents anchor, positive and negative respectively.

The $dist(x, y)$ function is a distance between x and y vectors. As a distance, Euclidean distance and cosine are commonly used. The margin is a threshold that separates the semi-hard triplets from the easy and hard negatives, proposed by Schroff et al. [[Schroff et al., 2015b](#)].

2.8 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction method widely used in statistics in multivariate analysis. From the reduction of dimensionality, it is possible to represent data of a larger dimension in a smaller dimension that is more understandable or easier to be modeled. The purpose of PCA is to extract important information into a smaller set of indices called principal components.

The dimensionality reduction by the PCA occurs by maximizing the variation present in the original dataset. This occurs through the principal components that make a linear transformation of the original attributes. The principal components are uncorrelated and ordered such that the first component retains the greatest percentage of variation from the original attributes [[Silva et al., 2019](#)].

2.9 Silhouette Score

The Silhouette Score or Silhouette Coefficient is a method for evaluating and interpreting clustering data. The general idea is that it validates how compact the clusters are (using the intra-distance cluster) and how far apart the clusters are (using the inter-cluster distance). [[Rousseeuw, 1987](#)]

The Silhouette value is in the range between -1 and +1. The closer to +1, the more the data points match their cluster and at the same time are poorly matched to neighboring clusters. If most of the data points are close to +1, then the cluster configuration looks good, otherwise, the configuration is bad.

The Silhouette score for data point i is given as:

In this context, b_i is inter-cluster distance defined as the average distance of the cluster closest to the data point i except for the cluster where i is a part. And a_i is the intra-cluster distance as the average distance from all other data

$$s_i = \frac{b_i - a_i}{\max(b_i, a_i)} \quad (2.14)$$

$$\text{where} \quad (2.15)$$

$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \quad a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \quad (2.16)$$

Silhouette score formula.

points that the i cluster is a part of. The overall Silhouette is computed from the average of the silhouettes of all data points in the dataset.

2.10 Evaluation metrics

Developing quality predictive models requires an important part: evaluation. For a good model evaluation, evaluation metrics are needed. From the metrics, the model building can continue to improve until it reaches a better performance. For this section we describe some of the metrics used in the literature for supervised models: confusion matrix, precision, recall, f1-score, mean reciprocal rank, and mean rank.

2.10.1 Confusion matrix, precision, recall, f1-score

Once a supervised model is trained, we must perform its validation. Validation through the confusion matrix consists of observing the output of the predictions and comparing them with the classes. This comparison consists of verifying if the predictions found the classes correctly and for which classes the model made the predictions wrongly.

Considering a model that makes predictions for a problem with N classes of predictions, a confusion matrix of $N \times N$ is required. Considering $N = 2$, for a binary classification problem, we can create the relation of prediction classes \times actual classes. Table 1 represent this.

		Actual class		
		Positive	Negative	
Predicted class	Positive	TP (True Positive)	FP (False Positive)	Positive predicted Value $\frac{TP}{(TP + FP)}$
	Negative	FN (False Positive)	TN (True Negative)	Negative Predicted Value = $\frac{TN}{(TN + FN)}$
		Sensitivity $\frac{TP}{(TP + FN)}$	Specificity $\frac{TN}{(TN + FP)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Table 1 – Confusion matrix for a binary classification problem.

Table 1 shows the total number of instances organized into predictions and actual classes. We can mention them as follows:

1. TP, refers to the number of instances correctly set for class Positive.
2. TN, refers to the number of correctly set instances for class Negative,
3. FP, refers to the number of instances predicted to be Positive but actually are Negative.
4. FN, refers to the number of instances predicted to be Negative but actually are Positive.

From these nomenclatures, we can define the metrics of precision (or Positive Predicted Value), Negative Predicted Value, Accuracy, Sensitivity (or Recall), and Specificity, defined in Table 1. It is worth mentioning that these metrics can have different importance depending on the problem. For a problem with unbalanced classes, that is, with several classes with great disparity, it may not be significant to observe the accuracy metric. For cancer classification problems, for example, it may be more important to look at the recall metric, to find the greatest number of people with cancer, even if there are a high number of false-positives.

In some situations, we may be required to obtain an optimal value between precision and recall. A good metric for this is the f1-score, which is defined by the harmonic mean of precision and recall. For this metric, harmonic mean is used, as it punishes extreme values. An f1-score can be defined as:

$$\text{f1-score} = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (2.17)$$

F1-score formula.

2.10.2 Mean Reciprocal Rank and Mean Rank

Mean rank (MR) and mean reciprocal rank (MRR) are statistical metrics used to evaluate a list of possible answers to a query in the area of information retrieval. MR and MRR are used in situations where there is only one correct answer. The rank in this case is defined as the position of the correct document for a given query. Therefore, the MR is the average of the correct positions for a query. The MRR is the inverse average of the correct positions for a query. The MRR in this case can be more advantageous than the MR, as it penalizes larger errors. MR and MRR can be mathematically defined as:

$$= \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \quad \text{MR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{rank}_i \quad (2.18)$$

MRR formula.

where rank_i refers to the rank position of the relevant document for the i -th query and Q is the total of queries.

3 Related work and literature review

This section discusses the main scientific works on the purchase prediction problem. We conducted a systematic review to extract related works. After the review was finished, we added up all the results and analyzed different methods related to our problem. Figure 14 shows the methodology of our literature review.

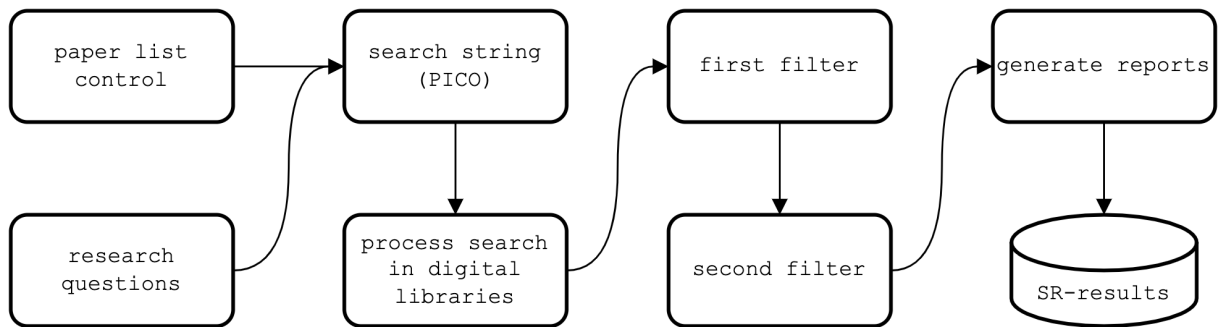


Figure 14 – Methodology of the systematic review. (Source: Own author)

Section 3.1 explains our systematic literature review for our problem. Section 3.2 shows results from our systematic review and analysis.

3.1 Systematic literature review

To conduct the review, we used advanced search in the libraries ACM, IEEE Explore, and ARXIV. We created a string with a pattern that represents our problem for searching in the selected libraries. The string creation was performed based on PICO (Population, Intervention, Comparison, and Outcomes) [Keele et al., 2007] and was based on technical terms and synonyms present in a created papers control list.

Table 2 – Inclusion criteria

IC01	Research papers that predict online user events (buy, browser, add-to-cart, etc) from unidentified users through deep learning.
IC02	Research papers that implement recommendation systems through unidentified user data using deep learning.
IC02	Research papers that implement recommendation system using implicit feedback.

Table 3 – Exclusion criteria

EC01	Papers that not related to e-commerce context.
EC02	Papers that use direct feedback to solve the problem. (e.g direct product rating, text review, comments, search string, etc.)
EC03	Papers published before 2014

After the search string is created, we filtered the results using some criteria to extract the primary works. The inclusion and exclusion criteria are present in tables 2 and 3. The filtering process occurred in two steps:

1. The first filter is to classify who will be present for the next filter, just matching the inclusion and exclusion criteria on keywords, the title, and abstract.
2. The second filter is a more in-depth selection of our results from the previous filter, but instead of just using the metadata, the full text is used to apply the criteria list.

3.2 Research results

Our search returned a total of 243 results, from which two filters were applied to reach a total of 19 articles. Unfortunately, of the 19 works selected for data extraction, not all solve the entire purchase prediction problem as we have defined it. Some of them solve the first stage; others solve the second or solve a similar problem. However, this does not mean that they can not serve as a basis for our research. To better classify our results, we can define four classes of works related to our problem:

- P1** Given user history interactions, who will be a buyer or not? We will call as *Buyer classification*.
- P2** Given user history interactions, which feature of the user or the product will be found on the next click? (e.g product category, user location, user payment, product id). For this, we will call as *feature prediction*.

P3 Given user history interactions, who will be a buyer or not? Among the buyers, which items they will buy? We will define this problem as *purchase intention prediction*.

P4 Given user history interactions, which items are more attractive for the user to buy?. This problem is called *next-item recommendation* or *next-click recommendation*.

Figure 15 shows the distribution of the described problems. The **P3** is the problem defined exactly equal to our problem. There is very little research on our problem in the literature. This shortage can be a hindrance since much of the basis for developing our research does not contain mature historical research in the literature. As the graph shows, either researcher solves only the first problem or the second, not both simultaneously, just like we want to. The problem **P3** is basically the concatenation of problems **P1** and **P2**: **P1** is the first stage of the **P3** problem and **P2** is the same problem of the **P3** second stage.

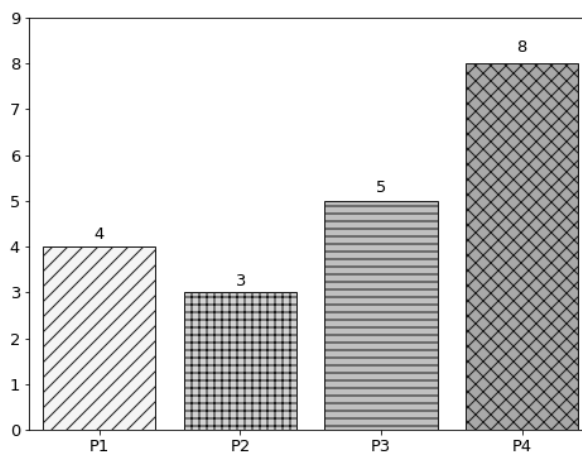


Figure 15 – Problem Class × Number of Publications. (Source: Own author)

P4 is not the same problem as **P2**, although they are similar. In **P4**, we are trying to convince new or same products for a user session. In **P2**, we will only predict the same products purchased in the past. Although different, **P4** models can be applied in similar ways in **P2** problems since recommendation tasks are also concerned with how to code e-commerce sessions for neural networks.

3.3 Research analysis

In the next section, we will briefly analyze the results for each type of problem defined previously.

3.3.1 Buyer classification (P1)

Ling et al. [Ling et al., 2019] solve **P1** as a binary problem classification using handcrafted features with a deep learning method, a fully connected neural network stacked with LSTM. The data contains different sources, and therefore, the network was designed, so that manage it. Unfortunately, the data is private, and the author only tries to predict the purchase intention in a promotion announcement.

Another work to comment on is Zhou et al, which solves **P1** proposing two attention networks [Zhou et al., 2019]. The first is a recurring network with global attention that uses a context vector to capture each activity’s attention. The second is a recurring network with logistic regression to update the attention weights in the network. Global attention outperforms the literature networks in conversion prediction problems, achieving an Area Under the Curve (AUC) of 0.739, a 7.0% lift above a Logistic Regression, and 0.9% lift above the local Attention Network from the literature in the RecSys 2015 dataset.

Sheil et al. [Sheil et al., 2018] solved the problem using stacked LSTM and GRU networks with hidden shared units, sequence reversal, event unrolling, and data augmentation. He also processed the data to input in an embedding network to represent the *items_ids*. Another fact from Sheil’s work is that the solution algorithm reached a 98% of Area Under the Curve (AUC) compared to Romov et al. [Romov et al., 2015] without any feature engineering.

3.3.2 Feature prediction (P2)

The **P2** problem class is closely related to our problem in the second stage. Liu [Liu et al., 2019] developed the DeepStore network to solve customers’ point-of-interesting problems to maximize the sales in a store based on multi-source data. The DeepStore is a stacked network that uses field embedding and attention-based spatial embedding to learn latent feature representations. Wen [Wen et al.,

[2018] works on location, time, payment amount, and product category prediction. To work on the problem, the researcher developed a probabilistic model that surpassed a Random Forest model. Wen used a private dataset from a Taiwanese bank.

Huang et al. [Huang et al., 2019] tried to predict the purchasing behavior by analyzing the final product category by customer. They operated on a private dataset from JD.com. Huang et al. developed a complex neural network: a novel multi-scale pyramid modulation network with a resolution-wise recalibration gating mechanism for fusing scale-specific pattern representations and automatically capturing the importance of each scale-view in the predictive model.

3.3.3 Purchase intention prediction (P3)

For works that solve **P3** robustly we can mention the Romov et al [Romov et al., 2015] work. Romov et al. won the RecSys Challenge in 2015 with a simple gradient boosting with categorical features, and as Sheil [Sheil et al., 2018], we believe that is current state-of-the-art but solves the problem differently as we want, using two stages algorithm. Another interesting work comes from Wu [Wu et al., 2015], which reached the seventeenth place in the competition. Although it was not among the first, it is interesting that it solved the problem using stacked Long Short-Term Memory (LSTM) Recurrent Network with fewer handcrafted features, unlike Romov et al [Romov et al., 2015].

The research of Yunghui [Chu et al., 2019] just uses a simple click rate to predict if a consumer will buy. The major work is found in the second stage when the author uses a Recurrent Neural Network with a K-nearest-neighbour to predict the next item. Myklatun [Myklatun et al., 2015] solved the RecSys 2015 Challenge in two stages through random forests and feature engineering. One of the most important features is the item probability feature that was created based on the amount of the item's presence in previous sessions.

3.3.4 Next-item recommendation (P4)

Of all the searched problem classes, **P4** contains the largest number of results. This is probably due to the great growth in deep learning in the recommendation area. Hidasi et al [Hidasi et al., 2015] were one of the pioneers in applying recurring networks to historical customer data. Hidasi et al applied a gated recurrent unit (GRU) with customized loss and session-parallel mini-batch. After this work, many improvement works were created. Tan et al [Tan et al., 2016] improved the GRU with the addition of data augmentation and retraining in more recent fractions. Wu et al [Wu et al., 2017] created a neural list-wise ranking with session information embedding to solve the next-item recommendation. In 2018, Hidasi et al [Hidasi et al., 2018] proposed a new sampling strategy and new customized loss functions to the 2015 GRU from Hidasi et al (2015) [Hidasi et al., 2015].

With the growth of new attention networks in the natural language processing area, some of these neural networks started to be deployed in recommendation systems. We can mention the work of Kang [Kang and McAuley, 2018] which created a neural network called SASRec self-attention layer with a point-wise feedforward network. Sun [Sun et al., 2019b] outperforms the SASRec network with a bidirectional self-attention network through the Cloze task called BERT4Rec.

Other interesting works occurred in 2015 on the recommendation problem. Hu [Hu and He, 2019] created a new concept of recommendation called *sets to sequential sets learning*, a generalization of *sequence to sequence learning* which treats the input and output encoding and decoding of the problem as a sequence of sets. The model is an RNN encoder-decoder that uses an element-embedding-based set embedding to represent each set and uses an attention mechanism to leverage the information in different input sets for different output sets accordingly.

Another interesting work is the Graph Neural Network for a session-based recommendation from Wu [Wu et al., 2019]. In this network, all session sequences are modeled as directed session graphs, where each session sequence can be treated as a subgraph. Then, each session graph proceeds successively, and

the latent vectors for all nodes involved in each graph can be obtained through a gated graph neural network. The graph neural network adopts the soft-attention mechanism to generate a session representation that can automatically select the most significant item transitions.

Conclusion

The systematic review showed that most of the scientific works focus on product recommendations for the users. And that researchers often don't solve the buyer and non-buyer customer classification along with product recommendation. This shows that there is a certain gap in scientific research on consumer purchase behavior modeling. Also, among the research works found for purchase prediction, none of them solved the problem in a single-stage efficiently. Another important fact observed is that most research work hardly focuses on using user, context, and product information to improve forecasting models, unlike our project which will focus only on the use of raw click-views.

The applied systematic review shows that there are several fields within consumer buying behavior predicting that can be explored. This enables the creation of new scientific interventions within the area.

4 Predicting purchase intention through single-stage siamese deep learning models

In this work, we are trying to predict consumer intent through e-commerce item clicks. In summary, we want to answer these two questions:

1. Which consumers are most likely to purchase at least one product (or item)?
2. Among those buyer (or *converted*) consumers, which items will they buy?

If this is solved using two distinct models, one specialized on each question, this approach is known as a **two-stage** approach. If we want to solve both questions at the same time, that is, using a single model, these are called **single-stage** approach. The purchase prediction problem is usually tackled in the literature by deploying two-stage models. Romov [[Romov et al., 2015](#)], for instance, deployed one model to classify who is a buyer and non-buyer and another to predict, among those sessions considered as a buyer, which items were bought. Chen [[Chen et al., 2016](#)] reported that a single-stage approach performed generally worse than deploying two-stage methods.

Although two-stage approaches have historically led to better results, they have some disadvantages in comparison with single-stage models. Since e-commerce environments may generate a lot of data continuously, training two distinct models for those tasks may require many optimization parameters and, therefore, much more human activity to operate them. Another disadvantage of two-stage models is that all prediction errors from the first model are default errors for the second model, i.e., if a first stage model wrongly decides that a session is non-buyer or buyer, the quality of the second is irrelevant, since its output is either disregarded (in the case of a wrongly classified buyer session) or wrong by default (in the case of wrongly classified non-buyer session). If the first model makes too many mistakes in its prediction, the second model may become practically useless.

The item purchase prediction problem is directly related to buyer classification. Once a single purchase of any item occurs, it follows that the session as a whole is a buyer session. Our intuition is that solving both tasks at the same time might help each other during training, and there is the possibility that the model can better understand session-wide characteristics that separate buyer and non-buyer sessions. Those characteristics might also help to identify which item was bought.

In this work, we focus on developing single-stage siamese-based networks aiming to provide a different view on how to solve this problem regarding previous solutions. Our solution addresses the problem by treating it as a single multi-class problem, classifying the output into three classes that will be presented in the 4.2 section: Si_{nn} , Si_{bn} , and Si_{bb} .

Section 4.1 defines our research problem formally. In the following section, 4.2, we present our design to model the purchase prediction problem as a single-stage classification of three classes. In section 4.3 we specify our approaches and the characteristics of our proposed models. Lastly, in section 4.4 we present the architectures used for our proposed models.

4.1 Formal problem statement

Let $c(s) = \{c_1(s), \dots, c_{n(s)}(s)\}$ be an array of items clicked during a user session s with length n in a e-commerce website such that all $s \in S$ are ordered over time.

Consider that each session s contains a set of items I_s clicked and $B_s \subseteq I_s$ denotes the set of items which have been bought in session s . Within each item click c_i assume that there is a set of records that describe it, such as click timestamp, product category of the item clicked, the item id, the location of the user, the item price, and any other features that describe the items clicked in the platform.

The following function $y(s)$ corresponds to the actual set of items B_s bought in a session s defined as follow:

$$y(s) = \begin{cases} B_s & \text{if purchase} \\ \emptyset & \text{otherwise} \end{cases} \quad (4.1)$$

Purchase problem function.

The sessions in S are divided into two sets: S_b which have at least one clicked item being bought, and S_{nb} which do not end in a transaction. We want to classify elements of S as belonging to S_b or S_{nb} . Moreover, if a session s results in a transaction (i.e. $s \in S_b$), we want to find the B_s products bought. In short, our task is to construct a model with a hypothesis $h(s)$ such that $h(s) \approx y(s)$.

4.2 The single-stage classes

An intuitive way to solve the purchase prediction problem is to solve it in two-stages, such as Romov [Romov et al., 2015] proposed. In that work, the first model classifies if a given session is a buyer or a non-buyer session. Then, the sessions classified as a buyer are used as input for the second model, which predicts, for each item clicked in the session will be purchased or not.

An issue with this approach is that there is a need to create two models for the problem. With more models, there is a greater amount of tunable parameters and therefore, it takes more time and complexity to optimize them. Furthermore, the two-stages approach filters sessions in the first stage and all wrongly filtered sessions will have no chance to be evaluated in the second stage.

In general, the literature solves the problem with two different approaches. The first consists of considering the whole session at once and thus first predicting whether the session ends in a purchase, and then predicting which item was bought among the clicked items. The second consists of creating session-item pairs, which consist of a mix of global session and item click specific features, and predicting, for each session-item pair from a buying session, if that specific item will be purchased or not.

Our approach to solving the problem in a single-stage is to consider session-item pairs as input, and then use these as input to predict three possible classes for each item in a session:

1. Si_{nn} : If the item is not in a buyer-session.
2. Si_{bn} : If the item is in a buyer-session but was not purchased.
3. Si_{bb} : If the item is in a buyer-session and was purchased.

4.3 Single-stage siamese networks approaches

In real scenarios, our research problem contains extremely unbalanced classes. For instance, in the RecSys dataset [Recsys, 2015], 89% of all inputs are of the Si_{nn} class, class Si_{bn} is about 5.5%, and class Si_{bb} is approximately 4.5%. This imbalance makes the problem very difficult to solve as unprepared models may show tendencies to predict class Si_{nn} much more frequently. Siamese networks can help solve this problem by comparing the fixed number of instances from different classes based on their similarity, in which the network itself generates an encoding (or an embedding) for the input, having the loss function based on comparing these encodings. Since every triplet (or quartet, as will be shown shortly) has a similar number of instances from each class in this scenario, the imbalance in the original dataset should not affect the training of the method.

The Figure 16 shows an overview of the classification process of our single-stage models. In it, we can observe three phases: Feature extraction (1), Embedding generation (2), and classification (3). We will detail it in the next paragraphs.

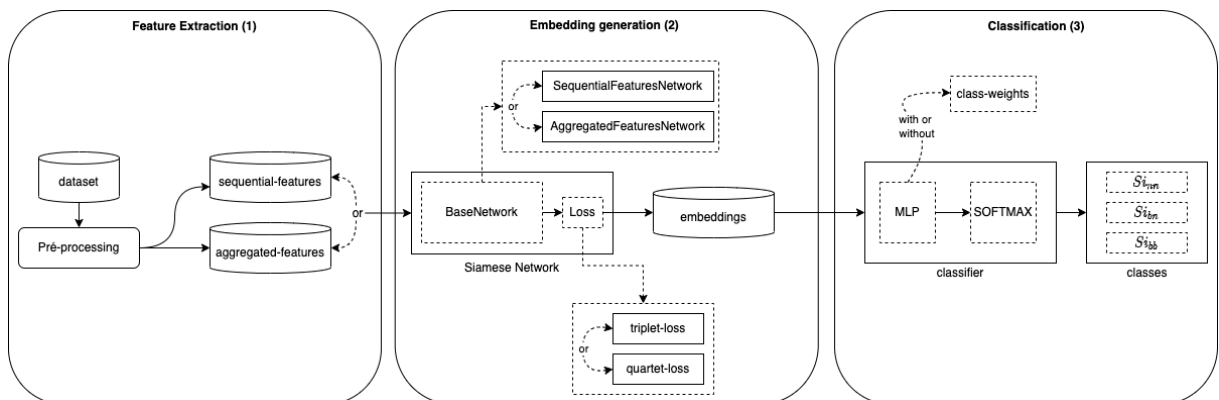


Figure 16 – Overview of the classification process for the single-stage model. (Source: Own author)

- **Feature extraction (1):** In this first phase, we perform the extraction of the two possible sets of session and item features we propose to use as input for

our models in this dissertation, sequential-features and aggregated-features. These two sets of features are discussed further in the [4.3.1](#) section.

- **Embedding generation (2):** In this second phase, the generation of embeddings by the Siamese networks takes place. First, an feature set is chosen as the input of the Siamese network, either sequential-features or aggregated-features. Given the type of input, we must choose the base of the Siamese network (BaseNetwork) adapted to the type of input. If we choose sequential-features we use SequentialFeaturesNetwork, if we choose aggregated-features we use AggregatedFeaturesNetwork. These base networks are detailed in section [4.4](#).

Next, we define the type of loss function to be used by the Siamese network, which can be either triplet-loss or quartet-loss. The input type in this case does not define the type of loss to use. Having defined the Siamese network architecture, we train the model to generate and store the representation (or embeddings) of our training session-items.

- **Classification (3):** In this last phase, the session-items embeddings are loaded into a multi-layer perceptron (MLP) model to be trained and predict the final classes Si_{nn} , Si_{bn} and Si_{bb} . The MLP in this case can have a variation with class-weights in its cost function to deal with the imbalance. The definitions of the number of neurons and layers, as well as other details, are described in the section [5.2](#).

In the next subsections, we will detail each phase of the single-stage model classification. In the [4.3.1](#) section we describe the input types and how the features were implemented. In section [4.3.2](#) we present how the two losses, quartet-loss, and triplet-loss, work. Finally, in the [4.4](#) section we show the two variations of BaseNetworks, SequentialFeaturesNetwork and AggregatedFeaturesNetwork implemented for the Siamese network.

4.3.1 Input features

We propose two alternatives for modeling the input sessions, sequential-features and aggregated-features. The next paragraphs describe those alterna-

tives:

4.3.1.1 sequential-features inputs

These features describe a session as simply a sequence of clicks as well as features from the considered item. They are comprised of ten features: target_item_id, item_id, category, day, hour, month, weekday, time between clicks (dwelltime), item_purc_rank, item_clk_rank, item_prob and item_session_prob. The first feature, target_item_id, represents which item_id the model should consider in relation to the whole session. All other features are relative to all clicks in the session.

target_item_id	item_id	category	item_purc_rank	item_prob	item_session_prob	item_clk_rank	weekday	day	hour	month	weekday	dwelltime
214536500	214536502	337	17.0	0.00	0.00	1625.0	1	8	11	5	16	0
	214536500	337	0	0.00	0.00	502.0	1	8	11	5	16	128
	214536506	337	3.0	0.04	0.16	62.0	1	8	11	5	16	64
	214577561	337	0	0.00	0.00	132.0	1	8	11	5	16	12
	214662742	337	13.0	0.00	0.00	3539.0	1	8	11	5	16	246

Table 4 – Example of a session described sequentially by sequential-features for section 4.3.1.1. We developed all features except for item_prob.

The features item_id and category represents which item and category are clicked over time. The features day, hour, month and weekday of the click represent temporal trends for the raw-features, and the dwelltime is the difference in seconds between two consecutives clicks.

The feature item_purc_rank represents the rank of items of all most purchased items in respect of quantity. To develop this feature we calculated the total amount that each item was purchased in the entire training database history. Each item is labeled with a value that determines the ranking where the higher this rank value is more often it is purchased.

The feature item_clk_rank implements the rank of an item concerning all most clicked items. To create it, we calculated the total number of times each item was clicked in the entire training database history. The ranking from this feature is created similarly to item_purc_rank. Each item is labeled with a value that determines the ranking where the higher this rank value more often it is clicked in the dataset.

The feature item_prob implements a feature created by Myklatun [Myklatun et al., 2015], being the probability that an item will be purchased in the

sessions that the item is clicked. That is, it is the total sessions in which the item was purchased divided by the total sessions in which the item was clicked.

In addition, we developed a similar feature called `item_session_prob`, which describes the probability of the item appearing in buyer-sessions. To calculate this feature, we must have the total number of times a certain item is in a buyer-session divided by the sessions in which it was clicked. The Table 4 shows an example of the mentioned raw features from a user session.

4.3.1.2 aggregated-features inputs

5 These features regard information from the whole session as well as from a particular item are the same features from the second-stage developed by Romov [Romov et al., 2015]. Romov’s method does not handle input sequentially but tries to compile all session information at once. That is, all features are calculated looking at the entire session, such as the total time of a certain item in the session. Romov features are divided into two subgroups session-features (SF) and session-item-features (SIF).

- **session-features:** The features describe the session as a whole, looking at all items and categories at once. From these features we can list:
 - Numerical time features for the timestamp of the first and last click in the session. That is, the hour, month, minute, weekday, week of the year, etc.
 - Categorical time features for the timestamp the first and last click in the session. For example, the concatenated string of the month, year, and day the click occurred.
 - Length of the session in seconds.
 - Number of clicks, unique items, categories and item-category pairs in the session.
 - Top 10 items ids and top 5 categories by the number of clicks in the session.
 - Id of the first and last item clicked at least $k = 1, 2, \dots, 6$ times in the session.

Group	Feature description	Number/Type
session-features	Numerical time features of the start/end of the session (month, day, hour, minute, second, etc.)	2×7 Num
	Categorical time features of the start/end of the session (month, day, month-day, month-day-hour, hour, minute, weekday)	2×6 Categ
	Length of the session in seconds	1 Num
	Number of clicks, unique items, categories and item-category pairs in the session	4 Num
	Top 10 items and top 5 categories by the number of clicks in the session	15 Categ
	IDs of the first/last item clicked at least $k = 1, 2, \dots, 6$ times in the session	12 Categ
	Vector of click numbers and total durations for 100 items and 50 categories that were the most popular in the whole training set	150×2 Num
session-item-features	Item ID	1 Categ
	Total and relative number of clicks in the session for the given item	2 Num
	Numerical time features of the first/last click on the item (month, day, hour, minute, second, etc.)	2×7 Num
	Categorical time features of the first/last click on the item (month, day, month-day, month-day-hour, hour, minute, weekday)	2×6 Categ
	Number of seconds between the first and the last click on the item	1 Num
	Total duration of the clicks on the item in the session and of all item's categories seen in the session	2 Num
	Number of unique categories seen in the session for a given item	1 Num

Table 5 – Features engineered for the model for the section .

- Vector of click numbers and total duration for 100 items and 50 categories that were the most popular in the whole training set.
- **session-item-features:** The session-item-features describe more specific characteristics of the item as a target. Given that, we can list:
 - The item_id (this is the same from target_item_id in sequential-features).
 - Total and relative number of clicks in the session for the given item.
 - Numerical time features for the timestamp in start and click on the item. That is, the hour, month, minute, weekday, week of the year, etc from the timestamp in the given item.
 - Categorical time features for the timestamp in start and click on the item. For example, the concatenated string of the month, year, and day the click on the given item.
 - Number of seconds between the first and the last click on the item.
 - Total duration of the clicks on the item in the session and of all item's categories pair seen in the session.
 - Number of unique categories seen in the session for a given item.

The Table 5 summarizes session-features and session-item-features.

4.3.2 Loss function

Our solutions may also deploy different loss functions, being either **triplet-loss** or **quartet-loss**, a new loss function proposed in this dissertation. The triplet-loss was discussed on Chapter 2. It's a common loss function for siamese networks developed by Schroff et al. [Schroff et al., 2015b]. The triplet-loss can be easily adapted for our problem, by setting an anchor and positive input as Si_{bb} and negative input with the buyer classes as either Si_{bn} and Si_{nn} .

The quartet-loss is a variation of triplet-loss proposed in this dissertation. Instead of only one negative example, we use two negatives. The quartet-loss receives quartets inputs containing anchor, positive, the first-negative, and second-negative. For our problem, the positive and anchor are two instances of Si_{bb} , the first-negative is an instance of Si_{bn} and the second-negative is an instance of Si_{nn} . Our preliminary experiments showed that this configuration for these classes achieves better results. Using quartets the loss minimizes the distance between the anchor and positive inputs whereas maximizing the distance between anchor and first-negative and anchor to the second-negative.

We believe that it might be interesting for the model to differentiate the negative classes Si_{bn} and Si_{nn} . Explicitly separating items of a *non-buying session* from items of a *buying session* that were not purchased might help the model identify specific characteristics of buying sessions and help it to differentiate specific items when looking for item purchases. With this, we hope that the quartet-loss separates the four instances in the latent space at once, so that it is easier to identify each class.

The quartet-loss can be formulated as follows:

$$(I_a, I_p, I_n^1, I_n^2) = \max(\alpha \cdot \text{dist}(I_a, I_p) - \beta \cdot \text{dist}(I_a, I_n^1) - \gamma \cdot \text{dist}(I_a, I_n^2) + \text{margin}, 0) \quad (4.2)$$

Quartet-loss formula.

where the I_a , I_p , I_n^1 and I_n^2 represents anchor, positive, first-negative and second-negative examples respectively. The $\text{dist}(x, y)$ and margin are the same function and parameter defined in triplet-loss in Fundamentals section 2.7.1 to

select semi-hard samples. The α , β and γ are weights created to better fine-tune how the model perceives each class.

In some tasks, the negative classes may not have the same semantic value and, therefore, may lead a siamese network encoding to focus on aspects of less relevant classes. For this reason, the quartet-loss compresses the input from one more negative class that will be used to differentiate one from another.

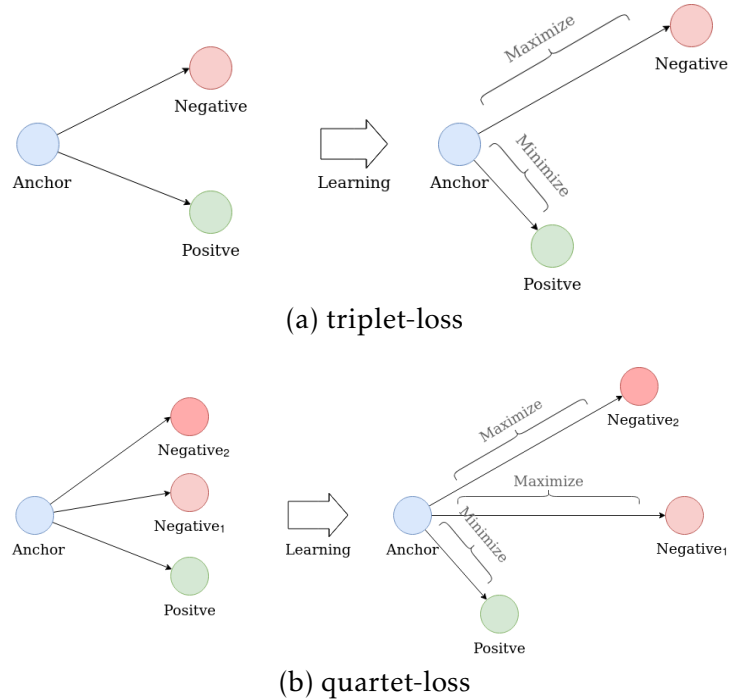


Figure 17 – Triplet-loss vs Quartet-loss. In triplet-loss we are separating only one negative while in quartet-loss we are separating two negatives. (Source: Adapted from [Schroff et al. \[2015a\]](#))

Depending on the problem, it might be more important to differentiate some classes from others. Thus, it is also possible to use the weights α , β , and γ to find the best level of separation between the positive, the first-negative, and second-negative. In our experiments in section 5.3 we present the configuration used for these weights. Figure 17 shows a representation of the difference between triplet-loss and quartets-loss. We can observe in the Figure how the loss works for maximizing or minimizing the inputs.

4.3.3 Classification

After the feature extraction process, a straightforward model should easily separate the three classes, since the Siamese network separates the features

concerning the classes in the latent space. Our experiments showed that the best classifier was an MLP neural network. The chosen configuration has 3 dense layers of 64 neurons and BatchNormalization to regularize the weights. To optimize the weights we use the categorical cross-entropy loss in training.

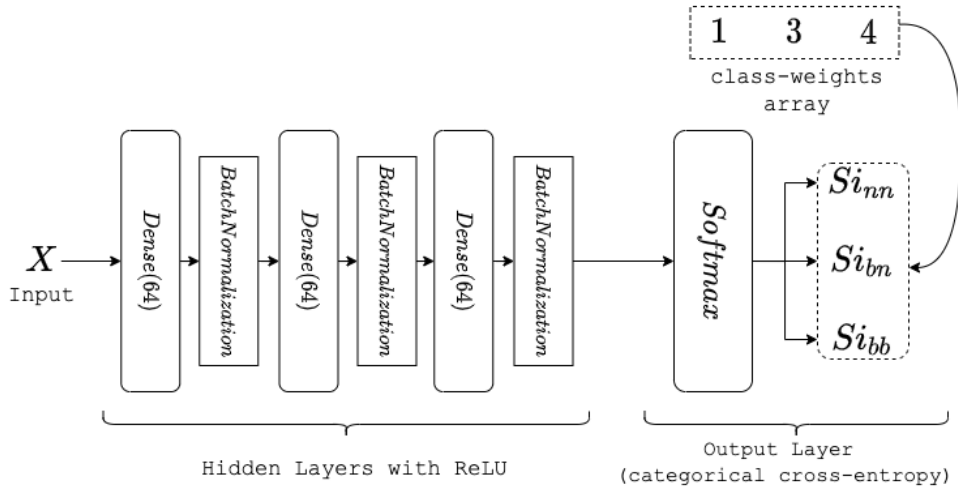


Figure 18 – MLP classifier architecture. (Source: Own author)

In the beginning of the section 4.3 we commented that the MLP has a variation on the class-weights technique for handling class imbalance. The class-weights is applied in the categorical cross-entropy loss. For the class-weights, we found a better configuration in the following parameters for each class: 1.0 for $S_{i_{nn}}$, 3.03 for $S_{i_{bn}}$ and 4.04 for $S_{i_{bb}}$. The Figure 18 shows the architecture of our MLP classifier.

4.4 Base network architectures

In section 4.3 we discussed our approach and phases to develop the classification for our problem using the Base Network. In this section, we will show the architectures used to create the mentioned Base Network.

Based on the two different representations of inputs (shown in section 4.3.1), we developed two Base Network architectures: SequentialFeaturesNetwork and AggregatedFeaturesNetwork. The base model SequentialFeaturesNetwork comprises an architecture made for sequential inputs. The base model SequentialFeaturesNetwork, along with our two possible loss functions, generate two variations of our proposed models TCNQuartet, TCNTriplet. The other archi-

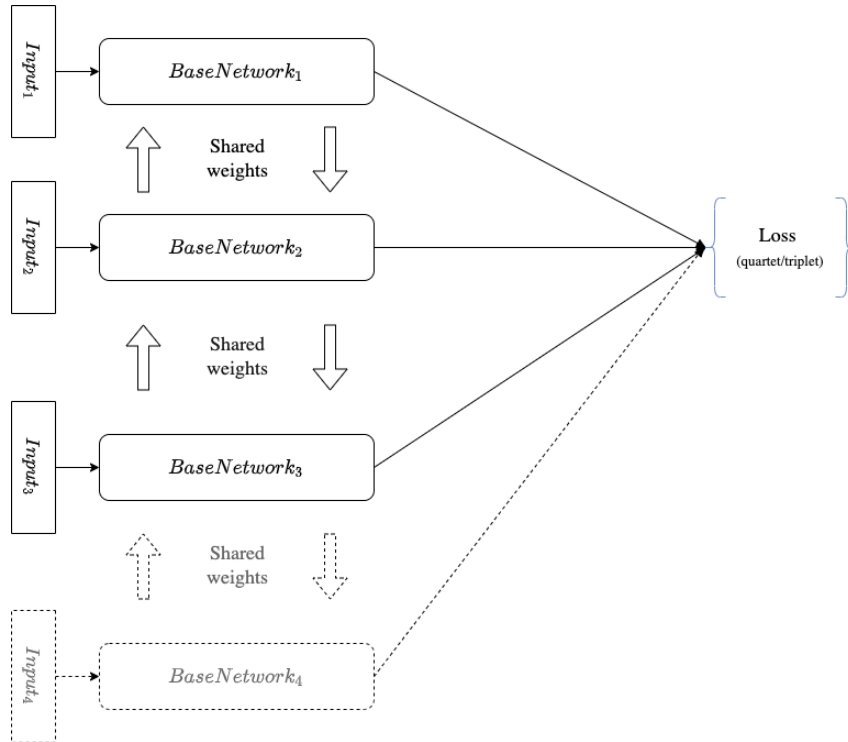


Figure 19 – Siamese network general architecture. BaseNetwork generates the latent representation for each input, and loss is calculated to update weights. The grey drawing represents the fourth BaseNetwork and Input for quartet-loss. (Source: Own author)

tectural base model AggregatedFeaturesNetwork integrates aggregated inputs aggregated-features. From there we have two more variations of our proposals: AGQuartet and AGTriplet.

As mentioned in the previous section, our Siamese network models can contain triplets and quartets. Figure 19 shows a general siamese network architecture where models with triplet-loss have three BaseNetworks and models with quartet-loss have four BaseNetworks. Next sections we present SequentialFeaturesNetwork and AggregatedFeaturesNetwork BaseNetworks.

4.4.1 SequentialFeaturesNetwork

For the Siamese network for sequential-features we developed a BaseNetwork called SequentialFeaturesNetwork. This network receives sequential-features inputs described in section 4.3.1. These features receive a concatenation of $F_{m,n}$ and a $target_item_id$. The $F_{m,n}$ is the features that represent clicks over time (item_id, category, day, hour, month, weekday, dwelltime, item_purc_rank,

item_clk_rank, item_prob and item_session_prob). The *target_item_id* is the item_id that indicates the current item the model is observing. The input $F_{m,n}$ is passed to a layer called FeaturesEncodingLayer composed of embeddings for categorical inputs and dense networks for numeric inputs. Table 6 describes which neural network is used to encode each feature and the total of the numbers of neurons used. After the FeaturesEncodingLayer, the output is passed to a TCN (Temporal Convolution Network [Oord et al., 2016]) to process the representation as a sequential input with 64 neurons. To process the input sequentially, several models were tested such as Transformers, GRU, LSTM, Graph-Neural-Networks, but TCN was the one that represented the best results. We suppress the results of other variations for the sake of brevity.

Feature	Encoding network	Number of neurons
item_id	embedding	100
category	embedding	10
item_purc_rank	dense	5
item_prob	dense	5
item_session_prob	embedding	5
item_clk_rank	dense	5
weekday	embedding	5
day	embedding	10
hour	embedding	5
month	embedding	5
weekday	embedding	5
target_item_id	embedding	100

Table 6 – FeaturesEncodingLayer details. The FeaturesEncodingLayer is the concatenation of multiples encodings networks.

After the TCN layer, we pass its output to a Bahdanau attention [Bahdanau et al., 2014] layer to improve our network encoding. In addition, we pass the output of attention to a dense network with 64 neurons conjugated with a batch normalization layer. These two layers can be stacked N times. Finally, we have an OutputLayer that can be a dense network with size $EMB_SIZE = 64$ or a dense network with a softmax (or sigmoid) activation to predict classes. Likewise, the *target_item_id* input is encoded by an embedding network and concatenated with the final layers of the network. The Figure 20 summarizes our description.

The OutputLayer representation has two possible output options because

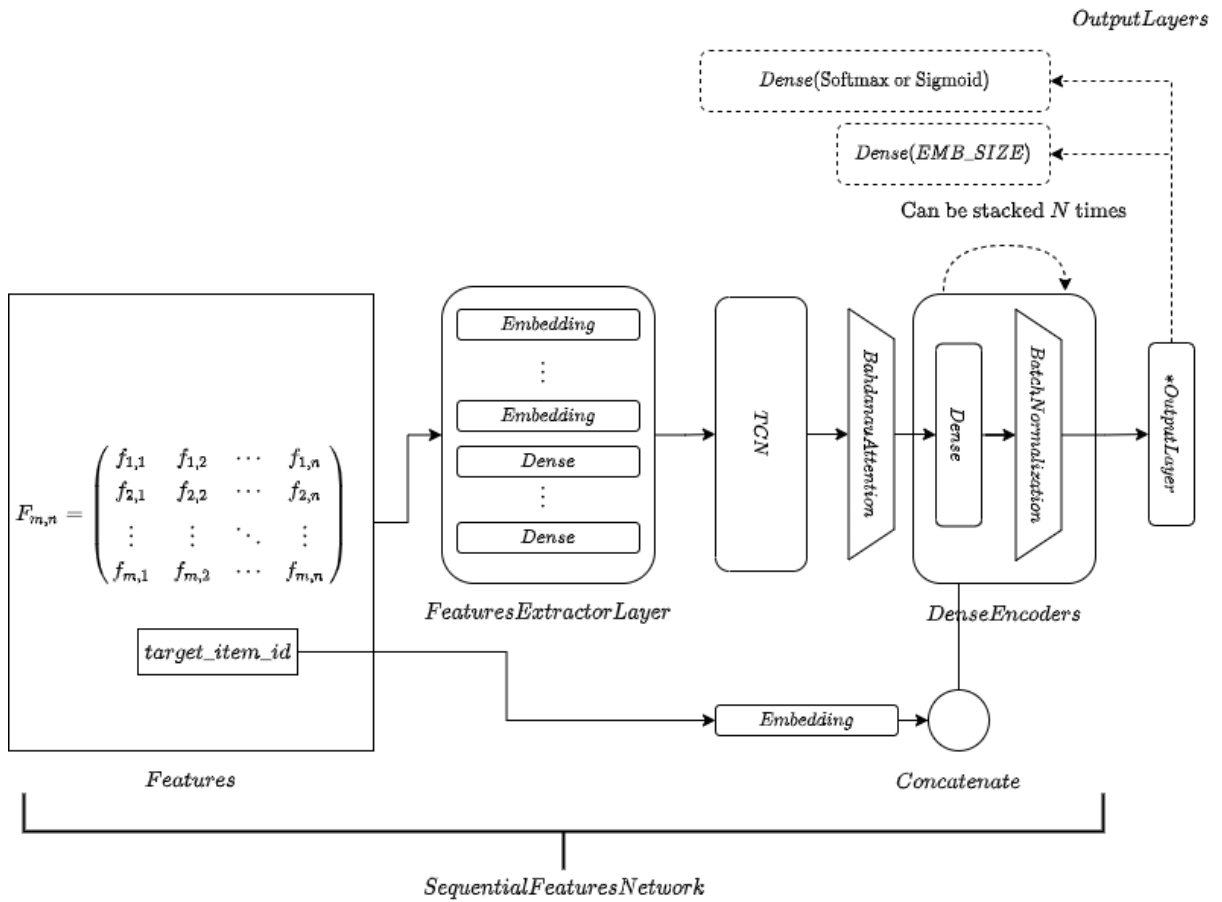


Figure 20 – SequentialFeaturesNetwork architecture. (Source: Own author)

we will also make use of this base architecture to implement some of our baselines, as is shown in Chapter 5. For the siamese implementation, the Outputlayer is a Dense network with output sized `EMB_SIZE` to generate latent representations with 64 output size, and for the baselines, the OutputLayer is a dense network followed either by a softmax or sigmoid activation. The baselines in single-stage have the OutputLayer with softmax activation and the baselines in two-stage have sigmoid activation.

In short, our models that use sequential-features have two loss variations, triplet or quartet (e.g TCNQuartet and TCNTriplet). These models use the SequentialFeaturesNetwork architecture to generate the latent representation of a session and a target item in the siamese network. These models have $N = 2$ DenseEncoders to generate the latent representation. In the Figure 19 we can observe that SequentialFeaturesNetwork is an implementation of the BaseNetwork.

4.4.2 AggregatedFeaturesNetwork

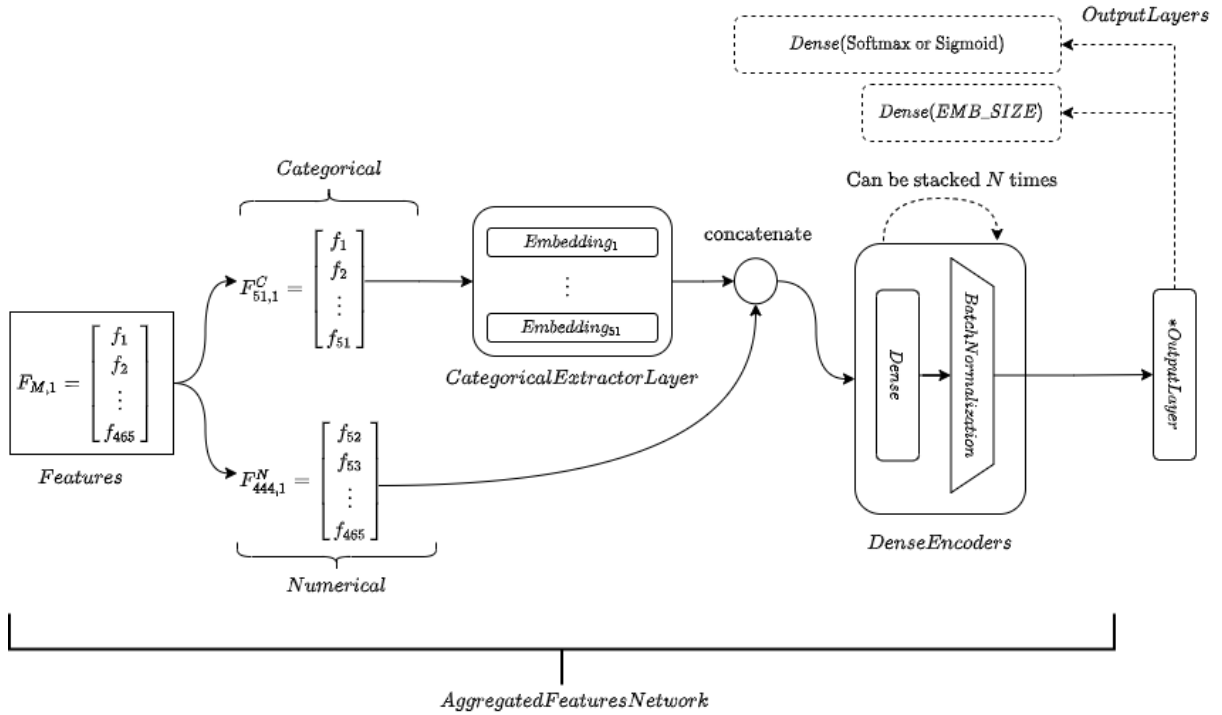


Figure 21 – AggregatedFeaturesNetwork architecture. (Source: Own author)

The siamese network for aggregated-features works similarly to the sequential-features siamese, however our input here is a single vector with 495 features, 51 of which are categorical. The aggregated-features are detailed in section 4.3.1.

The architecture splits the input into two vectors, one consisting of categorical features $F_{51 \times 1}^C$ and the other of numerical features $F_{444 \times 1}^N$. Each feature of the vector $F_{51,1}^C$ is passed to a set of embeddings called CategoricalExtractorLayer, as shown in the Figure 21. In this way, the output of the CategoricalExtractorLayer is concatenated with the vector of numerical features $F_{444 \times 1}^N$.

To set the output size of the embeddings layers in CategoricalExtractorLayer we use the following logic: if the categorical input contains more than 100 unique values, the output will be 50 otherwise the size of embedding output will be 10.

Thereafter, the concatenated output is passed through a dense network layer with 128 neurons and batch normalization to regularize the model. These two layers can be stacked N times. After this, the DenseEncoders are connected to the OutputLayer. In the same way as the SequentialFeaturesNetwork the Out-

putLayer contains two output options to both to implement the baselines (that can be a dense network with softmax or sigmoid activation) and to implement the siamese to generate latent representations with 64 output size (the Dense with $EMB_SIZE = 64$).

For aggregated-features we developed two variations of siamese models (e.g AGQuartet and AGTriplet) one with triplet loss and another with quartet-loss. These models have $N = 2$ DenseEncoders to generate the latent representation to a classifier model.

5 Methodology and Experiments

This Chapter presents the evaluation methodology and results of our proposed models compared with a previously proposed approach in the literature.

5.1 Dataset

In our experiments, we used the Recsys Challenge [[Recsys, 2015](#)] competition dataset from 2015. The company YOOCHOOSE provided a collection of consumer data that contains around six months of click sessions from a retail store in Europe. Each click session may or may not contain purchase clicks. To protect users' privacy and the retailer, the data was modified to conceal the identity of users.

The dataset is divided into three files in csv format: `clicks.dat`, `buys.dat`, and `test.dat`. The `clicks.dat` and `buys.dat` files contain data regarding the same user sessions, the first containing only views and the second with the purchased items in each session. The `test.dat` is similar to `clicks.dat`, with just session clicks and no buying information.

The `clicks.dat` file has information on 33 million view clicks and 1,15 million lines of purchase. This consists of a total of 9,249,729 sessions, of which 5.8% (509,696) of the sessions have at least one purchased item (referred to as a buyer-session). In `clicks.dat` file, only about 28% of the items present were purchased at least once. The test file follows a similar distribution as `clicks.dat`, containing a total of 2,184,173 non-buyer sessions and 128,259 buyer sessions, e.g 5.8% of the total. [Figure 22](#) demonstrates the session types distributions (buyer and non-buyer session) and items purchasing distribution.

In the previous Chapter, we defined the classes to be used with our proposed single-stage models. The major class is the Si_{nn} which represents non-buyer paired sessions-items, which represents 89% of the instances. The class Si_{bn} represents 5.5% of session-items and finally the class Si_{bb} has the lowest frequency with just 4.5% of all instances of paired session-items.

Although session sizes range from 1 to 200, most of them are small. Ses-

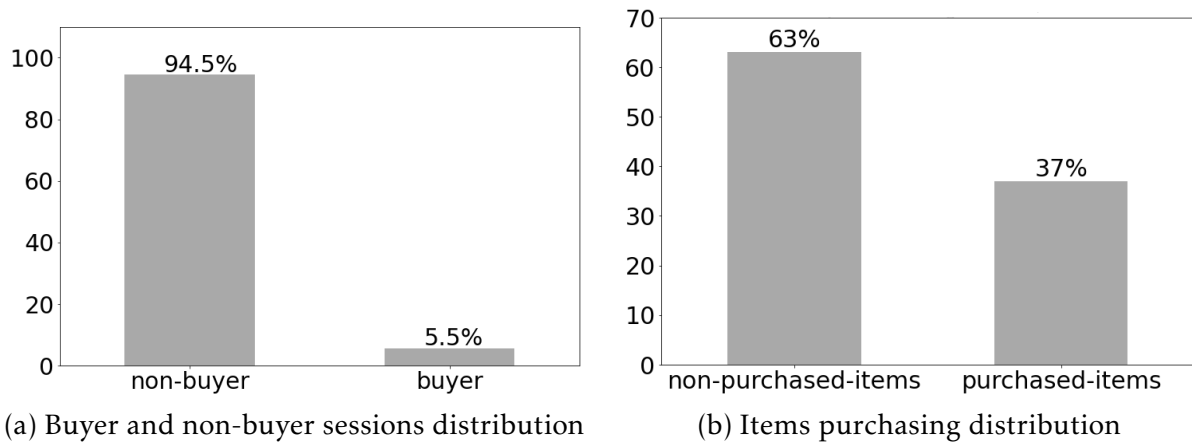


Figure 22 – Session types distribution and Items purchasing distribution. (Source: Own author)

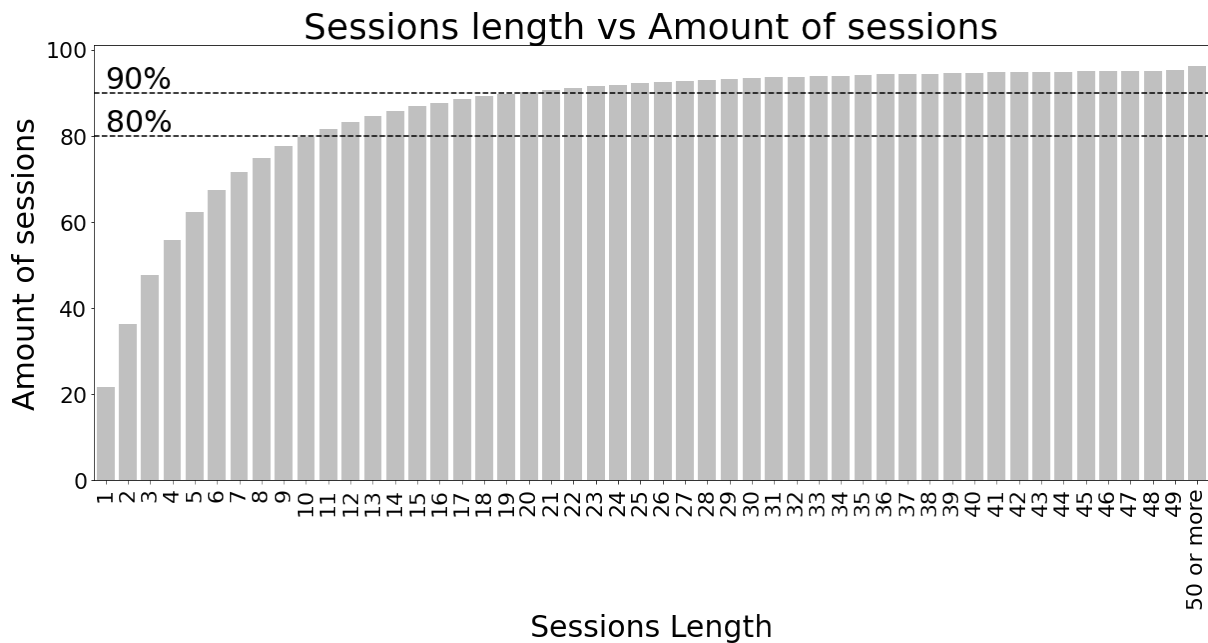


Figure 23 – Cumulative session’s length. (Source: Own author)

sions up to size 10 are 80% of sessions, and up to 20 are 90%. Figure 23 shows a cumulative sessions length percentage. An interesting feature in the dataset is that the larger the session size, the greater the purchase probability, that is, the buying rate. The Figure 24 shows this behavior.

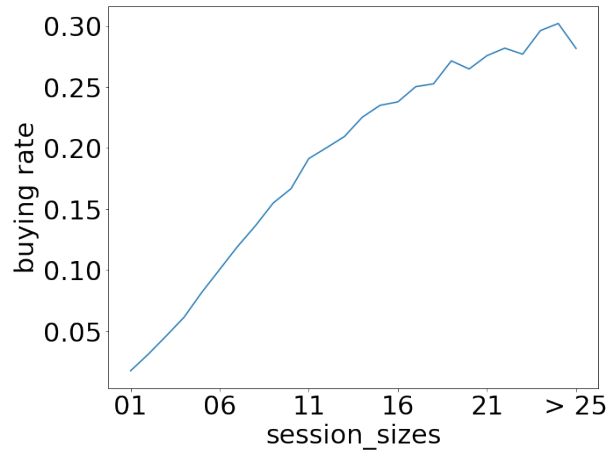


Figure 24 – Session sizes vs buying rate. (Source: Own author)

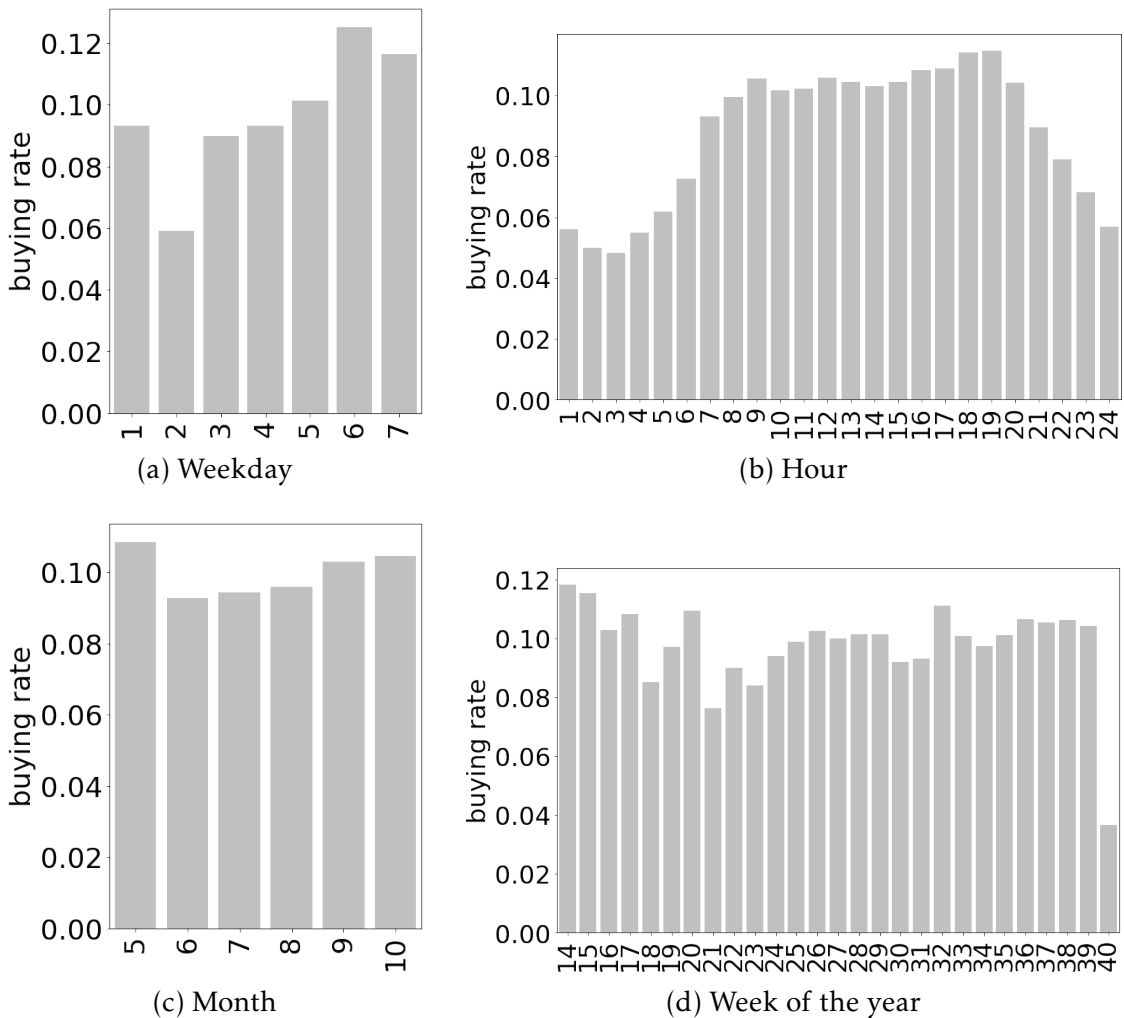


Figure 25 – Time features vs Buying rates. Weekdays are the days of the week, such as Monday, Tuesday, Wednesday. Weeks of the year refers to each of the 52 weeks in each year. (Source: Own author)

Temporal features are features that represent characteristics of time (e.g

month of clicked item). They can describe relevant information regarding purchasing behavior. For example, on days when a consumer receives their income they might be more likely to make purchases. Another example may be that at the weekends there may also be a greater probability of purchase than on working days since the buyer will have more time to fulfill their wishes. Specific months might have a larger volume of purchases than others, due to commemorative dates such as Christmas and Valentine’s day. In our analysis at Figure 25, we can see this trend.

Another interesting temporal feature is the time between clicks within a session, or *dwelltime*. Dwelltime can tell you how important a product can be depending on how long a user has been watching it. In our analysis in Figure 26, we have grouped dwelltime into a window of 200 seconds (or 3.33 minutes) to show the purchasing rate versus dwelltime.

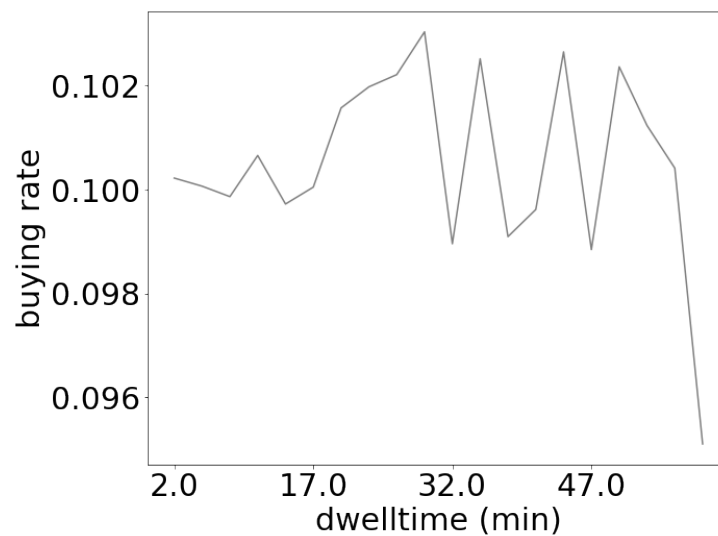


Figure 26 – dwelltime (min) vs buying rate. (Source: Own author)

Another important feature to analyze is whether the number of items purchased influences the purchase rate. Our intuition is that items that are frequently bought are also more likely to be purchased in future sessions than items that are rarely bought. From this idea, we can derive two features. A feature that is based on the number of times an item was purchased, called *item_purc_rank* and another on the number of times an item was clicked, called *item_clk_rank*. We can create a relationship between these two features concerning the buying rate. Figure 27 and Figure 28 show binned values from the features vs buying

rate.

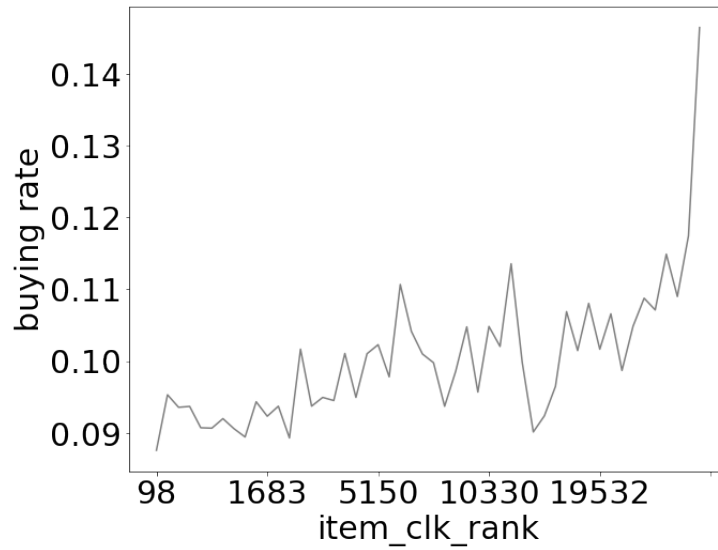


Figure 27 – Feature item_clk_rank vs buying rate. (Source: Own author)

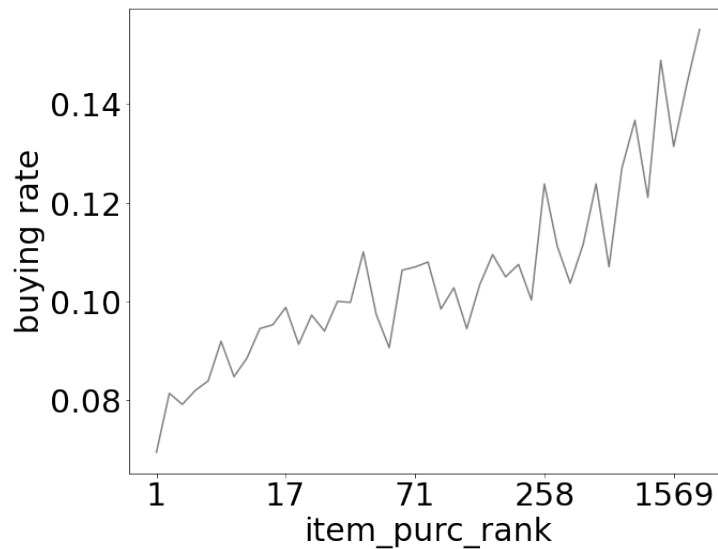


Figure 28 – Feature item_purc_rank vs buying rate. (Source: Own author)

The category of a product has a great influence on purchase intention. Some products might have a much higher demand than others. Figure 29 shows the buying rate for the top 20 categories out of the 338 existing categories. From the figure, you can observe the distribution seems to be very imbalanced and seems a long tail distribution.

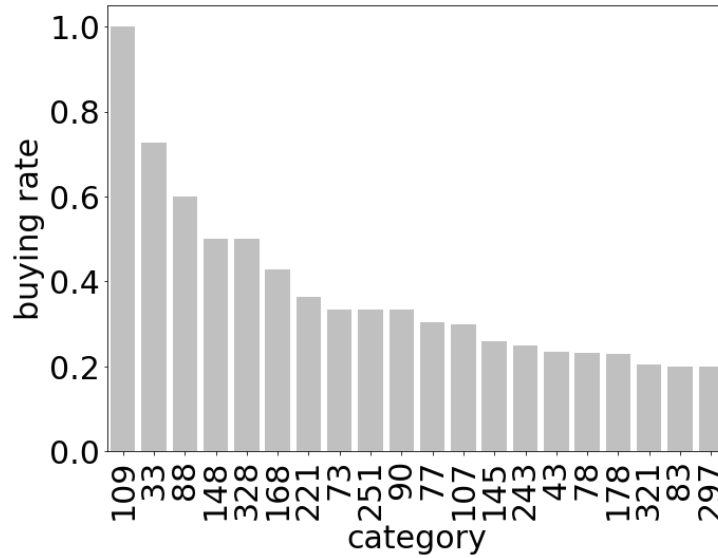


Figure 29 – Categories vs Buying rate. (Source: Own author)

5.2 Methodology

To evaluate the impact of our proposed single-stage methods in identifying buyer sessions and items, we compare them to some baselines. The baselines are described in the next paragraphs. We use 90% of the YOOCHOOSE dataset as a training set, 10% as validation, and the test.dat file as a test set.

The training of all methods occurred until the *overfitting* achievement and best weights were saved regarding validation holdout. We halt the training after the validation stops improving for about 50 epochs.

Romov presents in his paper and his code in GitHub [Romov, 2015] all the features proposed, the classifier, a gradient boosting method, and an input encoder called MatrixNet. Considering MatrixNet code was, unfortunately, not available, we implemented an adaptation of his method, called RomovNN-T, using a deep densely connected neural network to simulate the encoding of the features. We train the two-stage models independently, considering both purchase and non-purchase training sessions in the first stage and then considering only purchase sessions in the second stage.

To validate our experiments we implemented four baseline approaches, two using sequential-features and two of them using aggregated-features. The models RomovNN-T and RomovNN-S use aggregated-features as input. We

believe that RomovNN-T is the current state-of-the-art for finding the purchase problem and we performed its implementation to compare it with our proposed method. The models TCNBaseline-S and TCNBaseline-T were developed based on our sequential-features and they were created to observe how well they perform against our methods using the same features.

To identify the difficulties of implementing a single-stage versus a two-stage model, we implemented the version of RomovNN-T as a single-stage approach, which we called RomovNN-S. As we also want to know the impact of our features and we want to investigate the difficulties of implementing single-stage versus two-stage, we implement the mentioned models: TCNBaseline-S and TCNBaseline-T. The single-stage baseline models can highlight the efficiency of treating the unbalanced class concerning our proposed Siamese network models.

Next paragraphs describe our baselines:

RomovNN-T: This model is a simple neural network using two-stage algorithms to solve the purchasing intention problem. This implementation uses all features from [Romov et al., 2015] and runs on top of a deep neural network. As our proposed methods, this baseline makes use of AggregatedFeaturesNetwork as a base feature extractor, for both the first and second stage models. This baseline has the same parameters of AggregatedFeaturesNetwork, but, with $N = 2$ layers for the first-stage and $N = 1$ for the second-stage. After the AggregatedFeaturesNetwork both models have the OutputLayer layer with one neuron with sigmoid activation. In the first stage, the model uses *session-features* (SF) and for the second stage the second model uses *session-item-features* (SIF). Both are described in Table 5. To deal with the imbalance problem, the first stage model uses a focal loss [Lin et al., 2017] with Adam optimizer to train the model and penalize the negative class. In the second stage, we use a binary cross-entropy (BCE) loss with the Adam optimizer to train the model.

RomovNN-S: A single-stage neural network developed by us with Romov features based on AggregatedFeaturesNetwork, with same parameters, but with $N = 1$ number of layers, and receives aggregated-features as input. This implementation is quite similar to RomovNN-T second stage but instead of having the OutputLayer with one neuron, the model has the softmax activation and three

neurons to output class probabilities to represent $S_{i_{nn}}$, $S_{i_{bn}}$ and $S_{i_{bb}}$. To deal with the imbalance problem we applied a random under-sampling (RUS) method in the majority class $S_{i_{nn}}$. We decided not to present the focal-loss based results since undersampling had better results than focal-loss for this model. Also, we use sparse categorical cross-entropy (SCCE) with Adam optimizer to train the model.

TCNBaseline-T: A two-stage neural network proposed by us that uses sequential-features as input and has the SequentialFeaturesNetwork as base feature extractor, with same parameters although with $N = 2$. For both stages, after the SequentialFeaturesNetwork, the baseline has the OutputLayer with a single sigmoid activation. To deal with the imbalance problem in the first stage we applied a focal loss [Lin et al., 2017] with Adam optimizer to train the model. In the second-stage, the binary cross-entropy (BCE) loss with the Adam optimizer is used for training. The model solves the problem in the same way as RomovNN-T, first finding purchasing sessions and after finding items that were bought in that sessions.

TCNBaseline-S: A single-stage network proposed by us that uses sequential-features as input, with SequentialFeaturesNetwork as a base extractor, with same parameters but with $N = 2$ dense encoder layers. After the SequentialFeaturesNetwork, the baseline has the OutputLayer with softmax and three neurons to represent probabilities class of $S_{i_{nn}}$, $S_{i_{bn}}$ and $S_{i_{bb}}$. To train model we used a sparse categorical cross-entropy (SCCE) loss with Adam optimizer. To deal with the imbalance problem in we used a random under-sampling (RUS) to balance classes in quantity. The Table 7 shows more specific characteristics described in these last paragraphs for each baseline presented.

Model	Type	Input	BaseNetwork	N	Loss	OutputLayer	Imb. treat
RomovNN-T	two-stage	1st: session-features 2nd: session-item-features	SIFN	1st: $N = 2$ 2nd: $N = 1$	1st: Focal-loss 2nd: BCE	Both: Sigmoid	1st: Focal-loss 2nd: None
TCNBaseline-T	two-stage	sequential-features	SFN	Both: $N = 2$	1st: Focal-loss 2nd: BCE	Both: Sigmoid	1st: Focal-loss 2nd: None
RomovNN-S	single-stage	aggregated-features	SIFN	$N = 1$	SCCE	Softmax	RUS
TCNBaseline-S	single-stage	sequential-features	SFN	$N = 2$	SCCE	Softmax	RUS

Table 7 – Baselines configurations. SCCE: sparse-categorical-cross-entropy, BCE: binary-cross-entropy. RUS, random under-sampling. 1st: first-model, 2nd: second-model, Both: for both first and second stages. SIFN: AggregatedFeaturesNetwork, SFN: SequentialFeaturesNetwork.

As mentioned in the last Chapter, we want to separate the three classes Si_{nn} , Si_{bn} and Si_{bb} . Also, we know Si_{nn} is much more voluminous compared to others and can lead to biases in model training. Thus, we proposed models based on siamese networks to bypass this impact, as shown in Chapter 4. As we defined earlier, we have two types of models: Siamese-Triplet and Siamese-Quartet.

The triplet-loss is composed of three components: anchor I_a , positive I_p , and negative I_n . In this particular case, the anchor is represented by any pair session-item Si_{bb} , and the positive, by some other example of another Si_{bb} .

For the creation of triplets, we consider:

- As an *anchor* I_a any random item that was purchased in a session, Si_{bb} .
- As an *positive element*, also I_p any random item that was purchased in a session, Si_{bb} .
- As a negative element, two inputs are possible: one is an item that is not belong to a buyer session mentioned as Si_{nn} and another is an item that is belong to a buyer session but was not purchased within it, the Si_{bn} .

Our models that use quartet-loss solve the problem in a similar way to triplet-loss. As mentioned in the previous Chapter, we have the addition of one more negative, therefore, obtaining four elements: I_a , I_p , I_n^1 and I_n^2 . The explanation of how the quartet-loss works is described in Chapter 4.

The quartet-loss has the advantage that it doesn't treat the two classes Si_{bn} and Si_{nn} as a single negative element, instead of treating them as two distinct classes. Differently from Siamese-Triplet, our Siamese-Quartet proposals contain two negatives to avoid that alternating between two classes. Consequently, we determined the following settings:

- As an *anchor* I_a any random item that was purchased in a session, Si_{bb}
- As an *positive element*, also I_p any random item that was purchased in a session, Si_{bb} .
- As first-negative element I_n^1 any random item which was not is from a buyer session, or Si_{nn} .

- As second-negative element I_n^2 any random item that is from a buyer session but was not purchased within it, or Si_{bn} .

To train the siamese network models, we used all buyer session-items to generate the training triplets (or quartets), leading to about one million (1,000,000) paired session-items elements. This configuration is also maintained to generate triplets (or quartets) for validation, where we used about one hundred thousand (100,000) paired session-items.

After the generation of the siamese network inputs in the feature extraction phase, they are used to generate embedding representations, and then a multi-layer perceptron (MLP) model is trained to generate the classification based on those representations. The MLP model is trained with the Adam optimizer, with a learning rate of 10^{-3} and sparse categorical crossentropy loss function. This model is trained until overfitting, and the weights with the best results (e.g with the lowest loss) in the validation set are saved. For our stop condition, training is paused after the validation loss shows no improvement after at least 30 epochs. The set of weights with the smallest error obtained is saved in order to evaluate the model on test set.

As said about the models proposed in the previous Chapter, they are also divided by the input feature types: sequential-features and aggregated-features. Models using sequential-features as input are based on the base neural network `SequentialFeaturesNetwork`. This base network process input a time series with a fixed-length input. To handle the sessions of varying sizes, we decided to pad and truncate the inputs to size 20. We believe that 20 is an adequate size as it represents 90% of the existing sessions and showed the best results in preliminary experiments.

5.2.1 Model naming

We name the models according to their characteristics. That is, we name it according to the usage of the `BaseNetwork` type, `SequentialFeaturesNetwork` or `AggregatedFeaturesNetwork`, (which consequently selects the input type, sequential-features and aggregated-features), the type of loss used (triplet-loss or quartet-loss), and whether or not the MLP will have class-weights. In Figure

30 shows a diagram that demonstrates our model naming according to their characteristic. In this figure, we can verify that the model TCNQuartet, for example, contains input of type sequential-features, with siamese network with BaseNetwork SequentialFeaturesNetwork with quartet-loss and without class-weights in the model MLP. We can also check that we have its variation with class-weights in the final MLP model named TCNQuartet-CW.

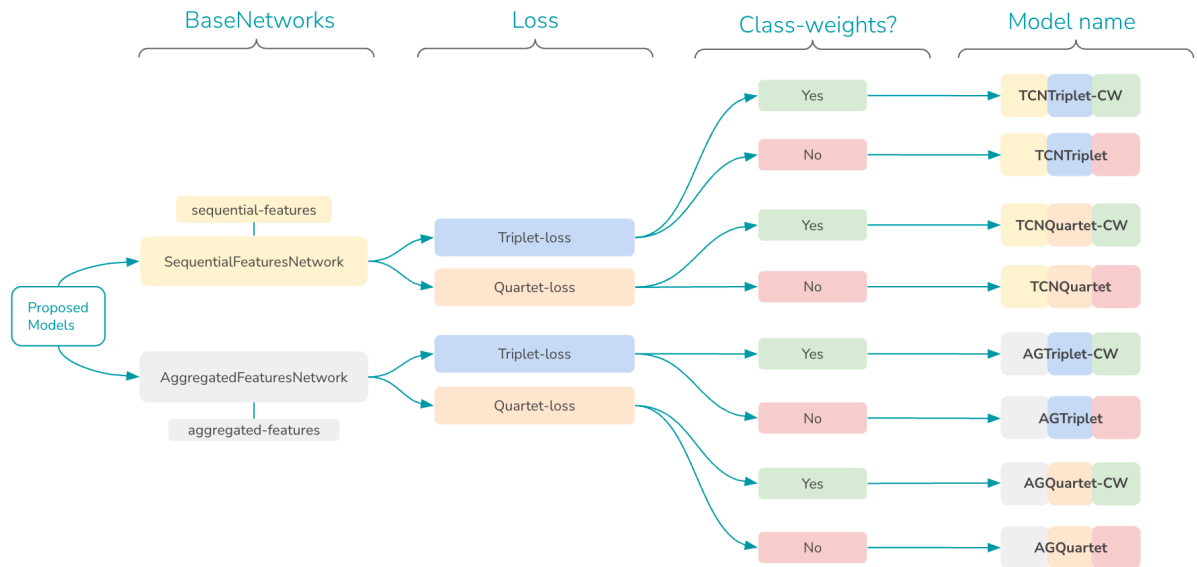


Figure 30 – Proposed models naming diagram.

For the baselines, we did the same. But for baselines, just the name demonstrates the type of BaseNetwork used and whether the model is single-stage or two-stage. In Figure 31 you can observe a diagram that express our baselines naming according to the characteristic. For example, the prefix 'TCN' in TCNBaseline-T, indicates that the BaseNetwork used is SequentialFeaturesNetwork, and the suffix 'T' indicates that the model is two-stage.

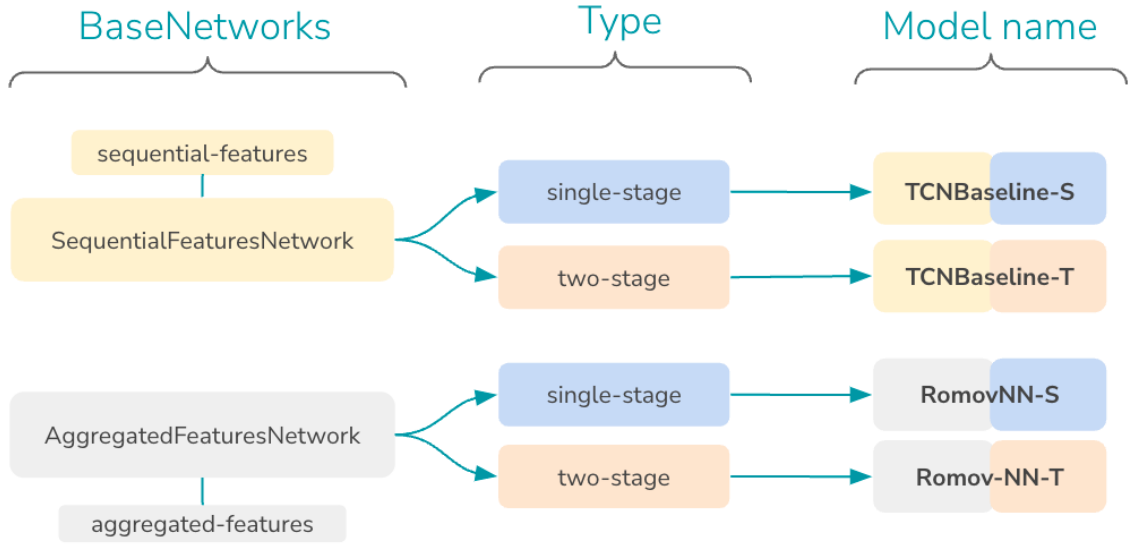


Figure 31 – Baseline naming diagram.

5.2.2 Hyperparameter configuration

The selection of hyperparameters for both the baseline and the proposed models was designed through multiple trial and error processes. We used the Adam optimizer with 10^{-3} as a learning rate with all methods. To avoid overfitting, all models presented in this work applied BatchNormalization to regularize weights as shown in the architectures presented in the previous Chapter.

As defined in Chapter 4 we have to define some important parameters for both quartet-loss and triplet-loss: α , β , γ and *margin*. The larger the parameters β , α and γ , the more importance the loss function gives to classes Si_{nn} , Si_{bn} , Si_{bb} , respectively. The following Table 8 summary the best parameters found for our models.

Model	margin	alpha	beta	gamma
TCNQuartet	0.5	1	0.5	0.7
TCNTriplet	0.1	na.	na.	na.
AGQuartet	0.7	1	0.5	0.7
AGTriplet	0.5	na.	na.	na.

Table 8 – Model proposals parameters for siamese networks. The "na" stands for "not applicable"

5.2.3 Model evaluation

In our problem, the classes are known to be very unbalanced, with a much larger number of non-buyer sessions. Therefore, it's necessary to select the comparison metrics meticulously.

We know that in an e-commerce system, finding the greatest amount of buyers is more important than computing the probability of purchase assertively. This means that the cost of having a false-positive buyer (it was a non-buyer but the model predicted it to be a buyer session) is low in comparison to missing a potential buyer session. Thus, in our case, it is more important to have a higher recall in the buyer classes than high precision. To simulate the model action in an e-commerce event we will use the metrics, precision, recall, f1-score (macro), and confusion matrices.

5.2.3.1 Simulating an online evaluation

In real e-commerce environments, user click sessions are not readily given in full as in the Recsys database. As the user clicks items, the set of clicks available as features for predicting increases, and the methods class predictions continuously increase. To verify how well our models would fit into online environments, where the session grows while the model is continuously trying to predict, we developed two metrics inspired by mean reciprocal rank (MRR) and mean rank (MR).

These metrics attempt to answer the following questions:

1. What is the lowest number of clicks that a method needs to correctly classify a purchase session?
2. What is the fewest number of clicks a method needs to correctly predict a purchased item?

Let $S = \{c_1, c_2, \dots, c_N\}$ be a session of size N , and $S' = \{S_1, S_2, \dots, S_N\}$ the all subsessions of S such that $S_i \subseteq S$ and $S_i \in S$ where S_i is a subsession composed by clicks from initial position 1 to position i . The Smallest number of clicks (SNC) indicates the smallest number of clicks needed to find the purchase. Considering the S' subsections, for example, assuming that a purchase is made in the click c_3

and the model needs 3 clicks to find the purchase then the SNC will be 3. We also developed RSNC which is the inverse of SNC to penalize sessions that the model takes to identify buyer-session or a purchased item. The SNC treats penalties equally while RSNC treats larger penalties as the session grows. Therefore, we can calculate the mean of SNC, called Mean of smallest number of clicks (MSNC) and the mean of RSNC, called Mean of reciprocal smallest number of clicks (MRSNC). Let S^T the set of all sessions clicks, we can mathematically define these metrics as:

$$MSNC = \frac{1}{|S^T|} \sum_{i=1}^{|S^T|} SNC_i$$

and

$$MRSNC = \frac{1}{|S^T|} \sum_{i=1}^{|S^T|} RSNC_i$$

where

$$SNC(x) = \begin{cases} length(x) & \text{if purchase} \\ 0 & \text{otherwise} \end{cases}$$

and

$$RSNC(x) = \begin{cases} 1/length(x) & \text{if purchase} \\ length(x) + 1 & \text{otherwise} \end{cases}$$

Note that when a purchase is not found in all sub-sessions, SNC penalizes it with zero, whereas RSNC penalizes with the session size plus one. It is important to note that we do this evaluation for both common sessions and paired session-items.

Through these metrics, we will have an understanding of how the model behaves in online systems, where predictions must occur as the user interacts with the environment. The higher the value of MRSNC, the quicker the model predicts purchase classes. For the MSNC metric the logic is the opposite, the smaller, the quicker.

5.3 Experimental results

In this section, we present our results in four parts. First, we show overall results for the proposed models. Then we describe recall levels visually according to session lengths. Then, we show the results of a simulated online evaluation. And finally, we present a visual plotting of our embeddings generated by siamese networks.

5.3.1 Overall results

To evaluate our overall results, we evaluated the following metrics f1-score (macro version), precision, recall, and confusion matrix. From the f1-score metric, we can obtain the general performance of the models concerning the classes. However, the f1-score can generate a false impression regarding minority classes Si_{bn} and Si_{bb} if the model has its predictions mostly in the class Si_{nn} . For this reason, we have to observe the trade-off between precision and recall metrics for different classes.

By analyzing precision, we can evaluate the quality of the predictions, and from recall levels, we can judge how good is the model at finding the correct items. For this research problem, it is much more interesting to evaluate the recall metric, since it is much more important in e-commerce systems to find the potential purchasing sessions than to have high precision in this decision. This means that the cost of detecting non-buyer-sessions as buyer-sessions (i.e. the false-positives) is much lower than detecting buyer-sessions as non-buyer-sessions (i.e. the false-negatives). Therefore, it is more advantageous for retail stores to have a higher number of false-positives and a low number of false-negatives because every time stores miss a potential buyer-session, they miss opportunities to use methods that are more focused on sessions that will indeed buy products and are not only "window shopping".

We know that most instances are of the Si_{nn} class and that most models would be able to have a high recall for this class. Therefore, what we should consider is the recall for the purchase classes Si_{bn} and Si_{bb} . The purchase classes are harder to find and are the classes of interest for purchase prediction.

Another interesting metric for unbalanced problems is the confusion

matrix. From there, it is possible to identify the summary of predictions for each class. Therefore, it can identify if they have focused on having high precision/recall in the most frequent class.

The Table 9 shows overall results for f1-score metric. The F1 column refers to the overall f1-score in its macro version. The columns $F1_{Si_{nn}}$, $F1_{Si_{bn}}$, $F1_{Si_{bb}}$ refer to the separate f1-scores of the classes Si_{nn} , Si_{bn} and Si_{bb} . The Table 10 show overall results for precision and recall metrics. Th. The columns $PR_{Si_{nn}}$, $PR_{Si_{bn}}$, $PR_{Si_{bb}}$ represent the precision metric of the classes Si_{nn} , Si_{bn} and Si_{bb} respectively. Finally, the columns $REC_{Si_{nn}}$, $REC_{Si_{bn}}$, $REC_{Si_{bb}}$ describe the recall metric of the Si_{nn} , Si_{bn} and Si_{bb} classes respectively.

Model	F1	$F1_{Si_{nn}}$	$F1_{Si_{bn}}$	$F1_{Si_{bb}}$
TCNBaseline-T	0.48	0.93	0.29	0.20
RomovNN-T	0.50	0.91	0.32	0.26
TCNBaseline-S	0.45	0.91	0.24	0.20
RomovNN-S	0.44	0.83	0.27	0.21
AGQuartet	0.50	0.91	0.30	0.27
AGQuartet-CW	0.42	0.80	0.25	0.22
AGTriplet	0.47	0.92	0.29	0.20
AGTriplet-CW	0.41	0.79	0.25	0.20
TCNQuartet	0.45	0.88	0.25	0.24
TCNQuartet-CW	0.40	0.79	0.21	0.21
TCNTriplet	0.48	0.91	0.29	0.25
TCNTriplet-CW	0.40	0.77	0.24	0.18

Table 9 – F1-score results for our baseline and proposed models. The first four models in the first part of the divider are our baselines.

The Table 9 shows the results for the f1-score of the models presented. From the table, we observed that the model AGQuartet and RomovNN-T were tied concerning the overall f1-score, reaching 0.50. It is worth mentioning that the general metric can be misleading, as it also encompasses the class with the highest quantity Si_{nn} . Regarding the metric $F1_{Si_{nn}}$ the highest values were obtained by the models TCNBaseline-T and AGTriplet. This may have occurred because these models do not treat the minority class differently, making the $F1_{Si_{nn}}$ metric to be higher.

Regarding the $F1_{Si_{bn}}$ metric, the model RomovNN-T obtained the highest score, in second place was our model AGQuartet. For the $F1_{Si_{bb}}$ metric our

model AGQuartet achieved 1 percent higher than the baseline RomovNN-T. The models TCNQuartet and TCNTriplet, achieved the similar results as RomovNN-T with 0.24 and 0.25, respectively. The results achieved by the AGQuartet model can be explained by our quartet-loss, which better separates each class, and also by the effectiveness of the aggregated-features that can better describe the raw features for our problem.

Model	PR_Si _{nn}	PR_Si _{bn}	PR_Si _{bb}	REC_Si _{nn}	REC_Si _{bn}	REC_Si _{bb}
TCNBaseline-T	0.94	0.25	0.21	0.92	0.34	0.21
RomovNN-T	0.96	0.23	0.20	0.87	0.49	0.36
TCNBaseline-S	0.94	0.19	0.16	0.87	0.35	0.26
RomovNN-S	0.97	0.17	0.14	0.72	0.61	0.49
AGQuartet	0.95	0.24	0.21	0.88	0.41	0.38
AGQuartet-CW	0.97	0.15	0.14	0.68	0.66	0.56
AGTriplet	0.94	0.23	0.21	0.91	0.40	0.20
AGTriplet-CW	0.97	0.16	0.12	0.66	0.60	0.60
TCNQuartet	0.96	0.18	0.16	0.81	0.40	0.46
TCNQuartet-CW	0.97	0.14	0.12	0.67	0.51	0.59
TCNTriplet	0.95	0.22	0.21	0.88	0.43	0.31
TCNTriplet-CW	0.97	0.15	0.11	0.64	0.58	0.63

Table 10 – Precision and recall results for our baseline and proposed models. The first four models in the first part of the divider are our baselines.

The Table 10 extends the results for the presented models, showing the precision and recall of each class. This Table shows that the models that showed better F1 results in the previous table, such as AGQuartet, RomovNN-T and TCNTriplet, were better in overall class precision, but not in the recall. Interestingly, the RomovNN-S model obtained better recall for Si_{bn} and Si_{bb} than the baseline RomovNN-T. This may be because single-stage models can help to reduce the information loss that exists in the two-stage model. In two-stage models, you must use the output of the first model to generate the output of the class of the second model.

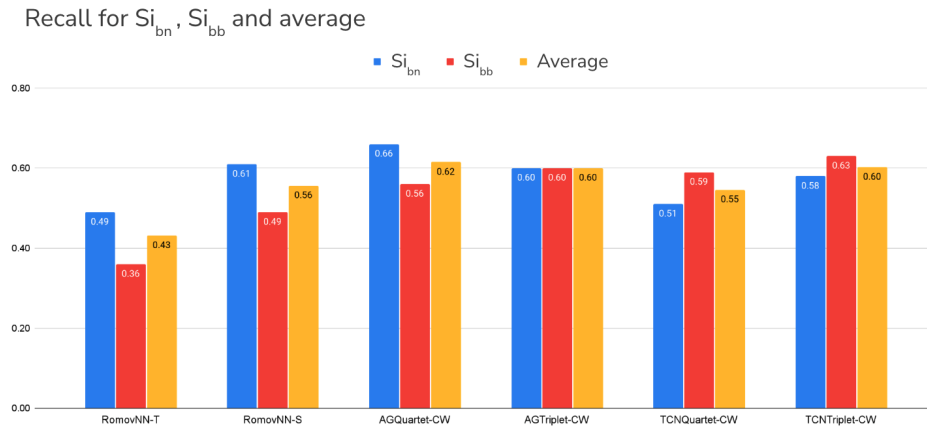


Figure 32 – Recall metrics for best models that were outstanding in Si_{bn} and Si_{bb} . (Source: Own author)

Another interesting fact is that all models with class-weights (AGQuartet-CW, AGTriplet-CW, TCNQuartet-CW, TCNTriplet-CW) were good overall in both recall for Si_{bn} and Si_{bb} , although they are worse in precision metrics. The model AGQuartet-CW found the best recall for Si_{bn} with 0.66 and the model TCNQuartet-CW found the best recall for Si_{bb} with 0.61 points. This was expected since models with class-weights force prediction on classes Si_{bn} and Si_{bb} .

As our interest metric is recall and interest classes are purchase classes Si_{bn} and Si_{bb} , we generated a Figure 32 for the models that excelled the most in this scenario. Note that we also added a REC_AVG metric which is the average of the two recalls (micro). Observing the REC_AVG metric, the AGQuartet-CW model was better than all the other models. This possibly shows that the quartet-loss models can generate better separation spaces than the triplet-loss ones for aggregated-features.

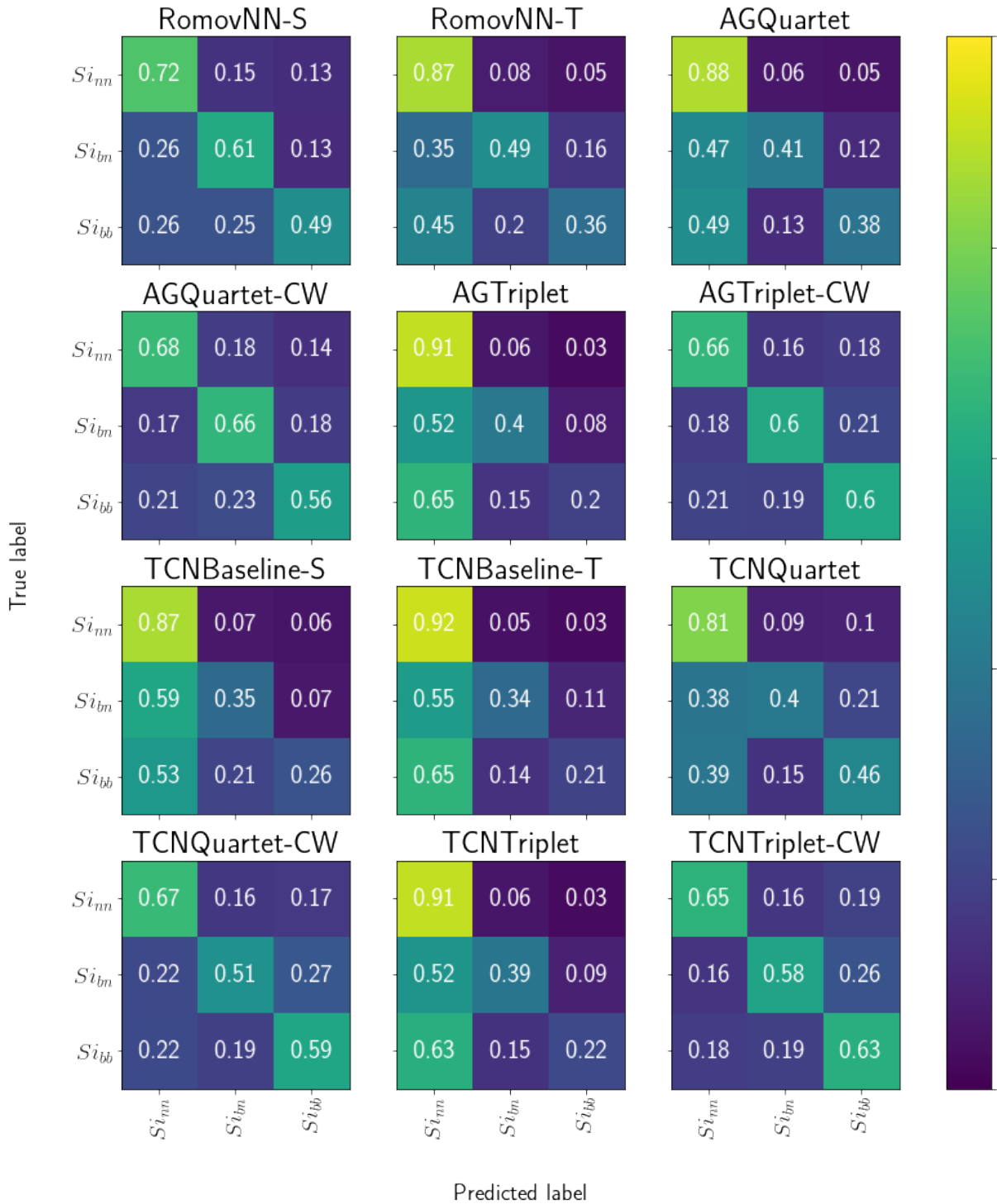


Figure 33 – Confusion matrices for the presented models. The lighter the cell color, the higher the score. (Source: Own author)

One way to explore more deeply the predictions and accuracy of the models is through the confusion matrix. The Figure 33 shows the confusion matrices for the models presented. Note that the lighter the cell, the higher the score. Also, the lighter the main diagonal cells of the confusion matrix, the better

the model.

We can see that the two-stage models RomovNN-T and TCNBaseline-T predicted the classes Si_{nn} and the purchase class Si_{bn} in their majority. From the prediction vector for the class Si_{nn} , it is possible to observe that the two-stage models predicted many purchase classes as a non-purchase class. This behavior also happens in single-stage models where class-weights are not applied. For instance, in models AGQuartet and AGTriplet, there is a high hit rate in the class Si_{nn} , but there is a high occurrence of wrong predictions of predictions of the class Si_{nn} as Si_{bn} and Si_{bb} purchase classes.

In the models with class-weights techniques, we have an interesting behavior. In the models AGQuartet-CW, for example, there was a high rate of correct predictions in the interest classes Si_{bn} and Si_{bb} , although with larger errors in the non-buyer class Si_{nn} .

5.3.2 Recall metric by sessions lengths

Another way to look at the impact of model predictions is to evaluate metrics by session sizes. Looking at how models evaluate session size can help inform which models take advantage of more clicked item information for their predictions. As mentioned in the last subsections, the recall metric is the most interesting for our context, as we want are more interested in maximizing true positives than to minimize false positives, especially in our interest classes. Figure 34 shows recalls by session sizes for baseline models. Model RomovNN-T seems to get the most stable results, getting better results on Si_{bb} in relation to others baselines. Also, the RomovNN-S obtained an interesting result for a higher recall for the purchase classes Si_{bn} and Si_{bb} in small sessions. The recall for Si_{bb} appears to be the most difficult to find compared to baselines. The baseline models with TCN (TCNBaseline-T and TCNBaseline-S) seem to lack stability against classes Si_{bn} and Si_{bb} as sessions grow.

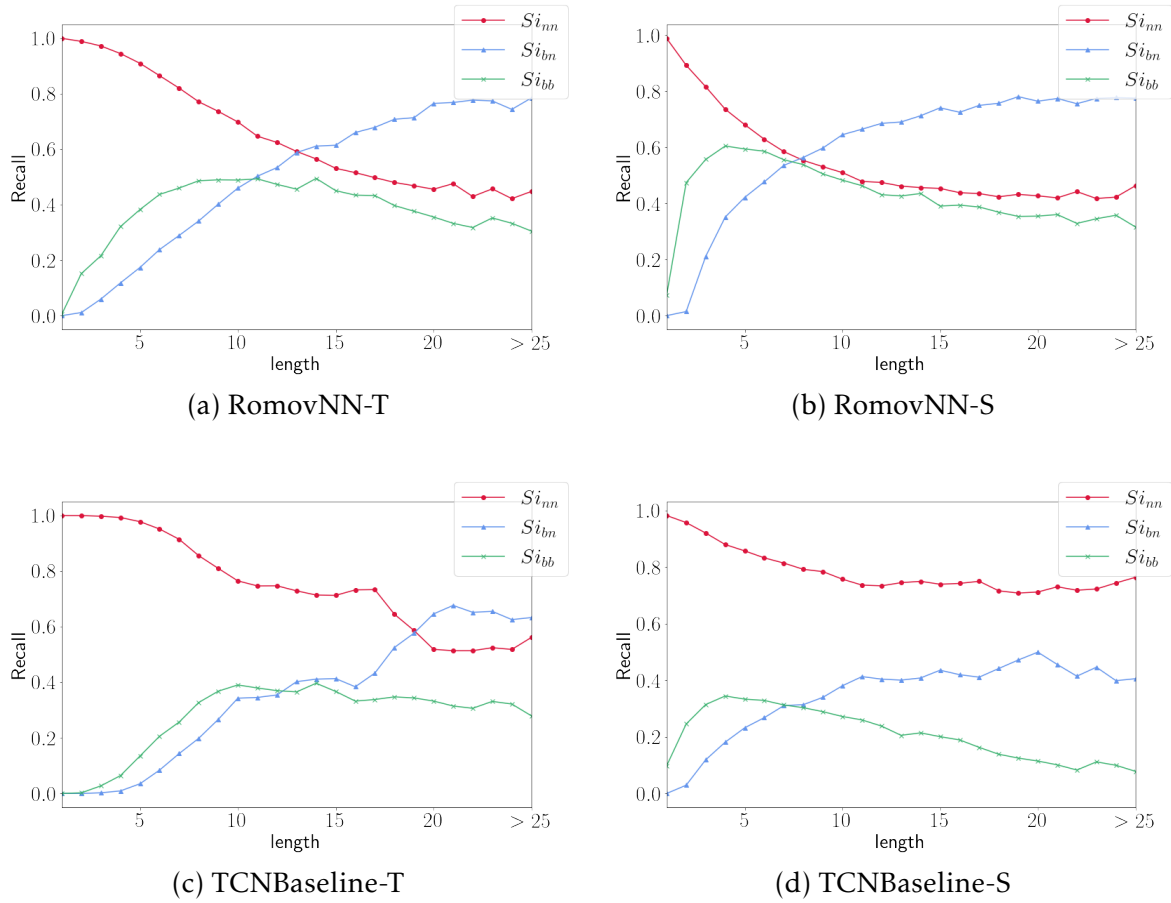
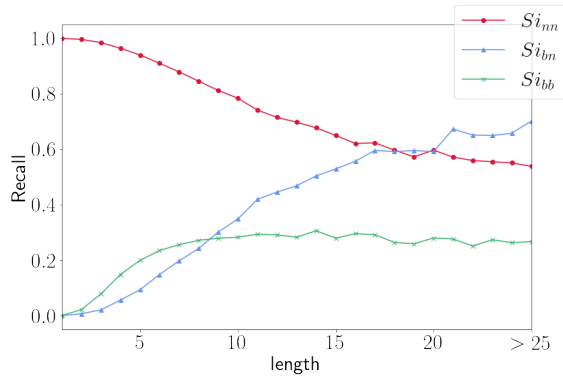
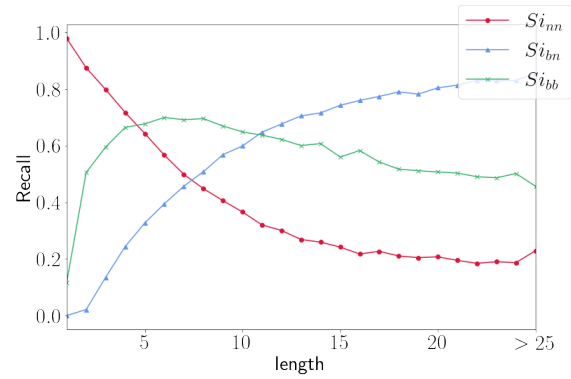


Figure 34 – Recall by session lengths for baseline models. (Source: Own author)

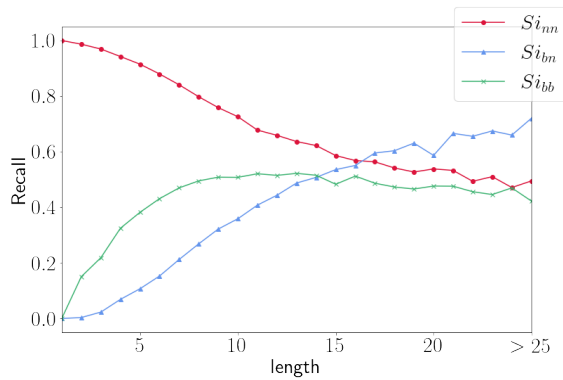
The Figures 35 and 36 show recalls by session sizes for our proposed models. Considering the Figure 35 we can understand that models with class-weights AGTriplet-CW and AGQuartet-CW delivered greater recall for purchase classes Si_{bn} , Si_{bb} as sessions lengths grow, especially between smaller sessions (size less than 5). This situation corresponds with these models, as the class-weights make the model give a bigger emphasis to correct predictions in the purchase classes. Another interesting point is that the model AGQuartet compared to AGTriplet is more stable in all classes and also obtained better results for the class Si_{bb} . This possibly shows a superiority of the quartet-loss for this class than the triplet-loss for this situation.



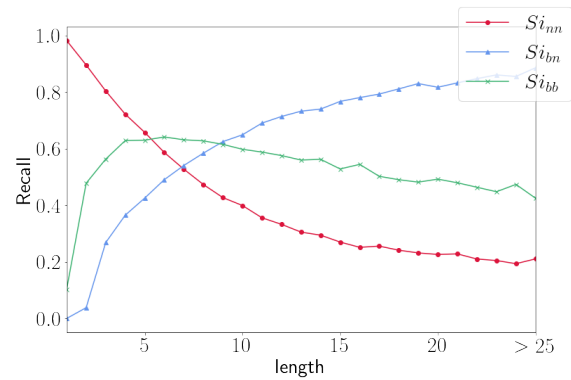
(a) AGTriplet



(b) AGTriplet-CW



(c) AGQuartet



(d) AGQuartet-CW

Figure 35 – Recall by session lengths for proposed models which uses aggregated-features. (Source: Own author)

In addition, the Siamese network also helped in the separation of classes, as the single-stage models that were not helped by the Siamese network (TCNBaseline-S, RomovNN-S), performed worse. Despite these positive results, these models also lose out in recall for the Si_{nn} non-purchase class as session sizes increase.

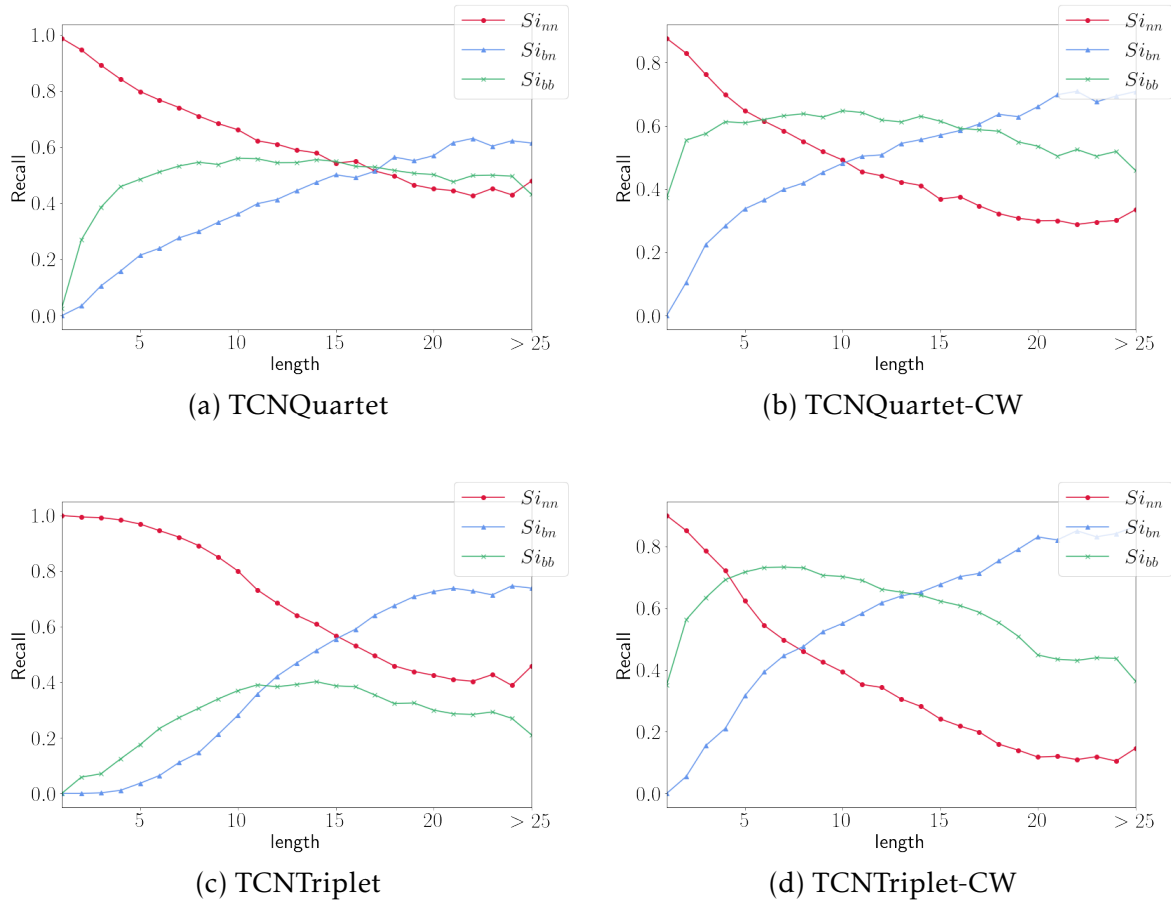


Figure 36 – Recall by session lengths for proposed models which uses sequential-features. (Source: Own author)

In the Figure 36 we see the models for sequential-features. Generally speaking, the models learned to classify more purchase classes as the size of the sessions grew. Something important to note is that the TCNQuartet model had a performance similar to the AGQuartet with balanced recall between the classes.

Also, the AGQuartet model as well as the AGQuartet model also performed better in the Si_{bb} class. This could also be because quartet-loss is possibly better than triplet-loss to find the recall in this class.

Another interesting fact, TCN models with class-weights, TCNTriplet-CW and TCNQuartet-CW, had stable recall for class Si_{bb} at all session sizes, even for small sessions. This can be useful for finding potential items that will be purchased for most sessions, as most user sessions are small.

5.3.3 Simulated online evaluation

As shown in the evaluation subsection, the metrics MSNC and MRSNC are two metrics that describe how quickly models find a purchased class as the click session grows. From them, it is possible to extract how a model would behave in online systems.

Model	MRSNC₁	MRSNC₂
TCNBaseline-T	0.029	0.026
RomovNN-T	0.090	0.059
TCNBaseline-S	0.149	0.059
RomovNN-S	0.290	0.112
AGQuartet	0.132	0.058
AGQuartet-CW	0.306	0.121
AGTriplet	0.070	0.032
AGTriplet-CW	0.322	0.132
TCNQuartet	0.162	0.085
TCNQuartet-CW	0.389	0.154
TCNTriplet	0.091	0.046
TCNTriplet-CW	0.354	0.153

Table 11 – Results for MRSNC for presented models. The higher the metric value, the better the result.

Model	MSNC₁	MSNC₂
TCNBaseline-T	5.395	12.461
RomovNN-T	4.568	11.808
TCNBaseline-S	4.239	12.727
RomovNN-S	3.510	11.824
AGQuartet	4.729	12.275
AGQuartet-CW	3.291	11.411
AGTriplet	5.369	12.913
AGTriplet-CW	3.200	11.108
TCNQuartet	3.802	11.257
TCNQuartet-CW	2.810	10.570
TCNTriplet	4.507	12.483
TCNTriplet-CW	2.894	10.783

Table 12 – Results for MSNC for presented models. The lower the metric value, the better the result.

Table 11 and Table 12 shows the results for all presented models. The columns MRSNC₁ and MSNC₁ describe MRSNC and MSNC metrics to find

whether a session is a purchase (either Si_{bn} or Si_{bb}). The columns $MRSNC_2$ and $MSNC_2$ describe $MRSNC$ and $MSNC$ metrics to find the specific purchased item(s), that is the Si_{bb} class. For the metric $MSNC$ the lower the value the better the model, for the metric $MRSNC$ the higher the value the better the model.

Surprisingly, our models with class-weights achieved great $MRSNC_1$ and $MSNC_1$ metrics, outperforming the baseline RomovNN-S, as expected, and our vanilla (i.e. without class weights) models. Overall metrics for columns $MRSNC_2$ and $MSNC_2$ are low, possibly because finding a specific purchase item Si_{bb} is a much more difficult task. Moreover, our models with class-weights also achieve the best results overall, highlighting the TCNQuartet-CW model as the one with the strongest results in these metrics. This is possible because single-stage models work better than two-stage models for the Si_{bb} class and because of the superiority of our quartet-loss over triplet-loss. The class-weights technique also has a significant impact, helping the simpler models to predict purchase classes more quickly, which improves these metrics.

Model	$MRSNC_{1_NB}$	$MSNC_{1_NB}$
TCNBaseline-T	0.003	3.636
RomovNN-T	0.020	4.103
TCNBaseline-S	0.037	3.379
RomovNN-S	0.102	3.564
AGQuartet	0.025	4.017
AGQuartet-CW	0.110	3.470
AGTriplet	0.013	4.124
AGTriplet-CW	0.127	3.392
TCNQuartet	0.050	3.237
TCNQuartet-CW	0.187	2.763
TCNTriplet	0.017	3.472
TCNTriplet-CW	0.137	2.907

Table 13 – Results for $MRSNC$ and $MSNC$ for non-buyer-sessions. For $MRSNC$ the lower, the better the result. For $MSNC$ the higher, the better the result.

Finally, Table 13 represents the metrics $MRSNC$ and $MSNC$ with regards to non-purchase-sessions. The columns $MRSNC_{1_NB}$ and $MSNC_{1_NB}$ describe $MRSNC$ and $MSNC$ metrics, respectively. These last metrics can tell us how much the model confuses the non-purchase session with a purchase session, as the num-

ber of clicks in the session grows. For the MRSNC metric, the smaller the better, because the closer to 1, the more the model wrongly predicts the non-purchase session as a purchase-session. The MSNC metric shows the average position at which the model mispredicts the non-purchase-session as purchase-session. That is, the bigger MSNC the better the model is, as it takes longer to consider non-purchase-sessions as purchase-sessions. As expected, all models that find more purchase classes performed worse in this table. The model with the best MRSNC₁_NB was TCNBaseline-T and the model with the best MSNC₁_NB was the model AGTriplet. These two results make sense, as the model TCNBaseline-T is two-stage, which makes the model more conservative in predicting purchase class. The AGTriplet model is also expected to have obtained better results for this metric, as it also does not contain the class-weights technique.

5.3.4 Embeddings visualization

Before generating the simple classifier with class-weights, our models perform feature extraction through the Siamese network to generate embedding representations of the sessions/session items. In this section, we will show the visualization of the encodings generated by our models using the Principal Component Analysis (PCA) method. Our visualizations were generated with opacity equal to 0.20 to facilitate the visualization of the overlaps. Also, we did undersampling in our dataset to balance classes, with 2000 elements in each class. As shown at the beginning of the Chapter, we know that the smaller the Section the more difficult it is to classify them. For this reason, we show the plots for small sessions (less than five clicks) and large-sessions (greater than five clicks). Sessions in sizes up to five represent about 60% of the database.

The Figure 37 shows the plots for the models TCNQuartet and TCNTriplet. From the plot of TCNQuartet it is possible to see a separation between the classes for both small-sessions and large-sessions. However, there doesn't seem to be much difference in quality between small-sessions and large-sessions for TCNQuartet. For TCNTriplet, the embeddings appear to be visually different in each figure. In large-sessions it is possible to see the separation better than in small-sessions.

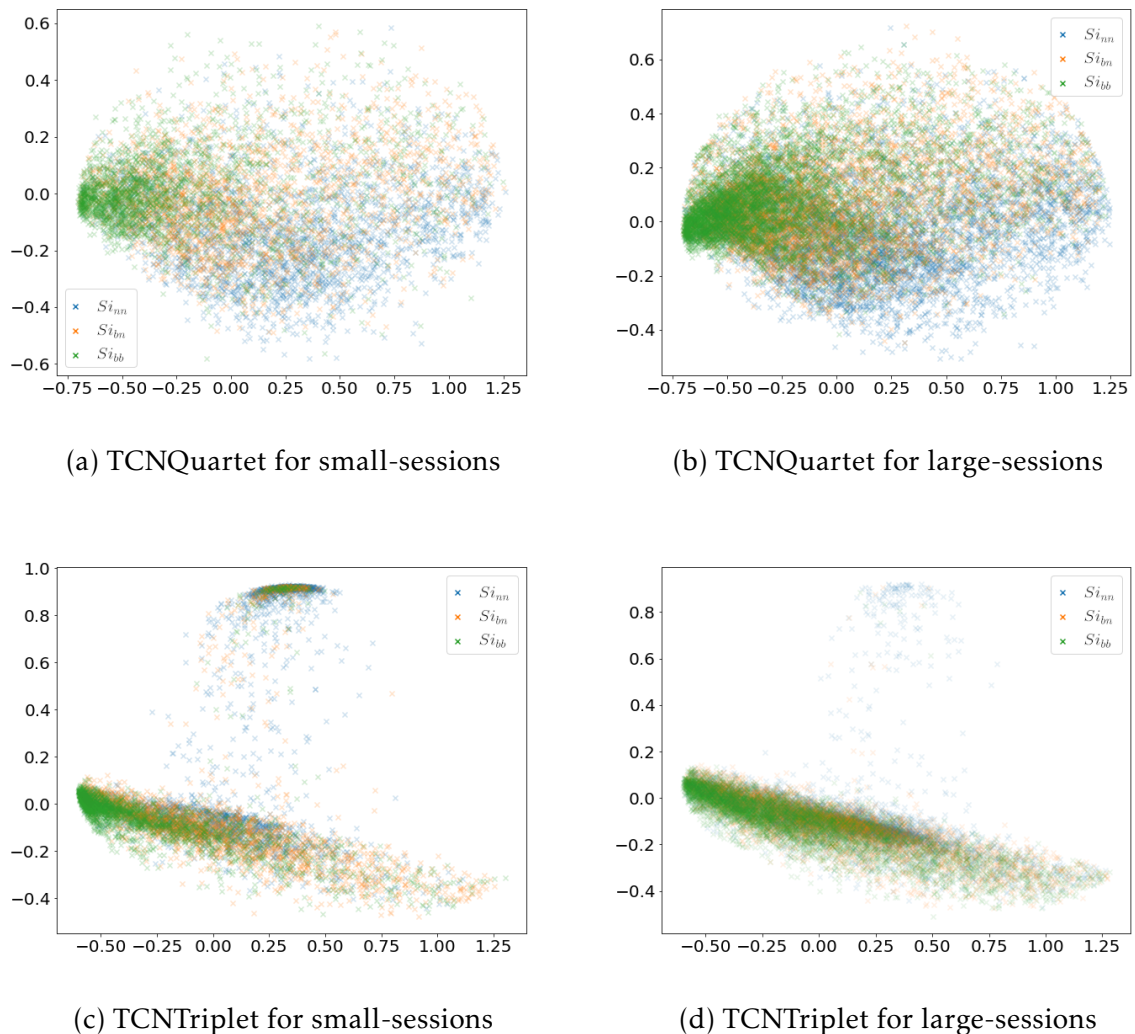
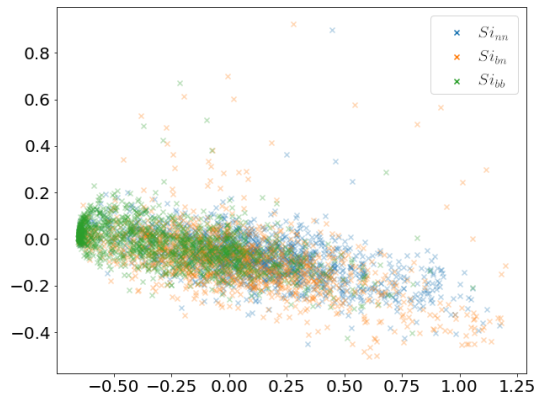


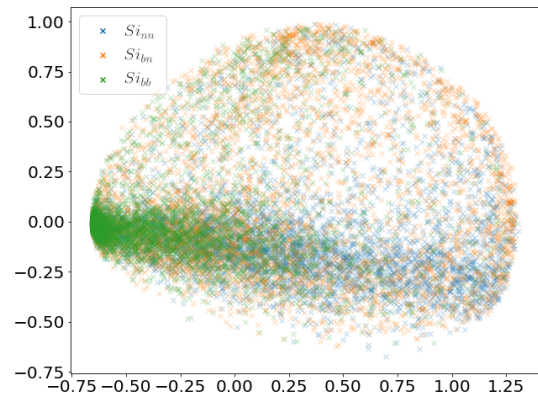
Figure 37 – Embeddings visualizations for TCNQuartet and TCNTriplet for small and large-sessions. (Source: Own author)

Figure 38 shows the plots for models AGQuartet and AGTriplet. From the embeddings of AGQuartet, we can see that in small-sessions the clusters seem to have more overlap than in large-sessions. This may explain why the model misclassifies small-sessions more. Large-session embeddings for AGQuartet seem to be more identifiable in the latent space than AGTriplet. In AGTriplet in both situations, it can be observed that the separation of classes happened, even with a lot of overlapping. Also, not much difference was observed in the visualization of small-sessions for large-sessions.

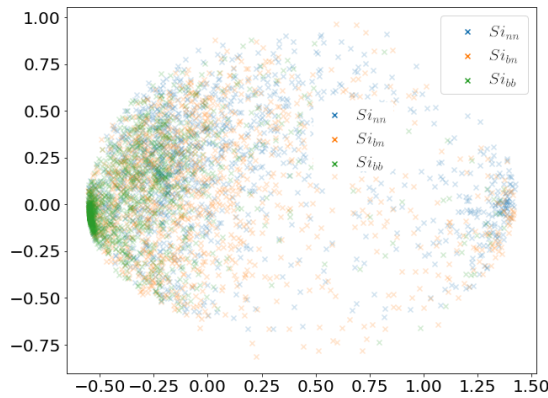
The overlapping of the classes in the embeddings can explain the difficulty of the problem in separating the classes. Many sessions can appear to be purchase-



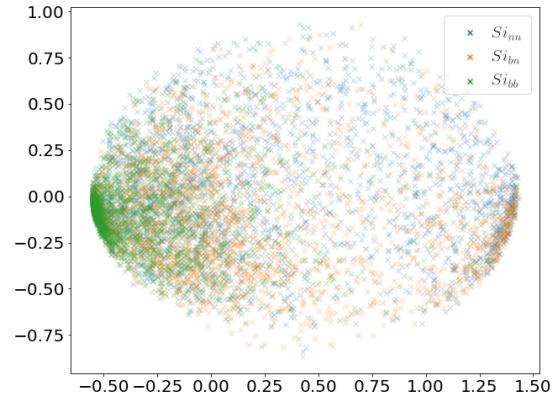
(a) AGQuartet for small-sessions



(b) AGQuartet for large-sessions



(c) AGTriplet for small-sessions



(d) AGTriplet for large-sessions

Figure 38 – Embeddings visualizations for AGQuartet and AGTriplet for small and large-sessions. (Source: Own author)

sessions even though with a low number of clicks. Even with this difficulty, the classes can still be seen as being separated to better represent this, we made a visualization through the k-means algorithm, to detect the easiest examples to be classified and therefore to be visualized in space. With this, we merge well-classified (which is around the k-means centroids) examples setting alpha equal to 1 in the image and with random examples with alpha 0.40. In this way, we can highlight the separations and observe in which positions the overlaps to occur more with the well-classified examples. Figure 39 shows the result of this representation created. From there, it is possible to verify with better quality the examples separated by quartet-loss or triplet-loss. From this image, it is also

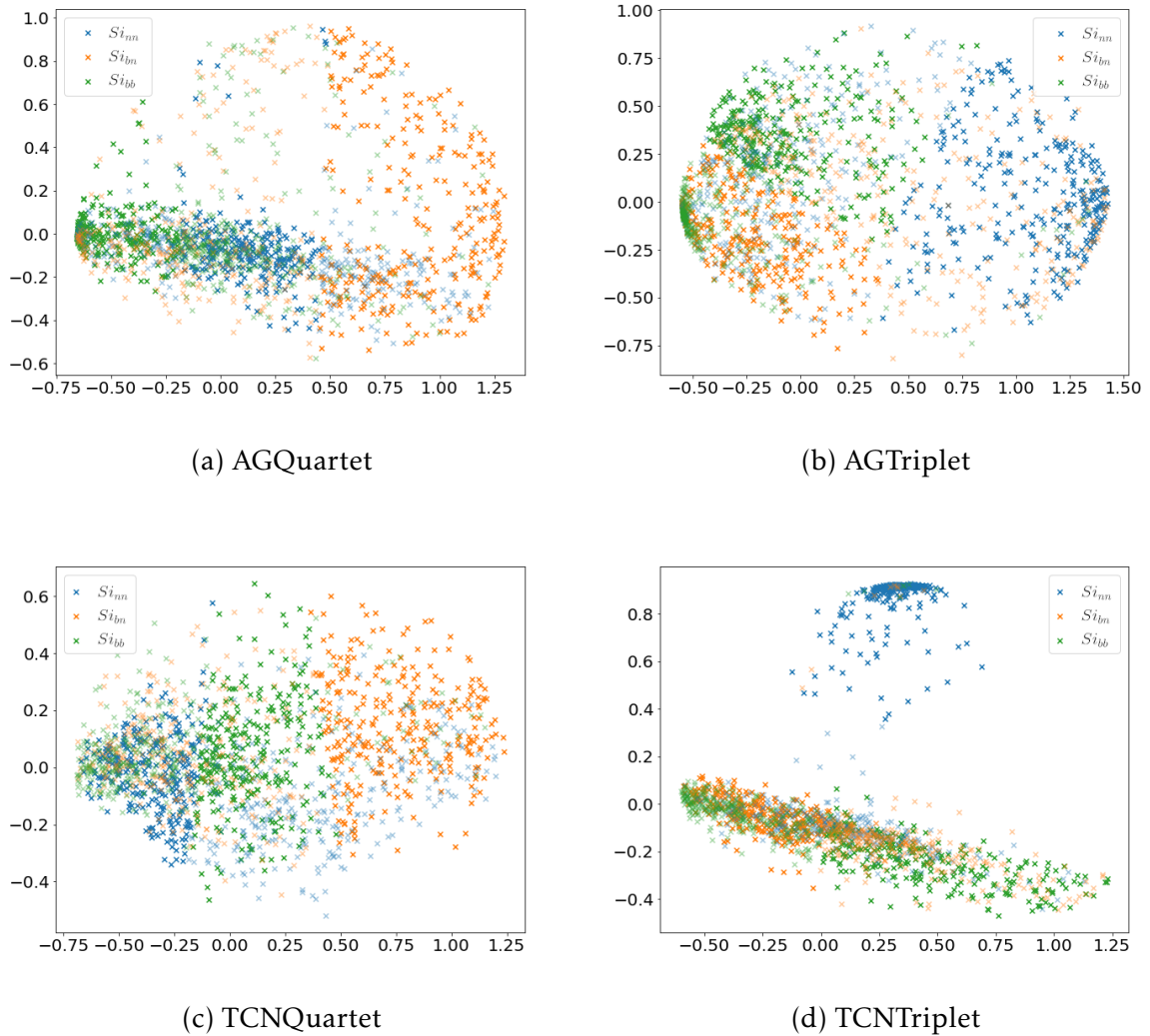


Figure 39 – Embeddings visualizations for proposed models with K-means. (Source: Own author)

Model	Silhouette Score
AGQuartet	0.015
AGTriplet	0.004
TCNQuartet	0.010
TCNTriplet	0.011

Table 14 – Results for Silhouette Coefficient for presented models.

possible to verify that there is a lot of confusion between the Si_{bb} and Si_{nn} classes.

About the Silhouette score, as can be seen in table 14, it can be seen that AGQuartet achieved better results in this metric, which can be a consistent explanation of the performance of this method concerning the other evaluated.

For our experiments, we use the Silhouette method with Euclidean dis-

tance. In the Table 14 we can see that the best value was for the model AGQuartet. This may explain why this model did better in recall metrics.

5.4 Final considerations

In this Chapter, we presented our experimental results of our proposed models. We propose models based on input type sequential-features and aggregated-features. Inputs of type sequential-features are more generic and describe data over time. Inputs of type aggregated-features are aggregated and handcrafted. Our proposed models achieved good results using both kinds of features, all while using a single-stage prediction model. In particular, our model AGQuartet delivered good results for f1-macro and recall for purchase classes Si_{bn} and Si_{bb} against the two-stage baseline RomovNN-T . A model that also gained prominence was the TCNQuartet-CW which, through our metrics MSNC and MRSNC, showed that they find purchase sessions faster than the baseline RomovNN-S.

From these results, the superiority of our proposed quartet-loss over the triplet-loss is evident. This is reinforced also through the embedding visualizations shown, classification metrics (Recall and F1-score), and through the Silhouette Coefficient, which proved to be better in the quartet-loss model AGQuartet. Furthermore, we believe that the session embeddings generated by the siamese networks can further be explored in other tasks related to user sessions in e-commerce environments.

6 Conclusions and Future work

In this section, we show the conclusions of our entire research project, the limitations of our work, and future projects that can be worked on.

6.1 Conclusions

In this work, we introduced several new methods to solve the item purchase prediction problem in a single stage. Through our experiments, we showed that solving the problem in a single stage is not reasonably easy, however, we managed to achieve better results than the baselines implemented even with two stages. Our proposed models achieved better results for finding buyer sessions and identifying purchased items. Because of class imbalance, the baseline models focused on their predictions of the non-purchasing class. And even with the imbalance problem, our models found a greater recall for the purchase classes.

In addition to presenting the impact of using the input representation based on the work of Romov [Romov et al., 2015] over our models, we also show how to represent sessions using sequential networks. We also show the influence of using different weights for minority classes in the classification after the extraction of features made by Siamese networks. Furthermore, we did extensive experimentation with all these possible variations and their effects on the final result.

Furthermore, we proposed the implementation of a new loss function, the quartet-loss, for Siamese networks. The quartet-loss proved to be more effective for our problem, as it adds one more negative than the triplet-loss. The quartet-loss seems more compatible with our problem since it avoids treating the two negative classes as the same, as occurs in triplet-loss. Using quartet-loss we observed better results concerning the triplet-loss.

Another important contribution in this work are the metrics MSNC and MRSNC. These metrics evaluate the model as the user clicks in session. This refers to what happens in the online scenario, where the session grows as the user clicks more items, and an earlier correct prediction may lead to potential

monetary gains for the platform. Through experiments, our models with class-weights proved to be more effective for online evaluation, with emphasis on the TCNQuartet-CW model, that obtained best results for MRSNC₁ and MRSNC₂.

6.2 Future works and Limitations

Although our experiments presented solid results for our methods, they were run using a single dataset. For future works, we intend to do experiments using new datasets, such as Retail rocket [Rocket, 2017] and Brazilian E-Commerce Public Dataset by Olist [oil, 2018].

The creation of embeddings by Siamese networks was effective for our problem. Moreover, it could also be used for a variety of other consumer problems. Among the possible tasks, we can list Churn prediction, Customer Lifetime Value prediction, Next basket recommendation, Product recommendation, etc.

In this work, we also proposed the use of two types of features, sequential-features and aggregated-features. We observed the different behaviors for each model and concluded that each set of features had an impact on our evaluation metrics. Models with aggregated-features had better purchase prediction metrics across the entire session, while sequential-features models had better metrics simulating online environments. One of the possible next tasks is to use both types of inputs at the same time, and check what would be the impact of a hybrid model on the ones presented.

From the visualizations of the embeddings generated by the Siamese networks, we observed that despite the separation between the classes, unfortunately, there is a lot of overlap between the encodings of the sessions. In this work, the characteristics of the sessions that lead the model to confuse the classes were not evaluated in depth. Therefore, it should be studied in more depth, why there are so many overlappings in the embeddings generated by the Siamese network.

Bibliography

- Brazilian e-commerce public dataset by olist. <<https://www.kaggle.com/olistbr/brazilian-ecommerce>>, 2018. Accessed: 2022-01-01. 95
- aaaa.
- Bahdanau et al. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 6, 30, 60
- S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018. 33
- J. Bennett, S. Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007. 22
- C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738. 23, 24, 25
- J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a " siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993. 6, 34, 36
- Chen et al. Predicting purchase behavior of e-commerce customer, one-stage or two-stage? *DEStech Transactions on Computer Science and Engineering*, (aics), 2016. 18, 48
- Y. Chu, H.-K. Yang, and W.-C. Peng. Predicting online user purchase behavior based on browsing history. In *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*, pages 185–192. IEEE, 2019. 45
- J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. 28
- A. Das, H. Yenala, M. Chinnakotla, and M. Shrivastava. Together we stand: Siamese networks for similar question retrieval. pages 378–387, 01 2016. doi: <10.18653/v1/P16-1036>. 6, 36
- J. Dean. *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners*. CreateSpace Independent Publishing Platform, USA, 2014. ISBN 1502462915, 9781502462916. 24, 25, 26
- J. Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 19

- J. Donahue et al. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014. 26
- J. Eggermont, J. N. Kok, and W. A. Kusters. Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1001–1005, 2004. 19
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016. 27, 28
- M. L. Ha and V. Blanz. Deep ranking with adaptive margin triplet loss. *CoRR*, abs/2107.06187, 2021. URL <<https://arxiv.org/abs/2107.06187>>. 36
- A. Hertzmann and D. Fleet. *Machine Learning and Data Mining Lecture Notes*. 2010. 22, 24
- Hidasi et al. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015. 46
- Hidasi et al. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018. 46
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, Nov. 1997. ISSN 0899-7667. doi: <10.1162/neco.1997.9.8.1735>. 28
- H. Hu and X. He. Sets2sets: Learning from sequential sets with neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1491–1499, 2019. 46
- C. Huang, X. Wu, X. Zhang, C. Zhang, J. Zhao, D. Yin, and N. V. Chawla. On-line purchase prediction via multi-scale modeling of behavior dynamics. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2613–2622, 2019. 45
- Johnson et al. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019. 19
- W.-C. Kang and J. McAuley. Self-attentive sequential recommendation. 2018. 46
- S. Keele et al. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007. 41
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: <10.1109/5.726791>. 23, 31

- T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. URL <http://arxiv.org/abs/1708.02002>. 70, 71
- C. Ling, T. Zhang, and Y. Chen. Customer purchase intent prediction under on-line multi-channel promotion: A feature-combined deep learning framework. 7:112963–112976, 2019. doi: <10.1109/ACCESS.2019.2935121>. 44
- Y. Liu, B. Guo, N. Li, J. Zhang, J. Chen, D. Zhang, Y. Liu, Z. Yu, S. Zhang, and L. Yao. Deepstore: An interaction-aware wide&deep model for store site recommendation with attentional spatial embeddings. *IEEE Internet of Things Journal*, 6(4):7319–7333, 2019. 44
- M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL <http://arxiv.org/abs/1508.04025>. 31
- T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072. 22
- Ø. H. Myklatun, T. K. Thorrud, H. Nguyen, H. Langseth, and A. Kofod-Petersen. Probability-based approach for predicting e-commerce consumer behaviour using sparse session data. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, pages 1–4. 2015. 45, 53
- A. v. Oord et al. WaveNet: A generative model for raw audio. 2016. 60
- A. Palmer, R. Jimenez, and E. Gervilla. Data mining: Machine learning and statistical techniques. 2011. 22
- Phillips et al. *Pricing and revenue optimization*. Stanford University Press, 2005. 16
- Recsys. Recsys. <https://www.kaggle.com/chadgostopp/recsys-challenge-2015>, 2015. Accessed: 2021-01-04. 17, 51, 64
- J. Richter. Temporal convolutional networks for sequence modeling. <https://dida.do/blog/temporal-convolutional-networks-for-sequence-modeling>, 2020. Accessed: 2022-01-24. 34
- R. Rocket. Retail rocket. <https://www.kaggle.com/retailrocket/ecommerce-dataset>, 2017. Accessed: 2021-01-01. 17, 95
- Romov et al. Recsys challenge 2015: ensemble learning with categorical features. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*. ACM, 2015. 18, 20, 21, 44, 45, 48, 50, 54, 70, 94
- P. Romov, 2015. URL <https://github.com/romovpa/ydf-recsys2015-challenge>. 69

- P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. 37
- T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016. 34
- F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015a. URL <<http://arxiv.org/abs/1503.03832>>. 7, 57
- F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015b. URL <<http://arxiv.org/abs/1503.03832>>. 20, 37, 56
- Sermanet et al. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 26
- Sheil et al. Predicting purchasing intent: Automatic feature learning using recurrent neural networks. *arXiv preprint arXiv:1807.08207*, 2018. 16, 44, 45
- C. C. D. Silva, S. P. Beckman, S. Liu, and N. Bowler. Principal component analysis (pca) as a statistical tool for identifying key indicators of nuclear power plant cable insulation degradation. In *Proceedings of the 18th International Conference on Environmental Degradation of Materials in Nuclear Power Systems–Water Reactors*, pages 1227–1239. Springer, 2019. 37
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL <<http://jmlr.org/papers/v15/srivastava14a.html>>. 34
- C. Sun, L. Huang, and X. Qiu. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence. *arXiv preprint arXiv:1903.09588*, 2019a. 19
- F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. BERT4rec: Sequential recommendation with bidirectional encoder representations from transformer. 2019b. 46
- I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. 29
- Tan et al. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016. 46
- Toth et al. Predicting shopping behavior with mixture of rnns. 2017. 17

- A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. 34
- Y.-T. Wen, P.-W. Yeh, T.-H. Tsai, W.-C. Peng, and H.-H. Shuai. Customer purchase behavior prediction from payment datasets. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 628–636, 2018. 44
- L. Weng. Attention? attention! *lilianweng.github.io/lil-log*, 2018. URL <<http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>>. 30
- Wu et al. Neural modeling of buying behaviour for e-commerce from clicking patterns. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*. ACM, 2015. 45
- C. Wu, M. Yan, and L. Si. Session-aware information embedding for e-commerce product recommendation. pages 2379–2382, 2017. doi: <10.1145/3132847.3133163>. 46
- S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353, 2019. 46
- J. Yosinski et al. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. 26
- H. Yuan, G. Liu, H. Li, and L. Wang. Matching recommendations based on siamese network and metric learning. In *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*, pages 1–6. IEEE, 2018. 35
- Zeiler et al. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 26
- Y. Zhou, S. Mishra, J. Gligorijevic, T. Bhatia, and N. Bhamidipati. Understanding consumer journey using attention based recurrent neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3102–3111, 2019. 44