FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

An Algorithm for Multimodal Recommender Systems

João Miguel de Almeida Oliveira

Mestrado em Engenharia Eletrotécnica e Computadores

Orientador: Carlos Manuel Milheiro de Oliveira Pinto Soares

Coorientador: Rafaela Garrido Ribeiro de Carvalho

Coorientador: João Nuno Castro Gonçalves

November 28, 2022

Abstract

Recommender systems (RS) are becoming increasingly more popular in the industry. These systems are used to predict user interests, helping them with their decisions, resulting in improved user experience and increased profit.

Most recommender system algorithms are based on user-item interactions. In this dissertation, we look at the impact of adding data from different modalities to these user-item interactions with the objective of improving the recommendations.

For this objective, we chose a multimodal fashion dataset, that had image, text and tabular data. We propose an algorithm composed of 4 modality blocks: the first one extracts features from images using a Convolutional Neural Network (CNN); the second extracts features from the text of the clothes descriptions, using a Natural Language Processing (NLP) model; the third uses tabular data features that try to capture the user characteristics and purchasing behavior; and the last one that uses embeddings to simulate the user-item interaction matrix. These blocks are put at the base of a Deep Neural Network (DNN), that scores how relevant an item is to a certain user. We test every combination of the modality blocks, by coupling or decoupling them of the algorithm.

We conclude that there is a potential for increasing the RS performance when incorporating modalities into the customer-item interaction matrix. In our experiments, the proposed and implemented algorithm clearly benefited from the addition of extra modalities, nonetheless it still performed better when adding only one modality (tabular data) as opposed to multiple modalities. We also see that the image and text modality complement each other, as the algorithms that had them both activated performed better than the algorithms with only a single one of them. Nonetheless, these algorithms (with only text and images) performed worse than using only tabular data.

Keywords: Recommender Systems, Multimodality, Machine Learning

Resumo

Os Sistemas de Recomendação estão a ficar cada vez mais populares na indústria. Estes sistemas são usados para prever os interesses dos utilizadores, ajudando-os com as suas decisões, e resultando assim numa melhor experiência para os utilizadores, e mais lucro para as empresas.

A maioria dos sistemas de recomendação são baseados nas interações utilizador-item. Nesta dissertação, analisamos o impacto de adicionar dados de diferentes modalidades a estas interações, com o objetivo de melhorar as recomendações.

Para este objetivo, escolhemos um conjunto de dados de roupas multimodais, com imagens, texto e dados tabulares. Propomos um algoritmo composto por 4 blocos, um para cada modalidade: o primeiro extrai características de imagens usando uma Rede Neuronal Convolucional; o segundo bloco é responsável por extrair as características do texto, a partir das descrições das peças de roupa, usando um modelo de Processamento de Linguagem Natural; O terceiro bloco usa dados tabulares para tentar capturar as características de cada utilizador, assim como os seus padrões de compra; o último bloco, usa embeddings para simular a matriz de interações utilizador-item. Estes blocos são colocados na base de uma Rede Neural Profunda, que prediz o quão relevante um item é para um determinado utilizador. Testamos todas as combinações dos blocos de modalidade, acoplando-os ou desacoplando-os no algoritmo.

Concluímos que existe potencial para aumentar o desempenho dos Sistemas de Recomendação ao incorporar modalidades na matriz de interação utilizador-item. Nas nossas experiências, o algoritmo proposto e implementado claramente que beneficiou da adição de modalidades adicionais. Ainda assim, este teve melhor desempenho quando se adicionou apenas uma modalidade (dados tabulares) em oposição a múltiplas modalidades. Concluímos também que as modalidades de imagem e texto se complementam, uma vez que os algoritmos em que apenas estas modalidades estão ativas tiveram um melhor desempenho do que os algoritmos com apenas uma delas. No entanto, estes algoritmos (só com texto e imagens) tiveram pior desempenho do que o algoritmo que apenas usa dados tabulares.

Palavras-chave: Sistemas de Recomendação, Multimodalidade, Aprendizagem máquina.

Agradecimentos

To my supervisors, Carlos Soares, Rafaela Carvalho and João Gonçalves, for all the support and feedback across all areas of this project, as their knowledge greatly enabled and improved the success of this work.

To my family, for the continued support and motivation across all areas of my life. In specific, to my parents and sister, for their love and support, and for dealing with me in the months leading up to the end of this degree.

To my friends, because without their laughs, aid and tears, I would not have finished this chapter of my life.

And to Fraunhofer, for fully supporting me during this research project, providing me with everything needed to accomplish this project.

João Oliveira

"There is only one way to happiness and that is to cease worrying about things which are beyond the power or our will."

Epictetus

Contents

List of Figures

List of Tables

Abbreviations list

RS	Recommender Systems
ML	Machine Learning
DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
CTR	Click Through Rate
BERT	Bidirectional Encoder Representations from Transformers
AUC-ROC	Area Under the Receiver Operating Characteristic
MRR	Mean Reciprocal Rank
MAP	Mean Average Precision
DCG	Discounted Cumulative Gain
NDCG	Normalized Discounted Cumulative Gain
iDCG	ideal Discounted Cumulative Gain
FM	Factorization Machine
RMSE	Root Mean Squared Error
MSE	Mean Squared Error
MSE	Mean Squared Error
NLP	Natural Language Processing
FCL	Fully Connected Layer

Chapter 1

Introduction

Recommender Systems (RS) are an increasingly pervasive technology in many areas, including e-commerce, retail, and entertainment [9]. These systems are used to estimate user preferences for items and to proactively recommend other items that may be of interest to them [10]. These systems have two main goals, the first one is to address the growing issue of information overload, by providing customers with personalized and relevant information and helping them make decisions [11]. The second goal is to increase value, both from the perspective of the company and of the user [9].

Most algorithms are based on user-item interactions, such as item evaluations (e.g. product ratings) or user clicks or purchases of items. However, there are other kinds of data, from other modalities, that can be used for recommendation purposes. Some examples are images (e.g. product images and videos), text (e.g. product description and reviews), signals (e.g. customer pulse when faced with the product) or statistics related to the user behavior or item characteristics. Thus, recommendation systems that are able to process multimodal information may be better in capturing user preferences [3].

1.1 Goals and Contributions

In this project, we develop a multimodal approach for RS that is able to make recommendations using multimodal data such as: the user-item interaction matrix, tabular, image and text data.

The proposed algorithm is composed of 4 modality blocks that can be coupled or decoupled on the algorithm, providing a way to test performance of the algorithm with every modality combination. The proposed method was tested on a public dataset from the fashion industry. In summary, the expected results of this project are the proposal and implementation of a multimodal RS, and the empirical evaluation of the proposed method on the following research question: "Will the recommender system algorithm perform better when we add data from multiple modalities to the user-item interaction matrix?".

This dissertation contributes to the state-of-the-art in multimodal recommender systems by designing a new multimodal algorithm and evaluating the impact that each combination of modality has on its performance. The main contributions are presented below:

- A new multimodal algorithm architecture that uses the user-item interactions, text, images and tabular data as inputs.
- A dynamic method of testing the various modality combinations of the algorithm by easily switching on and off each modality.
- A study of the impact that the various combinations of modalities have on each algorithm performance and their compute/inference time tradeoffs.

1.2 Dissertation Structure

This document is structured as follows. In Chapter 2 we present all the background information related to our project. In this chapter, we explore the concept o RS and its principal algorithms, as well as the concept multimodality and its implementations in RS. In Chapter 3, we propose the architecture of our multimodal RS algorithm, and all its implementation steps. In Chapter 4 we present, describe and analyze the results of our experiments, answering to our research question. Finally, in Chapter 5 we show our principal conclusions, and we propose the future work that can be done to expand on our research.

Chapter 2

Related Work

When reviewing the literature on the state of the art in RS, we encounter papers from three different backgrounds. The first two consists of academic and scientific industry papers. These are primarily focused on the use of Deep Learning techniques, relying on these algorithms to do most of the feature engineering work. The other type of papers come from competitions held in RS conferences. These are highly dependent upon the effectiveness of the hand made feature engineering process, and mostly use Boosted Decision Trees. The examined industry papers came from successful implementations of RS models by large companies, such as Google, YouTube, Facebook, and Alibaba.

We split this Literature Review into three main topics. First, in Chapter 2.1, we explain what is a RS and how we can approach this task. In this chapter, we also explore the most common metrics used when evaluating the performance of a RS, as well as the most common ways in which to handle the data splits, for a better performance estimation. Next, in Chapter 2.2, we explain in more detail the most common algorithms that are used in multimodal recommender systems. Lastly, in Chapter 2.3, we explain the concept of multimodality, along with its key challenges and representations. We also explore the characteristics of the datasets used in the reviewed literature, as well as the different data preparation and feature engineering techniques, and also the model architectures present in these state-of-the-art papers.

2.1 Recommender Systems

This problem can be defined as follows: consider a set of users U where $u \in \{1, ..., N_u\}$, and a set of items I, where $i \in \{1, ..., N_i\}$. The interactions between U and I can be presented as a $U \times I$ matrix called R, with N user-item interactions. This matrix is very sparse, since users typically only interact with small amounts of items. Given this, the objective is to recommend a subset of the items from I that are relevant to a user u [9]. These interactions can be explicit if they rely on ratings that a user has given on items. For example, user ratings on movies that may range from 0 to 5 stars. These user-item interactions can also be implicit, if they rely on user clicks, purchases, or browsing history. For example, a user-item interaction can be labeled with 1 if the user clicked

on an item, and 0 if the user saw the item but did not click in it [10]. The ground-truth label ratings can be represented as $r_{u,i}$, and the rating predictions given by the RS algorithm are represented as $\hat{r}_{u,i}$.

This problem of recommending items to a user can be addressed in two main ways [9]:

1. *Prediction*: The objective is to predict the rating value of a user-item combination. The training data indicates user preferences for items. Therefore, the goal is to predict the scores of items that users have not yet interacted with [9].

2. *Ranking*: The objective is to predict the top-k items for a user, ordered by their relevance. This task can be addressed with models developed for the previous one (i.e. sorting the ratings according to the predicted scores for the user). However, there are methods to address the ranking problem directly, which take the relative position of the items into account [9]. This will be further discussed in the section 2.1.1.3.

In order to achieve the broader business-centric goal of increasing revenue, Recommender systems have to achieve certain goals [9]:

1. *Relevance*: This is the most important goal of a recommender system, as users are more likely to acquire an item that they find interesting [9].

2. *Novelty*: The objective is to recommend items that the user has not seen in the past. If a Recommender System only recommends items that the user has already seen or interacted with, there may be a reduction in sales diversity and user interaction [9].

3. *Serendipity*: This goal focuses on recommending somewhat unexpected items to the user, giving the feeling of lucky discovery of a new interest. Serendipity differs from Novelty because the recommendations are truly surprising, rather than some related item the user just did not know about. One of the advantages of making serendipitous recommendations, is the possibility of starting a new field of interest for the user, increasing long-term sales. On the flip side, serendipity is a very hard goal to achieve, and systems that focus on it tend to provide irrelevant recommendations. So, there is a need to have a well thought strategy between short-term and long-term sales by balancing relevancy and Serendipity [9].

4. *Diversity*: If the recommended items are all similar to each other, there is the risk that the user might not like any of them, making the recommendations essentially redundant. If the items displayed to the user are of different types, there is a greater chance that a user might like at least one of them. This is done by addressing more than one interest of the user with a single set of recommendations [9].

As previously mentioned, Recommender Systems often use all historical user-item interactions to learn the preferences of each user, which assumes that all historical user-item interaction data is equally important when recommending an item to a certain user. This assumption may not always be true, as user preferences in items tend to change dynamically. This may imply that

the RS algorithm may not only depend on long-term historical data, but must also represent those short-term preferences of the user.

Session-based RS were created to address this situation. Each session is composed of multiple user-item interactions that happened together in a certain interval of time. For example, a session may be considered from the time when a user u_1 enters a website and starts to interact with different items $\{i_1, i_2, i_3\}$ and adding some of them to the cart, until finalizing the purchase [12]. The recommendations are made considering each session interactions, instead of considering the user previous purchase history. Therefore, each session is considered as a different user.

2.1.1 Recommender Systems approaches

There are several types of different Recommender systems. In this section we explain more standard ways in which to approach this task (Collaborative Filtering, Content-based and Hybrid systems). We also study the more recent strategies used in the industry and competitions (Click-Through Rate prediction and Ranking).

2.1.1.1 Traditional Recommendation approaches

There are three main standard approaches to RS [9][10] :

1. *Collaborative Filtering*: The core idea behind this algorithm is that two users u_1 and u_2 with similar tastes will have similar ratings on items, therefore, if user u_1 has liked an item that user u_2 has not yet seen, it is very likely that u_2 will also like it [9]. These techniques may be applied in session-based RS, by comparing different sessions.

2. *Content-based*: This algorithm is based in comparing items by their descriptive attributes. If the user u_1 has liked an item i_1 with certain attributes, then he might also like another item i_2 with similar features [9] [10].

3. *Hybrid systems*: This type of model combines two or more different types of recommendation models [10].

2.1.1.2 Click-Through Rate

This approach is based on trying to estimate the click-through rate (CTR), in other words, the probability that a user will click on a certain item [13]. This approach is very used in online advertising, as these companies profit every time an add display leads to a click or purchase [7].

The problem of predicting the CTR may be defined as follows: supposing the data is composed of N instances of user-item interactions, and every u and i have several features that describe them. Each user-item interaction has an associated label $r_{u,i}$ that indicates if the user u clicked on the item i or not (0 if the user has not clicked on the item, and 1 otherwise) [13]. The objective is to predict the probability score of u clicking on the next item, $\hat{r}_{u,i}$ [14].

2.1.1.3 Learning-to-Rank approach

Ranking is a fundamental task in recommendation systems, designed to provide users with a list of items ordered according to their relevance to the user. This approach is used after an item retrieval phase, in other words, after an algorithm that selects a variety of potential interesting items from the entire item corpus. For example, after performing collaborative filtering to select many potential relevant items, we use these algorithms to sort them from the most relevant to the least [15]. Each item in the ranked list has a correspondent rank index $rank_{u,i}$. The more relevant the item is, the lower the ranked index is. We use $rel(i)$, to indicate if the i^{th} item in the list is relevant or not, 1 if relevant and 0 otherwise. These notations come in handy, when analyzing the different metrics used to evaluate these systems performance.

In [15], these ranking algorithms are separated into three different approaches: pointwise, pairwise and the listwise approach.

2.1.1.3.1 Pointwise ranking approach

This approach is similar to the CTR problem explained above, taking single user-item interactions and predicting a relevance score of the ground truth label to each one of them, $\hat{r}_{u,i}$. After that, these predictions are sorted by their relevance score, and this ranked list of items is presented to the user in that order. As a result, these models consider the user-item interactions separate from each other, thus the position of each document in the ranked list is unknown to the model when training. Furthermore, the model does not consider the dependence among items, as the loss function does not take this into account.

2.1.1.3.2 Pairwise ranking approach

In this approach, the objective is not to attempt to accurately predict the relevance of an item, but to order a pair of items based on their relevance. This task is usually performed by classification models. These algorithms output a positive value if the items are in the correct order, the first item is more relevant than the second, and negative values otherwise. This approach differs from the previous one (Pointwise ranking approach), since it takes the both items as inputs and learns how to correctly order them.

2.1.1.3.3 Listwise ranking approach

This approach takes as input a list of multiple items, and outputs either a list of relevant scores that can be sorted just like in the pointwise approach, or the sorted ranked list of items. The advantage of using this approach is that its loss function takes into account the positions of each item on the list, measuring the distance between its predicted list and the ground truth.

2.1.2 Performance estimation

When dealing with any Machine Learning (ML) algorithm, it is necessary to divide the dataset into two parts, the *training* and *test* sets. As the name suggests, the *training* set is used when training an algorithm. And the *test* set is used to test the generalization capabilities of the algorithm on new unseen data [16]. This split is made in order to ensure that it is possible to obtain a reliable performance estimate of the algorithm by detecting if it is overfitting to the data. In other words, we are trying to detect if the algorithm has the generalization capabilities to deal with new unseen data effectively, instead of only memorizing the training data, and underperforming on unseen data.

There are several ways to perform this dataset partition:

- *Holdout*: Performing a single split of the data, by randomly allocating a percentage of it for training, and the rest for testing. For example, 80% for training purposes and the rest for testing. This method can cause overfitting because, there might occur some information leakage from the training data to the test data, making the model memorize certain interactions, instead of learning how to generalize [16]. For example, if we split the data randomly, the model will train on data that says that the user u_1 has seen Toy Story 2 and 3. and then it will recommend to the user Toy Story 1, making predictions with the basis of information that happened in the future. One common way of using this method in the RS context, is to gather data from a time period of a week, and take the data corresponding to the first 6 days to train the model, and using the data from the last day as the *test* dataset. This method was applied in [4] [14] [13] [5] [17] [18] [19].

- *Leave-one-out*: Putting one of the items related to every user or session in the *test* dataset, and training the model on the rest of the data. This method has the advantage of using almost all the available data for training, and the capability of obtaining a performance estimate of the algorithm based on all users/sessions [16]. This process can be seen in [20].

- *K-fold cross validation*: Dividing the users into k different groups. Then we take one of the groups as *test* dataset, and we train the model on the rest. This process is repeated k times until every group was used as *test* dataset. The performance of the model is measured by the mean of the performances of all k models on the test set. This method is very popular because it is the method that better represents how the model would behave on unseen data. However, this method has a very high tradeoff between its performance estimation capabilities and its high computational cost needed to train all k models [16]. This method can be seen in [21].

2.1.3 Metrics

There are three principal types of metrics that are used when evaluating the performance of a recommender system: Predictive Accuracy metrics, Classification Accuracy metrics and Ranking Accuracy metrics [22].

2.1.3.1 Predictive error metrics

These types of metrics, try to estimate how close the ratings predicted by the recommender system are to the ratings given by the users [22]. One of the most common error metrics is *Root Mean Squared Error* (RMSE), defined by the following equation:

$$\mathrm{R}MSE = \sqrt{\frac{1}{N}\sum_{u=1}^{N}(\hat{r}_{u,i} - r_{u,i})^2} \tag{2.1}$$

Where N is the total amount of user-item ratings, $r_{u,i}$ is the ground-truth label, the rating that the user u gave on the item i, and $\hat{r}_{u,i}$ is the value that the model predicted for that interaction [16][22].

2.1.3.2 Classification Accuracy metrics

These metrics focus on measuring the amount of relevant and irrelevant items recommended by the model to a certain user, ignoring the ratings or ranking of the items [22].

It is considered a True Positive (TP) when the recommended item is relevant to the user ($\hat{r}_{u,i} = 1 = r_{u,i}$), and False Positive ($FP$) when the recommended item is uninteresting to the user ($\hat{r}_{u,i} = 1 \neq \mathrm{r}_{u,i}$). On the other hand, if the system does not recommend an uninteresting item ($\hat{r}_{u,i} = 0 = r_{u,i}$), we have a True Negative (TN), and if the system does not suggest an interesting item ($\hat{r}_{u,i} = 0 \neq \mathrm{r}_{u,i}$) it is considered a False negative (FN) [16][22]. In Table 2.1, we have a confusion matrix that makes it easier to visualize the relationships between the predicted outputs and its ground-truth labels. It also facilitates the visualization of the most common metrics used when evaluating the prediction performance of the algorithm.

Table 2.1: Confusion Matrix

		Ground-Truth Labels		
		Atual: Positive	**Atual: Negative**	
Predicted Outputs	**Predicted: Positive**	True Positive TP	False Positive FP	Precision $\frac{TP}{TP+FP}$
	Predicted: Negative	False Negative FN	True Negative TN	MissRate $\frac{FN}{FN+TP}$
		Recall $\frac{TP}{TP+FN}$	Fallout $\frac{FP}{FP+TN}$	Accuracy $\frac{TP+TN}{TP+TN+FP+FN}$

The *Accuracy* measures the probability by which the predictions given by the algorithm conform to their ground truth labels. It is calculated by the ratio between all correct predictions and the total amount of predictions.

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.2}$$

One of the most common metrics is *Precision*, or in other words, the *true positive accuracy*. This metric measures the probability of recommending a relevant item to the user, among all

recommended items. This is calculated by the ratio between the relevant recommended items, and the total amount of recommended items [16][22].

$$\text{precision} = \frac{TP}{TP+FP} \tag{2.3}$$

Recall, or *true positive rate*, measures the ratio between the interesting items recommended to the users, with respect to the total number of interesting items, in other words, the probability of recommending a relevant item to the user among all relevant items [16][22].

$$\text{recall} = \frac{TP}{TP+FN} \tag{2.4}$$

Fallout, is the probability of recommending an irrelevant item to the user. This metric is calculated by the ratio between the irrelevant items that were recommended to the user and the total of irrelevant items [22].

$$\text{fallout} = \frac{FP}{FP+TN} \tag{2.5}$$

In contrast, the *Miss rate* is the probability of not recommending a relevant item to a user [22], equation 2.6.

$$\text{missRate} = \frac{FN}{FN+TP} \tag{2.6}$$

The *F1-measure* is the harmonic mean between the *precision recall*.

$$\text{F1} - \text{measure} = \frac{2 \times precision \times recall}{precision + recall} \tag{2.7}$$

One of the most common metrics applied in classification tasks is AUC-ROC meaning Area Under the Receiver Operating Characteristic. The ROC (Receiver Operating Characteristic) can be expressed as a graph in which the *Recall*, or *True positive rate*, is represented in the Y axis, and the fallout, or *False positive rate* is represented on the X axis, as we can see in the figure 2.1 [1] [22]. This metric can be seen applied in [5][4][13][14][17].

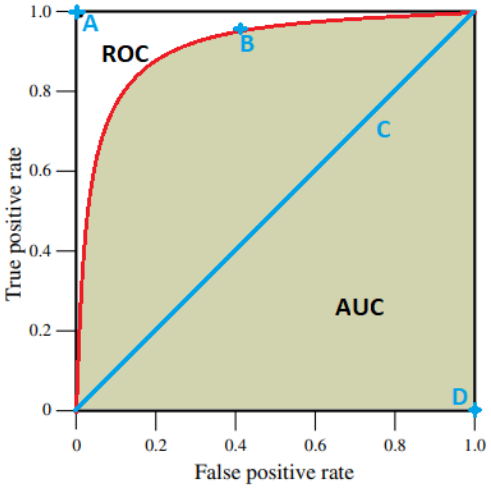

Figure 2.1: ROC graph, adapted from [1].

In binary classification, we usually define that the prediction is 1 if is grater than the 0.5 threshold, and 0 otherwise. The ROC curve is computed by varying this threshold and measuring precision and recall multiple times. By varying this threshold, we can determine the area under the ROC curve, which will give us the probability that the model is able to distinguish between a negative and positive sample [1].

If the area under the ROC curve (marked as green in the figure 2.1) is equal to 1, then it encompasses the point "A". This is the ideal situation, meaning that the system is predicting correctly 100% of the times. On the flip side, if the AUC is equal to 0, point "D", then the system is predicting the wrong class 100% of the times. The line "C" represents when the system has AUC equal to 50%, meaning that it can not distinguish any of the classes that it is predicting [1].

2.1.3.3 Ranking Accuracy metrics

When the RS provides a ranked list of items, we need to evaluate the extent to which the system can estimate the exact order of items according to a user preference [22]. One common ranking metric is the Mean Reciprocal Rank (MRR):

$$MRR = \frac{1}{N}\sum_{i=0}^{N}(\frac{1}{rank_{u,i}})^2 \tag{2.8}$$

Where N is the total number of user-item interaction samples, and $rank_{u,i}$ is the rank of the first correct (relevant) item i in the sample list for the user u. This metric was applied in [19] [21] [20] [18] in the context of ranking hotel recommendations to users.

In contrast to MRR, that only cares about the position of the first highest-ranked item on the list, the Mean Average Precision (MAP) considers the positions of all the relevant ranked items in

the list. This metric is defined by the following equation:

$$\mathrm{MAP} = \frac{1}{U}\sum_{u=0}^{N_u}\sum_{i=0}^{n} precision(i) \times rel(i) \tag{2.9}$$

Where N_u is the number of users u, n is the number of items in the list, $precision(i)$ is the precision calculated from the first to the i^{th} item of the list, and $rel(i)$ indicates if the i^{th} item in the list is relevant or not, 1 if the item is relevant and 0 otherwise [8].

Another commonly used metric is the Discounted Cumulative Gain (DCG). This metric takes into account the position and relevance of each item in the ranked list and adds a logarithmic penalty associated with each position. This is done for proportionally penalizing relevant items that are further away from the top of the ranked list. As we can see in equation 2.10, considering that the list has n items, the DCG is calculated by the sum of every relevance $rel(i)$ of every item in the ranked list, equation 2.11,and divided by its correspondent logarithmic discount, equation 2.12, associated with every position [23].

$$\mathrm{DCG} = \sum_{i=1}^{n} G_i \times n_i \tag{2.10}$$

$$\mathrm{G}_i = 2^{rel(i)} - 1 \tag{2.11}$$

$$\mathrm{n}_i = \frac{1}{log_2(i+1)} \tag{2.12}$$

DCG has a clear disadvantage, as it is impossible to compare this metric across different systems or sets of users, because it varies accordingly to the length of the recommendation lists. For example, two systems with different sized lists will end up with DCG values that can not be comparable. Therefore, we use the Normalized Discounted Cumulative Gain (NDCG) instead of the DCG. This metric is calculated by dividing the normal DCG by the ideal Discounted Cumulative Gain (iDCG), equation 2.13. The iDCG is calculated the same way as the normal DCG, but the items in the ranked list are sorted in their ideal order given by their ground truth labels. The final NDCG score is the average NDCG across all customers. Being that, the NDCG is always a value between 0 and 1. If the NDCG is equal to 1, then it means that the ranking model is having the perfect performance, as the DCG is equal to the iDCG [23].

$$\mathrm{NDCG} = \frac{DCG}{iDCG} \tag{2.13}$$

2.2 Algorithms

In this section, we analyze and briefly explain the most common algorithms used in multimodal RS.

2.2.1 Neural Networks

A Neural Network (NN) is composed of several simple and highly interconnected elements called neurons. These neurons are organized into layers, and all neurons in any given layer are connected to all neurons in the preceding and subsequent layers. There are 3 types of layers, as seen in figure 2.2:

1. *Input Layer*: There is only one Input Layer, and it is responsible for receiving the data.
2. *Hidden Layer*: A NN can hold several hidden layers stacked on top of one another, each containing different amounts of neurons. The algorithm performance may vary according to the amount of stacked layers and neurons. A NN with multiple hidden layers is called a Deep Neural Network (DNN).
3. *Output Layer*: This layer is the last in the NN. In the case of classification tasks, this layer has as many neurons as the number of classes on which the DNN is being trained on.

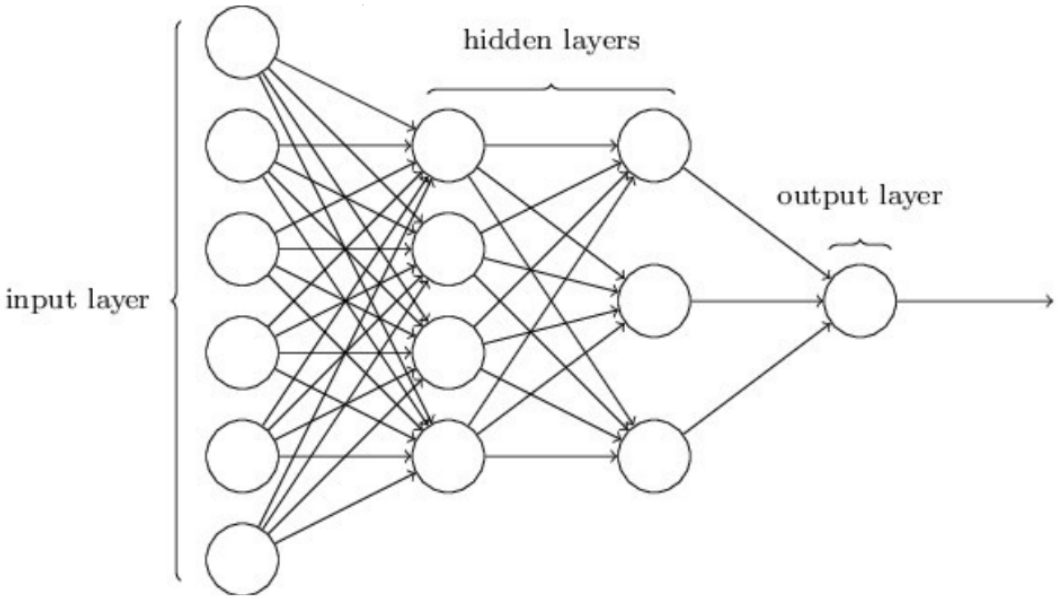

Figure 2.2: DNN architecture, from [2]

Each neuron a in layer l performs the following computation:

$$\mathrm{a}^{(l+1)} = f(\sum_{l} \mathrm{W}^{(l)}\mathrm{a}^{(l-1)} + \mathrm{b}^{(l)}) \tag{2.14}$$

where f is the activation function, most of the time rectified linear unit (RELU), and a,b and W are the inputs, bias and the weights that are multiplied by each input. These weights represent how important that specific input is for the output of the neuron. The training of the DNN consists of changing these W and b, optimizing the network in accordance with the task at hand [5].

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of Deep Learning model that is used when the input data has a matrix like shape, such as images. These models are designed to automatically learn from low-level features, such as small edges or colors present in an image, and extracting high-level features, detecting for example the features that describe an image of a dog. These networks are mostly composed of a stack of the following 3 types of layers: Convolution, Pooling and Dense layers.

1. *Convolution Layers*: These are responsible for the feature extraction, by performing the convolution of a small matrix numbers known as *Kernels* through its input, usually with $n \times n$ size, originating what is called a feature map. This process is repeated, using different kernels resulting in multiple amounts of feature maps. Each feature map represents one characteristic of the base input. For example, consider a kernel that extracts vertical lines from the input image. This kernel creates a feature map where each position has a greater or lesser value depending on the similarity of the image, with a vertical line on each point where the convolution between the kernel and the image was performed [24].

2. *Pooling Layers*: These layers do a down-sampling operation, reducing the dimensions of the feature maps. This is important, as it decreases the amount of parameters of the network. Additionally, it also makes the network invariant to translations or distortions of the inputs [24].

3. *Dense Layers*: These layers form essentially a DNN that sits on top of the first block of layers. This DNN performs the task of prediction, and takes as input the high-level features that were learned in the previous block [24].

These layers are arranged into blocks. The first block is composed of convolutional and pooling layers. These layers are commonly arranged into multiple stacks of several convolution layers followed by a pooling layer. The second block is made up of dense fully connected layers [24].

2.2.3 Decision Trees

Decision trees are one of the most common algorithms used for decision-making. As the name implies, these algorithms have a structure that looks like a tree. At the base of the tree we have the root node, each node makes a test on an input variable and crates branches that connect into two other nodes, according to the decision that was made on said test. If the node reached a decision, it stops splitting, and it is called a leaf [25].

Currently, there are many variations of the base decision tree algorithms that improve on its performance, such as Random forests and Gradient Boosted trees.

A Random Forest is an ensemble model made of multiple decision trees. In other words, these join different trees in parallel, and each one of these trees makes a prediction on the input data. The output of the random forest is based on the majority of the predictions of the individual trees [26].

Gradient Boosted trees combine multiple different small decision trees, that only have one node and two branches, in a sequence. This algorithm, ensures that each subsequent tree corrects the errors of the previous one by computing the gradient of the training objective. This process is used to produce optimal combinations of the trees [26]. The most common implementations of these algorithms encountered during this state-of-the-art analysis were XGboost [27] and LightGBM [28].

These Gradient Boosted trees can be used to perform pointwise ranking, working like a binary classifier, or to do pairwise or listwise ranking. For example, the XGboost algorithm library provides the XGBRanker, that performs these ranking approaches.

In [8], we can see these Gradient boosted algorithmS applied in a multimodal recommendation competition. Most of the top ranked competitors used these algorithms to do the scoring and ranking of the recommendations after manually generating features from multimodal data such as tabular, image and text data.

2.2.4 Factorization Machine

A Factorization Machine (FM) is used to model pairwise feature interactions, by calculating the inner product of the latent embedding vectors of a pair of features. One of the advantages of this algorithm, is that on the contrary to other models, where the features can only be trained if they have interacted with each other (appeared in the same data record), in FM these interactions are not needed, as it already computed the product of the two embeddings during training. This algorithm is mostly used when the features are sparse. In summary, FM has the ability of learning feature interactions that have never or seldom appeared in the training dataset [13].

2.3 Multimodal RS

A modality is a way in which something happens or is experienced. For example, humans interact with the word around them by hearing sounds, feeling textures and seeing things, and each one of these examples is a different modality [3]. That being said, recommender systems that use multimodal data have the potential to be better suited in analyzing the behavior of a wide range of customers. This is mainly because they may be better in capturing multiple simultaneous references of the same interest that a user may have. For instance, if a user is interested in a particular style of clothing, we may use the product description to extract the brand and the fabric materials, and the image characteristics to extract its style [11].

There are five central challenges to any multimodal Machine Learning problem [3]:

1. *Representation*: Given the heterogeneity of multimodal data, it is difficult to represent and summarize it so that they complement each other [3]. Taking the previous clothing recommendation example, the image of a product and the text that describes it are represented in two different ways, one by an RGB matrix of pixel values and the other by a text string.

2. *Translation*: How to translate data from one modality to another. For example, there is more than a single way of describing the image of a dress, depending on who is describing it. [3].

3. *Alignment*: identify relationships between different modalities. For example, the image and the description have to be converted to the same representation in order to extract relationships between them [3].

4. *Fusion*: Joining the different modalities to perform predictions. For example, joining the two converted representations of the image and the description, in order to serve them as the input features of a predictive model [3].

5. *Co-learning*: How a model that has been trained on one modality can help train and improve another model trained on a different modality [3].

2.3.1 Multimodal representations

To deal with the challenge of multimodal representations, there are two main proposed categories of multimodal representations: *Joint* and *Coordinated*, as we can see in Figures 2.3 and 2.4 [3].

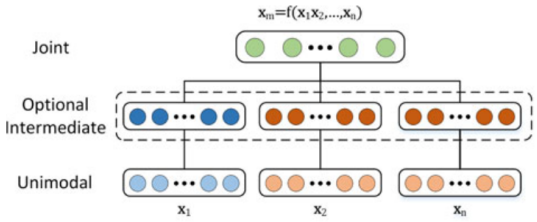

Figure 2.3: Joint representation, from [3]

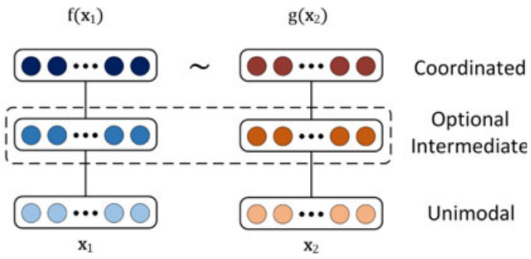

Figure 2.4: Coordinated representation, from [3]

The joint representation combines different unimodal types of data in the same representation space. That representation can be expressed mathematically through the equation 2.15 [3],

$$V_m = f(v_1, ..., v_n) \tag{2.15}$$

where V_m is the multimodal representation obtained by computing different unimodal representations $v_1, ..., v_n$, through the function f, this function in normally a deep neural network. To construct a joint representation using a neural network, each modality starts with multiple individual dense layers, these wide layers are then joined together at the base of a neural network. This neural network can have several hidden layers and an output layer that is used to make a prediction [3].

In the coordinated representation, each single unimodal data is dealt with individually, learning separate representations for each modality. However, similarity constraints are imposed on each modality, rendering them equal in a coordinated space. These similarities are very important as they make it possible to compute similarities between two modalities. For example, it is

possible to measure the distance between a word and its representative image using cosine similarity, measuring how similar the two different representations are. These models can be expressed mathematically by the equation (2.16) [3],

$$f(x_1) \sim g(x_2) \tag{2.16}$$

where there is a different function (f and g) for each modality (v_1 and v_2) and the space coordinated between them is indicated by $\sim$, [3].

2.3.2 Datasets

Machine Learning based systems require vast quantities of data in their learning process [9]. The following table 2.2 is the compilation of the characteristics of public datasets that can be used in multimodal Recommender Systems. The focus was on the collection of datasets with a large number of users, items and user-item interactions (ratings). These datasets were obtained from the reviewed bibliography and from public dataset websites. We collected the main types of modalities present in each dataset and the amount of raw features provided by each one of them [29]. By analyzing this table, we conclude that most datasets are provided by large tech companies. These industry corporations have access to the biggest datasets, as they collect them themselves. On the other hand, they do not disclose them to the public, nor do they reveal the exact amounts of data or different features that they possess, except when they create smaller datasets to be used in competitions.

When looking into the "Types of modalities" column, we can observe that the most common modalities of data is *tabular data*, meaning data that is structured in a table. Each column of the table represents a different feature, each row represents a user, item or user-item interaction, and each data entry on a row-column combination represents a feature value for said row. Another common modality is "text". This modality is represented by strings, and it is often used in item descriptions, reviews of items, names and countries. The image and Video modalities are also a common modality that is used in RS. These modalities are used to extract information from images of products or advertisements.

Table 2.2: Analysis of the characteristics of multiple RS Datasets.

Datasets	Users	Products	Ratings	Types of modalities	Raw features	Availability	Ref
Huawei - Criteo Advertizing	-	-	45 000 000	Tabular	39	No	[13]
Alibaba – Undisclosed Commercial advertising platform	-	12 2050	-	Tabular + Text + Images	9+	No	[7]
YouTube	-	-	-	Audio + Video + Images + Text + Tabular + Dates + Timestamps + Location	-	No	[4]
Alibaba - Taobao	298 349 235	12 166 060	47 556 271 927	Tabular + Location + Dates +Timestamps	-	No	[14]
Amazon	6 458 419	2 685 059	32 292 099	Images + Text + Tabular+ Dates + Timestamps	19	Yes	[29]
Trivago	730 803	927 000	1 124 000	Text + Tabular + Dates + Timestamps + Location	24	Yes	[29]
Google Local Reviews	4 567 431	3 116 785	11 453 845	Location + Tabular Timestamps + Text	15	Yes	[29]
EndoMondo	1 104	-	253 020	Timeseries + Text	15	Yes	[29]
H&M Personalized Fashion	1 371 980	45875	1 362 281	Text + Images + Tabular + Dates	37	Yes	[8]
Movielens	138493	27278	20 000 263	Timestamps + Tabular + text	16	Yes	[30]

2.3.3 Data preparation/ Feature enginnering

Data preparation is one of the most important steps when dealing with ML systems, this consists in the preprocessing of raw data and its transformation into useful features that can be feed into ML algorithms. Feature engineering, consists on leveraging these preprocessed features, and creating new ones that may provide more information to the model. Different algorithms require more or less feature engineering. For example, DNN models are designed to do most of the feature engineering by themselves. On the other hand, models based on boosted decision trees, require extensively engineered features as input. This process is made by hand, relying on different techniques [4] [5].

2.3.3.1 Tabular data

Tabular data is data that is structured in a table. These tables can store different kinds of values (features), where each column corresponds to a different feature, and each row to a different input sample. These features may need to be transformed from their raw form in order to be used in ML algorithms.

These tabular features can belong to one of two types, Categorical or Continuous features.

Categorical features are features that contain a fixed number of discrete values. One of the most common ways to encode these types of features is to use one-hot encoding. Considering that a certain feature has multiple possible discrete values, these are separated in to that many columns, each one corresponding to a different value. Each of these columns is then assigned a binary value, 1 on the column that represents the original feature value, and 0 on the rest of the columns. This

technique has the disadvantage of generating multiple different columns to represent only one column, which might generate a very large amount of features [20] [5].

As seen in [5] [6] [4] [14] [20], when dealing with DNN, it is common practice to convert this sparse and high-dimensional categorical features into a low dimensional dense vector, called embedding vector. For example, in the case of YouTube [4], they create a different embedding for each video identifier, representing the summary of its characteristics. After this, when trying to create a feature that represents the tastes of a certain user, they calculate the average of all videos that the user has already seen and feed it to the DNN.

Continuous features, in contrast to categorical features, can have any value in a certain interval. When using DNNs, it is important to properly normalize continuous features, because the scale of these features can greatly interfere with the training of such networks. With this in mind, it is necessary to scale all continuous features to the $[0,1]$ interval. The following equation shows a possible a case where we transform a rating from its current interval (for example $[0,10]$) to the $[0,1]$ interval.

$$\mathrm{r}_{u,i} = \frac{r_{u,i} - min(r_{u,i})}{max(r_{u,i}) - min(r_{u,i})} \tag{2.17}$$

where $r_{u,i}$ is the rating value, and $max(r_{u,i})$ and $min(r_{u,i})$ are the maximum and minimum values of that the rating may have. This technique is widely used, as we can see in [5] [4] [19] [21] [20] [18].

2.3.3.2 Text

Machine learning algorithms are usually not able to take as input text data. Hence, it is necessary to represent these text modality variables in a fixed length feature vector. One of the most common fixed length vector representations algorithms for text is called bag-of-words. This representation measures the presence and frequency of certain words, that are part of a known vocabulary, in a sentence. However, this representation has a few drawbacks. One of them is that it does not take into account the order in which the words were used in a sentence. Thus, different sentences that use the same words, but with a different purpose, have the same representation and are considered exactly the same [31]. There are more variants for these representations, however we will focus on deep representations, as they are the most common nowadays.

At Google, Quoc Le *et al* [31], proposed an unsupervised framework called Paragraph vector, or Doc2vec. This algorithm learns vector representations for pieces of text, addressing some of the weaknesses of the bag-of-words algorithm by preserving the semantic and context of words. This network can be applied to texts from multiple lengths, from sentences and paragraphs to large documents. This work was inspired by an algorithm called word2vec, proposed by Tomas Mikolov *et al* [32]. In this algorithm, every word in a sentence is mapped to a unique vector. These words are then concatenated, being used to predict the following word in a sentence. After the training of the model, these word vectors can be used to capture the similarities between themselves, by using simple vector arithmetic. For example, we can observe that the relation between the words “Man” and “Woman” is the same as between “King” and “Queen” by performing the following operation

$King - man + Woman = Queen$, with the vector representations of these words. In Doc2Vec the same concept is applied to paragraphs, by using the same word2vec model with an added vector called paragraph ID. This new variable is responsible for representing and memorizing the missing context information of the topic of the paragraph.

This approach has also been applied in [21] by Malte Ludewig. Where the Doc2Vec algorithm is used to obtain the vector representation of all items clicked in a given session, by assuming each sentence is a session and each word of the sentence is a different item. After getting this session vector representation, it is possible to calculate the cosine similarity between each session pairs. Using this collaborative filtering technique, it is possible to find similar sessions, and if so, recommend items that they may have not yet seen to each other.

In[17] Sumit Sidana uses a model called Bidirectional Encoder Representations from Transformers (BERT) proposed at Google by Jacob Devlin *et al* [33]. This model was used to get the feature vectors of different tweets. The BERT model uses a Transformer to learn the contextual relationships between words in a sentence. This model is called bidirectional because, in contrast to other models that read the text input from left-to-right or right-to-left, it looks at the context of a word based upon its whole surroundings.

2.3.3.3 Images

As seen in [14], Qiwei Chen *et all* proposed the use of a CNN to extract a feature vector from an image, so that it can be used jointly with other modalities of data. The design of this network consists of 17 convolution layers. The first layer uses 5x5 convolution kernels, and the following layers use 3x3 convolution kernels.

When designing such networks, it is important to consider the trade-off between performance and training time, as increasing the number of convolutional layers until a certain point, increases the performance of the network at the cost of computing time. The authors also propose to pre-train the CNN before assembling it to the rest of the model, as it decreases the training time.

2.3.4 Model architecture

Next, we will look at some RS model architectures from our reviewed literature.

YouTube's recommender system architecture, proposed by Paul Covington *et al* [4], is made up of two deep neural networks, one for candidate generation and one for ranking, as we can see in image 2.5. In this implementation, YouTube mainly uses tabular features to represent the user and video characteristics. They also use user and video embeddings, to representing their characteristics and how they interact with each other.

1. *Candidate Generation Network*: This network performs the item retrieval. It contains previously recorded events from user-to-video interactions as inputs, and extracts a small subset (hundreds) of candidate videos to be recommended to a user from a large corpus. This softmax output layer is not needed when the algorithm is being used in production, as the user

embedding vectors are learned offline, so this problem is reduced to a collaborative filtering approach by using a Nearest Neighbors Search, as we can see at the top left of the figure 2.6.

2. *Ranking Network*: Assigns a score to each video retrieved from the candidate generation network, and presents to the user the highest scoring videos, ranked by their score. During this task, the model has access to many more features describing the video and the user relation to the video, since only a few hundreds of videos are being scored instead of millions.

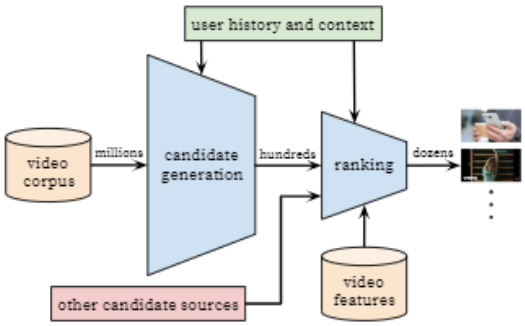

Figure 2.5: YouTube's Recommendation system architecture, from [4].

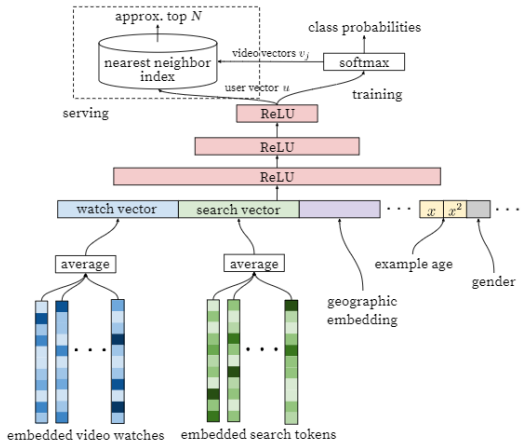

Figure 2.6: Deep candidate generation model architecture, from [4].

At Google, Heng-Tze Cheng *et al* [5], proposed a Wide and Deep approach to a recommender system. This approach attempts to tackle a big challenge in recommender systems, by trying to achieve both memorization, exploiting the frequent recurrence of items or features in historical data, and generalization, exploring new feature combinations that may have never or rarely occurred in the past. This is accomplished by jointly training a linear model component (Wide part, responsible for the memorization) and a deep neural network component (Deep part, responsible for the generalization), as shown in figure 2.7. This proposed algorithm is also used after an item retrieval phase, scoring the retrieved items and ranking them using a pointwise ranking approach.

1. *Wide part*: Represented at the left in Figure 2.7, the Wide part is used in order to achieve memorization. It is important to transform the raw input features using the cross-product transformation. This model calculates its output based upon $y = Wa + b$.

2. *Deep part*: At the right in Figure 2.7 we have the Deep part. This model consists of a Deep neural network which takes as input an embedding of sparse features. This model is used because it can generalize well to previously unseen data, by learning high dimensional feature interactions. However, it has a difficulty learning low dimensional feature interactions, capturing user specific preferences or niche items with a narrow appeal.

The combined model output is given sum of the outputs of both parts after passing it through an activation function.

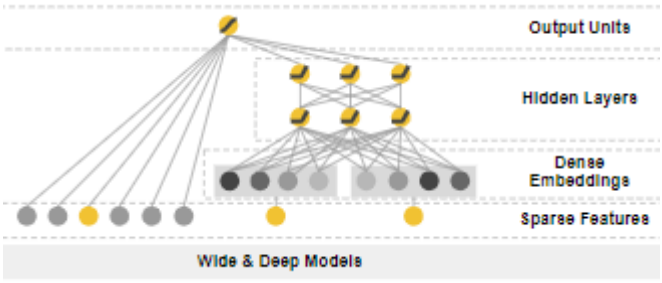

Figure 2.7: Wide and Deep model representation, adapted from [5].

At Huawei, Huifeng Guo *et al* [6], expand on the work made by Heng-Tze Cheng *et al* [5], by proposing to substitute the linear model located on the wide part of the model, by a factorization machine (chapter 2.2.4), this change is represented on the left side of the figure 2.8. This change was made in order to better capture low dimensional features interactions. Another reason for the change, was that the model proposed in [5] requires feature engineering on the input of the wide part (cross-product). In contrast, the factorization machine uses the same embedding as the deep part as its input, as we can see by in the inputs of the model represented in the figure 2.8.

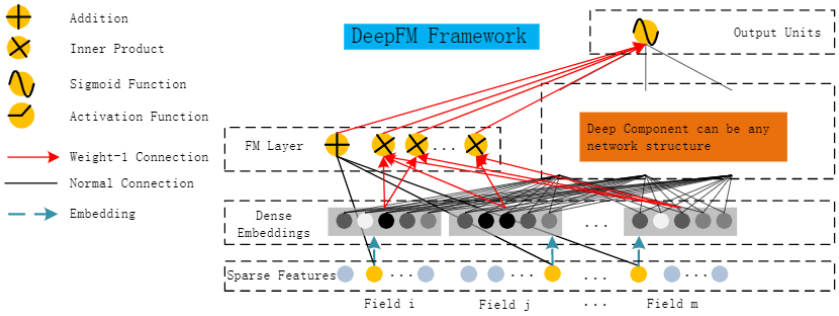

Figure 2.8: Wide and Deep model representation, adapted from [6].

At Alibaba, Junxuan Chen *et al* [7], proposed a model architecture that takes as inputs, not only sparse tabular data, but also images, for CTR prediction in the context of advertising. Since images and tabular data have different formats, they cannot be simply concatenated into one another. Given this, the authors break down the model into two different sub-networks, the *Convnet* and the *BasicNet*, that tackle a different modality each and then join them into a joint multimodal representation (chapter **??**), as we can see in figure 2.9. The *Convnet*, in the blue block at the bottom left of the image 2.9, takes an image as its input, and forwards it through a Convolutional Neural Network. The output of the last convolution layer is the feature vector of the raw image. The Basic net applies a fully connected layer to the tabular features in order to reduce their dimensionality. Finally, the outputs of these two sub-networks have the same modality and can be concatenated into a single vector and fed into two fully-connected layers to obtain the CTR prediction (top of the figure 2.9).

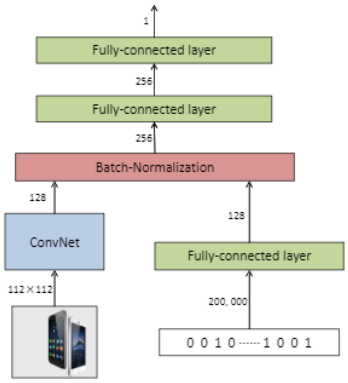

Figure 2.9: Architecture of the network, using a CNN to extract features from an image [7]

At RecSys Challenge '19, Sumit Sidana [17], Sara Rabhi *et al* [19], Malte Ludewig [21], Huang Steeve [20] and Paweł Logicai *et al* [18] all use an ensemble of boosted decision trees to preform CTR prediction and ranking, using mostly XGboost and LightGBM models. This ensemble of models architecture is very different from the joint training presented earlier in [5]. In an ensemble of models, each model is trained separately without knowing each other and their predictions are combined at prediction time by doing the weighted average of the predictions of each model. For example in [20] the authors join a DDN with a LightGBM XGboost models and do the weighted average of the predictions using the following ratios: $1 \times DNN : 7 \times LightGBM : 4 \times XGboost$. By comparison with the joint training, these models need to be larger (having more layers, neurons or leaves) in order to achieve better accuracy, which may lead to a slower prediction time.

Chapter 3

Empirical Study Multimodal Recommender Systems

The aim of this project is to develop a recommendation system algorithm which takes input from different modalities (images, text and tabular data). As seen in the previous chapter, these systems usually contain two phases, the candidate retrieval and the classifier/ranker. At first, these systems have an item retrieval algorithm that extracts a set of candidate items to a user, and then they score each one of them using a classifier/ranker with the addition of many features in order to better differentiate the user preferences. That being said, the focus of our implementation will be in the classifier/ranker algorithm, as it is the step that makes the most use of multiple features and modalities [4].

3.1 Dataset

In order to study the performance of a recommender system with multimodal data, it is necessary to choose a good multimodal dataset to train it on. That said, one dataset was selected from the collected datasets presented in Table 2.2. The following topic describes the features of this dataset and the reasons behind its selection.

3.1.1 H&M Personalized Fashion Recommendations

This dataset comes from a data science competition hosted on the Kaggle website [8], that has more than 2 960 teams. The H&M Group, the owner of this dataset, is a family of fashion brands and businesses with 53 online markets and approximately 4 850 stores. Their objective with this competition, is to enhance their user shopping experience, and to improve the sustainability of the brand by reducing returns, and thereby minimizing emissions from transportations. This is a large dataset with 1 371 980 users, 105 542 different items and 31 788 324 transactions (user-item interactions). This dataset only contains implicit and positive user-item interactions. In other words, we only have user-item interactions that resulted in purchases. Therefore, it is impossible to know what items the users saw but did not buy.

One of the reasons for choosing this dataset was the fact that it contains data from 3 modalities, images of products (Figure 3.1), text and Tabular data. Tables 3.1, 3.2 and 3.3 introduce the main features present in the dataset, referring to the Article, Customer, and Transaction data, respectively. These tables contain the names and descriptions of the features, along with an example of a dataset entry.

Figure 3.1: Examples of images from the H&M Personalized Fashion dataset [8]

Table 3.1: Article Metadata, H&M Personalized Fashion dataset [8].

Articles Metadata		
Feature name	**Description**	**Example**
article_id	A unique article identifier	108775015
prod_name	A unique identifier of every product and its name(articles may have the same name)	Strap top
product_type_name	Group of items that the article belongs to	Vest top
graphical_appearance_name	Type of pattern in the item	Stripes
perceived_colour_value_name	Color	Off-white
department_name	Department that the item is in	Jersey Basic
index_name	A unique identifier of every index and its name	Ladieswear
section_name	Section that the item is in	Womens Everyday Basics
detail_desc	Item description	Jersey top with narrow shoulder straps.

Table 3.2: Customer Metadata, H&M Personalized Fashion dataset [8].

Customer Metadata		
Feature name	**Description**	**Example**
customer_id	A unique customer identifier	00000dbacae5abe5e238858 ...
age	Customer current age	42
postal_code	Anonimized Customer Postal Code	52043ee2162cf5aa7ee79 ...
fashion_news_frequency	How often the customer receives news from H&M	Regularly

Table 3.3: Transaction Metadata, H&M Personalized Fashion dataset [8].

Transactions Metadata		
Feature name	**Description**	**Example**
customer_id	Customer identifier	00000dbacae5abe5e238858
article_id	Article identifier	108775015
price	Price of the article	0.050831
t_dat	Transaction Date	2018-09-20
sales_channel_id	Chanel through which the customer preformed the purchase	2

3.1.2 Exploratory Data Analysis

We conducted an exploratory analysis of the data on the selected dataset in order to better understand the data. We also aimed to understand potential correlations between features, in order to engineer new features that may enhance the model performance.

As seen in Figure 3.2, there is a large difference in the amount of customers between age groups, as most of the costumers are between 20 and 40 years old. This also reflects on the amount of purchases between age groups, as seen in Figure 3.3, where the customers between 25 and 50 years old have 58.8% of the total purchases. The main user preferences of articles does not seem to change between age groups, the two most bought article category groups are garment upper body and garment lower body, as portrayed in Figure 3.4. On the other hand, the next three most popular categories vary across age groups. This indicates that the age of the customer could be an important feature.

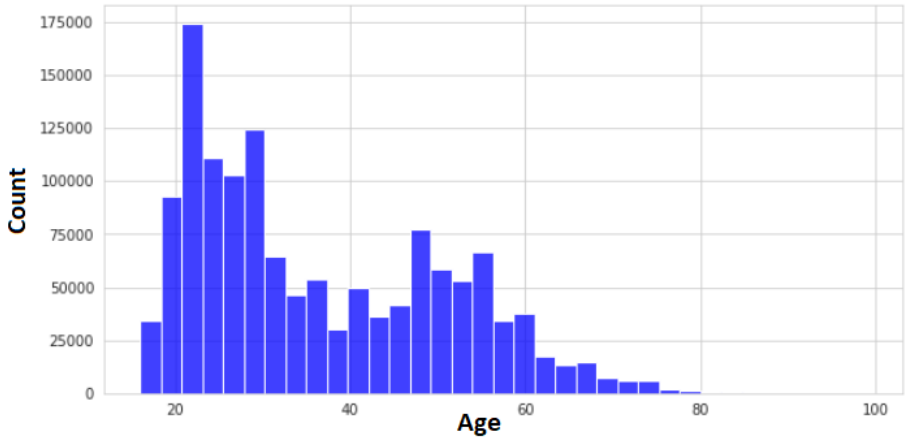

Figure 3.2: Number of customers by age group

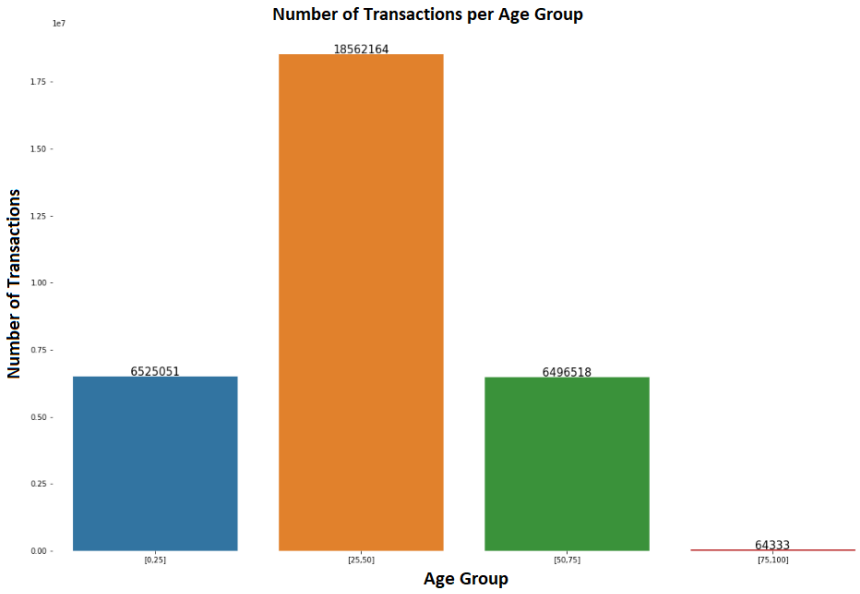

Figure 3.3: Number of transactions by age group

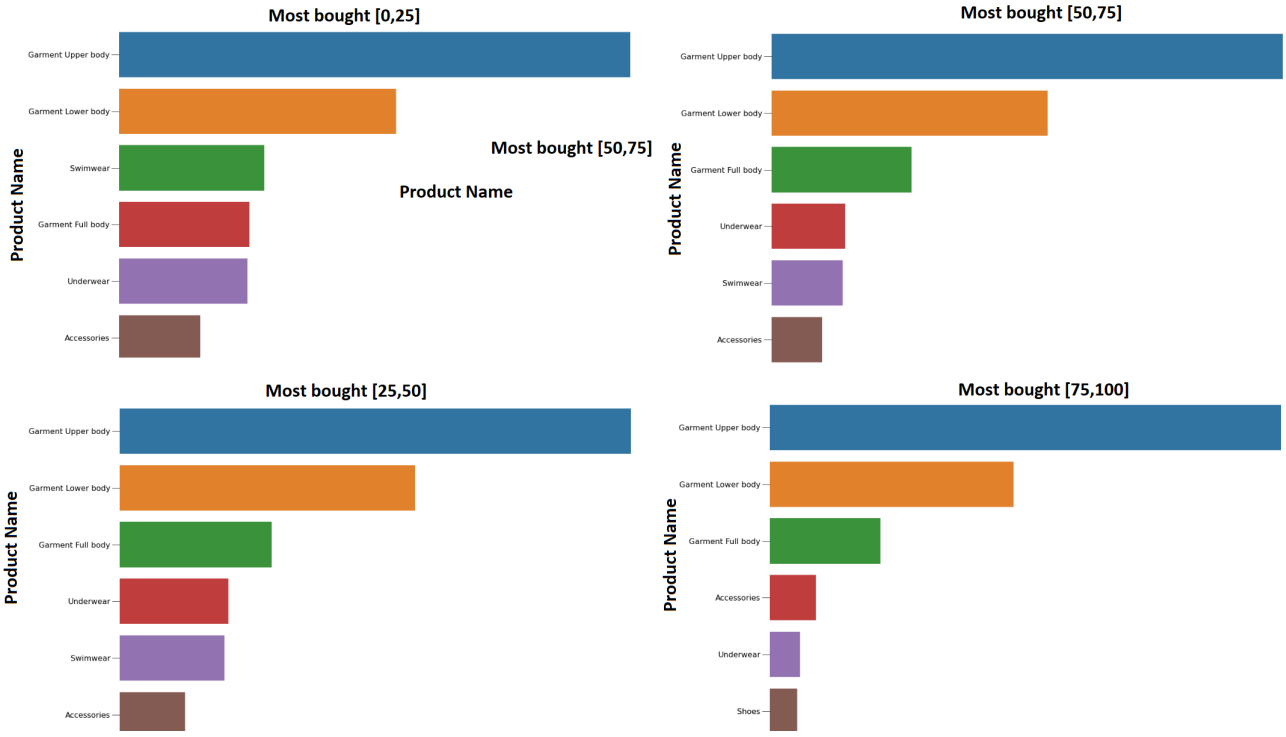

Figure 3.4: Most Bought groups of items by age interval

Figure 3.5 shows the amount of purchases per day. As we can see, there are many days when the number of purchases registered is higher than their surrounding days. This may indicate some seasonality or trends in the customer's purchasing behavior. Therefore, we may conclude that

features representing the customer's buying behavior in their past may be important to improve the algorithm.

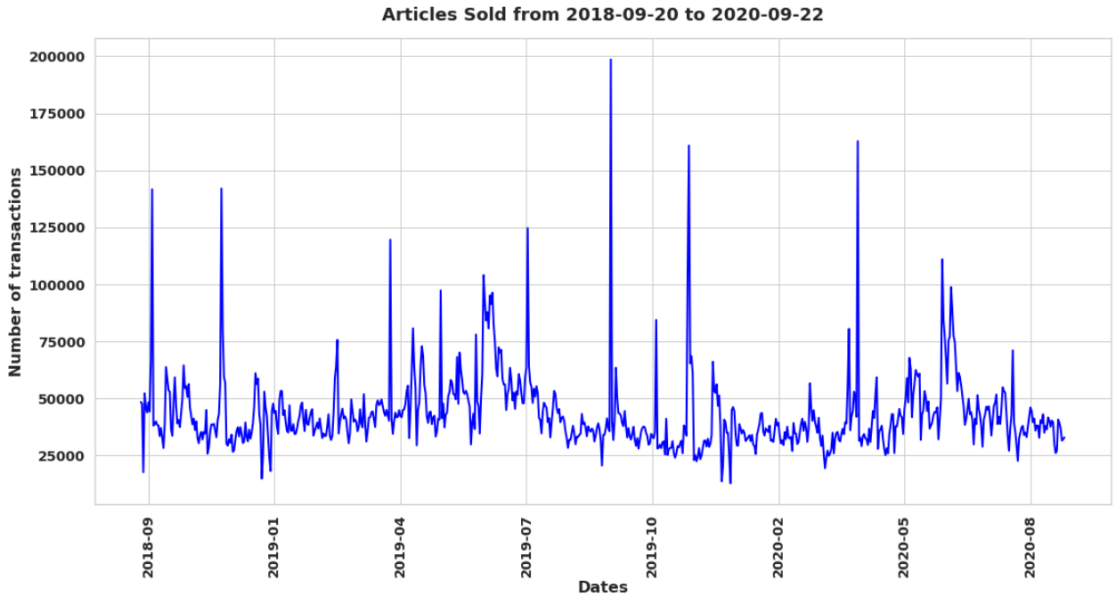

Figure 3.5: Number of transactions by age day

3.2 Model architecture

We choose to approach this recommendation problem in a pointwise ranking approach. Therefore, we implemented a classification algorithm that outputs a relevance score of a customer-article pair, and later we sorted every prediction in order to perform the ranking. We chose this approach, because it enabled a faster start of the experimentation process. This is due to the fact that we can tackle the ranking process as a binary classification problem, requiring less complexity when designing the training process of the algorithm. The model we proposed had 3 requirements:

- It had to process the 3 previously presented modalities as input data;
- Encapsulate every algorithm responsible for each modality in to a different block to be bundled into a single general model;
- It had to have the capability to easily enable the activation and deactivation of each modality block, in order to train and test every combination;

We developed or algorithm using the multimodal joint representation explained in Chapter 2.3.1, by separating and processing each modality separately, and then joining them at the base of a DNN. The Figure 3.6 illustrates the architecture of the implemented algorithm, and it is composed of five blocks. The image modality block of the algorithm is represented in yellow (bottom left), consisting of a CNN and a Fully Connected Layer (FCL). The block responsible for the text data is represented in red (second from the bottom left), and it is composed of a Natural

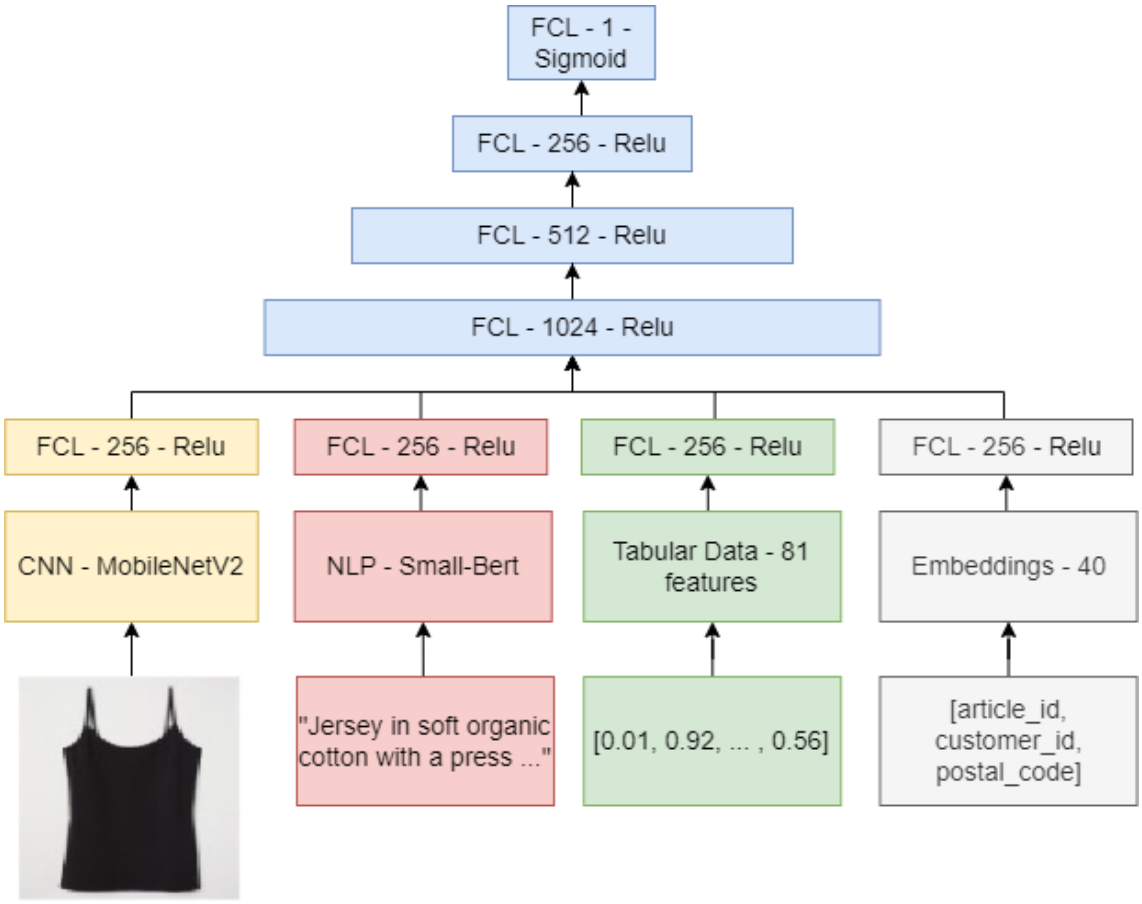

Figure 3.6: Implemented Algorithm architecture

Language Processing (NLP) model called Small-Bert, and once again, a FCL. In green (third from the bottom left), we have the tabular data block of the algorithm. Lastly, in gray (bottom right), we have the customer-item interaction block, composed by the embeddings of the identifiers of the customers and items.

These four blocks are then concatenated at the base of a DNN that is composed of three FCL. The sizes of these FCL were chosen based on the results of different tests presented in [4]. These tests were performed with a dataset of 1 million transactions. In this paper, Paul Covington *et al* [4], structured their DNN by stacking multiple fully connected layers, where the bottom layer was the widest and each subsequent layer was reduced by half the number of neurons. The results showed that the best combination was stacking 3 FCL with 1024, 512 and 256 neurons respectively. We used the Relu activation function in all FCL except for the output layer, where we used the sigmoid activation function.

3.2.1 Customer-Item interaction embeddings

This block, represented in gray (bottom right), is composed of 3 embeddings for the *article_id*, *customer_id*, and *postal_code*. As discussed before in Chapter 2.3.3.2, each value of these features

is seen as a different word to be represented into a dense vector. Similar words will end up having similar representations (ex: similar articles with different identifiers will be similarly represented). These dense vector representations are called embeddings, and each value of this vector is seen as a weight value, and is trained in the same way as fully connected layers. The size of these embeddings can be selected. When choosing the embedding dimensions, we have to take into account that higher dimensional embeddings have the potential to better represent each word, with the drawback of needing more data in training and having a larger inference time. The implemented embedding layers are 40-dimensional. We chose this value with the aim of it not being too small, not allowing for the good representation of the articles because of its low dimensionality, nor too big that it can not learn the representations. Nonetheless, this dimension value has to be tested in order to determine the impact that it has on the algorithm performance. This block simulates the customer-item interaction matrix, and therefore they are never decoupled from the model. In other words, they are used in every experiment, and they serve as the baseline model when all the other modalities are decoupled.

3.2.2 Text modality

We implemented the BERT model in order to extract features from the text metadata that describes the products. The full model architecture is explained in [33]. We chose this model because it reports better performances when compared with unidirectional NLP models, as discussed in Section 2.3.3.2.

In order to speed up the training time, we selected a pretrained model that is a smaller version of the BERT, called small-BERT. This version was selected because it has fewer and smaller transformer blocks, which allows for a faster inference time and less memory usage at the tradeoff of performance.

The architecture and the expected outputs of the text modality block of the algorithm are represented in Figure 3.7. This block is composed of an input layer that receives the text in its raw form. Then, this text in preprocessed in the next layer that converts each word of the input text in to a list of numeric token identifiers (*token ids*). These *token ids* are unique to each word, transforming the text into the input type that the model expects. This step is also important, as it filters out words that the model was not trained on. Nonetheless, we have to keep in mind that this removal of words might worsen the algorithm performance, as it eliminates information. Next, the small-BERT model takes as input the *token ids* list, and returns 3 different outputs: the *encoder_outputs* that represents all intermediate activations of the layers of the model; the *sequence_output* that represents a contextual embedding for each word of the sentence; and the *pooled_output* that represents the embedding of the entire context of the input text. The output vector dimensions are (1, 512). In this implementation, we only use the *pooled_output*, and we feed it into a FCL with 256 neurons, in order to reduce its dimensionality.

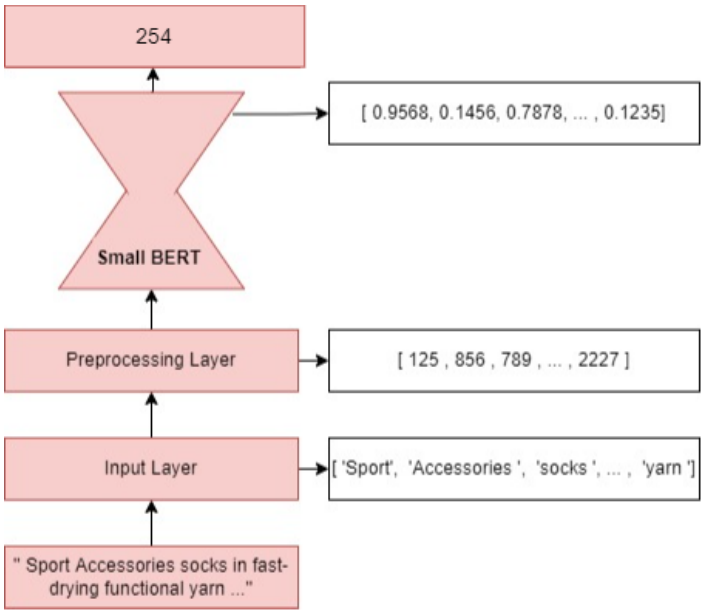

Figure 3.7: Text modality architecture

3.2.3 Image modality

In order to extract information from images, we followed the same architecture adopted in [14]. For that, we selected a pretrained CNN model, replacing its fully connected layers by a single one with 256 neurons. As explained in Chapter 2.3.3.3, this is done in order to retain all the feature extraction capabilities of the CNN, by keeping all the convolution layers of the model and using them to perform another task that is different from the one that it was originally trained for.

Because of computational restrains, a lightweight model called MobileNetV2 was selected in order to speed up the training process and have a shorter inference time. This CNN was especially produced for mobile and resource constrained environments, significantly decreasing the number of operations and memory needed during inference, while remaining as accurate as other larger models. The full architecture explanation of this algorithm can be found in [34].

3.2.4 Tabular modality

In this block, represented in green in Figure 3.6 (third from the bottom left), we take all tabular features described in Chapter 3.2.5.3, and we feed them into a fully connected layer with 256 neurons. We have a total of 81 features, which try to capture the customer and item characteristics and buying behavior.

3.2.5 Data Preparation/ Feature Engineering

As discussed before, data preparation and feature engineering crucial steps when dealing with ML algorithms, as most data cannot be fed into these algorithms in its raw form. We applied

feature engineering in 5 different steps: image preprocessing, text preprocessing, tabular feature generation, feature encoding and negative sampling.

3.2.5.1 Image Preprocessing

MobileNetV2 expects images with shape $224 \times 224 \times 3$ and pixel values normalized between -1 and 1. However, the images present in the dataset have varied shapes, and its pixel values range between 0 and 255. Therefore, all images were resized to $224 \times 224 \times 3$, and converted into the desired pixel value range. Figure 3.8 shows the images before and after the resizing operation. This resizing operation, deforms the shapes of the clothes in the not squared shaped images, which may cause problems in extracting characteristics from them.

Figure 3.8: Image Resizing

3.2.5.2 Text Preprocessing

As illustrated in Tables 3.1, 3.2 and 3.3, we have many features represented in strings, such as color, department graphical appearance, item description, etc. In order to extract information from these text features, we have joined all of these into a single string in order to be fed into the BERT model. The following text is an example of this process:

"Lawrence Reflective Low Sock 3 Socks Socks & Tights Solid Black Dark Black Men Sport Acc Sport Men H&M Sport Accessories Sports socks in fast-drying functional yarn with ventilating hole-knit sections over the feet and an ankle-height shaft with reflective details. The polyester content of the socks is recycled"

3.2.5.3 Tabular features generation

For each transaction, we extracted several new features in order to add aggregated behavior and time related information to each purchase. These features we created are based on 3 time windows: one based in all historical data, one based in the 5 weeks prior to each transaction, and the last one based on the week prior to each transaction. The generated features are presented in Table 3.4:

Table 3.4: Engineered features.

Feature Name	Description
day	Day of the month that the transaction occurred
month	Month that the transaction occurred
years_since	How many years since the purchase
days_since	How many days since the purchase
week	Week number of the purchase
price_mean_last	Mean of the prices of all products that a certain customer has bought in the last week
price_hist_max	Price of the most expansive item that the customer has ever bought
price_hist_min	Least expansive item that the customer has ever bought
rate_sales_channel_hist	Mean channel that from which the user has bought the items
rate_sales_channel_recent	Mean channel that from which the user has bought the items in the last 5 weeks
n_buy_hist	How many different items the customer has ever bought
n_buy_recent	How many different items the customer has bought in the last 5 weeks
n_buy_last	How many different items the customer has bought in the last week
n_buy_hist_prod	How many purchases items customer has made ever
n_buy_recent_prod	How many purchases the customer has made in the last 5 weeks
n_buy_hist_ptype	How many items the customer has bought with the same type as this product ever
n_buy_recent_ptype	How many items the customer has bought with the same type as this product in the last 5 weeks
n_buy_hist_graph	How many items the customer has bought with the same graphical appearance as this product ever
n_buy_recent_graph	How many items the customer has bought with the same graphical appearance as this product in the last 5 weeks
n_buy_hist_col	How many items the customer has bought with the same color as this product ever
n_buy_recent_col	How many items the customer has bought with the same color as this product in the last 5 weeks
n_buy_hist_dep	How many items the customer has bought from the same department as this product ever
n_buy_recent_dep	How many items the customer has bought from the same department as this product in the last 5 weeks
n_buy_hist_group	How many items the customer has bought from the same group as this product ever
n_buy_recent_group	How many items the customer has bought from the same group as this product in the last 5 weeks
n_buy_hist_sec	How many items the customer has bought from the same section as this product ever
n_buy_recent_sec	How many items the customer has bought from the same section as this product in the last 5 weeks
n_buy_hist_garm	How many items the customer has bought from the same garment group as this product ever
n_buy_recent_garm	How many items the customer has bought from the same garment group as this product in the last 5 weeks
days_after_buy_prod	How much timer has passed since the customer bought the last product
days_after_buy	How much time has passed since the customer bought this product
days_after_buy_ptype	How much time has passed since the customer bought an item with the same type as this product
days_after_buy_graph	How much time has passed since the customer bought an item with the same graphical appearance as this product
days_after_buy_col	How much time has passed since the customer bought an item with the same color appearance as this product
days_after_buy_dep	How much time has passed since the customer bought an item from the same department as this product
days_after_buy_group	How much time has passed since the customer bought an item from the same group as this product
days_after_buy_sec	How much time has passed since the customer bought an item in the same section as this product
days_after_buy_garm	How much time has passed since the customer bought an item with the garment group as this product

3.2.5.4 Feature Encoding

As reported in Chapter 2.3.3.1 we used Equation 2.17 to transform the values of all continuous features to the interval [0,1]. We applied normalization to the training data, and then used the maximum and minimum values to transform the validation and test data. We also proceeded to categorize all string features like the "customer_id".

3.2.5.5 Negative Sampling

As we discussed before, all transactions (user-item interactions) are implicit and positive, meaning that we only have the user-item interactions that resulted in a purchase. Knowing this, it is necessary to generate negative user-item interactions, items that the user might have seen but decided not to purchase. This is a necessity because if we trained the model with only the positive interactions, it would simply always predict the maximum score as the output, and would not generalize in the real world.

Hence, we generated ten negative user-item interactions for each user in each week that they performed a purchase. From these ten negative samples, five of them are items chosen at random

from the top 100 most purchased items from the week before. These samples are intended to simulate the most likely items that the user may have seen but decided not to buy, these items were selected at random from the top 100 to ensure that the model does not overfit by memorizing the items. The last five items are completely random out of all the articles available in the dataset.

As this process drastically increases the size of the dataset, computing costs and training time also increase. As a result, we choose ten as the number of negative samples, attempting to find a compromise between these two factors.

3.3 Performance Estimation Methodology

To proceed with the training and test of the performance of the algorithms, we first had to split the dataset into three parts: train, validation and test, as explained in Chapter 2.1.2. We only used a single fold of the data in this performance estimation process, because the training time of the algorithms took longer than anticipated. We used the holdout method, splinting the data considering the time evolution as explained in Chapter 2.1.2. The dataset has a total of 104 weeks of data, thus we used the interval between [5, 100] as training data, the week number 101 as validation data, and 102 to 104 as test data. We decided to downsize the dataset to 50% of the users, and to exclude all cold start users (users that have a very small amount of transactions in the training data, or that only appear in the validation or test weeks). This was done because the training time of the algorithm was unreasonably long, and it would not have been possible to train all modality combinations inside the time constrains of this dissertation. This is because the dataset became very large with the addition of the negative samples. Table 3.5 shows the amount of transactions in the final train, validation and test datasets, as well as the amount of customers and articles within these transactions.

Table 3.5: Dataset distribution.

Dataset	Transactions	Customers	Articles
Train	2 650 173	63 833	34 846
Validation	228 650	19 194	20 517
Test	620 976	32 920	27 009

We trained the algorithm with 12 modality combinations of tabular, text and image modalities plus the baseline algorithm. In specific, we could couple or decouple the tabular modality block, as well as the text modality part. Finally, in order to study the impact that the different modalities have on the performance of the algorithm, we created a baseline model that has all modality blocks decoupled, and only uses customer-item interaction embeddings. This baseline algorithm consists of the embeddings of the *customer_id*, *article_id* and *postal code*. As explained in Chapter 3.2.1, these embeddings are always coupled with the model across every experiment.

Each combination was trained for a maximum of 15 epochs, or until the validation loss stopped to decrease, in order to avoid overfitting. We selected a performed ranking with a pointwise

ranking approach, as explained in Section 2.1.1.3.1, using binary classification models to predict scores, and sorting them after the training of the algorithms.

During training, we used Mean Squared Error (MSE) as the loss function, and we tracked the RMSE (Equation 2.1) and accuracy (Equation 2.2) of each epoch. Although MSE is often used in cases where the customer has given a rating score to each item (0 to 5 stars ratings, for example), we used this same metric in this binary classification case. This was possible because we considered the binary ground truth labels 0 and 1 as possible ratings given by the user. We used the MSE to compare the continuous scores that the DNN outputs with these binary labels. Nevertheless, given that we are approaching this recommendation problem by doing binary classification, in the future it is important to perform these experiments using the Binary Cross-entropy loss, as it is better suited to this approach.

In order to perform the ranking step, we predicted the relevance scores for every transaction in the test data. We grouped these transactions into ranked lists, where each list corresponds to a set of transactions that occurred in a given day for each user. We sorted these lists by their predicted score. The ranking lists were limited to 10 transactions each, because we considered that to be a sufficient amount of recommendations.

For the purpose of assessing and comparing the performance of the algorithms, we computed several metrics: NDCG, AUC, RMSE, Precision, Recall, TP, TN, FP and FN.

3.4 Experimental pipeline

Figure 3.9 represents the structure of the Machine Learning pipeline developed for this project. This pipeline enables fast experimentation, by dividing the full process into several customizable steps. At the start of each step, some configurations can be customized by changing flags and variable values, allowing for quick and easy parameter experimentation.

The first stage covers all the *feature engineering* procedures described in Chapter 3.2.5. It takes as input the original raw dataset, and converts it to the shape that the model is expecting. In this step, it is possible to configure the number of negative samples per customer per week, the amount of weeks used in training, the percentage of users to use (reducing the dataset size) and how to split the data between *train*, *validation* and *test* datasets. This processed data is stored separately from the original data, in order to allow for more experiments with different parameters.

Next we have the *model configs* block. This block is used to perform the training experiments, by changing between the different modalities combinations after every training cycle. It is possible to configure everything concerning the training and evaluating of the algorithms, such as what *train*, *validation* and *test* datasets are to be used, as well as the number of epochs, batch size and patience (number of epochs before forcing the training to stop), and others.

The *training* block is composed of three different blocks. The first block is the *data loader*, it receives as input the configurations defined in the *model configs* block, and it is responsible to load the datasets into memory selecting only the features that are needed for the current experiment. For example, if the experiment only needs tabular data, then the text features and images will not

be loaded, enabling faster training times. This module is also responsible for creating the batches that will be fed into the model during training, and to shuffle the data after each epoch.

The *model training* block, is responsible for the setup and training of the algorithms. It switches on or off the different modality blocks, depending on the experiment that is being performed.

Lastly, we have the *Evaluation* block, that computes all the test metrics of the algorithm. This block also stores the model, and the computed metrics in a database. This guarantees that the data from the experiments is never lost, as well as allowing the reproducibility of the tests of each trained model.

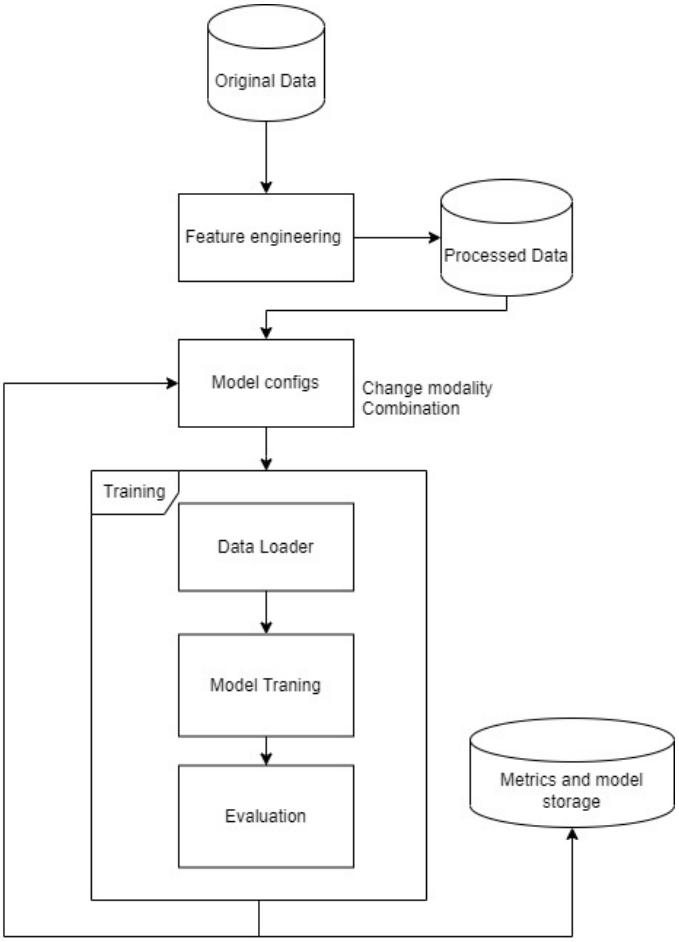

Figure 3.9: Experimental pipeline

3.5 Implementation Details

We developed this project using the Python 3 language. All data wrangling and feature engineering was done using the Pandas and Scikit-learn libraries. We used GitHub to do code versioning. Finally, we used the TensorFlow library to develop, train and test the algorithm.

Chapter 4

Evaluation

In this chapter, we present the results of the experiments with the proposed algorithm on the H&M Personalized Fashion Recommendations [8] dataset. In Section 4.1, we present the performance results of all experiments with the multiple modality combinations. Next, in Section 4.2, we formulate and discuss three hypotheses, in order to better answer to our research question. Next, in Section 4.3, we evaluate the training and inference time across all experiments. Finally, in Section 4.4, we do the final analysis of our experiments, answering to our research question.

4.1 Results

To test our research question "Will the recommender system algorithm perform better when we add data from multiple modalities to the user-item interaction matrix?", we trained and evaluated a different model for every modality combination. The metrics that we computed to perform these tests are compiled in Table 4.1. Each table row contains a different experiment that can be identified by the flags present in the *Tabular*, *Text* and *Images* columns. To increase the readability of the table, we organized its rows into 3 groups, each group corresponds to the combination of 0, 1, 2 or 3 modalities. The ✓symbol indicates that we are using that modality for the experiment, and the ✗ indicates the opposite. In the case of the experiments with the image modality, the ✓symbol indicates that we are training the CNN, and the *dt* (meaning "don't train") symbol indicates that we are using the pre-trained CNN without training it.

As discussed before, the metrics that we computed belong to two different categories, ranking and classification metrics. The classification metrics, such as Precision, Recall, RMSE, AUC, TP, TN, FP, and FN, are used to evaluate the correctness of each prediction given by the algorithm. Nonetheless, these classification metrics might not be sufficient, as what really matters is not that the model can accurately classify if a user-item pair has a 0 or 1 score, but if it can correctly order a list of items from the most to the least relevant.

Table 4.1: Computed metrics for every experiment

Number of modalities	Tabular	Text	Images	NDCG (%)	AUC (%)	RMSE	Precision (%)	Recall (%)
0	×	×	×	48.5	47.7	51.0	28.8	14.2
1	✓	×	×	**69.4**	**68.0**	**45.9**	**47.9**	**41.5**
	×	✓	×	48.5	46.6	49.9	26.9	10.9
	×	×	✓	49.4	45.9	51.0	26.8	16.7
	×	×	*dt*	49.7	48.9	49.6	26.0	14.0
2	✓	✓	×	56.3	52.8	46.3	43.5	1.3
	✓	×	✓	49.0	47.5	49.7	17.0	6.2
	✓	×	*dt*	58.3	55.9	52.4	33.1	38.5
	×	✓	✓	**52.1**	**53.9**	**47.7**	23.0	16.2
	×	✓	*dt*	50.6	49.5	48.9	**26.1**	**22.9**
3	✓	✓	✓	**57.0**	**57.5**	**46.3**	25.2	**2.3**
	✓	✓	*dt*	54.9	52.8	47.4	**45.4**	2.1

For this analysis, we chose to focus primarily on one ranking metric, and we complement this analysis with three classification metrics. For the ranking process, we compare the NDCG across all algorithms, as it measures the capability of scoring the most relevant items higher than the least relevant ones [23]. We compare the AUC of the different algorithms, as it measures the probability that the algorithm will score a positive sample higher than a negative one [1]. We also compare their precision, measuring the probability of recommending a relevant item to a user among all recommended items. We also analyze the recall, as it tells the percent of relevant recommended items that the algorithm recommended across every relevant items [16].

4.2 Research question

As said before, our aim is to answer the following question: **"Will the recommender system algorithm perform better when we add data from multiple modalities to the user-item interaction matrix?"**. To help us answer this question, we formulated 3 different hypotheses:

- *Hypothesis 1*: Adding data from a single modality will improve the performance of the algorithm.
- *Hypothesis 2*: Adding data from more than one modality will improve the performance of the algorithm.
- *Hypothesis 3*: Ignoring tabular data, the text, and image modalities complement each other.

Hypothesis 1

Adding data from a single modality will improve the performance of the algorithm:

We start this analysis with the baseline model, when all modality blocks are turned off, and we only have the customer-item embeddings block (corresponding to the user-item matrix). By

looking at the NDCG values of Table 4.1, we can conclude that it has the lowest NDCG value. This means that whenever we add a new modality to the algorithm, we will be adding useful information that will enhance performance.

Hypothesis 2

Adding data from more than one modality will improve the performance of the algorithm:

The best performing model is highlighted in green. This model only uses the tabular data block. Looking at the NDCG values, it is 11.1 percent points higher than our second-highest performing model, and 20.9 percent points higher than the baseline. This means that when we add the text and image modalities to the model, we decrease its performance. Therefore, we can conclude that, in our experiments, adding more than one modality did not increase the model performance.

These results are consistent with the top 15 solutions from the Kaggle competition that this dataset is from. In these solutions, they only used tabular data instead of combining CNN and NLP models into their algorithms. Rather, they engineered tabular features that represented similarities across different item images and text descriptions [8]. We can clearly see that the other computed metrics also support this analysis, as it has the largest AUC value of every experiment, as well as the lowest RMSE value.

Hypothesis 3

Ignoring tabular data, the text and image modalities complement each other:

Now, we will be looking only at the models that do not have tabular data, as the models that have tabular data outperformed every other modality combination. Starting with the model that only have a single modality combination, we can observe that when using the image modality, we achieve 1.2 higher percent points in NDCG than the model that only uses text data. We also observe a small NDCG difference of 0.3 percent points between when we train the CNN or not. This difference is too small, so it is likely to be a random effect.

When looking at the combination of the text and image modalities (without tabular data), we observe that these have a higher NDCG than the ones with a single modality, highlighted in yellow in Table 4.1. There is also a 1.5 percent points difference between the pre-trained and the trained CNN, with advantage for the latter. The NDCG difference between the best model with 2 modalities (text and images, training the CNN) and the best model with one modality (images without training the CNN) is 2.4 percent points. This difference also reflects itself in the other computed metrics, 5 percent points in AUC and 1.9 RMSE difference.

A similar difference in NDCG, between training or not training the CNN, can be seen when looking at the model that use all three modalities, highlighted in blue in Table 4.1. In this case, we see a 2.1 percent points difference of NDCG between not training or training the CNN, also with the advantage for the latter. Which means that there is an increase in performance when we

fine-tune the CNN. If we see the other computed metrics, there is a difference of 4.7 percent points in AUC and 1.1 RMSE.

From this discussion, we can conclude that even though the addition of the image and text modalities to the model only seems to worsen its performance, when they are separated from the tabular data, they complement each other, favoring the cases of when we train the CNN.

4.3 Training and Inference Time Evaluation

As in any system, the tradeoff between predictive performance and computing time is very important, in order to get the most value from the time and computing cost that was invested into the system. Therefore, we decided to record the mean epoch time, the total training time, and the inference time of each experiment. The results of these experiments are displayed in Table 4.2.

The training of the models was made using two separate machines, therefore the training times can not be directly compared across machines. These two machines were not equivalent in terms of hardware, and Machine B has a better GPU than Machine A, which reduces its training time. We also have to keep in mind that these machines were not being exclusively allocated to the training of these algorithms, as they were being used simultaneously by other colleagues to perform other tasks. Nonetheless, we can clearly see the patterns in training time across experiments. We did the experiments that had the same number of modalities, and with the image modality active in the same machine, providing a more correct comparison in the tradeoff between training and not training the CNN. All the inference time tests were made using the same machine, so they can be directly compared between each other. For these inference time measurements, we took the average prediction time across 50 different predictions.

Table 4.2: Inference/Training time results

Number of modalities	Tabular	Text	Images	Mean epoch time (hours)	Machine	Total training time (hours)	Inference time (ms)
0	×	×	×	1.28	A	6.6	31.7
1	✓	×	×	1,04	B	5,2	29.1
	×	✓	×	6,59	A	32.9	53.0
	×	×	✓	6,72	A	26,9	82.1
	×	×	*dt*	4.16	A	20.8	83.7
2	✓	✓	×	6.45	B	23.3	54.3
	✓	×	✓	6.77	B	32.9	98.8
	✓	×	*dt*	3.52	B	17.6	92.2
	×	✓	✓	12.34	A	61.6	104.6
	×	✓	*dt*	7.20	A	36.0	117.3
3	✓	✓	✓	12.83	B	64.2	108.2
	✓	✓	*dt*	7.72	B	38.6	104.7

By inspecting the table, we can clearly see the difference between the experiments with no modalities or with only tabular data, and the rest of the experiments. These two models, took much less time than the models with other modality combinations, being several hours shorter.

The model that only contains the tabular data modality has the clear advantage of having better predictive performance, needing much less training time than any other combination. This makes it possible to experiment multiple variations of it, like varying the number of tabular features or architectures, in the same interval of time that any other model took for training only one model. The inference time values also follow this trend, being clearly shorter than the rest of the modalities combinations.

We can also clearly see that when using the image modality, the training time is considerably reduced in the cases where we do not train the CNN. This reduced training time comes at the cost of the performance of the model, as we have seen is the previous section. The difference in inference time of these two cases does not seem to be significant.

Comparing the cases where we have the image modality (training the CNN) or the text modalities active, we can observe that the training times are very similar. Nonetheless, the model that have the text modality and no image modality, need less inference time when compared to the ones with image modality and no text modality. As seen in the previous section, this diminished inference time comes at the cost of performance when compared with the models with image modality and no text modality. However, if we compare the cases where we have the image modality without training the CNN, we can see that it takes considerably less training time than the model with only the text modality, having better performance at the cost of inference time.

By comparing the epoch and inference times for the combinations of the tabular modality with the models that use text or images, we conclude that tabular data does not seem to add any significant additional time.

Comparing the inference time of each group of modality combinations, we can observe that it increases when we combine the image and text modality.

4.4 Final Evaluation

In summary, the model performs better when we only use tabular data, and when we add other modalities its performance decreases. When looking at the models that only have a single modality, we have an increased performance when compared to the baseline model. In the case of single modalities, we also see that the use of images performed better than text and the difference between training or not the CNN was very small. The models with two modalities have better performance than the single modality models. Furthermore, it is beneficial to train the CNN as it allows for an increase in performance. This same performance increase occurred when looking at the algorithms with 3 modalities.

Looking at the training and inference times, the tabular data model outperforms all other combinations while having the lowest training and inference times. We can also conclude that, taking into consideration the training time, it is preferable to use a pretrained CNN when compared with the models that use text data, as it has better performance and takes less training time.

Will the recommender system algorithm perform better when we add data from multiple modalities to the user-item interaction matrix?.

From our experiments, we can conclude that our algorithm performed better when using only a single modality (tabular data). However, we cannot conclude that the use of multiple multimodal data is useless, as we only tested this hypothesis on a single dataset and with a single algorithm architecture. In order to give a definitive answer to this question, we need to do tests on different datasets and different architectures. However, we can see that the baseline algorithm that only used the user-item interaction matrix clearly benefited from the addition of extra modalities.

Chapter 5

Conclusions

Recommender Systems are really useful in this day and age. These provide users with the most relevant items, increasing the business revenue and increasing sustainability by decreasing item returns, and therefore reducing waste.

The objective of this dissertation was to investigate the impact of using multiple data modalities and their combination in the development of a RS.

We proposed a recommender system algorithm that takes input data from multiple modalities, such as tabular data, text, and images in addition to the customer-item matrix. We used the multimodal joint representation, separating and processing each modality separately, and then joining them at the base of a DNN. Our algorithm is composed of 4 modality blocks, that encapsulate every algorithm responsible for each modality. The algorithm is designed in such a way as to facilitate easy experimentation of different modality combinations, by easily switching on or off each one of them. We evaluated the ranking and classification capabilities of this algorithm for every modality combination. From these experiments we could conclude that, in our case, the use of multiple modalities worsened the model performance, as our best performing model was the one that only used a single additional modality (tabular data). However, we see potential in the use of multimodal data, as the models that combined text and images performed better than the ones with these modalities separate. From our experiments, we also see an increase in performance when we retrain the CNN, which gives us an indication that if we also retrained the NLP model, the model performance might also increase. The addition of multiple modalities also considerably increases training and inference time of the models when compared to our best performing model (tabular data).

Finally, the study of the impact of multimodal data on recommender systems still needs more experiments with more datasets and different algorithm architectures in order to make a definite answer to our research question.

5.1 Future Work

For future work, we intend to test our proposed algorithm in other multimodal datasets. We also intend to try the following changes to the proposed algorithm in order to evaluate if those increase its performance:

- Trying other popular and heavier CNN and NLP models, in order to evaluate if it brings performance to the algorithm;
- Changing the depth of our DNN, as we only tried one DNN configuration;
- Changing the number of neurons in our FCL and embeddings, to see if the model benefits of larger or smaller sizes;
- Changing our ranking paradigm from pointwise to listwise ranking, making the model learn how to rank instead of learning how to score user-item pairs.
- Retrain or NLP model instead of only using a pretrained version of it, because our experiments indicate that it will increase the algorithm performance.
- Test other architectures and datasets, in order to provide a definitive answer to our research question.

References

[1] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, 6 2006. [Online]. Available: https://doi.org/10.1016/J.PATREC.2005.10.010

[2] S. Akthar, "A study on neural network architectures," vol. 7, 2016. [Online]. Available: www.iiste.org

[3] T. Baltrusaitis, C. Ahuja, and L. P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 423–443, 2 2019. [Online]. Available: https://doi.org/10.1109/TPAMI.2018.2798607

[4] P. Covington and J. Adams, "Deep neural networks for youtube recommendations," youtube. [Online]. Available: http://dx.doi.org/10.1145/2959100.2959190

[5] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. S. G. Inc, "Wide deep learning for recommender systems," 6 2016. [Online]. Available: https://arxiv.org/abs/1606.07792v1

[6] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, and Z. Dong, "Deepfm: An end-to-end wide deep learning framework for ctr prediction." [Online]. Available: https://arxiv.org/abs/1804.04950

[7] J. Chen, B. Sun, H. Li, H. Lu, and X. S. Hua, "Deep ctr prediction in display advertising," *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pp. 811–820, 9 2016. [Online]. Available: https://arxiv.org/abs/1609.06018v1

[8] "Hm personalized fashion recommendations | kaggle." [Online]. Available: https://www.kaggle.com/c/h-and-m-personalized-fashion-recommendations/overview

[9] C. C. Aggarwal, *Recommender Systems*. Springer International Publishing Switzerland, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-29659-3

[10] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys*, vol. 52, 7 2017. [Online]. Available: http://arxiv.org/abs/1707.07435http://dx.doi.org/10.1145/3285029

[11] V. K. Singh, S. Sabharwal, and G. Gabrani, "Comprehensive analysis of multimodal recommender systems," pp. 887–901, 2021. [Online]. Available: https://doi.org/10.1007/978-981-15-8530-2_70

[12] S. Wang, L. Cao, Y. Wang, Q. Z. Sheng, M. Orgun, and D. Lian, "A survey on session-based recommender systems," vol. 9, p. 39, 2019. [Online]. Available: https://arxiv.org/abs/1902.04864

[13] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for ctr prediction," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 0, pp. 1725–1731, 2017. [Online]. Available: https://doi.org/10.24963/ijcai.2017/239

[14] Q. Chen, H. Zhao, W. Li, P. Huang, and W. Ou, "Behavior sequence transformer for e-commerce recommendation in alibaba," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 5 2019. [Online]. Available: https://arxiv.org/abs/1905.06874v1

[15] T. Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, pp. 225–231, 2009. [Online]. Available: https://doi.org/10.1561/1500000016

[16] E. Campochiaro, R. Casatta, P. Cremonesi, and R. Turrin, "Do metrics make recommender algorithms?" *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pp. 648–653, 2009. [Online]. Available: https://doi.org/10.1109/WAINA.2009.127

[17] S. Sidana, "Leveraging user embeddings and text to improve ctr predictions with deep recommender systems," 2020. [Online]. Available: https://doi.org/10.1145/3415959.3415995

[18] P. J. Logicai, L. K. Logicai, P. S. Logicai, and M. W. Logicai, "Boosting algorithms for a session-based, context-aware recommender system in an online travel domain," *Proceedings of the Workshop on ACM Recommender Systems Challenge*, 2019. [Online]. Available: https://doi.org/10.1145/3359555.3359557

[19] S. Rabhi, T. S. Evry, F. W. Sun, J. Perez, N. Seattle, U. R. M. B. Kristensen, N. Copenhagen, D. J. Liu, N. Pittsburgh, U. E. Oldridge, N. Vancouver, W. Sun, M. R. B. Kristensen, and J. Liu, "Accelerating recommender system training 15x with rapids," 2019. [Online]. Available: https://doi.org/10.1145/3359555.3359564

[20] H. R. steeve and rosettaai Yi-Fu, "A-ha: A hybrid approach for hotel recommendation," 2019. [Online]. Available: https://doi.org/10.1145/3359555.3359560

[21] M. Ludewig, "Learning to rank hotels for search and recommendation from session-based interaction logs and meta data," 2019. [Online]. Available: https://doi.org/10.1145/3359555.3359561

[22] G. Schröder, M. Thiele, and W. Lehner, "Setting goals and choosing metrics for recommender system evaluations," *Proceedings of the RecSys 2011 Workshop on Human Decision Making in Recommender Systems*, 2011. [Online]. Available: https://www.researchgate.net/publication/268381252

[23] T. Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, pp. 225–231, 2009. [Online]. Available: https://doi.org/10.1561/1500000016

[24] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, pp. 611–629, 8 2018. [Online]. Available: https://doi.org/10.1007/S13244-018-0639-9/FIGURES/15

[25] "Overview of use of decision tree algorithms in machine learning," *Proceedings - 2011 IEEE Control and System Graduate Research Colloquium, ICSGRC 2011*, pp. 37–42, 2011. [Online]. Available: https://doi.org/10.1109/ICSGRC.2011.5991826

[26] J. Heaton, "An empirical analysis of feature engineering for predictive modeling," 2020. [Online]. Available: https://arxiv.org/abs/1701.07852v2

[27] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 13-17-August-2016, pp. 785–794, 3 2016. [Online]. Available: https://arxiv.org/abs/1603.02754v3

[28] "Lightgbm: A highly efficient gradient boosting decision tree." [Online]. Available: https://doi.org/10.5555/3294996.3295074

[29] "Recommender systems datasets." [Online]. Available: https://cseweb.ucsd.edu/~jmcauley/datasets.html

[30] "Movielens | grouplens." [Online]. Available: https://grouplens.org/datasets/movielens/

[31] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, pp. 2931–2939, 5.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1 2013. [Online]. Available: https://arxiv.org/abs/1301.3781v3

[33] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, 10 2018. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[34] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 1 2018. [Online]. Available: https://arxiv.org/abs/1801.04381v4