Content-based Recommender System for Detecting Complementary Products

Evaluating Siamese Neural Networks for Predicting Complementary Relationships among E-Commerce Products

MARINA ANGELOVSKA

Master in Data Science
Date: August 18, 2020
Supervisor: Sina Sheikholeslami
Examiner: Amir H. Payberah
School of Electrical Engineering and Computer Science
Host company: E-commerce platform
Company Supervisor: Bas Dunn
Swedish title: Innehållsbaserat rekommendationssystem för att upptäcka kompletterande produkter

Abstract

As much as the diverse and rich offer on e-commerce websites helps the users find what they need at one market place, the online catalogs are sometimes too overwhelming. *Recommender systems* play an important role in e-commerce websites as they improve the customer journey by helping the users find what they want at the right moment. These recommendations can be based on users' characteristics, demographics, purchase or session history.

In this thesis we focus on identifying *complementary relationship* between products in the case of the largest e-commerce company in the Netherlands. *Complementary products* are products that go well together, products that might be a necessity to the chosen product or simply a nice addition to it. At the company, there is big potential as complementary products increase the average purchase value and they exist for less than 20% of the whole catalog.

We propose a content-based recommender system for detecting complementary products, using a supervised deep learning approach that relies on *Siamese Neural Network (SNN)*.The purpose of this thesis is three-fold; Firstly, the main goal is to create a SNN model that will be able to predict complementary products for any given product based on the content. For this purpose, we implement and compare two different models: *Siamese Convolutional Neural Network* and *Siamese Long Short-Term Memory (LSTM) Recurrent Neural Network*. We feed these neural networks with pairs of products taken from the company, which are either complementary or non-complementary. Secondly, the basic assumption of our approach is that most of the important features for a product are included in its title, but we conduct experiments including the product description and brand as well. Lastly, we propose an extension of the SNN approach to handle millions of products in a matter of seconds.

As a result from the experiments, we conclude that Siamese LSTM can predict complementary products with highest accuracy of $\sim 85\%$. Our assumption that the title is the most valuable attribute was confirmed. In addition, transforming our solution to a K-nearest-neighbour problem in order to optimize it for millions of products gave promising results.

Keywords: Machine Learning, Deep Learning, Siamese Neural Networks, E-Commerce, Recommender Systems, Complementary Recommendations

Sammanfattning

Så mycket som det mångfaldiga och rika utbudet på e-handelswebbplatser hjälper användarna att hitta det de behöver på en marknadsplats, är onlinekatalogerna ibland för överväldigande. *Rekommendationssystem* en viktig roll på e-handelswebbplatser eftersom de förbättrar kundupplevelsen genom att hjälpa användarna att hitta vad de vill ha i rätt ögonblick. Dessa rekommendationer kan baseras på användarens egenskaper, demografi, inköps- eller sessionshistorik.

I denna avhandling fokuserar vi på att identifiera *komplementära förhållanden* mellan produkter för det största e-handelsföretaget i Nederländerna. Kompletterande produkter är produkter passar väl ihop, produkter som kan vara en nödvändighet för den valda produkten eller helt enkelt ett trevligt tillskott till den. På företaget finns det stor potential eftersom kompletterande produkter ökar det genomsnittliga inköpsvärdet och de tillhandahålls för mindre än 20% av hela katalogen.

Vi föreslår ett innehållsbaserat rekommendationssystem för att upptäcka kompletterande produkter, med en övervakad strategi för inlärning som bygger på *Siamese Neural Network (SNN)*. Syftet med denna avhandling är i tre steg; För det första är huvudmålet att skapa en SNN-modell som kan förutsäga kompletterande produkter för en given produkt baserat på innehållet. För detta ändamål implementerar och jämför vi två olika modeller: *Siamese Convolutional Neural Network* och *Siamese Long Short-Term Memory (LSTM) Recurrent Neural Network*. Vi matar in data i dessa neurala nätverk med par produkter hämtade från företaget, som antingen är komplementära eller icke-komplementära. Det andra grundläggande antagandet av vår metod att de flesta av de viktiga funktionerna för en produkt ingår i dess titel, men vi genomför också experiment inklusive produktbeskrivningen och varumärket. Slutligen föreslår vi en utvidgning av SNN-metoden för att hantera miljoner produkter på några sekunder.

Som ett resultat av eperimenten drar vi slutsatsen att Siamese LSTM kan förutsäga komplementära produkter med högsta noggrannhet på $\sim 85\%$. Vårt antagande att titeln är det mest värdefulla attributet bekräftades. Därtill är omvandling av vår lösning till ett K-närmaste grannproblem för att optimera den för miljontals produkter gav lovande resultat.

Acknowledgements

I would like to start by thanking my examiner, Asst. Prof. Amir H. Payberah for his expert feedback and support of my ideas. A great appreciation to my supervisor Sina Sheikholeslami for his enthusiastic encouragement and help during the whole process of this research work. His knowledge and guidance were of a great assistance starting from submitting my proposal until the final submission.

I am very grateful to my host company for providing me with a unique internship experience where I truly felt like part of the community. I extend sincere gratitude to my supervisor at the company, Bas Dunn for the unconditional support, constant feedback and guidance. His contribution goes beyond the work for my thesis as he has been selflessly sharing his knowledge with me throughout the whole internship. I also want to thank the amazing team I was privileged to be part of.

Big thanks to my family and friends for their unceasing support and enthusiasm. Last but not least, special thanks to my boyfriend Vilijan Monev who unreservedly believed in me. His ideas and interest in the field were of a valuable help during this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis discusses the design and implementation of a recommender system using Siamese Neural Networks for predicting the complementarity between any two given e-commerce products in the case of an e-commerce platform in the Netherlands. In this Chapter we will discuss the motivation for implementing such system, the context of the problem and its potential, the implemented approach, the research questions, the ethical and sustainability aspects of the approach and the general outline of the thesis.

1.1 Motivation

A very important part of the online platforms nowadays is the ease of finding relevant items and making decisions in the incredibly overwhelming catalogs that they offer. E-commerce platforms such as Amazon, movie platforms as Netflix, music apps as Spotify are a few examples of such online platforms where billions of users daily have to make a decision for what to purchase, watch or listen. Recommender systems play big role in making this process convenient and as effortless as possible for the users. Different recommendation techniques have been discovered and applied in various use cases. Amazon recommends products based on what the user previously purchased, viewed or rated, Netflix shows personalizied movies suggestions the user might like based on the movies watched before, and Facebook presents advertisements focusing on user's browsing history. Nowadays, recommender systems are becoming more and more attractive to the online platforms and the research area. Most of these recommender systems are focusing on the user's previous behaviour, choices, ratings and profile. The ultimate target of recommendations in the online world is increasing the profit and decreasing

the platform traffic by helping users find the items they like, eliminating the enormous amounts of items offered in the platforms.

In the Recommender Systems Textbook [1] there are four different operational and technical goals of recommender systems stated: *Relevance*, *Novelty*, *Serendipity* and *Increasing recommendation diversity*. *Relevance* refers to the fact that users are more likely to consume items they find interesting. In addition, *Novelty* suggests that users find it very helpful when the system offers them something, which they did not think of. *Serendipity* is different from novelty in the way that the recommendations are truly surprising to the user, rather than simply something they did not know about before [1]. Finally, the key in *Increasing recommendation diversity* is that the system should not offer top K items, which are similar to each other as it increases the risk that the user will not decide on selecting either of them but instead offer diverse products. Now, having in mind these four general goals of recommender systems, companies are designing their offers and suggestions differently for different parts of the platform.

A specific case of recommender systems that existed long time before the e-commerce era are complementary products. According to Cambridge Dictionary [2], complementary products are products that are sold separately but that are used together, each creating a demand for the other. It is worth to mention that complementary products are different from *substitutable* products. For instance, when looking to buy a smart phone, the user might be offered an alternative - similar product having similar specifications from a different brand. On the other hand, the products that are offered once the user decides to buy the specific product are called complementary products. In the example of buying a smart phone, complementary products could be a screen protector, a case, earphones, wireless charger, etc.

When going to a traditional retail store, shoe sellers offer sprays for maintaining the quality of the shoes, which are usually placed near the cashier. In fact, this has been the oldest retail trick in the book, which enables competitive prices for the main products, but have high margins on the complementary products. The exact place and timing of the complementary products offer in retail have big importance as this have been associated with impulse buying. Once the customer decides do buy something more expensive, he/she is more likely to spend a little extra money on a product that goes well together with it and the complementary product might feel like a necessity at the moment of purchasing.

When speaking of recommendations on e-commerce platforms, despite the frequently bought together items and previously viewed sections, we can

clearly see the existence and need of complementary products. Taking the example of traditional retail, online retails often offer the complementary products once the user decides to buy the specific item.

1.2 Problem Statement

This thesis' focus will be on recommdender systems in the online retail industry in the case of the largest e-commerce company in the Netherlands. Due to the company's policy, we do not reveal the company's name in this thesis but we will refer to it as "the company" or "the e-commerce platform".

The company serves more than 2.5 million visits every day and has over 20 million products in the online store. Recommendations play an important and big part at the company as they prevent users from being overwhelmed by the big offer while searching for products in the online catalog. Detecting complementarity in an online catalog consisting of millions of products is very different than detecting it in retails with specified domain of products. Some of the complementary products might not be as obvious as the ones in the example of the smart phone. An interesting scenario would be to suggest mosquito spray if someone is purchasing a book for travelling to the forests in Africa. This task of detecting complementarity among products has been explored in recent years but there is still a lot of room for experiments and improvements.

Currently, the company offers different recommendations in multiple screens across the website that rely on user's search history and frequently bought together items. All of these existing recommender systems at the company are used as up-selling methods, which help the customers find products they might need quicker. However, there is a need for improving the recommender systems, which present complementary products to the user. The complementary products, also known as *add-ons* in the online world, usually appear once the users make the decision to put the product in their basket.

Figure 1.1 shows how product add-ons appear once clicked on the `"Add to basket"` button. As seen on Figure 1.1, such recommender systems exist at the company. For each product in the database, there is a list of complementary products referencing to other currently offered products. Figure 1.2 shows how this system approximately looks like.

The problem with the current way of handling complementary products' recommenders at the company is that it is done manually, thus it takes a lot of time and requires human assistance. It is done firstly by querying the items that were bought together more than a few times. Then, human validation is

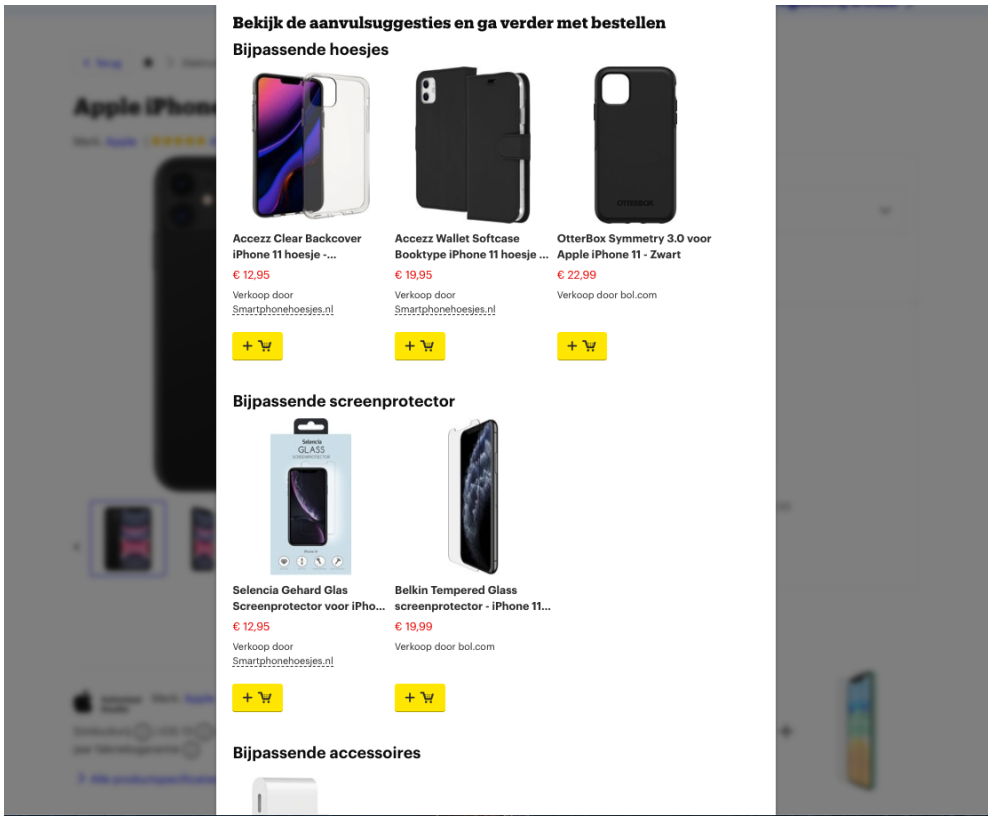


Figure 1.1: Example of a product page at the company's website showing add-on suggestions for an Apple iPhone.

performed as it is not necessarily true that all items that were bought together are complementary to one another. In addition to the time requirements, the main limitation of such approach is its focus on already popular and frequently bought items. In relation to the work of this thesis, the currently used method of manually querying and validating complementary pairs of products gives us a "perfect" base of a labelled dataset taking into account that it was done by industry experts within the company.

Potential

The aforementioned process is done for less than 20% of the 24 millions products offered at the moment. The success of the company is measured using different metrics, including the average purchase value. The average purchase value is the amount of money that a user spends on one single purchase. Based on analytics inside the company, the orders with an add-on page in the journey

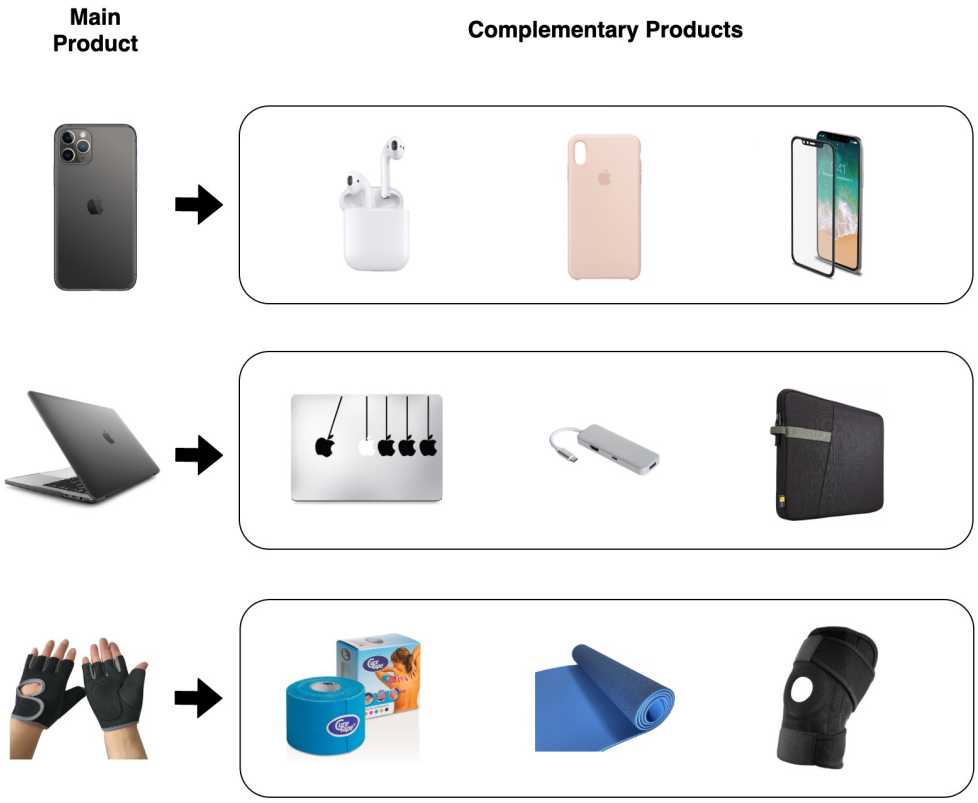


Figure 1.2: Complementary product examples that currently exist at the company.

have increased purchased items per order using the current system of generating complementary recommendations whose focus is mainly on popular items. Nevertheless, these facts prove that there is a big need and potential in implementing a smart model, which will automatically generate the add-ons for all of the products.

At this point it is worth to mention that at the company only the most popular and frequently bought items have add-ons, meaning that the items, which do not have many purchases will stay undiscovered. Easier customer journey and higher items per order (thus, increased average purchase value) are the company's primary goals for automating the complementary products' suggestions.

1.3 Approach

Predicting complementarity is a complex task and it is not as straight forward as detecting similarity. It is not enough that two products were viewed or purchased together to be able to detect their complementarity. Thus, we need to have clear data input, good model and distance measure to predict which products fulfil the conditions of being complementary to a given product.

To address the above complexities, challenges and potential of recommending complementary products, in this thesis we propose a supervised deep learning approach using *Siamese Neural Networks (SNN)*. SNNs are widely used for comparing two inputs, hence, it is considered to be a suitable approach for experimenting with e-commerce products. As SNN is a concept for using neural networks with multiple inputs, we can use any type of neural network in the model architecture. In this thesis the focus is on Siamese Convolutional Neural Network (CNN) and Siamese Long Short-Term Memory (LSTM) Recurrent Neural Network.

The pipeline of the model is shown on Figure 1.3. We use manually labelled data from the company in the format `MainProductID`, `AddOnProductID`, `Label(Y/N)` where different product attributes are extracted for each pair such as the product title, description, brand, sub-category, etc. This data is extracted from the company's data warehouse using Google BigQuery. These product attributes are then used in the neural networks for creating embeddings and getting their vector outputs. The final part is the actual distance between the two given products, which gives us the final result whether the two products are complementary or not.

A rather challenging part of this pipeline is making it scalable so that this is a suitable approach for the millions offered products at the company as well as for determining the model hyperparameters and functions. In the case of the company with a catalog of more than $20M$ products, typical, single neural networks will be traversed $(20M)^2$ times in order to get the complementarity score for each possible pair in the catalog. On the other hand, Siamese architecture will traverse the neural networks for each product only once, which sums up to $20M$ runs as each product is treated separately (not in combination with its pair product). Then, the last part for calculating the similarity distance for getting the complementarity score will be done $(20M)^2$ times for each pair of products. One can conclude that performing similarity score is much faster and scalable compared to traversing the whole neural network as many times.

The proposed model is expected to give us good results because the pipeline's main functionality is to extract features from two given items, thus match the

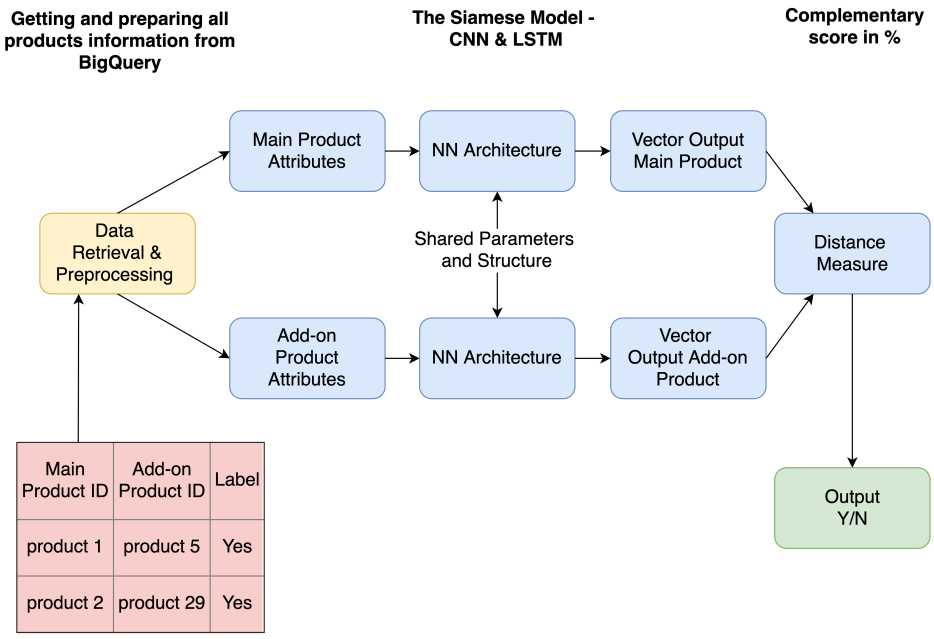


Figure 1.3: The proposed model pipeline using SNN.

items based on their content complementarity. Expert knowledge will not be required in the proposed model as the labelled data was built by experts in the field in combination with what customers purchase together. By taking this dataset, we will be able to learn what makes two products complementary and train the model to detect this complementarity on future data inputs for all items, including non-frequently bought (unpopular) products.

In Figure 1.4 an overview of the report steps is represented. The first area (yellow part) is representing the *Data Preparation*, the red area is part of the *Data Preparation* but it represents the *Storage* (local and on cloud) and lastly, the blue area represents the main part of the thesis, which is the *Model implementations and Comparison*. In the first part of this thesis work, we start with gathering the data from BigQuery, which is the data warehouse that the company uses. We need four different databases from the company's cloud warehouse, therefore, the next step is to combine the data sources into a database containing all information that we need for the supervised learning approach.

Initially, we only have positive labels, hence the third step from this part is to generate negative samples, which will indicate that the pair of products is not complementary. Lastly, after we do data cleaning, the final dataset will be ready to be used in the models, which will be designed in the second part

marked with blue background. The model implementation part will start with splitting the data into training and testing. As we are dealing with textual data, embeddings for the input data will be calculated before we give the data as an input to the models. We will discuss the reasons and need for this later on in Section 3.4. Implementation of Siamese CNN and Siamese LSTM neural networks will follow, as well as their comparison during a few different experiments. Once we get the model architecture that gives higher accuracy, hence is more promising, we will continue using only that model in the further experiments. We will conduct multiple experiments testing the proposed model in this thesis in comparison to other baselines, as well as experiments showing how including multiple product attributes can affect the model performance. Last but not least, the proposed Siamese approach will be transformed into a scalable solution such that it can handle big data scenarios. All of the mentioned steps in Figure 1.4 will be explained in details during this report, with a big focus in Section 3.4. Big part of the source code including the model implementation, architecture and experiments is available on GitHub [1].

1.4 Research Question

Many e-commerce websites are offering complementary products in some form [3, 4, 5], and this has been the subject of many research efforts [6, 7, 8, 9, 10, 11, 12, 13, 14]. Therefore, it is an interesting topic to explore from both, practical (business) and theoretical aspect. There have been different approaches using recommender systems for detecting complementary products, which will be described in details in Chapter 2.5. The general research question in this thesis is as follows:

General Research Question. *How can we use Deep Learning to improve the process of detecting complementary products based on content having the end goal of increasing the average purchase value on the e-commerce platform?*

So, the main approach is to use a *Supervised Complementary Recommender* using deep learning models. This general research question tackles the main goal of this thesis, which is designing a model, which will give us the complementarity score for any given two products from the online catalog. Furthermore, we are interested in the performance of the chosen model as well as the features included in the model. Therefore, there are a few more concrete and technical sub-questions, which this thesis will answer:

[1] https://github.com/marinaangelovska/complementary_products_suggestions

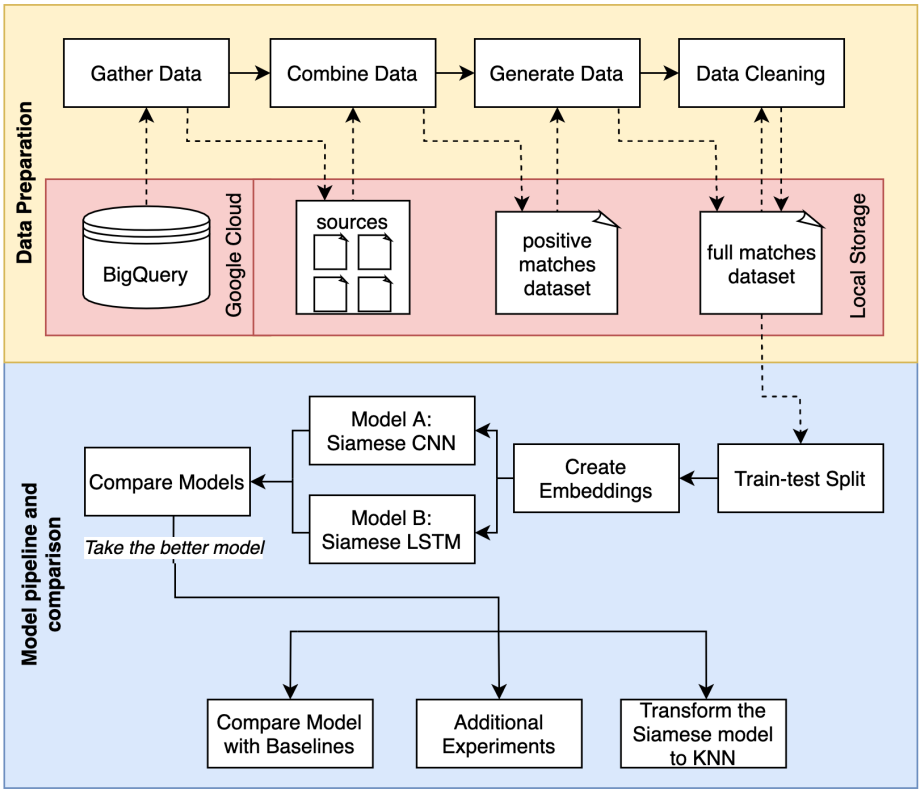


Figure 1.4: An overview of the steps taken in this research including the needed data sources for each step.

Research Sub-question 1. *Which of the proposed two models - Siamese LSTM and Siamese CNN predicts complementarity among products with higher accuracy based on the content?*

Research Sub-question 2. *Is the title most valuable attribute for content-based complementary recommender systems?*

Research Sub-question 3. *Can the proposed SNN be transformed in such way that it can handle millions of pairs of products in a timely manner? How well does it perform compared to the manual (human) pipeline?*

By answering the general research question, we will be able to answer all of the aforementioned sub-questions. In addition, by answering those sub-questions, we will be able to learn much more about products' complementarity at the company, which might open different discussions. These questions will not only help us improve the company's offer and average purchase

value, but also the results and conclusion from these questions would benefit future research in the field of recommender systems, similarity/complementarity measures as well as any scientific work related to the e-commerce world.

1.5 Ethics and Sustainability

The implementation of a machine learning system that will generate complementary products for a chosen product is fully transparent. More precisely, our system will not make use of any personal information about the user as the add-ons suggestions will be identical for every user. This means that the products suggested to the user only depend on the user's choice of a product to purchase, but it does not depend on the user's profile, demographics nor characteristics. This thesis uses the labelled dataset consisting of product pairs, which were paired with the help of anonymized data for frequently bought together products at the company.

The aim of this work and experiments is to improve the way people are purchasing products online. It is not in any way stimulating unwanted user behaviour as it only suggests products that go well together, products that are necessary for each other (e.g. batteries for an electric product) and products that the user most likely needs. One of the company's main objectives for providing such recommender systems is to save customer's time, thus improve their journey and reduce website traffic. By purchasing two or more items at a time instead of ordering them separately over a period of time, the environmental and economical costs for delivery are reduced, providing a more sustainable offering. In general, at the company all products that are violating the sustainability rules are removed and banned from the platform. Such examples could be products made out of animal ingredients, products tested on animals, plastic non-reusable cutlery, etc.

1.6 Report Structure

This thesis is organized in the following way. Chapter 2 consists of the background knowledge the reader needs in order to understand the work done during this thesis. More precisely, it includes theoretical explanation for Machine Learning concepts and SNNs. It also includes information about the metrics, platforms and frameworks used in the experiments. In the same Chapter we present the related work. Chapter 3 presents the goals of the thesis, the data (including the data retrieval and data generation) and the methodology, which is

the implementation of the proposed system. Furthermore, Chapter 4 includes the results from the experiments together with a discussion. We then conclude this thesis with Chapter 5, where we give concluding remarks including the final conclusion of our work and directions for further research.

Chapter 2

Background

Recommender systems for complementary products are present at online retailers such as Amazon, AliBaba, Zalando, Netflix, etc. and these recommender systems are the basis and inspiration for a lot of research focusing on different approaches including collaborative filtering, product embeddings, neural networks and frequency co-counting. This thesis will use the knowledge from the academic world based on previously done work in the field of recommender systems and the gathered knowledge from the company. The main idea of this thesis is to use Machine Learning (ML) and Deep Learning (DL) techniques to tackle the challenging problem of detecting complementarity in comparison of the majority of related work, which do not use ML techniques. In addition, we want to compare the performance of two different techniques. Furthermore, we want to design a complex model, which could potentially be used for solving different tasks. Therefore, we are interested in proposing a solution, which can be reusable and scalable. In addition, the models, experiments and results during this process would benefit the research field to (dis)prove whether ML and DL can solve such problems with high efficiency.

In this section, we will firstly explain the main idea behind *ML and DL*, their usage and benefits over traditional recommender systems' approaches. In addition, *Convolutional Neural Networks*, *Recurrent Neural Networks* and *SNNs* as specific type of neural networks will be introduced and explained as they are the core idea behind our proposed solution. The next section Platforms and Frameworks will define the libraries and platforms used during this thesis, focusing on the usage and benefits of Keras and TensorFlow. In the last subsection of this part of the thesis we will focus on the related work within the field of finding complementary and/or similar e-commerce products as part of

recommender systems.

2.1 Machine Learning and Deep Learning

ML is a branch of Artificial Intelligence (AI) and it is a scientific study of algorithms and models for completing a specific task without using explicit instructions by relying on patterns and inference. ML enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings [15]. There have been multiple definitions and explanations of what precisely ML is, thus a more concise definition would be the one from the Machine Learning book by Mitchell [16], which is as follows: "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."

Learning Approaches and Data Splitting

Within the field of ML we distinguish two main types: *supervised* and *unsupervised* learning. The difference between the two approaches is that in supervised learning we are using a ground truth, meaning we have prior knowledge about the samples whereas in unsupervised learning we do not know anything about what the output values for our samples should look like. Moreover, in supervised learning we know the labels of the samples, thus, we are trying to teach the model to predict those values based on the truth that we feed the model with such as in the case of regression and classification. Common algorithms used in supervised learning tasks include Logistic Regression, Naive Bayes, Support Vector Machines, Artificial Neural Networks and Random Forest. The problem we are trying to solve in this thesis is a supervised learning task as we are trying to predict the complementarity of the products based on previous knowledge. On the other hand, unsupervised learning, does not have labeled outputs, so its goal is to deduce the natural structure given in the training data. Some of the most common tasks with unsupervised learning are clustering, representation learning and density estimation. It is mainly used in exploratory analysis.

When we are dealing with labelled data (hence, supervised learning), we split the data into *training*, *validation* and *testing* set. The training set is used as the input data during the learning process of the model. In the case of a neural network, the weights and biases will be updated using the training set. The validation set is used to evaluate the given model in the process of training. It

is a small portion of the training set, usually 10%, which is used to provide an unbiased evaluation of a model fit on the training data while tuning the model hyperparameters. This means that the model occasionally sees this data, but never does the learning using it. The goal of a ML task is to train a model that will give good results on unseen data, thus evaluating the final performance of the model using the training set, the data that was used to improve the model is called *overfitting*. Overfitting means that the model will give great results on the train or validation data, but performs very poorly on new, unseen data. For this purpose, we introduce the test set, which provides unbiased evaluation of the final model fit on the training set. It is a good standard to evaluate the model and it is used only once the model is completely trained.

These three datasets are formed by splitting the original dataset at the beginning into three parts having different sizes. The actual size of the split depends on the problem, but usually it is done in the proportion 80:10:10 for the training, validation and test set.

Artificial Neural Networks

DL is a ML technique based on Artificial Neural Networks (ANN) with representation learning. It teaches computer systems to learn by example, something that comes naturally to humans. In fact, as their name suggests, artificial *neural* networks are biologically inspired computer programs designed to simulate the way in which real human brain gains knowledge by detecting patterns in the surroundings and learning through experience. Neural network consists of input and output layers, as well as hidden layer(s) consisting of units that transform the given input into something that can be used in the output layer. A sketch of an ANN is shown on Figure 2.1. An ANN contains multiple layers, which are formed from hundreds of single units or artificial neurons connected with weights. Each of these neurons has weighted inputs, transfer function and one output.

On Figure 2.2 the detailed flow of the neural network is presented. Basically, as an input enters the node, it gets multiplied by a weight value and the resulting output is either observed, or passed to the next layer in the neural network. It is common that the weights of the neural network are part of the hidden layers. On Figure 2.2 we can also see the bias term, which represents how far off the predictions are from their intended value. The activation functions within each layer are of a crucial importance for the ANN to learn and make sense of something complicated, which has non-linear properties. There are different activation functions. Therefore, there are many things to

take into account when choosing the right one for the model and task. The simplest ANN architecture is the one-layer perceptron, but adding more layers brings the capability of solving more complex tasks.

In fact, neural networks (perceptrons) have been around since the $1940s$ but they became a major part in AI only in the last decades [17]. This is due to the *backpropagation* technique, which enables networks to learn and adjust their hidden layers of neurons when their output does not match the real (true) value. The process of an ANN in a supervised task (a task where we know what is the true value of the sample) starts with comparing the network's produced output with the actual (desired, expected) output. The learning actually happens by trying to lower this difference between the actual and produced output and this is done using the backpropagation algorithm. In simple words, the network works backward going from the output units to the input unis to adjust the weights associated with each neuron until the difference between the actual and predicted outcome produces the lowest possible error or until some stopping criteria is reached. This process of adjusting the weights and the other hyperparameters of the model in order to have the lowest possible error rate is called *learning process*.

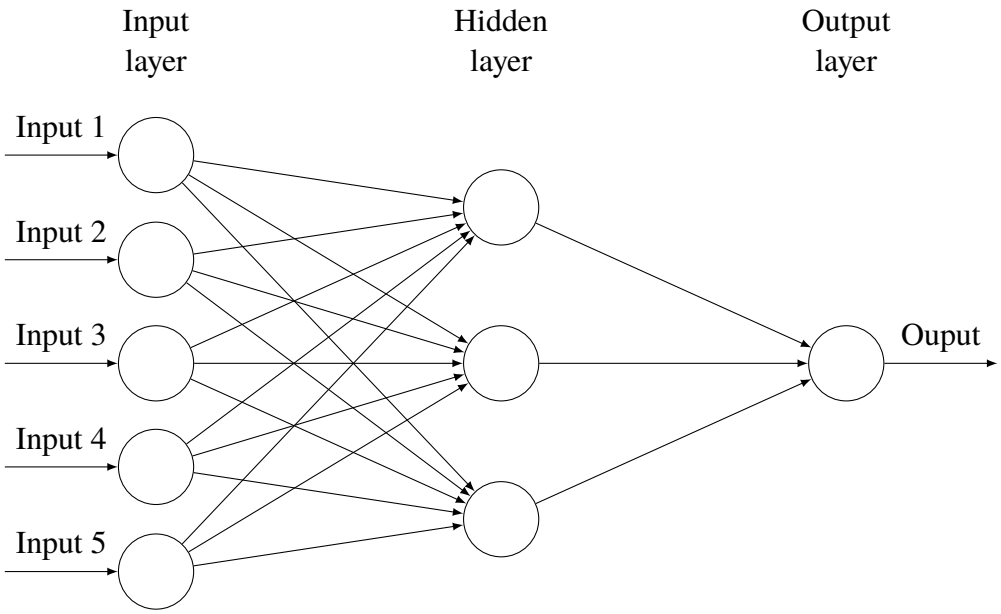


Figure 2.1: Feedforwaed ANN with one hidden layer.

Nowadays, DL is becoming more and more popular as we have enormous

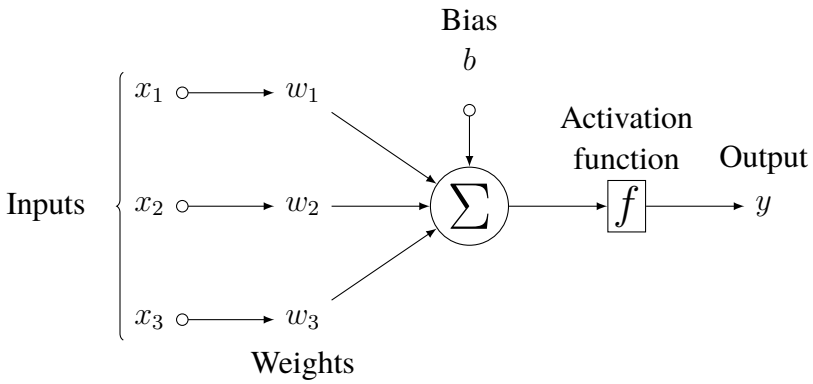


Figure 2.2: Detailed concept of a single neuron in an ANN.

amounts of data and larger neural networks to train the data on. Moreover, the performance of ANNs grow as they are bigger, meaning that the networks have more hidden layers, more connections and more neurons. There are different types of neural networks, which perform better in different scenarios. They all have their strengths and weaknesses, and one need to fully understand their internal structure to know which would fit a specific task the best. In the following part, we will explain two complex ANN structures, which this thesis will be focusing on: *Convolutional Neural Networks (CNN)* and *Long Short-Term Memory Networks (LSTM)*.

Convolutional Neural Networks

CNN is a deep feedforward (not-recurrent) neural network containing one or more *Convolutional Layers*. In recent years, CNNs have achieved state-of-the-art results in isolated character recognition [18, 19], large-scale image recognition [20, 21], text-classification [22, 23] and modeling sentences [24]. CNNs require very little preprocessing compared to other classification algorithms, they have the ability to learn a lot of characteristics and are easily applicable to any language. The name "Convolutional Neural Network" indicates that the network applies a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are neural networks that use convolution in the place of general matrix multiplication in at least one of their layers. Due to the convolutional operation, the CNN can be much deeper than standard feed-forward neural networks but with less parameters. The convolutional layers can be fully connected or pooled [15]. The objective of the convolution operation is to extract high-level features. Similar to the convolutional layer, the *Poling Layer* of a CNNs is responsible for re-

ducing the size of the feature map, thus it decreases the computational power needed for processing the data through dimensionality reduction. In addition, dominant features can be extracted using the pooling layer. Furthermore, the *Fully Connected Layer* is responsible for learning the non-linear combinations of the high-level features. At the end of a CNN, the classification is achieved by a dense layer having *softmax* or *sigmoid* activation function. CNNs have shown tremendous success in their application in video and image recognition, natural language processing tasks and recommender systems. Figure 2.3 presents how a CNN for image classification looks like.

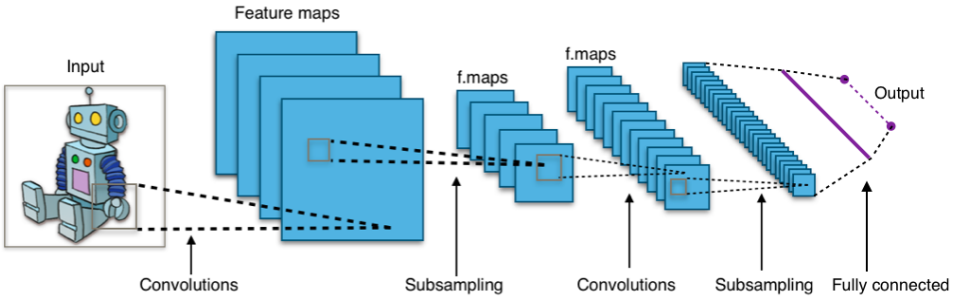


Figure 2.3: Example of a CNN architecture presenting some of the main layers. Image by Aphex34 [1].

Long Short-Term Memory Networks

Recurrent Neural Networks (RNNs) are a family of neural networks for processing sequential data. Much as a CNN is a neural network that is specialized for processing a grid of values X such as an image, a recurrent neural network is a neural network that is specialized for processing a sequence of values $x^1 ... X^t$ [15]. In RNNs, the output of one layer is saved and fed to the input of the following layer. Therefore, from one time-stamp to another, each node remembers some information that it had in the previous time-stamp. These kinds of neural networks capture a challenging design, which overcomes traditional neural networks' limitations, which appear when dealing with sequential data, time series, videos, etc. The RNN architecture is shown on Figure 2.4.

Despite their benefits, there are some disadvantages such as the gradient vanishing and exploding problem when using *Vanilla RNNs*. Gradient vanishing problem can happen when the input sequences are very long. Moreover, the gradients carry information used in the RNN parameter update and when

[1] Aphex34 - Own work, https://commons.wikimedia.org/w/index.php?curid=45679374

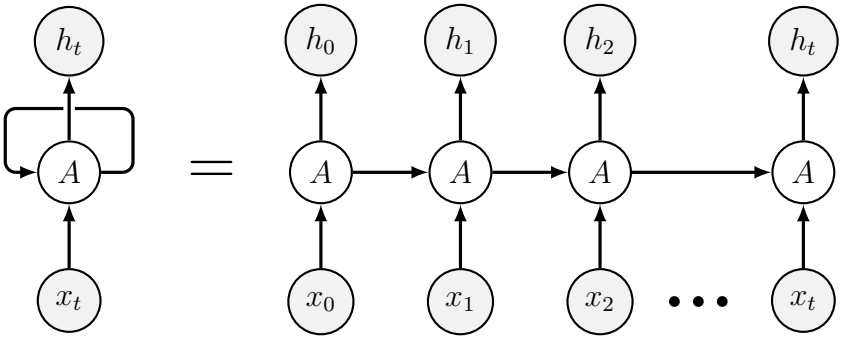


Figure 2.4: Example of a RNN architecture presenting how each output from the timestamp $t-1$ is passed to the following timestamp t together with the current input x_t.

the gradient becomes smaller and smaller by going deeper in the network, the parameter updates become insignificant, which means no real learning is done.

LSTM [25] is a type of RNN architecture, which is capable of maintaining information for long periods of time and has better control of the flow in general, thus it solves the vanishing gradient problem. LSTM implements three types of gates: *input*, *forget* and *output* gate. The input gate discovers which values from the input should be used by using the *sigmoid* and *tanh* activation functions. The forget gate determines what should be eliminated from the block by using the *sigmoid* function. Finally, the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The exact difference between the structure of the cell in an LSTM compared to a Vanilla RNN is shown on Figure 2.5.

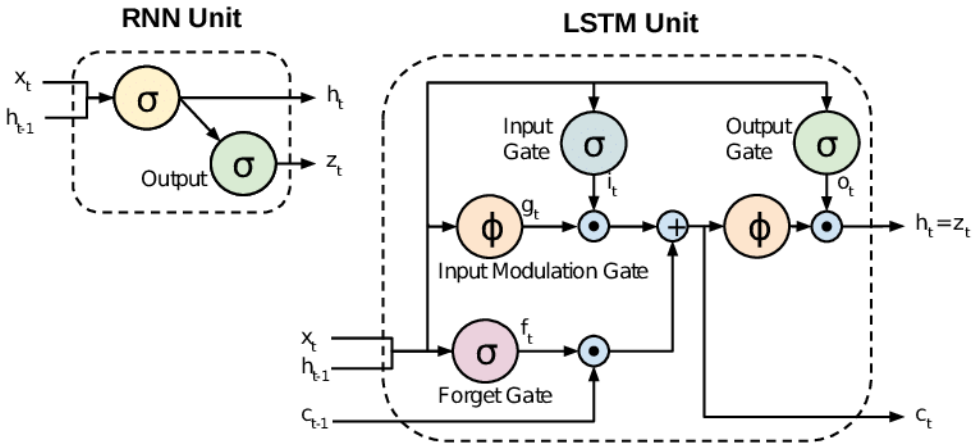


Figure 2.5: The difference between RNN (left) and LSTM (right) cell [26].

2.2 Siamese Neural Networks

SNN, which is basically a twin neural network, is an ANN composed of two separate neural networks sharing the same architecture and the same weights. In other words, the SNN is a neural network architecture capable of learning similarity knowledge between cases in a case base by receiving pairs of cases and analysing the differences between their features to map them to a multidimensional feature space [27]. The interesting part of having such approach is that the idea behind SNN explains the part that the two neural networks work in tandem but there is no limitation on the actual architecture of the neural network used. By receiving two different inputs on the train and test level, the main goal of such network is to develop similarity knowledge between the two produced outputs. Figure 2.6 shows an overview of a SNN. The outcomes of the two identical networks are the feature vector outputs for each of the inputs.

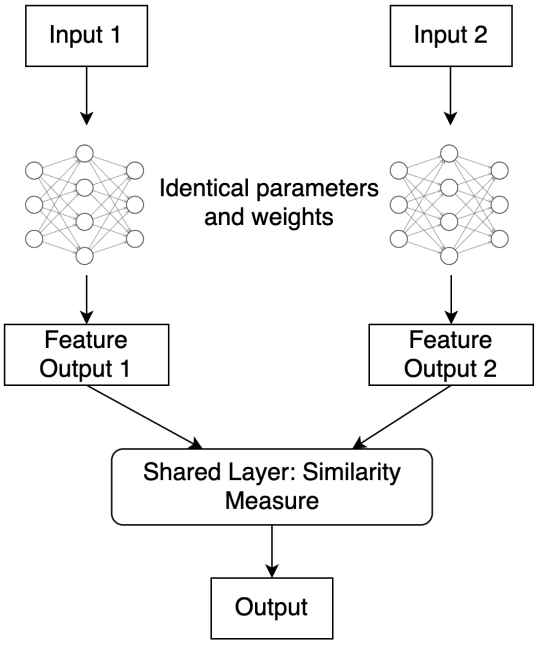


Figure 2.6: SNN architecture.

In the case of binary classification as in the example of this thesis, *Binary Crossentropy* is considered as the most suitable loss function as the goal is to see how far the predicted value is from the original value. Binary Crossentropy measures how far away from the true value (which is either 0 or 1) the

prediction is for each of the classes. Then, it averages these class-wise errors to obtain the final loss.

More About SNNs and their Advantages

Up until now, SNNs have been widely used for *One-Shot Classification*. In comparison to standard classification where the input image is fed to the neural network and the network outputs the probability of the image belonging to each of the classes, one-shot classification takes only two images as an input and outputs the similarity probability, thus classifying if they are the same or not. In the former approach, we need to have a lot of training instances as one can assume. The advantage of using a SNN architecture is that it does not require a lot of training examples. In fact, we only need one item per class in order to train the network properly.

One such example can be the task of face recognition in a private part of a company. Now, to achieve this we only need one image from each person having access to that part of the company. In order to detect whether a person is allowed to enter the building or not, the system requires only one shot of the person entering. Then, the SNN compares each picture form the database with the taken shot and determines whether that person has access or not based on the similarity measure (in other words, if that person is recognized in another image).

However, the power and advantages of SNN can be applied beyond image classification. Recently, SNNs have been used in Natural Language Processing (NLP) tasks as well. In fact, a lot of classification tasks can be represented in a siamese-like architecture if the problem definition allows so. Such examples include detecting if two sentences have the same semantic meaning including questions, titles, book summaries, etc. In relation to our specific recommender systems problem in this thesis, the images and sentiment analysis from the previous examples can be replaced by e-commerce products, while the problem-solution fit remains the same.

There is no clear distinction between the different types of SNNs as it is mainly a concept of how any neural network can be used in a pair of its copy for given two separate inputs. But in their book, Fiaz, Mahmood, and Jung [28] categorize SNN in three groups based on the position/time of merging the layers: *late*, *intermediate* and *early* merge, which are shown on Figure 2.7. To start with, late merge is the architecture shown on Figure 2.6, which basically shows that the output vectors of each network are merged at the last dense layer. Furthermore, the intermediate merge suggests that the outputs

of the two networks are merged in the middle of the network and processed together as one output in the last layers, which could be of any type (not just a fully connected layer). Lastly, as the name suggests, the early merge type SNN merge the two inputs right before the actual network, thus resulting in a single-like neural network architecture.

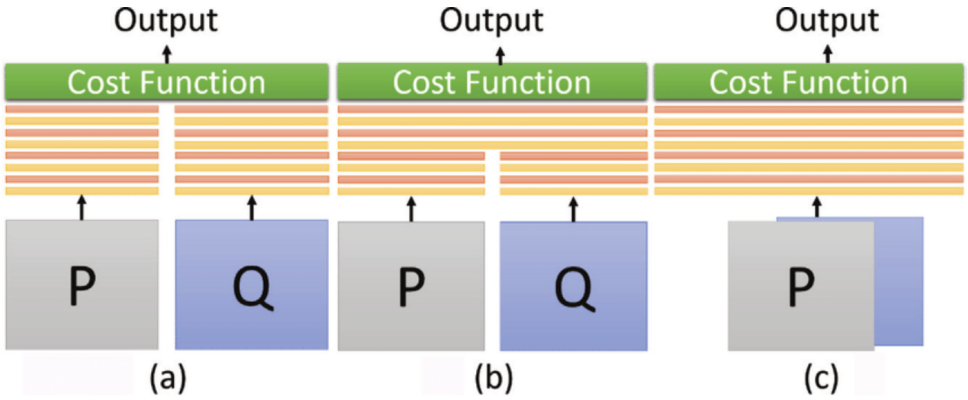


Figure 2.7: Types of Siamese networks (a) Late merge, (b) Intermediate merge and (c) Early merge [28].

In the use-case of dealing with millions of data points, as in the case of the company's product catalog and this thesis, we consider developing a network that will have high efficiency (performance) and very importantly, be able to scale up for millions of product pairs at the same time. In this thesis we focus on SNN architecture having intermediate or late merge characteristics. Moreover, by doing so, the model process each input (pairs of products) to the network and compares their complementarity at the end. Such architecture is the advantage over traditional single neural networks where each pair of two products needs to be processed separately for all of the millions of possible pairs available. In fact, by training the same model architecture for both, a Siamese model and a traditional model of the same type, we expect to obtain the same results. The main advantage is the scalablity and ability of the SNN to process each item in the network once and then compute the similarity/compatibility score for each of the pairs, which has lower complexity compared to training the whole model for each pair. As suggested by Zhao et al. [9] in their Deep Style for Complementary Products paper, we seek items whose representations are close to the main product representation under the linear kernel distance, thus transforming the problem into a K nearest neighbours problem.

2.3 Result Metrics

For the purpose of analysis and comparing the performance of the two proposed models as well as comparing their performance with baselines models, we introduce a few different metrics:

- **Confusion matrix** is a performance measurement for classifications tasks where the output can be two or more classes. Actually it is a table having four different combinations of predicted and actual values. Figure 2.8 shows how a confusion matrix looks like. *True Positive (TP)* value represents the number of correctly predicted values for the positive class (1, which in our case is the case where the model says that the pair is an add-on when it really is). *False Positive (FP)* is the case where the model predicts that the class has label 1 when in fact the real label is 0. We have *False Negative (FN)* if the model predicts 0 but the actual label is 1. Lastly, *True Negative (TN)* shows the number of correctly predicted cases when they belong to the negative 0 class.

  The reason why we calculate these occurrences is for calculating *Recall* and *Precision*. Recall represents how many labels the model predicted correctly out of all positive classes. It is also known as *True Positive Rate (TPR)*. Mathematically it is calculated as:

  $$Recall(TPR) = \frac{TP}{TP + FN}$$

  Precision on the other hand represents how many are actually positive out of all the positive classes the model predicted correctly:

  $$Precision = \frac{TP}{TP + FP}$$

  Another very important term coming from the confusion matrix is the *False Positive Rate (FPR)* defined as

  $$FPR = \frac{FP}{TN + FP}$$

  It corresponds to the proportion of negative data points that are mistakenly considered as positive with respect to all negative data points. Generally, in the case of our scenario, we are trying to maximize recall.

- **Accuracy** shows out of all the classes, how many we predicted correctly, hence we want to maximize it. This metric works well if we have a balanced dataset. The values the accuracy can have are are in the range $[0, 1]$.

	Class 0 Predicted	Class 1 Predicted
Class 0 Actual	**TN**	**FP**
Class 1 Actual	**FN**	**TP**

Figure 2.8: Confusion Matrix.

- **AUC-ROC Curve** is a very widely used metric for models evaluation. We are using this as it is used for binary classification problems and it shows the probability that the model will rank a randomly chosen positive example higher than a randomly chosen negative example. It can have values in the range $[0, 1]$. For the purpose of visualizing the problem we are using the aforementioned terms: TPR and FPR. The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis as shown on Figure 2.9.

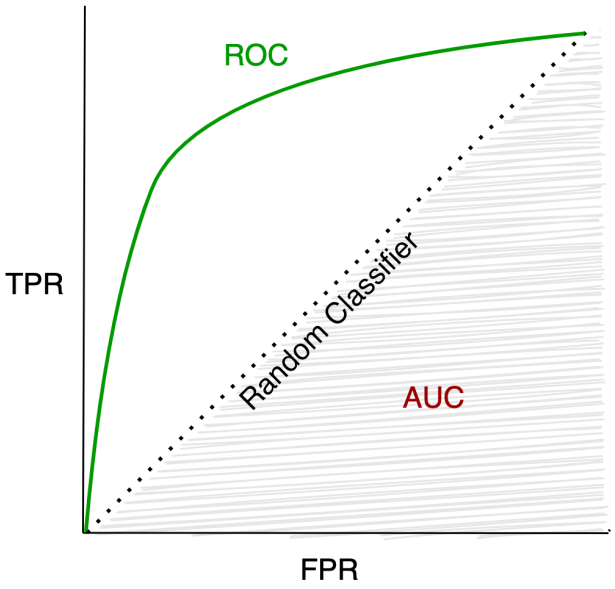


Figure 2.9: Example of AUC-ROC Curve where False Positive Rate (FPR) is shown on the x-axis and True Positive Rate (TPR) on the y-axis.

This shows how much the model is capable of distinguishing between

the two classes. The higher the ROC curve is, the better the model is as predicting. The worst scenario is when the curve is overlapping with the dotted line on Figure 2.9, which means that the model cannot distinguish the two classes at all.

2.4 Platforms and Frameworks

As this thesis focuses on ML and DL solution, we are using open-source APIs for looking into big amounts of data and developing ANN architectures. We use Python (version 3.7.4) as a programming language together with multiple libraries and frameworks such as *tensorflow*, *keras*, *sklearn*, *pandas*, *nltk*, *matploit* and many more. Therefore, in this section Google BigQuery, Keras and TensorFlow will be briefly explained as they are the main pillars of this thesis' development.

2.4.1 Google BigQuery

Google BigQuery [29] is a serverless, highly scalable and cost-effective cloud data warehouse, which works in conjuction with Google Cloud Storage. The company is using Google Cloud services, where different IT teams are updating and analyzing different datasets. This means that there are teams who make sure that the data in Google Cloud is always up-to-date and reliable. Therefore, as the company is using Google Cloud services, using BigQuery during this thesis enabled quick understanding of the available data as well as efficient data retrieval and analysis. BigQuery enables querying millions of rows in a matter of a few minutes using SQL queries.

2.4.2 Keras and TensorFlow

Keras [30] is a high-level open-source library for implementation of neural networks. It can be run on top of different platforms such as TensorFlow [31], CNTK [32] or Theano [33]. By using Keras, we are able to easily prototype and change the network architecture within a few lines of code. It is user-friendly, modular and extensible. Another advantage of using Keras in this thesis is that it includes different layers, which helps for the creation of different types of ANN, such as the ones mentioned in Section 2.1.

As previously mentioned, Keras can be run on different backends including *TensorFlow*, which we use in this thesis. TensorFlow [31, 34] is an end-to-end open-source platform for machine learning. It enables developers to

easily develop and scale ML-powered applications. TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research [31].

2.5 Related Work

As previously mentioned there are different approaches to measure similarity and complementarity among two or multiple items. In general, the most obvious distinction between the methods is splitting them into two groups: *unsupervised* and *supervised* learning approaches.

At the beginning of their existence, complementary recommenders mainly relied on unsupervised learning. These solutions focused on finding complementarity between products based on their co-purchase history. One of the most common methods using unsupervised learning is the *Frequent Pattern (FP) Growth* [35] algorithm, which has been widely used in recommenders' tasks. Basically, the assumption behind such model is that if two items were bought together more than n times, there is a high probability that those items are complementary to one another. *FP Growth* algorithm is based on the FP tree, making it possible to efficiently search for all frequent patterns in the purchase history of all users.

Other groups of research focus on using the paradigm of *Word2Vec* [36] from NLP field in the case of unsupervised learning. Basically, these models are taking into account the sequence of previously searched or purchased items and are predicting the following items that are most likely to be bought. In the *Prod2Vec* paper, Grbovic et al. [37] propose a *Word2Vec* model, which learns product representations from sequences of retrieved past orders. The *Prod2Vec* model involves learning vector representations of products from e-mail receipt logs by using a notion of a purchase sequence as a "sentence" and products within the sequence as "words", borrowing the terminology from the NLP domain [37]. The *Meta-Prod2Vec* paper by Vasile, Smirnova, and Conneau [38] extends the *Prod2Vec* model by taking into account additional side information in the input and output space of the Neural Network. The *Meta-Prod2Vec* model outperforms the standard *Prod2Vec* model, especially when combined with the standard *Collaborative Filtering* approach.

The main problem with models such as the aforementioned is the possible lack of ground truth, known as the cold-start problem in the case that there were no purchases done yet. Complementarity among products cannot be ac-

curately detected based only on purchase history due to the noise introduced. For instance, identical items having different sizes or colors are likely to be bought together, and are pure substitutes instead of complementary products. Improvement of the approaches, which only make use of purchase data is implemented in the paper by Trofimov [6] introducing BB2Vec. The *BB2Vec* model uses both, the browsing and purchase session data, thus it eliminates the cold-start problem. The *BB2Vec* model is a combination of several *Prod2Vec* models, which are learned simultaneously with partially shared parameters [6]. Although the core idea of using *Prod2Vec* model still remains, the proposed model is additionally relying on browsing data and it outperforms its predecessor models.

There have been several different approaches using supervised learning. Some of these studies focus on image data, product text attributes or both. *SCEPTRE* is a model introduced by McAuley, Pandey, and Leskovec [7] and stands for Substitute and Complementary Edges between Products from Topics in Reviews. The main goal of the model is topic modelling using Latent Dirchlet Allocation (LDA) [39] and edge detection of related topics. *SCEPTRE* uses Amazon data of frequently viewed and bought together items, meanwhile collecting the ground-truth. This paper mainly focuses on using review data for topic modelling and it outperforms typical LDA.

Another interesting approach is making use of multi-modal input (making use of image, text and user ratings). *ENCORE* or the Neural Complementary Recommender explained in the paper by Zhang et al. [8] suggests a three step algorithm: 1) detecting the complementarity among products based on their embedding distances of image and text attributes, 2) taking into account user preferences (ratings) for detecting validity of each complementarity distance and 3) training a neural network with the outcomes of the previous two steps [40]. *ENCORE* is a supervised learning approach as it uses Amazon purchase history (sections "Also-bought" and "Also-viewed") for labelling the data and it outperforms its baselines and alternatives mentioned in the paper [8].

So far, we have seen different approaches in the field of finding complementary products using purchase data, Word2Vec paradigms from NLP, topic modelling as well as graphs. In Kalchbrenner, Grefenstette, and Blunsom [24] Dynamic Convolutional Neural Network (DCNN) have been extensively explored for semantic modeling for sentences using CNNs. The most similar approach to this thesis work is the paper using SNNs for detecting complementarity [9]. This paper's focus is on CNNs for detecting complementarity

among given two products using only product titles as attributes. When compared to previously mentioned related work, despite the main model, it differs in the way that it gets two products as an input and only outputs the probability of those products being complementary while simultaneously learning and sharing the neural network parameters for both products. We consider this research as an excellent baseline for our thesis experiments.

Chapter 3

Methods

This Chapter will tackle the data used and the applied methods for solving the problem of this thesis. Firstly, we will state the requirements, goals and hypotheses of this work. Furthermore, the data retrieval, data generation and data analysis will be explained and presented. Section 3.4 will focus on the implemented models and conducted experiments. Moreover, we will present the architectures of the two models in details, as well as the hyperparameters and loss functions used. Last but not least, we want to construct models, which will be usable in reality, therefore Section 3.4.6 will show the implementation on how to make the models scalable and usable in real world scenarios when we have millions of data points to process.

3.1 Requirements and Goals

In this thesis we attempt to develop *highly accurate*, *scalable* and *usable* solution for detecting complementary products for the company's online catalog, which will eliminate any need of human assistance. One of the biggest challenges is to provide scalability, as in reality we will be dealing with millions of products, which will need add-on suggestions generated in a matter of seconds. To address the problem and our end-goal, we need to fulfil the following requirements and goals, ordered by chronological order:

- Retrieve and prepare the labelled dataset for one category of interest.
- Generate negative samples for the aforementioned dataset.
- Conduct data exploration in order to be able to better understand the data and see possible flaws or improvement points.

- Apply data preprocessing, especially because we are dealing with different kinds of input data. Moreover, transform the data in a format that be can used by a neural network.
- Include additional word embeddings using Word2vec.
- Implement and test Siamese CNN in order to get complementarity scores for given two products.
- Implement and test Siamese LSTM in order to get complementarity scores for given two products.
- Hyperparameter tuning in order to find the parameters of the model that gives the best performance.
- Comparative analysis between Siamese CNN and LSTM.
- Comparative analysis between Siamese LSTM and the baselines.
- Test whether the title is the most valuable parameter.
- Measure and improve the ability of the model to handle millions of products.
- Present the results, conclusions and guidelines for future research.

3.2 Hypotheses

Before diving deep into the data and the experimentation part, it is necessary to gain some domain knowledge and think about the bigger picture of the work. Formulating hypotheses will help us to better understand the outcomes that we are trying to achieve and it will structure the problem definition. The hypotheses, which this thesis is trying to prove are as follows:

Hypothesis 1. *Siamese LSTM outperforms Siamese CNN for predicting complementary products using the same text attributes.*

Hypothesis 2. *More product attributes will increase the model accuracy in both cases. However, the product title is the most valuable attribute for determining the complementarity.*

Hypothesis 3. *SNN can scale up to handle millions of data inputs and provide highly accurate solution for detecting complementarity among e-commerce products.*

3.3 Dataset

The data is one of the most important parts of the pipeline, especially because we are using supervised learning approach. As previously mentioned, the company has around $24M$ products in their online catalog and most of these products do not have add-ons. The company as an online catalog has multiple shops such as Electronics, Health, Beauty, Fashion, etc. As the names suggest, these shops are in a way product categories, but for simplicity and maintaining the same naming convention, we will refer to them as product shops. Therefore, we are interested in picking a shop that: a) already has some add-on matches for the products and b) where add-ons are of a big importance and there is a clear need at the moment.

Having in mind these two conditions, as well as the importance of scoping down the initial experiments, the matches from *Garden and Christmas* shop will be used as the training, testing and validation data in our model. In the following subsections of this Chapter, we will explain in details the data retrieval part, focusing on *Google BigQuery* tables and the data gathering. We will continue more in details about the negative samples generation in Section 3.3.3. In Section 3.3.2 we will present some interesting findings and very important data analysis, which might have high impact on the upcoming experiments in this work.

3.3.1 Data Retrieval

The data that is being used in this thesis is gathered from the company's data warehouse - BigQuery, which is used in the company for data and business analysis. The four data sources (tables) that were used for gathering the needed data were `offers`, `products`, `product categories` and `orders`. Figure 3.1 shows the overall structure of these data sources. The underlined parameters are *primary keys* meaning that they uniquely identify the rows in that dataset.

- **Products.** The products table holds information on products. Each row is identified by a unique internal `productId` and it consists of the title, description, brand, images and many other product attributes. The column `product-Attributes` holds information about more specific product attributes, which may vary among products, meaning that not all products have the same attributes. Such attributes can be dimension, size, color, etc.

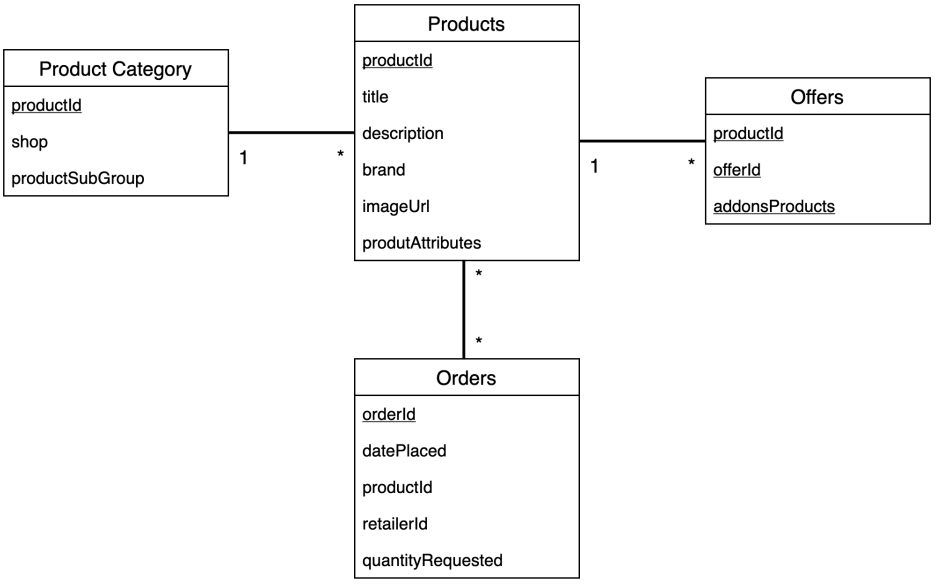


Figure 3.1: Class diagram representing the four main data tables used in this thesis with some of their main attributes.

- **Product Category.** This table holds information about the product categories for each product. In fact, despite the main shop division, products can belong to different `subCategories` within that shop.
- **Offers.** Each product can have multiple offers from different retailers. For example, a few different sellers can sell an iPhone for different or even the same price. The same Offers table holds information about related products for that offer which are actually the add-on products. In some cases retailers can specify the add-on products for the product they are offering, but in most cases at the company retailers do not specify the add-on products manually, which means they are either generated by the company, thus the add-ons are specified for the product regardless of who the seller of that product is or not generated at all.
- **Orders.** The orders table shows every order made by a customer. Each row has a unique identifier based on the `orderId`, `sellerId` and `productId`. In fact, there can be multiple products within an order. We use this table as we can extract information about which items were frequently (not) bought together but we do not gather any information for the users.

Initially, we are mainly interested in getting pairs of matches in the format `(mainProductId, addonProductId)` for each product in the *Garden and Christmas* shop that has at least one add-on. This means that if a product has multiple add-ons, it will appear multiple times as the `mainProduct` in the table. On the other hand, a product might be an add-on to multiple different main products. We name this table consisting of the pairs of product `matches dataset`. After gathering these product matches from the shop, we are interested in getting all of the information (attributes) for those products. We refer to this table from the database as `content dataset`. Furthermore, we are using the `orders` table because we are interested in the ordering frequency of the products. The convenience in using Google BigQuery for retrieving this data is the simplicity that it offers, the high speed performance as well as the opportunity to locally save the newly generated (merged) datasets in pickle files, which can be further used in the python notebooks. Once we have these two main datasets, we want to merge their contents to finally get the dataset having the columns `(mainProductId, addonProductId, mainProductTitle, addonProductTitle, mainProductDescription, addonProductDescription, mainProductBrand, addonProductBrand)`.

3.3.2 Exploratory Data Analysis

In this part of the report we will present the results of the exploratory analysis of the data, some interesting numbers as well as the analysis of the newly generated datasets including the negative samples generated using some rules. To begin with, *Garden and Christmas* shop in total has 113.397 products separated in 13 sub-categories. Some of the sub-categories are *Flowers and Plants*, *Garden furniture*, *Christmas*, *Ponds* and so on. Figure 3.2 shows the distribution of the products within the sub-categories in this shop.

Moreover, we assume that the three most important features are the product title, description and brand. Table 3.1 shows the minimum, average and maximum length of textual attributes: the title, description and brand. Note that these numbers might slightly change once we perform the data cleaning part where we remove unneeded tokens from these textual attributes. Figure 3.2 shows the number of unique categories, the most and least common brand category.

We will now give a more detailed overview of the information behind the products that are included in the `matches dataset`, which includes all existing matches for that shop together with their textual and categorical at-

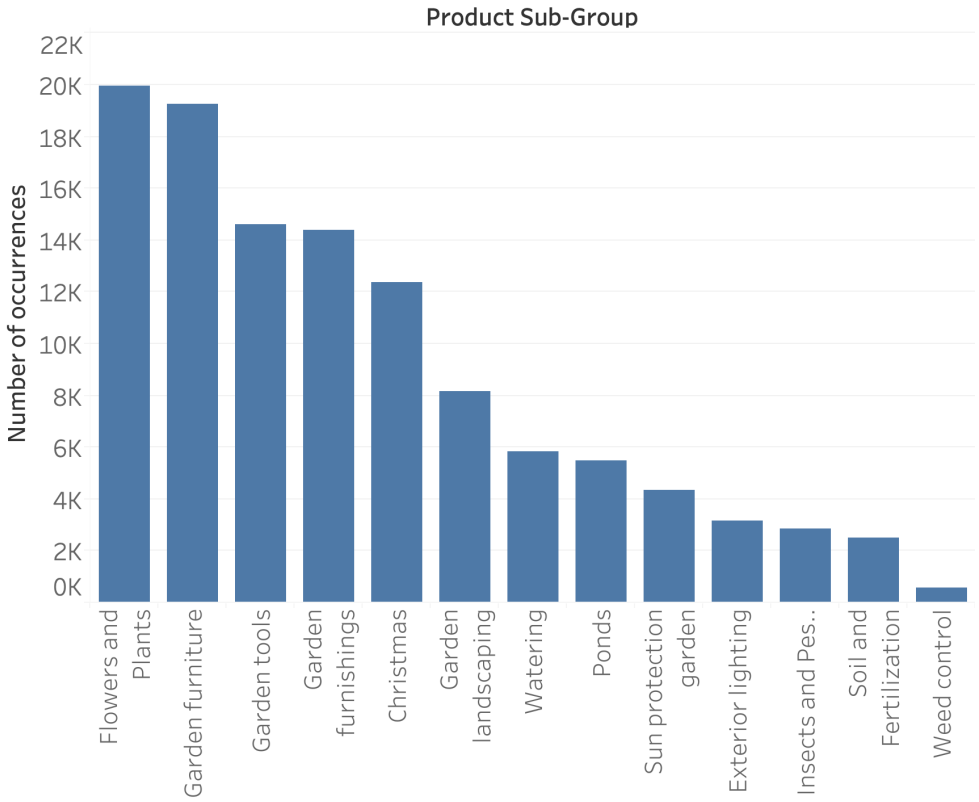


Figure 3.2: Bar chart presenting the sub-categories in the *Garden and Christmas* shop.

Attribute	Max length	Min length	Average length
Title	53	1	8
Description	2.641	1	116
Brand	8	1	1

Table 3.1: Analysis about the product title, description and brand in terms of words. The data is taken from the *Garden and Christmas* shop.

tributes. There are 18.346 pairs of products for which we know that they are complementary. There are 7.478 unique products in total. The maximum value one main product appears in this dataset is 9 times, whereas the maximum value that the same product appears as an add-on is 1.417. On the other hand, the mean value that a product appears as a main product is 2.6 and 13.7 as an add-on. This tells us that there are some products that appear many times as add-ons to different main products.

Attribute	Unique categories	Most common	Least common
Brand	5.397	18.844	1

Table 3.2: Analysis of the brand attribute for the *Garden and Christmas* shop.

More detailed view on this is show on Figure 3.3. The blue line indicates how a line of equality should look like in the case of a "perfect" distribution of the add-ons. The orange line represents the distribution of the add-ons in our dataset. We see that around 5% of the add-ons (which is approximately 68 products) take up around 80% of the whole add-ons dataset. This means that there are a few products out of the 1.366 unique products that appear as an add-on very often. This is also represented in Figure 3.4 where we see bubble chart of the add-ons distribution. The bigger the bubble is, the bigger portion of the data that add-on takes. The biggest bubbles take up to 1.417 points as shown. The dark blue colors represent the most common add-ons and the light blue color the least common.

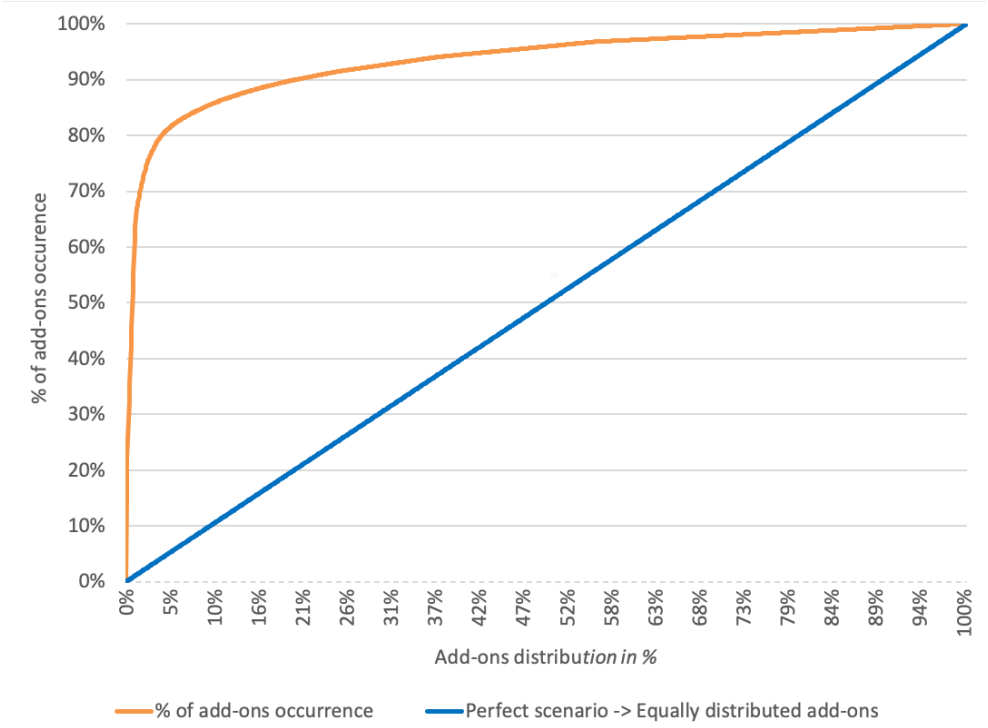


Figure 3.3: Gini coefficient graph presenting the distribution of the add-on products.

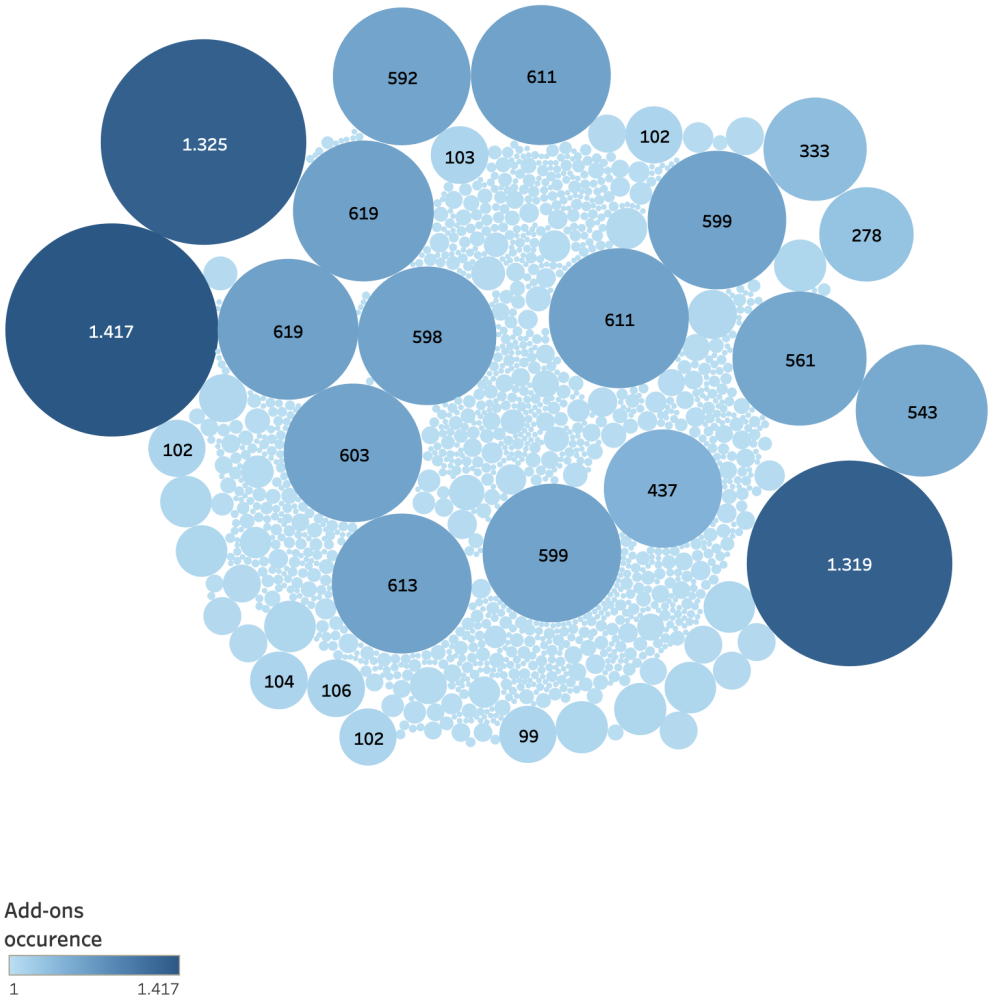


Figure 3.4: Bubble chart of the add-ons distributions. The labels of the bubbles are representing the occurrence of that product as an add-on product.

In fact, these products that appear more than few hundreds times as an add-on are mostly Christmas tree decorations, which go well-together will any product from the sub-category Christmas. These are quite important findings, which we use for generating the negative samples. For instance, if we do not include any other information about the product that appears 1.417 times as an add-on in the training set, but we then want to test if some products go well together with this item, the model will always say yes (label 1) because it did not get any information where this product is not an add-on. The model would

have learnt that it is very frequent add-on and it would have concluded that if it only appears as add-on to that many different products then it must be an add-on for every other given product. As one can image, this can be very wrong in real life scenarios (the Christmas tree decoration does not go well with snow slides or a garden snowman sculpture from the same shop), but it might be even more wrong when talking about other more complex and diverse shops. Such shop can be the *Electronics* shop: if tripods go well together with any camera or phone, we know for a fact that they do not go well together with other electronic devices such as TVs.

3.3.3 Data Generation

The dataset that we use in this thesis, does not have any negative samples. By negative samples we mean pairs of products (rows of the table) where the label is set to 0 meaning that the add-on product is not an add-on to the main product. As we are dealing with supervised learning approach, it is obvious that we need these negative samples although they are not stored in any of the retrieved datasets. If we do not do the negative samples generation, the model will conclude that everything is a complementary product (label 1) for every given main product, simply because we did not teach the model that there are also products, which are non-complementary. It is only natural that the company does not have these information as they are not of any interest to the company. However, we need these negative labels for training the model in this thesis and for conducting the experiments. The most accurate way of dealing with this would be manual labelling, but the disadvantage of it is that it requires many labelling experts with domain knowledge and it is very time consuming. Therefore, in this thesis, we are proposing two different but also complementary approaches for generating the negative samples in our database:

- The first approach is to generate the negative samples randomly. We take the assumption that if a pair of two products is not in the database, then it is a non-match, meaning that the second product in the pair is not an add-on to the first product in the pair. Of course, we might lose lots of valuable information by doing so, because it does not necessarily mean that if a pair of products is not in the database it is not a match. For instance, the database shows that a flower vase and wooden chairs are add-ons to the a wooden table and there is not any information yet in our database about the relation between the vase and the chairs. Taking this random approach, we would put the wooden chairs and the vase as non-complementary products, simply because we do not hold any

information about their relation. However, it might turn out that those two products go well together because they are from the same brand and are in the same colour, or simply because they are a complementary pair. However, in the previous work done in this field, there have been approaches where the negative samples generation was done randomly [8] [9], which proves that this approach can be reliable and useful.

- The second approach, which is basically an extension of the first approach is based on the assumption that if two products were never bought together, we consider them as non-complementary. For doing this, we are taking into account the `bought together` occurrence for each pair of products in the dataset and we take that pair if the product pair is not in that table, meaning that those two products were never bought together in the same order/purchase. This approach is more intuitive especially considering the fact that the add-ons at the company, the add-ons that we are using as a labelled dataset have been generated using the opposite assumption. That is the assumption that if two products were frequently bought together, they *might* be an add-on. We emphasize *might* as this is the first part of the human labelling process.

We need to be very careful of the distribution of positive and negative samples in the dataset as well as the distribution of the unique products. It was stated that in reality one single product can appear to be an add-on to many different main products and vice versa, but this can potentially be a big factor in the model overfitting and generalization. For the reason of understanding the data in details and getting insights about the possible effects it might have on the model performance, we did the data analysis showed in Section 3.3.2 and we iterate this process of negative data generation until we are certain that the data is in good shape and can produce promising results.

Generating Negative Samples

After analyzing the catalog of products in the shop of interest, as well as the matches' distributions, we could generate the negative samples following some rules. So, in a perfect scenario, we would like to have $50-50$ positive-negative ratio, meaning that the number of rows in the dataset having label 0 should be the same as the number of rows with label 1.

For this purpose, following the second assumption based on never bought together items we get $50 - 50$ positive-negative ratio. However, we get the scenario shown on Figure 3.5. The left table shows an example of pairs of

products from the new dataset, where `Product 1` only appears as a main product where the label is set to 1 (Yes). This might be due to the fact that we were randomly selecting products from the shop and only checking whether that pair is never bought together and is not in the dataset. Because this is done by random choice, `Product 1` might never be in a pair where the label is 0. Then, if we have a look at the table on the right on Figure 3.5, which is representing a possible test case, we see that if that `Product 1` appears in a pair with a completely new `Product 5`, which originally have the label 0, our model will say that it is 1 with a very high certainty close to 1 because it has never seen a case where `Product 1` was paired with a non-addon.

Main ProductID	Add-on ProductID	Label
Product 1	Product 2	Yes
Product 1	Product 3	Yes
Product 1	Product 4	Yes

→

Main ProductID	Add-on ProductID	Pred score	Pred label	Real label
Product 1	Product 5	1.0	Yes	0

Figure 3.5: Showcase of a possible training and test set where there is overfitting due to limited main products data. The table on the left is showing a subsample of a training set. The table on the right is showing a possible test case scenario.

We have the same problem for the add-ons column. The left table on Figure 3.6 presents the possible train set where `Product 5` appears as add-on two times and always having the negative label 0. In the right table on Figure 3.6 we see that for a test case where `Product 5` appears with a new unseen main product our model will say that it is not an add-on because it is very likely that this product is not an add-on to any product. This comes from the fact that this product did not go together with any of the products during the training.

For preventing data overfitting, we want to create the data in such way that the distributions of each main product and add-on product are equal for both labels. We handle these two criteria in the following way. We iterate through the add-ons list and for each add-on we make sure that we generate as many negative samples as there are positive. Given `Product 5`, which is an add-on product to 10 different main products, we find another 10 products $2, 3, 4...$ for which product `Product 5` will not be an add-on, meaning that we put the label 0 to the newly created pairs. For finding those 10 new main products, we use the assumption that if `Product 5` and `Product 1` were never bought

Main ProductID	Add-on ProductID	Label
Product 1	Product 4	Yes
Product 1	Product 5	No
Product 2	Product 6	Yes
Product 2	Product 5	No

→

Main ProductID	Add-on ProductID	Pred score	Pred label	Real label
Product 9	Product 5	1.0	No	1

Figure 3.6: Showcase of a possible training and test set where there is overfitting due to limited add-on products data. The table on the left is showing a subsample of a training set. The table on the right is showing a possible test case scenario.

together, they are non-complementary, therefore we put it in the dataset. After making sure that we have $50 - 50$ ratio in terms of the labels for each of the add-ons in our dataset, we repeat almost the same process again but now taking care of the main products. In a nutshell, we want each main product to have same number of positive and negative samples. If main `Product 1` always has label 0 regardless of the add-on product, there obviously will be overfitting. Therefore, we want to balance the data from both sides in the same way.

At the end we end up having 60.442 total pairs of products out of which 42.096 negative and 18.346 positive samples. There are 35.978 unique products overall. This number significantly increased because we introduced completely new products from the same shop while generating the negative samples based on the *never bought together* assumption. After making sure that we have $50 - 50$ positive-negative ratio for the main and add-on products, the dataset ended up $24 - 76\%$ ratio for the overall data. Ideally, we would like to have a balanced dataset where the proportion of positive and negative samples is (almost) equal.

There are different techniques for dealing with such problems, and some of the most beneficial methods suggest augmenting data by using the data that we already have [41]. This would mean that new "fake" products will be generated based on the already known content and what goes well together with that. This is a rather complex problem on its own as it has not been widely explored for textual data, meaning that there is not a framework or simple enough method that could be applied. Therefore, in this thesis for achieving simplicity in this part we use *Random Over Sampling* method, which creates duplicates of the minority class. There are advantages and disadvantages of such process.

On the positive side, compared to other methods it is fast and easy to generate these duplicates. On the other hand, by doing this we do not give any new information to the model because we are simply overpopulating the data with product titles that were already there.

3.4 Methodology

This section includes the main part of the thesis, which is the actual implementation [42] of the suggested work and concepts. First, the data prepossessing part will be explained. This step is needed in order to remove the noise and unneeded tokens from the text. Previously in this thesis, we mentioned SNN, which is the model that we are designing, analyzing and experimenting with in this thesis. In the second subsection of this part, we will explain in details the structure of the SNNs used, more precisely Siamese CNN and Siamese LSTM. Initially, we are interested in the differences between these two proposed architectures, both in the Siamese setup. We want to design two different pipelines where both of them will have the same input, output and general structure, but the building blocks of one part of the network to be different. More precisely, to have different types of layers, Convolutional layers or LSTM layers respectively.

The reason why we are interested in this comparison comes from the fact that RNN/LSTM architecture is proven to be one of the best methods for extracting valuable information out of textual data. In related work, Han et al. [10] used (Bidirectional) LSTMs for detecting fashion compatibility based on text. Similarly, in Han et al. [10] LSTM is introduced for detecting fashion compatibility.

On the other hand, we also want to compare this approach with CNN as besides their well known and proven usage for extracting features from image data, they are becoming popular in the field of extracting features out of textual data as well. The core idea is that once the textual data is transformed in the feature space, the model will see each word same as it would see a pixel from an image and therefore, iterate through the sentence similarly as it would iterate across image pixels. Recent work introduced using CNN for text classification [22], detecting complementary products [9] and for time-series data [27].

Therefore, we want to compare these two methods and check their performances on e-commerce data when the textual attributes are not that rich and contain very unique words, tailored for the e-commerce platforms and customers. For both of these approaches, we use the same techniques, data and we conduct the same experiments. After analyzing their performance, we are

interested in testing and exploring the scalability that such Siamese architectures allow. So, the last subsection of this part thoroughly explains how can our solution be transformed for real-life scenarios with big data.

3.4.1 Data Preprocessing

Data Cleaning

Data preprocessing is an essential part of the pipeline, especially because we are dealing with textual data, which has a lot of noise and requires time for proper understanding. We apply a few different techniques to prepare the data. The following steps are applied to the whole dataset:

- Small-cased all letters
- Removed punctuation signs
- Removed all digits
- Removed measurements (e.g. "cm", "m")
- Removed stop words

During the experimental work, we are checking if all of these steps are necessary and more importantly, whether they improve the model performance. The outcome is that the combination of all of these steps is essential for getting the best possible results from the data given. Of course, these steps might differ among categories. We are using *Garden and Christmas* shop, a frequent format of a product title (original, in Dutch language) is the following:

"Triumph tree Forest Frosted kunstkerstboom - 60 x 46 cm - Met jute kluit"

The title shown above starts with a capital letter, has more than nine tokens and includes dimensions (digits), measurements and signs. We need to remove these digits as they are presenting more noise than actual new information about the products. However, if we are using some other shop, for instance the *Electronics* shop, the numbers indicating the smart-phone model might be of crucial importance, therefore in that case we would not exclude the digits from the product titles without further investigation. In fact, excluding the digits from the products in our dataset, improves the performance by 10% regardless if it is Siamese CNN or Siamese LSTM.

Data Split

In previous sections, we explained how and why the data needs to be split in three different sets, namely: train, test and validation set. There is a standard way of splitting the dataset using a python library, which allows to specify the proportion of the data you want to use in each test set as well as the randomness of selecting those. However, by doing the typical *train-test split*, we cannot guarantee where specific products will end up. Since we are dealing with textual data and embeddings, we need to make sure that the models do not overfit and that they generalize well. What this means is that if we split the data randomly in two parts, we might have `Product 2` that appears in the train set only in the pairs where it has a label 0. Then, if the model sees the exact same product `Product 2` in the test set it will predict 0 because that is the only thing it learnt. On the other hand, if the data is randomly shuffled in such way that for each product we have equal number of positive and negative labels in both, the train and the test set, there is a problem that the model will give us high accuracy and very good results for the test and validation set, but once we try it on completely unseen data, it will perform very poorly. We do not want such scenario as we want to report and analyze the model performance objectively. For this purpose, we use *Group Shuffle Split* in order to make sure that each product that will appear as an add-on in the train set will not appear as an add-on in the test set. Therefore, we make sure that the model's performance is calculated on new unseen data.

Tokenization

Tokenization is the process of segmenting text data into individual tokens (words). Before the text data can be presented to the neural network, we first encode it so that each word is represented by a unique integer. Then, we are forming sequences out of those integers representing the product titles. For this task, in this thesis, we were using Tokenizer API provided by Keras. Note that we do this once we have the train-test set in place. So, this tokenization does not take into account any frequency of the words compared to other encoding techniques, but it only replaces words with a unique integer. We also add padding to make all vectors of the same length and for this we use the maximum length from all data, which in our case is 30 when we only use the product titles as attributes. After we perform this step, the data is ready to be used by the SNN.

Layer	Input Shape	Output Shape	Param #
Embedding	(None, 30)	(None, 30, 300)	6219900
ZeroPadding1D	(None, 30, 300)	(None, 34, 300)	0
Conv1D	(None, 34, 300)	(None, 32, 100)	90100
MaxPooling1D	(None, 32, 100)	(None, 6, 100)	0
ZeroPadding1D	(None, 6, 100)	(None, 8, 100)	0
Conv1D	(None, 6, 100)	(None, 7, 100)	20100
MaxPooling1D	(None, 7, 100	(None, 2, 100)	0
Dropout	(None, 2, 100)	(None, 2, 100)	0
Flatten	(None, 2, 100)	(None, 200)	0
Dense	(None, 200)	(None, 100)	20100
Dot	(None, 100)	(None, 1)	0
Dense	(None, 1)	(None, 1)	2

Table 3.3: Model A: Siamese CNN architecture and hyperparameters.

3.4.2 Model A: Siamese CNN

The main building block of the task is the neural network architecture. The first choice for the experimental and scientific analysis is a Siamese CNN, which will map two product titles from an item pair into representation vectors, which are then used for computing the compatibility between the two products. After the preprocessing is done, the main Siamese CNN implementation takes place.

Model Architecture

We will split the implementation part in three different blocks, namely: a) CNN implementation, b) Merging implementation and c) Compiling implementation. An overview of the architecture of the Keras model can be seen on Table 3.3.

CNN Implementation

The first part of the network is to create an architecture, which will support pair of two inputs (product titles) and return a similarity score. We are building a *Sequential* model in Keras, having in total 11 layers out of which one is an embedding layer and two convolutional layers. We will briefly explain the function of each of the layers in the case of our specific problem together with the hyperparameters:

- **Embedding layer** is the first layer of the neural network architecture and it creates embeddings for each of the words in the product titles. It basically learns the vector representation of the textual data. In Keras, this layer requires three arguments: `input_dim`, `output_dim` and `input_length`. The first parameter is the length of the vocabulary in the data, which is basically a vocabulary of all unique strings in the data. The second parameter is the size of the vector space in which words will be embedded, and this is a hyperparameter, which can be adjusted based on the task. In our case, we choose the output dimension to be 300, meaning that each word will be represented in 300 different features in the multi-dimensional space. Finally, the last parameter is the length of the inputs, which in our case is 30 when we are only using product titles. This is related to the padding that we did before, so here all of the inputs have the same length of 30 tokens. Other than these three parameters, there are two more very important parameters: `weights` and `trainable` parameter, which can be used if we use pretrained embeddings and we do not want the embedding layer to start the learning from scratch. If these two parameters are omitted, the default behaviour will be to start with random weights and perform the training.

- **ZeroPadding1D** layer is used to fill the surrounding of the input with zeros in order to help in preserving features that exist in the edges of the original input matrix and control the size of the output. The parameter indicates how many zeros to add and in our case it is set to two.

- **Conv1D** layer is the first convolutional layer in the network. It extracts the features from the input data and preserves the relationship among consecutive parts of the product titles. There are a few hyperparameters, which are very important and need to be tuned for solving the task. The `filters` are representing the dimensionality of the output space, `kernel_size` is specifying the length of the 1D convolution window, `padding` is set to "valid", `activation` function is "relu", the `kernel_regularizer`, `bias_regularizer` and `activity_regularizer` have all `L2 regularization penalty` with the default 0.01 value. `L2` regularization pushes the sum of the squares of the parameters to be small in comparison of `L1` regularization, which pushes the sums of the absolute values of the parameters to be small. Changing the regularization type significantly increases the performance of our model.

- **MaxPooling1D** layer is used always after the convolutional layer and it downsamples the input representation by taking the maximum value over the window defined by hyperparameter `pool_size`, which in our case is set to five. The window is shifted by strides. The resulting output when using "valid" padding option has a shape of:

$$output\_shape = \frac{input\_shape - pool\_size + 1}{strides}$$

- **ZeroPadding1D**, **Conv1D** and **MaxPooling1D** layer are basically the same layers that we mentioned earlier in the steps. We are repeating these three steps by only changing the `filter_length` with sizes two and `pool_size` with three.

- **Dropout** layer is used to prevent overfitting. Its hyperparameter is set to 0.01, which indicates that 1% of the input units will be dropped.

- **Flatten** layer as the name suggests is used to flatten the output so that we have a simple vector output.

- **Dense** layer is a fully connected layer with 100 neurons with the "relu" activation function. It means a linear operation on the layer's input. In this case, it applies the rectified linear unit activation function.

Merging

The second part of the pipeline is to merge the two inputs once they have been processed through the neural network separately. This part consists of two layers:

- **Dot product** layer is also provided by Keras and it allows us to give two inputs in order to get the cosine similarity among each given pair of inputs.

- **Output layer** is the last layer, which is a dense layer having the "sigmoid" activation function as we are solving binary classification problem. The output of this layer will be the probability of the pair to have the class 1, meaning the probability that the second input is an add-on to the first input. We set the threshold to be 0.5, such that each score that is ≥ 0.5 will have the prediction that the second input is an add-on to the first input (product) and vice versa.

Layer	Input Shape	Output Shape	Param #
Embedding	(None, 30)	(None, 30, 300)	6219900
LSTM	(None, 30, 300)	(None, 30, 150)	270600
Flatten	(None, 30, 150)	(None, 4500)	0
Dot	(None, 4500)	(None, 1)	0
Dense	(None, 1)	(None, 100)	200
Dropout	(None, 100)	(None, 100)	0
Dense	(None, 100)	(None, 1)	101

Table 3.4: Model B: Siamese LSTM architecture and hyperparameters.

Compiling

This part explains the last part of the pipeline and it is not a layer but just the way that the model is being compiled. In this part there are two important parameters: the `optimizer` and `loss function`. We used "Adam" optimizer and "binary_crossentropy" loss function as we have a binary classification and it has been proven through the experiments that it is the best fit for the way the problem was formulated.

3.4.3 Model B: Siamese LSTM

The second but as important choice for the experimental and scientific analysis is a Siamese LSTM, which will map two products' titles from an item pair into representation vectors, which are then used for computing the compatibility between the two products. After all of the preprocessing is done, the main Siamese LSTM implementation is done in the same way the Siamese CNN is implemented.

Model Architectuee

Again, same as in the previous subsection, the implementation will be divided into three blocks. The illustrated picture of the LSTM architecture can be seen on Table 3.4. Note that some parts of the two Siamese approaches might be similar or almost the same, however, hyperparameter tuning is performed independently for the two pipelines.

LSTM Implementation

We are building a *Sequential* model in Keras, having in total seven layers from which one is an embedding layer and one LSTM layer. We will briefly explain the function of each of the layers in the case of our specific problem together with the hyperparameters:

- **Embedding layer** is the first layer of the neural network architecture same as in the Siamese CNN and basically there is no alteration on its structure. Therefore, for more information about what each of the parameters does check Section 3.4.2. In short, `input_dim` is set to the vocabulary size, the `output_dim` is set to 300 and `input_length` equals to 30.
- **LSTM** layer is the core part of this pipeline and it learns the sequential characteristics of the words in the product titles. There are a few hyperparameters, which might affect the model performance a lot. The first hyperparameter are the `units` and that represents the dimensionality of the output space. This is set to 150 neurons. We again use "relu" for the `activation` function in the layer. The `kernel_regularizer`, `bias_regularizer` and `activity_regularizer` have all `L2 regularization penalty` with the default 0.01 value.
- **Flatten** layer is used when we have multidimensional output such as in the case of the LSTM output and we want to make the output linear so that we can pass it to the Dense layers.

Merging

The second part of the model is to merge the two inputs same as in the Siamese CNN.

- **Dot product** layer is used in the same way here having `normalize` parameter set to "True" so that we get the cosine similarity between the two product vector representations.
- **Dense** layer is a fully connected layer with 100 neurons. In comparison to the previous Siamese CNN pipeline where, in this case we apply the dense layer after we perform the merging of the two inputs. More about this will be explained in Section 4.
- **Dropout** layer is used to prevent overfitting. Its hyperparameter is set to 0.01, which indicates that 1% of the input units will be dropped.

- **Output layer** is the last layer, which is a dense layer having the "sigmoid" activation function as we are solving binary classification problem. Same as in the Siamese CNN, the outcome will be a score indicating the probability that the second input is an add-on to the first input. We set 0.5 as the threshold for the decision making.

Compiling

This part is the same in both approaches as we are interested in getting the same output. What this means is that we use the same "Adam" optimizer and the logically the same "binary_crossentropy" loss function. As we are using the "sigmoid" activation function in the last dense layer it is intuitive that we need to use the "binary_crossentropy" loss function as they are dependant.

3.4.4 Additional Embeddings

In addition to the previous steps of the data preparation, we also create embeddings before we give the data to the models. Embeddings are methods for learning vector representations of the data. They are most commonly used for working with textual data. *Word2vec* [36] and *GloVe* [43] are two popular frameworks for learning word embeddings. What embeddings do, is they simply learn to map the one-hot encoded categorical variables to vectors of floating point numbers of lower dimensionality than the input vectors. For example, one-hot encoded vector representing a word from vocabulary of size 50.000 is mapped to real-valued vector of size 100. Then, in the case of the thesis, the embeddings vector is used in the neural network.

We previously mentioned that the Embedding layer has a parameter *weights*, which can either be left to the default value, meaning that the weights of the network will be randomly initialized and then they will be updated through the learning process of the network or they can be initialized with some starting values. However, by applying some pre-trained model or training these weights before the actual neural network takes place can improve the process of predicting complementarity among products. Of course, this highly depends on the problem and the size of the dataset.

We are training the embeddings using Word2vec before the Embedding layer, completely separated from the neural network architecture. Once we have the embeddings for each of the words in the corpus, we add those weights to the `weights` parameter in the Embedding layer. Moreover, when we are using Word2vec for learning the word embeddings, we are not avoiding the Embedding layer in the neural network but instead we give these learnt weights

as an initial weights in the layer instead of letting the layer start from random numbers for the weights. In practical terms, pre-trained Word2vec embeddings can be used as features of any neural network (or other algorithm). On the other hand, there are examples showing that learning the embeddings only from our data, optimized for a particular problem, may be more efficient in some cases [44].

3.4.5 Product Attributes

Furthermore, we try to include multiple product attributes to answer the second research question of whether the title is the most and only significant attribute or perhaps the description and the brand might increase the model performance. The description is another textual attribute and it is usually seen as a complementary text to the title. Therefore, in the later experiments we concatenate each product description with the title and use it as one textual attribute. The brand is also textual attribute and there are around 5.000 different brands of product in the shop of interest. However, in general the brand can also be treated as a separate categorical attribute, but in our case the diversity of the brands limits us in using it as categorical value and it introduces some other problems. Therefore, we concatenate each product title with its brand in the experiments.

3.4.6 Advantages of the Siamese Architecture

The third research question was connected to whether the proposed solution can be transformed in such way that it can handle big data scenarios. Indeed, the company offers more than $20M$ products in its online catalog, therefore being able to prove scalability is of crucial essence for the proposed solution. In fact, one of the benefits of using SNN, as suggested by Martin et al. [27] is that it can be easily transformed into *K-Nearest-Neighbour (KNN)* problem. The proposed pipeline's results are indicating the probability of the second product being complementary to the first product, for any given pair.

In this part when we refer to the proposed solution, pipeline or neural network we refer to any of the two approaches Siamese CNN or Siamese LSTM as they both produce the same type of output and offer roughly the same complexity. Thus, if we have pairs of products, both of the suggested solutions (Siamese CNN and Siamese LSTM) will be able to produce the compatibility of the pair representing the chances that the second product in the pair is an add-on to the first target product. But, in a real-life scenario, we do not get pos-

sible pairs of products for which we want to check this, but instead we would get a list of main products for which we want to find add-on products from a specific shop, subcategory, brand, etc. We are usually given target products set $Q = q_1, q_2, q_3...q_N$ and candidate set for the add-ons $C = c_1, c_2, c_3, c_4, c_5...c_M$ where N and M will probably have values bigger than 10^6, indicating a few millions of products, which is a valid scenario taking into account the size of the company's catalog. Thus, if we use the proposed solution we would need to create $N * M$ pairs of products and give those products as inputs through the neural network. But in a best case scenario, we would just want the top K candidate products for a given target product as e-commerce websites usually offer only a few add-on products.

Our approach can be easily extended to handle these big data scenarios and complete the task of finding top K most complementary products for a given product in a way which was proposed by Martin et al. [27]. Considering we have a target product q and a candidate add-on product c. We first generate their vector representations using only the first part of the proposed SNN. This means that once the network is trained to create proper vector representations for each product, we are only interested in the weights that the network produce before we apply the dot product (basically, before we merge the two inputs). This is achieved with the Siamese setup as each product is treated separately until the merge point in the model, thus for each of the products in Q and C we can get the weights and store them using the Siamese setup. Once we have the vector representations X_Q and X_C for each product from Q and C respectively, we can compute the cosine similarity. From this point on, we have a KNN problem, so once we get the cosine similarities in a matrix format where each of the columns represents a product from Q and each row represents an add-on product from C, we can easily get the top K add-on products for a given product q sorted based on their probability of being an add-on product.

We want to point out what would happen in terms of complexity if we use the predictions by the neural network when we introduce millions of products. As mentioned, in a real-life scenario we need all possible combinations as we do not know which are possible pairs. Therefore, we could achieve this by combining all possible pairs of products from the target and candidate list and give those pairs as an input to the neural network. While explaining the computational complexity, for simplicity, we will use the number of products introduced in the test set we use during the experiments. This is 4.209 unique candidate products and 9.218 unique target products. In total there are 12.733 unique products from both sets. In the test set there are 16.256 pairs of products (rows). However, in a more realistic scenario we would not know which

are candidate products, thus we might want to test all possible combinations between these 12.733 unique products in total, which results in $162.116.556$ combinations in total excluding the diagonal of the matrix as it the cosine similarity between the same two products, which results in similarity score of 1. Perhaps we could indicate business rules based on price to filter out some of these products and indeed create a candidate list set, which will contain less than 12.733 products.

Furthermore, Figure 3.7 illustrates how the Siamese architecture can be used to extract the weights of each of the products of interest, where the embeddings part is done only once for each product separately. Then, taking the idea from transfer learning [45], we save those weights in forms of matrices and apply the dot product between the two matrices having the weights for each target and candidate product, representing their cosine similarity.

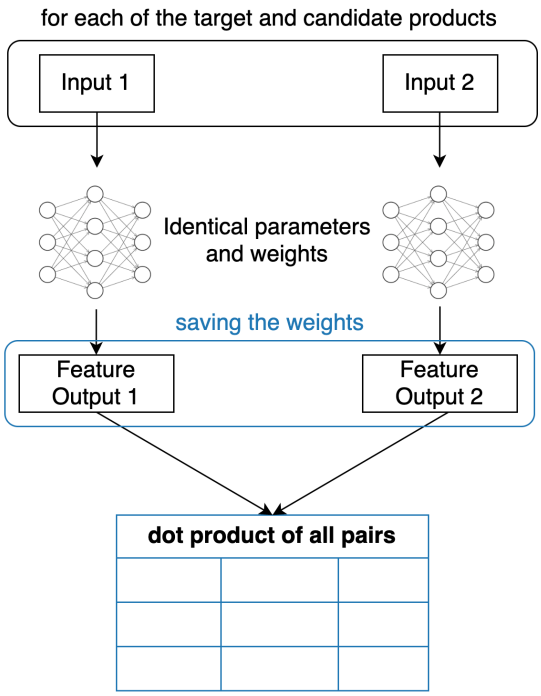


Figure 3.7: Illustration of where we save the weights from the SNN and apply the dot product between the two matrices of target and main products.

Chapter 4

Results and Discussion

In this section, the previously introduced models will be applied on the real data taken from the company to predict complementarity among e-commerce data. The results from various experiments will be presented as well as some comparative analysis. The data for these experiments is always split in proportion $80 - 20$ for the train and test set, while 10% of the train data is used for validation during training. As stated in the earlier sections, we are only making these experiments using one category/shop, which is the *Garden and Christmas* shop. Unless stated otherwise, we do the experiments using the products title as attributes because the titles are short but have a very concise content, which describes the products in the best way. Later on, we will also include other product attributes in the experiments, which will be accordingly presented. The hyperparameters of the models and the actual implementation was introduced in Chapter 3. Therefore, in this section the main focus will be exploring and discussing: a) *the models* in terms of their performance, their (dis)advantages and structure and b) *the data* in terms of how the data can impact the model performance and analyze possible drawbacks.

In the comparative analysis we will see how LSTM outperforms CNN using textual product attributes and Word2vec embeddings. An interesting insight is that using only the product titles produces most promising results in terms of accuracy and time complexity. Later on, by using transfer learning we extend the proposed Siamese LSTM approach to KNN problem.

4.1 Comparative Analysis

In this part of the thesis we do comparative analysis in terms of the proposed models, their architectures, including additional embeddings, baseline com-

parisons as well as results when including `description` and `brand` as an addition to the `title` attribute.

4.1.1 LSTM vs. CNN

To answer the first sub-research question regarding the performance of Siamese CNN and Siamese LSTM, we conducted a few different experiments. We are trying to compare the performance/accuracy of both models and conclude which one solves the problem of finding complementary products better.

The first experiment showcases the different accuracy when we implement the merging - dot layer in different places. We explained that there can be three types of merging of the two vector outputs from the SNN: early, intermediate and late merge. In the case of this thesis we are interested in testing the intermediate and late merge. Figure 4.1 illustrates the difference in intermediate and late merge in the case of our implementation. When speaking of Intermediate Merge (IM) we mean that the Dot layer is applied right after the Flatten layer. What this means is that there is one Dense layer after the merging is done and before the final Output layer is reached. Late Merge (LM) is representing the architecture when the Dot layer is implemented just before the final Output layer.

Having this in mind, we conduct two experiments for each of the proposed architectures to check their best performance in terms of the place of merging. From the results shown on Table 4.1 we see that Siamese CNN performs better when there is late merge implemented, whereas Siamese LSTM performs better when having intermediate merge. $Recall - 0$ and $Recall - 1$ on Table 4.1 are presenting the recall of each of the two classes respectively. We see that Siamese CNN with late merge outperforms Siamese CNN with intermediate merge. On the contrary, when speaking about Siamese LSTM, having intermediate merge gives better results than having late merge. In fact we can see that CNN - IM and LSTM - LM have quite low recall for class 1, meaning that both approaches have problems detecting pairs of products which are complementary. Siamese LSTM with intermediate merge outperforms all other models' architectures and that is the architecture we will use in the further analysis.

Once we got the results out of the experiments where each of the models performed the best, we run the models to see how the accuracy and loss are evolving over 10 epochs shown on Figure 4.2. Figure 4.2 represents how the two models learn over time.

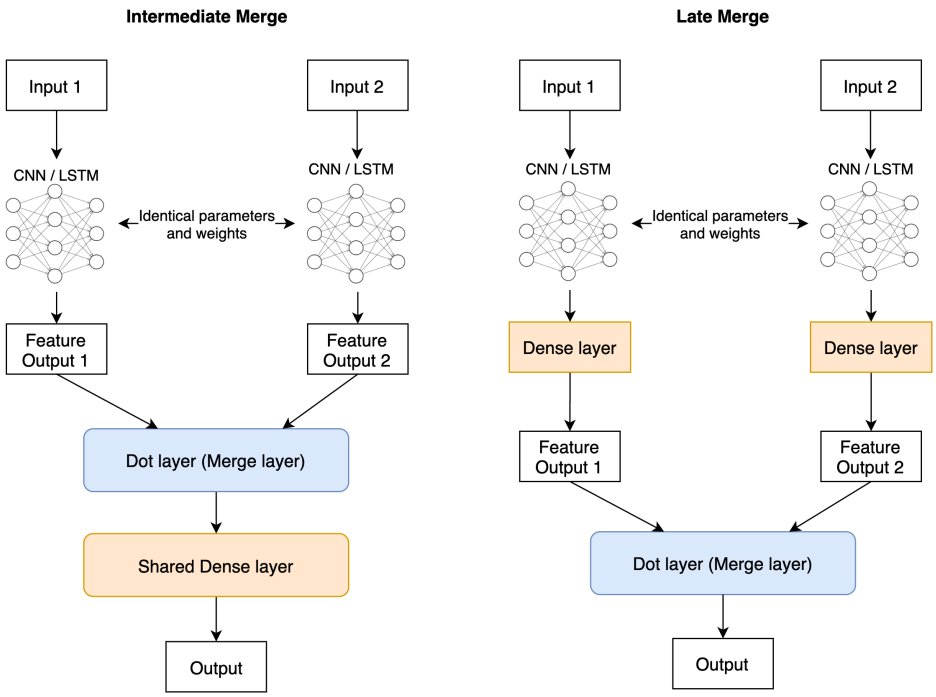


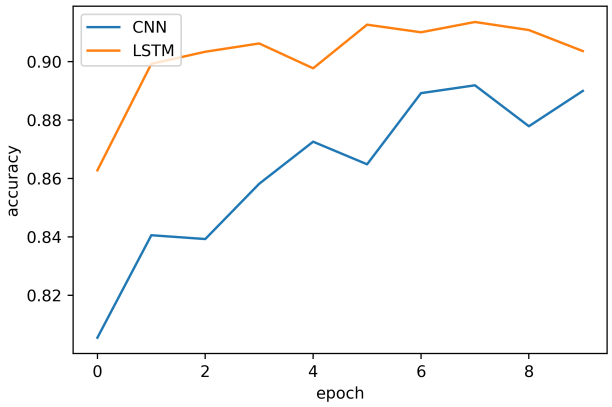
Figure 4.1: The difference between intermediate (left illustration) and late (right illustration) merge in the implementation of the proposed model.

4.1.2 Analyzing the Embeddings

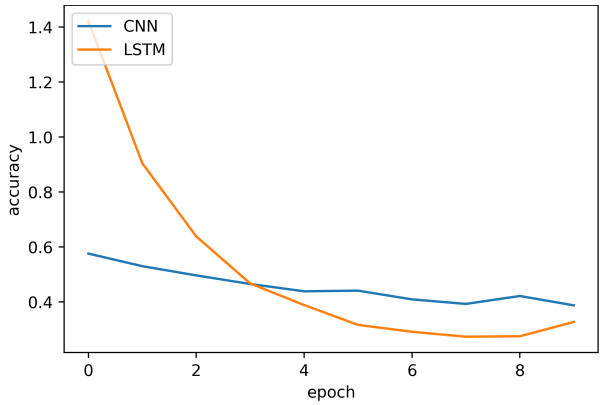
As shown before, Siamese LSTM performs better for the task we are trying to solve in this thesis, thus from now on we conduct all of our experiences only on the Siamese LSTM architecture. In this part, we are reporting the results after applying the method described in Section 3.4.4 where additional embeddings using Word2vec before the Embedding layer were introduced. This was applied for all of the tests shown on Table 4.1. Then, we perform experiments to check how removing the Word2vec model will impact the models' accuracy. We apply this in the LSTM network, taking into account the architecture (merge layer), which gave most promising results from Table 4.1. The outcome of these experiments is that training the word embeddings on the train corpus, taking into account all possible titles from all exiting products in that shop and applying those weights in the neural network architecture increases the accuracy and decreases the learning time. This approach is usually used when the training set is not that large and when we want to help the model to generalize better on unseen data.

Siamese model	AUC	Accuracy	Recall-0	Recall-1
CNN - IM	72%	65%	99%	22%
CNN - LM	82%	78%	63%	93%
LSTM - IM	93%	85%	88%	91%
LSTM - LM	80%	75%	93%	51%

Table 4.1: Comparative results showing the performance of Siamese CNN and Siamese LSTM based on the place of merging the two product outputs.



(a) Comparing accuracy



(b) Comparing loss

Figure 4.2: Accuracy and loss over 10 epochs for Siamese LSTM model with intermediate merge and CNN with late merge.

Results for the accuracy over the validation set are shown on Figure 4.3. We see that when there is no Word2vec applied, the starting point of the ac-

curacy for the model is quite low and it rapidly increases and outperforms the model when there is Word2vec applied. However, this is reasonable to happen since this is tested on the validation data. Furthermore, the model where the word embeddings are learnt in the network is naturally more tailored for the given data. Once we introduce the new unseen data to the model, the model where we use the pre-trained word embeddings using Word2vec outperforms the model where the weights are randomly initialized at the beginning. This is shown on Table 4.2.

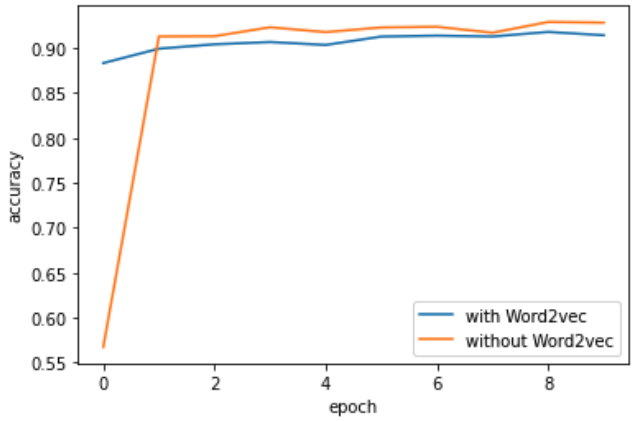


Figure 4.3: Accuracy over 10 epochs when we apply Word2vec compared to when start the training with random weights on Siamese LSTM.

Embeddings	Accuracy	AUC	Training time
With Word2vec	85%	93%	8min
Without Word2vec	76%	81%	12min

Table 4.2: Accuracy and AUC score for LSTM with intermediate merge based on the additional Word2vec embeddings.

4.1.3 Comparing to Baselines

In this part of the results, we want to showcase how Siamese LSTM performs compared to some of the baselines, which are in general frequently used methods. The methods that we are comparing the Siamese LSTM model with are *Random Forest*, *Single LSTM* network having the same layer architecture as our Siamese LSTM and *Vanilla NN* where the first layer is the Embedding

layer and then there is one Dense layer before the final output layer. In the following part we will briefly explain the architectures of the three baselines.

- Random Forest [46] has been widely used for classification tasks such as this one, it is easy to implement and it performs well on textual data. Martin et al. [27] are comparing their proposed model with Random Forest, thus it is a well-defined baseline for our experiments, especially taking into account that we want to continue and upgrade the research done in their paper. The Random Forest implementation starts with combining the inputs (product titles) from each sample in the database and tokenization of the product titles using *Count Vectorizer*. We then apply simple Random Forest algorithm, with 1000 estimators.

- On the other hand, we are also implementing a single (we say single to emphasize that it is not a Siamese) LSTM neural network. By doing so, we want to see the difference when implementing Siamese compared to having the same architecture but for a single neural network. The main different between these two approaches is that in the Siamese approach the inputs are treated separately until the moment of merging the two vector outputs. This means that the embeddings for each product title are done separately and independently from the other product in the pair. On the contrary, in single neural networks, the two product titles for each of the product pairs are concatenated since the beginning and are treated as one input during the tokenization, word embeddings and finally, in the model architecture. For that reason, the `input_length` is set to 60, which is $2 * 30$ where 30 represents the maximum title length in our dataset. The exact layers of the LSTM were presented in Section 3.4.3.

- Lastly, we also implement and test vanilla NN, which is a simple neural network and has the main component - the Embedding layer. Basically, we have in total six layers: Input, Emedding, Flatten, Dense, Dropout and Output layer. The embedding layer uses the exact same parameters as introduced in the Siamese LSTM, but the `input_length` parameter is set to 60 as the two titles are treated as one. The Dense layer has 100 neruons and 'relu' activation function. Furthermore, the dropout rate is set to 0.01. The final Output layer has the 'sigmoid' activation function. We use 'binary_crossentropy' as the loss function and 'Adam' as the optimizer. The purpose of such implementation was to see the importance of the Embedding layer and whether only having the Embedding layer without any specific CNN or LSTM layers will give promising results.

The experiments in the implemented baselines are conducted using exactly the same data with identical preprocessing and train-test split as in the original implementation. The results showing the accuracy and AUC score for each of these models are shown on Figure 4.4.

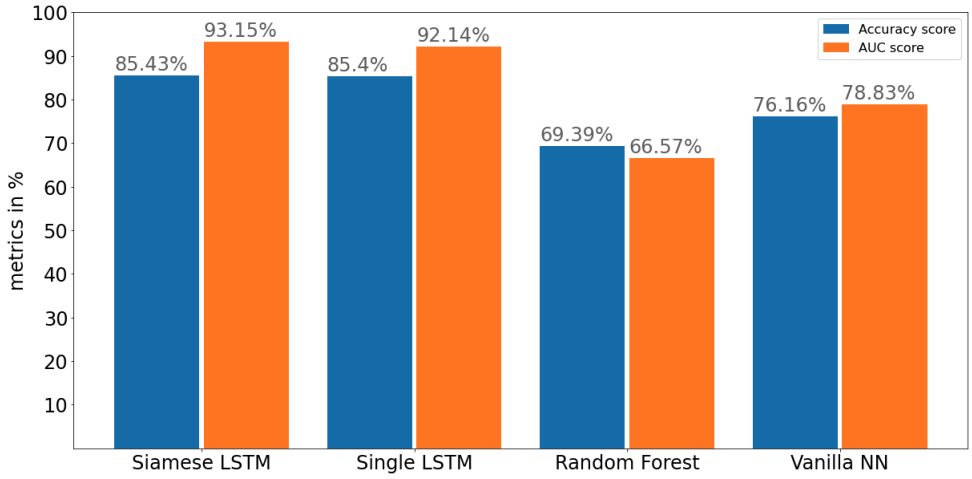


Figure 4.4: Comparative results showing the accuracy and AUC for Siamese LSTM, Single LSTM, Vanilla NN and Random Forest.

From Figure 4.4 we see that the Siamese LSTM and the Single LSTM network perform with the same accuracy. This is due to the fact that we have the same architecture in both cases and the way the predictions are done is identical. The difference and advantage of using SNN is that it can be transformed for big data scenarios and it is more scalable (fast). This will be further discussed in Section 4.2. Furthermore, Random Forest performs the worst in this scenario with accuracy of 66%. The two-layered neural network outperforms Random Forest but performs worse than LSTM as expected. Although the network has the Embedding layer, which handles the textual data well, the network is not specifically trained to handle text sequences, thus there is some loss of information.

The plot on Figure 4.5 is representing the distribution of the model predictions. Red color represents the True Negatives, the inputs that were correctly classified as negative. The bars with green color represent the True Positives, the inputs that were correctly classified as positive (add-ons). We see that sometimes there is overlapping with the colors. Ideally, we would want all green bars to be on the right side of the plot, above 0.5 and the other way around for the red ones. We say 0.5 because this is the standard threshold for binary classification, which is also the one we use in this thesis. As the out-

come of the model is the probability that the input belongs to the positive class, we formulate it in the way that if the models gives $0.5+$ certainty that the pair belongs to the positive class, then we label is as positive, otherwise we label is as negative. However, the "perfect" distribution that we would want is not the case with most of the model outcomes. Figure 4.5 shows the predictions for the final Siamese LSTM. Here, there are a few cases that are wrongly classified and we see that some green bars are positioned at the left side of the chart close to 0. Even if we cannot get the perfect 100% accurate classifier, a better outcome would be to see that the wrongly classified inputs were somewhere around 0.5 representing that the model was not sure about the decision.

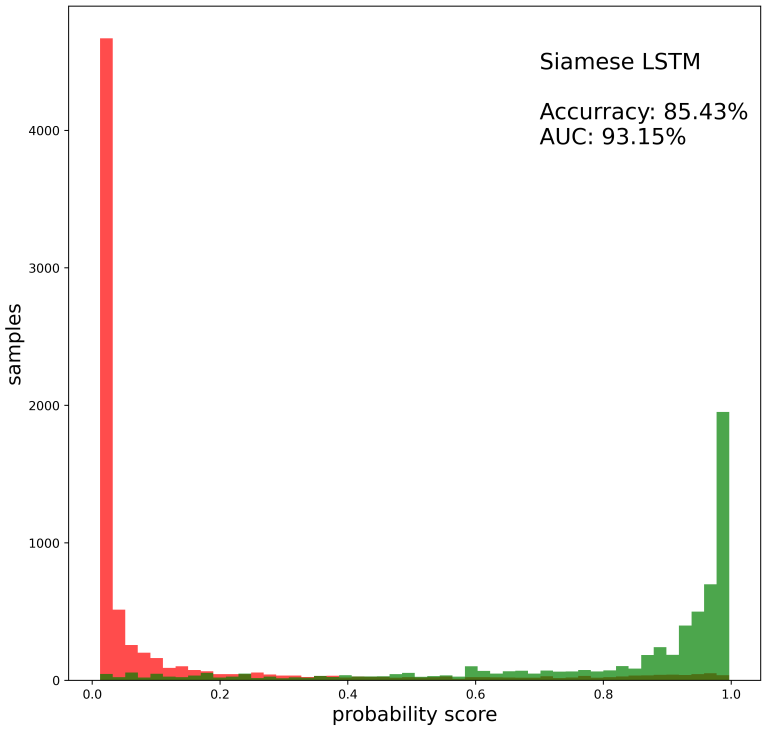


Figure 4.5: Predictions graph for Siamese LSTM.

The reason why the model is so sure that something is not an add-on when in fact it is and vice versa, is simply because it was trained in that way. By further investigation, we see that there are cases where the training data states that two products are complementary, but we see that they should not be marked as complementary based on the product titles and our common sense. This is due to the fact that the original data labelling process was done by human content experts and some business rules. So, even if something might not seem

as an add-on, the experts might have put it as an add-on simply because of the purchase frequency of those two products or some previous experience with the products in general. Another problem is the noise in the data introduced by products, which have very short one or two-word titles, which are basically giving no information about the product. In the case of the company, this is usually done by partner sellers who do not spend a lot of time on the appearance of their products in the company's catalog and have very few information about the products they sell. Therefore, having this kind of noise and inconsistency in the data might prevent the model to formulate general rules, which will always be correct.

Figure 4.6 explains the AUC-ROC curve for the same model. We see that the ROC curve for both classes is in the upper-left corner, which gives us information that the model performs well, hence it is trustworthy.

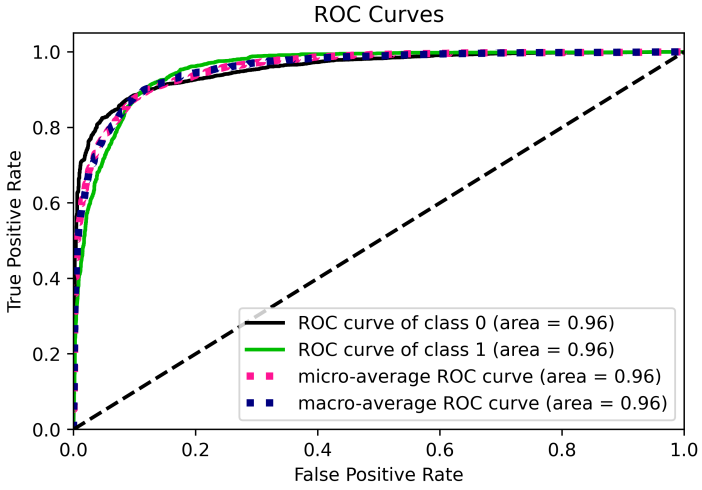


Figure 4.6: AUC-ROC curve for Siamese LSTM.

4.1.4 Testing Product Attributes

We included different product attributes in order to investigate the second research question. The results are shown on Table 4.3. When using only the title, the input length in the Embedding layer (the maximal length in the input) is set to 30. When we include the description to the title or when it is used as standalone the input length is set to 200. In the cases where the description is included, the longest length of the product attributes (title+description) is 800. However, we treat these long sequences as outliers because they are just a few

cases, and it slows down the training time for approximately four times. On the other hand, when we use the brand as an attribute, we set the input length to 35 as after performing the data cleaning the longest brand name is five words long.

Product attribute(s)	Accuracy	AUC	Training time
Title	85%	93%	13min
Title + Description	89%	95%	58min
Description	72%	81%	58min
Title + Brand	80%	83%	14min

Table 4.3: Comparing accuracy, AUC score and training time for Siamese LSTM using different product attributes when the training was done on 10 epochs.

4.2 Transforming the Siamese LSTM into KNN

In this part we will start by showing the difference in the prediction time using a single LSTM network, Siamese LSTM architecture on its own and the proposed transformed/upgraded approach which is a combination of a Siamese LSTM and KNN approach. Then we will be showcasing the results obtained from the experiments when using the transformed approach. Note that we only take into account the testing set for these experiments. In a real-life scenario we would run this method between all products from the chosen shop or even between cross-shops.

In this part we are interested in real-life scenario situation, therefore we are not interested in testing only the ~ 16.000 pairs of products that are included in the test set as in reality we would not have pairs of possible complementary products. Instead, we are selecting all the unique products from the test set which are in total around 13.000. We are interested in the add-ons for all of these 13.000 products as we would not know which are possible target and candidate products. In total, if we want to pair 13.000 products with each other, we would get roughly $170M$ pairs of products for which we want to know their complementarity relationship. For the purpose of this thesis and the experiments, due to the limitations of the hardware, we only take $1M$ pairs of products in the following experiments.

The three approaches that we are comparing here would treat this scenario as follows. The single LSTM network is traversed $1M$ times for each pairs of products. The Siamese LSTM network is traversed $1M$ times as well, but with a slightly higher time performance due to its Siamese architecture and the ability to learn faster. Finally, the transformed approach which is based on transforming the Siamese LSTM into KNN, will be traversed only 13.000 times, for each product once. Then the cosine similarity will be calculated for the $1M$ pairs, which is a very fast operation as it is basically a matrix multiplication. The time analysis for the three approaches are shown on Figure 4.4. If we mark the number of unique products with N, then the time complexity for the Siamese LSTM and single LSTM neural network would be $O(N^2)$ while the time complexity for the transformed approach would be only $O(N)$ as the neural network will be traversed only once for each product.

Model	Prediction time	Time complexity
Single LSTM	11min	$O(N^2)$
Siamese LSTM	8min	$O(N^2)$
Transformed Siamese LSTM	10sec	$O(N)$

Table 4.4: Comparing the time needed for predicting complementarity among $1M$ pairs of products.

From this point on, we focus on analysis of the last approach which is highly scalable for millions of data points. Therefore, we want to analyze its performance further. Once the model weights are extracted from the neural network and the cosine similarity is applied among all possible pairs of the given products, we get the results in the format shown on Figure 4.7. In the outcome, each row is representing the main products identifiers and the columns are representing the candidate add-on products identifiers, which in this case are all products from the test set. For the purpose of analyzing and discussing the results we put pictures instead of the original product identifiers for one of the products shown on Figure 4.8. The target product is the colorful hammock in the top left cell. We select five add-on products as based on discussions with content experts it is the most common number of add-ons to be shown on e-commerce websites.

Figure 4.8 shows top five detected add-ons for the target product - the hammock in this case. We see that the first suggested add-on is actually another type of a hammock, meaning that it is not a complementary product but rather a substitute product. Furthermore, the second suggested product is a real add-on

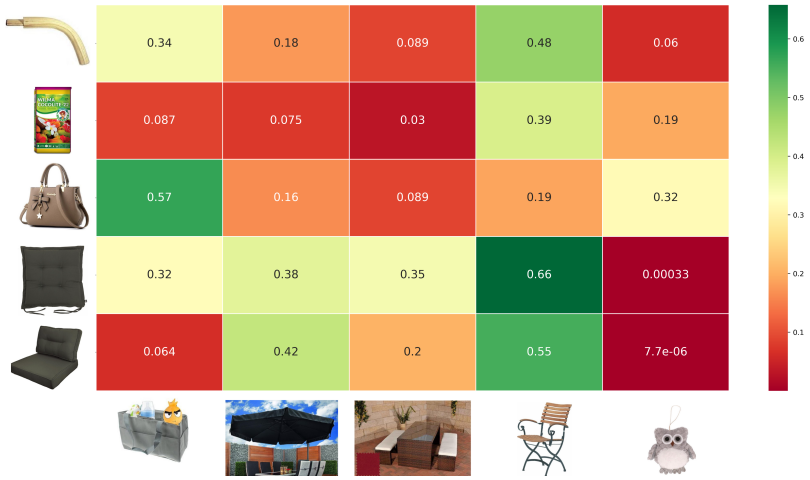


Figure 4.7: Heatmap of the cosine similarity between five target products and five candidate products where the green color indicates high score and the red color indicates no complementarity between the products.

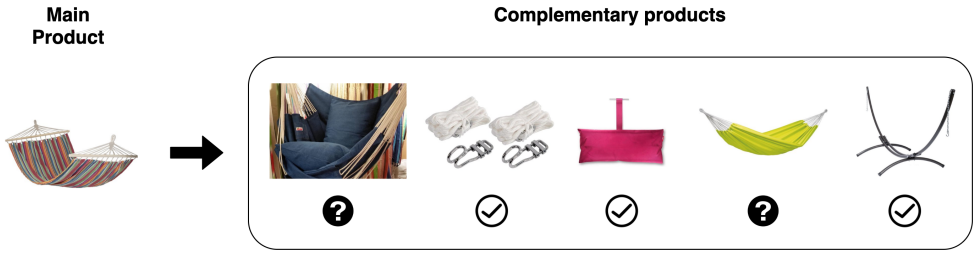


Figure 4.8: Example of suggested top five add-on products for the hammock being the target product.

product, which was predicted correctly. With further analysis and comparisons of multiple products, we can see that in most of the cases the top products that are suggested are either real add-ons products or substitute/alternative products having the same characteristics. This shows that the model can clearly detect real complementary items for a given main product, but there are also a lot of false positives.

On the Figure 4.8 we put either question mark or a tick, where question mark indicates that the cosine similarity $might$ be wrong and tick where we clearly see that it is an add-on. Now, having the transformation of the Siamese model to a simple cosine similarity (dot product) between the model weights has some advantages and disadvantages.

Table 4.5 shows the prediction scores obtained by the two different approaches for the same product, which was shown on Figure 4.8. When columns have the `None` value it means that these pairs were not in the test set, thus they would not have been suggested at all if we use the currently applied method of manual labelling products at the company. The third product is a hammock pillow, which is a really interesting add-on. The results show that the extended approach for millions of products detects products, which were not suggested with manual labelling based on the frequently bought together filtering. On the other hand, for the products that were spotted as add-ons by both methods, we see that the predictions by the transformed approach are with lower probability.

The reason why the second approach is suggesting substitute products as well is simply because it was trained to do that using the given ground truth. In some cases, the add-on suggestions that we have as part of the ground truth consist of similar/alternative products to the target product. Such case is presented on Figure 4.9. We see that the add-ons suggestions for the vase are other vases in different/similar shapes. Now, this is reasonable for the vases as people usually tend to buy a few of these products. But, if the user decides to buy a hammock, this would not be the case because it is an expensive product, which is needed as one piece. The fact that there are cases in the training set where alternative products are allowed for some products but not for other ones is considered as incorrect data because ideally, we would not want our model to suggest these cases. However, this is set as a learning point in the model.

The model implemented in this thesis is not able to distinguish whether for `Product A` we can suggest similar products, but we should not to that for `Product B`. The network learns from the data given, and the data suggests in multiple inputs that having similar products as add-ons is a good idea. Therefore, besides detecting actual add-on products, our model suggests similar products as well. Ideas on how this can be further improved will be included in Section 5.2.

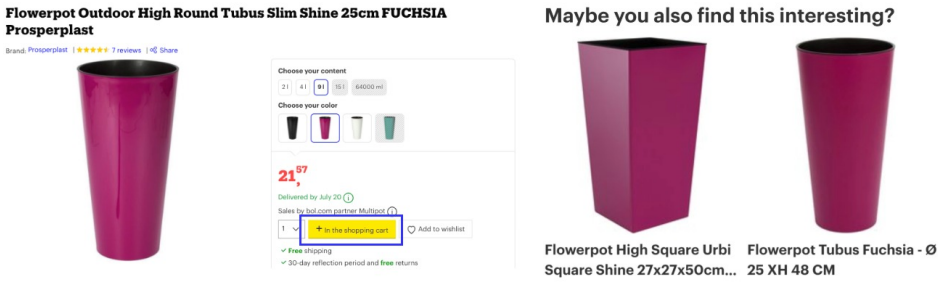


Figure 4.9: Example of the ground truth when similar/alternative products are considered as good add-ons. The two vases on the right are suggested add-ons for the vase on the left.

Siamese LSTM predictios	None	92%	None	None	92%
Transformed approach	83%	72%	69%	68%	66%

Table 4.5: Comparative results in terms of complementarity score between Siamese LSTM for testing given pairs of products and the extended approach when we compare all possible pairs of products. The add-on products shown are suggested when the colorful hammock is the main product of interest.

Chapter 5

Concluding Remarks

In this section, we want to give final remarks regarding the work done during this thesis. First we will start with the Conclusion for the proposed models based on the experiments conducted. Then, in the Future Work section we will give ideas and suggestions for future improvements and experiments. The inspiration for these guidelines for the future work came from the outcomes, challenges and limitations during this thesis work.

5.1 Conclusion

In this work we conducted a research on content-based complementary products recommendations in the use case of the largest e-commerce company in the Netherlands. More specifically, the scope of this thesis was to improve the current manual way of detecting complementary products by suggesting a deep learning solution using SNN, which was expected to provide promising results and scale up easily.

We were using a supervised learning approach using the manually labelled data (matches of complementary products) from the *Garden and Christmas* shop at the company. Chapter 3.3 explains the dataset used during this thesis. In short, after retrieving the data from the company's online warehouse and combining the data sources of interest, we only had the positive classes for the supervised learning approach as the company does not hold information about the negative classes (the pairs of products, which are not complementary). Therefore, in Section 3.3.3 we explain how the negative samples were generated as part of the work in this thesis as well as the problems encountered during the process. After the data was ready and preprocessed, we implemented and tested the performance of Siamese CNN and Siamese LSTM,

both having the same Siamese structure, which accepts pairs of products as inputs and produces a complementarity score as the output of the model. More details about the implementation of the proposed pipelines, methods and metrics were explained in Section 3.4. In addition, we conducted experiments including additional word embeddings using Word2vec as well as several experiments including the description and brand on top of the main attribute - the title. The proposed model was also compared to other baselines in terms of the accuracy and AUC score. Last but not least, Section 4.2 explains how we managed to transform the proposed Siamese approach to handle big data scenarios, thus be scalable and easily applicable to real-life scenarios at e-commerce platforms.

For all of the aforementioned experiments, we accordingly documented the results in Chapter 4. As a conclusion of the experiments, on Table 5.1 we present in dark letters the final solution to the problem that we were trying to solve in this thesis.

Siamese Model	Word2vec	Attributes	Accuracy	AUC
LSTM	**Yes**	**Title**	**85%**	**93%**
LSTM	Yes	Title+Description	89%	95%
CNN	Yes	Title	78%	82%

Table 5.1: The final model and settings that gave most promising results.

Although the Siamese LSTM including the description as an addition to the title attribute increases the accuracy and AUC score, we conclude that a trade-off between $accuracy - vs - time$ needs to be made since including the description attribute slows down the training process for about four times.

This thesis' work was focusing on answering the general research question and the three sub-questions formulated at the beginning of the work stated in Section 1.4.

To start with the general question: *How can we use Deep Learning to improve the process of detecting complementary products based on content having the end goal of increasing the average purchase value on the e-commerce platform?*. We proved that by using Siamese LSTMs and cosine similarity using only the product titles we can detect add-on products and suggest products to the user, which will encourage him/her to purchase more than one item at a time.

The first research sub-question was: *Which of the proposed two models - Siamese LSTM or Siamese CNN predicts complementarity among products*

with higher accuracy based on the content?. Based on the results in Section 4, we come to the conclusion that Siamese LSTM gives higher accuracy. We conducted different experiments by modifying the Siamese CNN and LSTM structures, but at the end we conclude that Siamese LSTM is most suitable model for our problem as it learns complementarity using the words in the product titles as long sequences and it generalizes well.

The second research sub-question was: *Is the title the most valuable attribute for content-based complementary recommender systems?*. The answer is "Yes", the title is indeed the most valuable attribute as shown on Figure 4.3. This is due to the fact that titles are the most informative parts of the products at the company. They are often short and very concise, consisting of only the most relevant information for the user, hence for the model as well. Basically, we can see titles as a summary of the product descriptions.

Last but not least, the third research sub-question: *Can SNNs be transformed such that it scales to millions of pairs of products? How well does it perform compared to the manual (human) pipeline?*. We managed to transform the Siamese LSTM into a solution, which can handle big data scenarios. Compared to the current manual approach, it is scalable (easily extendable) to all categories and shops in the company and it does provide new information, which were not retrieved in the current process. On the other hand, the proposed extended solution does give some false positives, suggestions that are incorrectly classified as add-ons but instead they are substitute products. Further improvements regarding this will be suggested in Section 5.2.1.

In addition to the research questions in this thesis, we also formulated hypotheses for each of the three research sub-questions in Section 3.2. We proved the first hypothesis that *Siamese LSTM outperforms Siamese CNN for predicting complementary products using the same text attributes*. The second hypothesis was that *More product attributes will increase the model accuracy. However, the product title is the most valuable attribute for determining the complementarity*. We did prove this hypothesis as the `title` was the most valuable attribute as expected. In addition, when combining the product title with the description, we got higher accuracy but this was not the case with the brand attribute. Lastly, by answering the third research sub-question we proved the third hypothesis, meaning that *SNNs can scale up to handle millions of data inputs and provide highly accurate solution for detecting complementarity among e-commerce products*.

5.2 Future Work

We split the future work in two parts, the Qualitative Interpretation at the company, focusing on user testing and how the experiments can be further performed for testing the direct impact to the company. Then, we will suggest improvement points regarding the data and the model as some ideas for further experiments.

5.2.1 Qualitative Interpretation at the Company

For proving the real value of the proposed pipeline to the company, we want to compare the proposed Siamese LSTM pipeline to the current approach at the company in terms of the impact on increasing the average purchase value as that is the metric by which the success of recommender systems is valued. Showing the capability of the model in real-life scenarios and proving that the manual labelling of the add-ons can be fully automated in the company is of a big importance. For this purpose, we cannot run experiments in the experimental setup and metrics we had set up during this thesis work because the current process at the company for detecting add-on products gives almost 100% accuracy as it is labelled by domain experts. However, by proving that the solution can easily scale up for millions of products and does not require human assistance, a trade-off based on the $accuracy-vs-time$ can be made. With the current labelled data available, we can only validate already existing pairs of products automatically. This means that for all new matches, the data itself cannot tell us whether the model is good or bad.

For this purpose, we propose *A/B testing*. A/B testing is a method of comparing two versions of the same page or content against each other to check which one performs better or which one increases the company's evaluation metrics more. In the case of this thesis, the split between the two versions for testing can be formulated in such way that one group of users will use the previous version of the add-ons for the *Garden and Christmas* shop. In the first group, some of the products will have "perfect" manually added add-ons (the dataset that we used in the supervised learning approach) but on the other hand for a lot of products there will be no add-ons suggested. In the second group of testing users, we will show the add-ons, which our model suggested. By doing so, after a few weeks of testing, we will be able to make a conclusion for the last experiment - whether the proposed solution increases the average purchase value as it is the parameter of highest importance when talking about qualitative interpretation at the company. In Section 4.2 we stated that the

proposed model is suggesting a lot of false positives or alternative/substitute products. This is wrong in the definition of what a complementary product is. However, we are trying to increase recall, so we expect that suggesting a few non-addons as an addition to the real add-ons products will have higher impact on the average purchase value increase than having no add-on suggestions at all. But, if we keep suggesting irrelevant add-on suggestions to the user, especially without suggesting anything interesting and truly complementary, it might backfire on the customer satisfaction and happiness level.

Therefore, if conducting A/B testing as suggested is not an option at the company, our solution can be used as a valuable starting point in the current process of detecting complementary products. Having such solution instead of the currently used business rules will significantly reduce the labelling time as we know for a fact that it will only suggest most relevant items (similar or complementary).

5.2.2 Data and Model Improvements

In Section 4 we mentioned some obstacles and problems with the labelled data that we use for training. More specifically, throughout the experiments we stated that some of the incorrectly classified candidate add-ons were due to the data quality. The problem comes from the fact that the train set consists of only positive samples, which we take as ground truth, while the negatives sample were generated with proposed methods Section 3.3.3. However, for the negative samples generation we take quite extreme cases where two products are obviously not add-ons as we are starting from the assumption that if two products were never bought together they are probably not complementary. Therefore, we omit all of the cases where similar products were bought or viewed together but in fact are not complementary.

Another interesting approach would be to take into account user click history to check for items that were viewed in the same session, usually alternative items, which are very similar in terms of titles, and take these pairs into the negative training samples. However, the scope of this thesis did not take into account any user click/history data as the focus was to explore the SNNs given pairs of products. Hence, an alternative approach for the future would be to include user click data in the negative samples generation part.

Including more product attributes is always a good idea for experimentation. Therefore, we propose including the `price` attribute in addition to the title, description and brand. The price can be a good indicator of the threshold when a product can be a substitute and when not in the cases where substitute

products were suggested by the proposed KNN solution in this thesis. Taking the examples given in Section 4.2, from the data we have in the training set it is correct that two similarly shaped vases are an add-on to the chosen oval vase, but suggesting hammocks in similar shapes is wrong when the user decided to buy a hammock. By adding the price as an attribute, this problem might be fixed as the model will be able to learn when it is accepted to suggest similar items as add-ons. Although, in reality this should not be the case as by definition, add-on suggestions should not be similar products in any case even though people might buy some of these similar products together in bulk. There are two ways of including the price as an attribute: a) the first one is by adding it as a separate feature in the model while the learning is being done and b) adding a price threshold once the suggestions from the proposed KNN solution are done. By applying the latter approach, we might be able to analyze its impact more and learn more about the data provided.

Another interesting attribute which might have high impact on eliminating the substitute products is including the products sub-subcategory. If we eliminate all of the products of the same type, the problem of suggesting substitutes can be fixed. In the example of the hammock, all products in the hammocks or swings sub-subcategory could be eliminated.

Last but not least, including image data in the categories where it is relevant might be a good idea. Aggarwal et al. [11] have done this for learning the style compatibility in furniture by combining text and image data. However, this is very costly compared to using only textual data and might not be applicable to all product categories, especially where style is not that important such as in the *Garden and Christmas* shop used in this thesis.

Bibliography

[1] Charu C. Aggarwal. *Recommender Systems, The Textbook*. Springer, 2016.

[2] *Cambridge Dictionary*. URL: https://dictionary.cambridge.org/. (accessed: 21.04.2020).

[3] *Amazon online store*. URL: https://www.amazon.com/. (accessed: 17.07.2020).

[4] *Alibaba online store*. URL: https://www.alibaba.com/. (accessed: 17.07.2020).

[5] *Zalando online store*. URL: https://zalando.com/. (accessed: 17.07.2020).

[6] Ilya Trofimov. *Inferring Complementary Products from Baskets and Browsing Sessions*. Sept. 2018.

[7] Julian McAuley, Rahul Pandey, and Jure Leskovec. "Inferring Networks of Substitutable and Complementary Products". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 785–794. ISBN: 9781450336642. DOI: 10.1145/2783258.2783381. URL: https://doi.org/10.1145/2783258.2783381.

[8] Yin Zhang et al. "Quality-aware neural complementary item recommendation". In: Sept. 2018, pp. 77–85. DOI: 10.1145/3240323.3240368.

[9] Kui Zhao et al. "Deep Style Match for Complementary Recommendation". In: (Aug. 2017).

[10] Xintong Han et al. “Learning Fashion Compatibility with Bidirectional LSTMs”. In: *Proceedings of the 2017 ACM on Multimedia Conference - MM ’17* (2017). DOI: 10.1145/3123266.3123394. URL: http://dx.doi.org/10.1145/3123266.3123394.

[11] Divyansh Aggarwal et al. *Learning Style Compatibility for Furniture*. 2018. arXiv: 1812.03570 [cs.CV].

[12] T. Zhao et al. “Improving recommendation accuracy using networks of substitutable and complementary products”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 3649–3655.

[13] Yunzeng Wang. “Joint Pricing-Production Decisions in Supply Chains of Complementary Products with Uncertain Demand”. In: *Operations Research* 54 (Dec. 2006), pp. 1110–1127. DOI: 10.1287/opre.1060.0326.

[14] Ruiliang Yan and Subir Bandyopadhyay. “The profit benefits of bundle pricing of complementary products”. In: *Journal of Retailing and Consumer Services* 18 (July 2011), pp. 355–361. DOI: 10.1016/j.jretconser.2011.04.001.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[16] Tom M Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1997. ISBN: 0070428077.

[17] Luke Dormehl. *What is an artificial neural network? Here’s everything you need to know*. URL: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network. (accessed: 01.06.2020).

[18] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. *Multi-column Deep Neural Networks for Image Classification*. 2012. arXiv: 1202.2745 [cs.CV].

[19] “A further step to perfect accuracy by training CNN with larger data”. English. In: *Proceedings - 2016 15th International Conference on Frontiers in Handwriting Recognition, ICFHR 2016*. Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR. United States: Institute of Electrical and Electronics Engineers Inc., July 2016, pp. 405–410. DOI: 10.1109/ICFHR.2016.0082.

[20] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[21] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[22] Alexis Conneau et al. *Very Deep Convolutional Networks for Text Classification*. 2016. arXiv: 1606.01781 [cs.CL].

[23] Shailza Jolly et al. *How do Convolutional Neural Networks Learn Design?* 2018. arXiv: 1808.08402 [cs.CV].

[24] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 655–665. DOI: 10.3115/v1/P14-1062. URL: https://www.aclweb.org/anthology/P14-1062.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[26] Jeff Donahue et al. *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*. 2014. arXiv: 1411.4389 [cs.CV].

[27] Kyle Martin et al. "A Convolutional Siamese Network for Developing Similarity Knowledge in the SelfBACK Dataset". In: *ICCBR*. 2017.

[28] Mustansar Fiaz, Arif Mahmood, and Soon Jung. "Deep Siamese Networks toward Robust Visual Tracking". In: May 2019. ISBN: 978-1-78985-158-8. DOI: 10.5772/intechopen.86235.

[29] *Goole BigQuery: Cloude Data Warehouse*. URL: https://cloud.google.com/bigquery. (accessed: 01.04.2020).

[30] *Deep Learning for humans.* Nov. 2019. URL: https://github.com/keras-team/keras. (accessed: 01.04.2020).

[31] *An Open Source Machine Learning Framework for Everyone*. Apr. 2020. URL: https://github.com/tensorflow/tensorflow. (accessed: 01.04.2020).

[32] *CNTK*. Apr. 2019. URL: https://github.com/microsoft/CNTK. (accessed: 02.05.2020).

[33] *Theano*. July 2020. URL: https://pypi.org/project/Theano/. (accessed: 02.05.2020).

[34] Martin Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[35] Han Jiawei et al. "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach". In: *Data Mining and Knowledge Discovery*. Ed. by Springer. 8. 2004, pp. 53–87.

[36] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: Jan. 2013, pp. 1–12.

[37] Mihajlo Grbovic et al. "E-commerce in Your Inbox: Product Recommendations at Scale". In: (June 2016). DOI: 10.1145/2783258.2788627..

[38] Flavian Vasile, Elena Smirnova, and Alexis Conneau. "Meta-Prod2Vec - Product Embeddings Using Side-Information for Recommendation". In: (July 2016). DOI: 10.1145/2959100.2959160.

[39] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 993–1022. ISSN: 1532-4435.

[40] H. Yu et al. "Complementary Recommendations: A Brief Survey". In: *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD IS)*. 2019, pp. 73–78.

[41] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. *Data Augmentation Using GANs*. 2019. arXiv: 1904.09135 [cs.LG].

[42] Marina Angelovska. *Github source code*. 2020. URL: https://github.com/marinaangelovska/complementary_products_suggestions.

[43] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[44] Ye Qi et al. *When and Why are Pre-trained Word Embeddings Useful for Neural Machine Translation?* 2018. arXiv: 1804.06323 [cs.CL].

[45] Chuanqi Tan et al. “A Survey on Deep Transfer Learning”. In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by Věra Kůrková et al. Cham: Springer International Publishing, 2018, pp. 270–279. ISBN: 978-3-030-01424-7.

[46] Andy Liaw, Matthew Wiener, et al. “Classification and regression by randomForest”. In: *R news* 2.3 (2002), pp. 18–22.