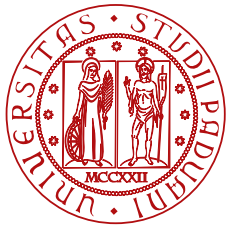


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master Thesis in Computer Engineering

Design and Evaluation of Joint Information Retrieval and Recommender Systems

Master Candidate

Simone Merlo

Student ID 2076747

Supervisor

Prof. Nicola Ferro

University of Padua

Co-supervisor

Dott. Guglielmo Faggioli

University of Padua

Academic Year
2023/2024

Date: 03/09/2024

Abstract

Information Retrieval (IR) and Recommender Systems (RecSys) represent two fields of research that are constantly growing since the necessity to satisfy user's information needs in large collections of documents and resources is becoming more and more important. IR and RecSys traditionally are two separate fields and two distinct kind of systems. In this thesis, we explore how IR and RecSys can be joint together in order to better address users' information needs. In particular, we studied the literature of this innovative field, with a major focus on the current state-of-the-art framework that is the Unified Information Access (UIA) framework and we understood the major issues related to this domain. To do this, we reproduced the work related to the UIA framework, we tested UIA to seek for potential problems in terms of architecture and/or dataset preprocessing and we performed several experiments, adapting and modifying the system, to give a proof of the found issues or to understand if we could improve its performance.

Sommario

L'Information Retrieval (IR) e i Recommender Systems (RecSys) rappresentano due ambiti di ricerca in costante crescita in quanto la necessità di soddisfare il bisogno degli utenti di trovare informazioni in grandi collezioni di documenti e risorse sta diventando sempre più importante. Tradizionalmente, l'IR e i RecSys corrispondono a due ambiti separati e a due tipi di sistemi distinti. In questa tesi, si esplora come l'IR e i RecSys possono essere combinati in modo da soddisfare più efficacemente i bisogni dell'utente. In particolare, abbiamo studiato la letteratura riguardante questo ambito innovativo, focalizzandoci principalmente sul sistema che attualmente è considerato essere lo stato dell'arte, ovvero lo Unified Information Access (UIA) framework e abbiamo capito i problemi e le mancanze principali di questo dominio. Per fare tutto questo abbiamo riprodotto il lavoro riguardante l'UIA framework, abbiamo testato il sistema per trovare eventuali problemi in termini di architettura e/o di manipolazione dei dataset e abbiamo svolto vari esperimenti, adattando e modificando il sistema, per fornire una dimostrazione dei problemi trovati o per capire se potevamo migliorare le sue prestazioni.

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

IR Information Retrieval

RecSys Recommender Systems

UIA Unified Information Access

KS Keyword Search

QBE Query By Example

CIR Complementary Item Recommendation

ANN Approximate Nearest Neighbor

APN Attentive Personalization Network

SE Search Engine

ML Machine Learning

DL Deep Learning

NNs Neural Networks

BERT Bidirectional Encoder Representations from Transformer

GPTs Generative Pre-Trained Transformers

nDCG Normalized Discounted Cumulative Gain

MRR Mean Reciprocal Rank

1 Introduction

Every people has some information need that can be related to work tasks, personal interests, entertainment or other reasons. Up to two decades ago the way in which humans shared and had access to information was based on libraries and on word of mouth. However, when computers and the internet started to become popular everything changed because these new technologies allowed to share large amount of documents. In order to be able to look for some specific piece of information inside the huge collection of documents (which was constantly growing), some systems started to be created and used: Search Engines (SE). Later, people began to use the web also for commercial purposes and this favored the development of Recommender Systems (RecSys).

Search Engines are mainly focused on obtaining the most suitable piece of information from a collection of documents, based on the information need of the user. Historically, these systems were though to work with text, thus, the information need was a short textual string, while the documents composing the collection corresponded to textual documents. Nonetheless, nowadays SEs are used everywhere and they are able to work also with images, videos and other types of information. Google, Bing, DuckDuckGo are some classical example of SEs but also when search for something inside some website (*e.g.* a product on amazon) we are using a Search Engine. The research field that is concerned with studying how to effectively retrieve information is Information Retrieval (IR).

Recommender Systems, instead, are strongly based on the users, in fact, they aim to suggest the most relevant documents from a collection based on the user previous interaction. Nowadays, we use RecSys very frequently, in fact, when

we enter a website and perform some operations we are suggested with other actions that we could do or with other pages that we can visit. For example if you watch a video on YouTube then the site recommends to you some other related videos or if you buy a product on Amazon or eBay than the page shows to you other similar products.

Today it is quite common that the results of IR systems and RecSys are combined together, in fact, when you search for something on Google, in addition to the standard IR results you obtain also some sections/boxes that contain the output of RecSys. The results of these two types of systems are starting to be displayed together in order to be able to provide the user with a more complete answer to their information needs. Nonetheless, historically IR and RecSys have been developed independently but since the nineties some researchers argued that there were several common aspects between these two research fields. Based on these considerations, in this thesis we analysed the novel and promising topic of "joint retrieval and recommendation" and in particular we focused on the system/model that is the current state-of-the-art that is the Unified Information Access (UIA) framework.

The aim of this thesis was to reproduce and deeply analyse the UIA framework in order to discover its strengths and weaknesses so that it is possible to fix/adapt it to use it as a starting point to develop working systems or some new "joint IR and RecSys" models. In Chapter 2, we first focus on describing what IR, RecSys and "joint IR and RecSys" models are, how they work and how they are evaluated. Furthermore, since the current state-of-the art models for both IR and RecSys are based on neural networks and, in particular, on the architecture of the transformer, in Chapter 2 we also introduce all these aspects and how they can be applied to IR and RecSys. In Chapter 3 we deeply explain the architecture and the optimization process of the UIA framework. In addition to that, in this chapter we focus also on the dataset that we used and on how we processed it in order to be able to use it to train the framework. In Chapter 4 we talk about how we were able to reproduce the work of UIA paper and about the main discrepancies between what was explained in the paper that introduced UIA and what we were able to discover from the code shared by the authors. Eventually, in Chapter 5 we explain the potential issues that we found in the architecture of the framework and/or in the datasets used and we reported some of the tests and the experiments that we performed.

2

Related Work

Nowadays everyone has some kind of information need that can be related to some work task, entertainment or other fields. In the vast majority of cases to address these necessities everyone relies on the web technologies and, in particular, the core components that are used are IR systems and RecSys.

In this chapter, in Sections 2.1 and 2.2 we will explain how IR and RecSys work, respectively. In Section 2.3 we will present the most common matching models for IR/RecSys. In Section 2.4 we will introduce the fundamental concepts of dense retrieval and the related neural network concepts. In Section 2.5 we will provide an overview about the history, the motivations and the structure of joint IR and RecSys models. Eventually, in Section 2.6 we will report and explain the main measures used to evaluate IR and/or RecSys.

2.1 Information Retrieval

IR systems are widely diffused and are used everyday by millions of people to find answers to their information needs. In Section 2.1.1 we describe the main structure and purpose of such systems while in Section 2.1.2 we provide an overview of the methodologies that are used in IR.

2.1.1 Information Retrieval Systems Purpose & architecture

IR is mainly concerned with retrieving the most relevant documents given some user input. In particular, as can be seen from Figure 2.1 an IR system takes

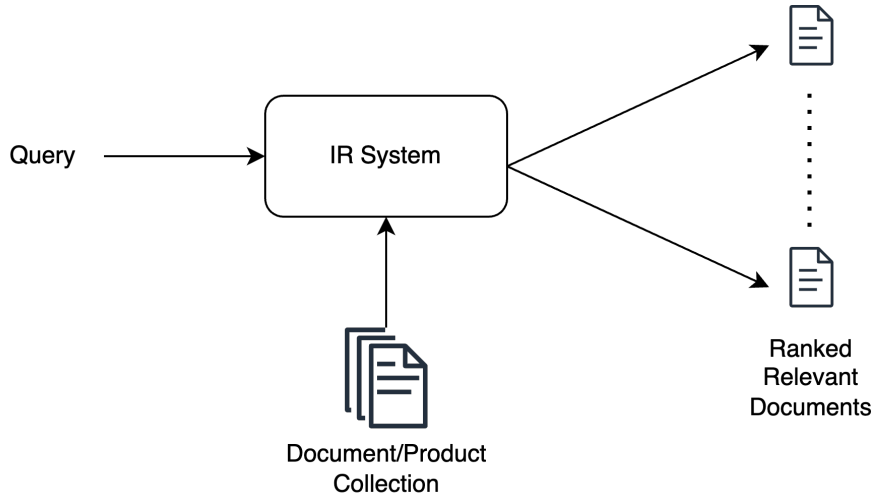


Figure 2.1: Base structure of IR systems.

as input a query and a document collection. The query is usually a short textual string (sequence of keywords) which represents the information need of the end-user while the document collection is a set of textual documents and corresponds to the set of all possible pieces of information available and in which the user might be interested. In real applications the documents could be some websites, some products or some other elements depending on the domain of the application. The IR system provides in output a list of ranked relevant documents that contains the most relevant documents in the collection with respect to the query provided in input, ordered from the most to the least relevant. A classic example of IR is reported in Figure 2.2 which represents the everyday use that we do of a Search Engine (SE) like Google, Bing etc.

2.1.2 Information Retrieval Methodologies

IR systems need to retrieve the most relevant document in a collection according to a user query (see Section 2.1.1 for more details). To do this, those system apply some pre-processing to the document collection in order to be faster and more accurate. In particular, this pre-processing transforms the document collection into an index which represents a way of storing data that uses specific data structures, allowing to reduce the time needed to perform search operations. When a user provides a query in input to the system, the query is also processed in order to be able to use it to access the index and retrieve the most relevant documents according to it.

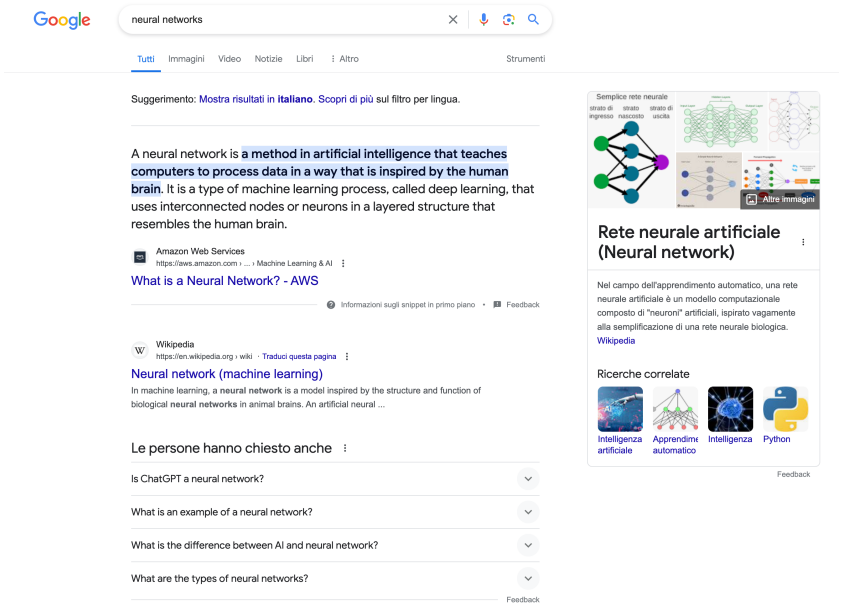


Figure 2.2: IR example.

To preform all these operations, there are several techniques that IR systems can use. The main approaches adopted by IR systems are:

- Keyword based search.
- Vector based search.

In keyword based search the textual documents are split into tokens (that could be single words or even sub-words), these tokens are processed, filtered and then the index is created on the basis of those. The queries, when submitted to the system, are processed in a similar way to the documents and to compute the relevance of the documents to the queries some statistical metrics are used like term frequency (tf) and inverse document frequency (idf) (see Section 2.3).

In vector based search the documents in the collection are processed and turned into some vector representations, usually exploiting neural networks (see Section 2.4). The index is then created on the basis of those vectors. The queries are also mapped to some vector representations and to compute the relevance of the documents to the queries, usually, metrics based on the spatial distance between vectors are used (*e.g.* dot product, cosine similarity). Normally, the documents corresponding to the closest vectors to the one representing the query are the most relevant (see Section 2.3).

2.2 Recommender Systems

RecSys are frequently used by every user but usually people do not perceive that they are making use of those systems because, in the wast majority of cases, RecSys don't require an explicit user input. In Section 2.2.1 we describe the main structure and purpose of such systems while in Section 2.2.2 we provide an overview of the methodologies that are used by RecSys.

2.2.1 Recommender Systems Purpose & Architecture

RecSys are mainly concerned with retrieving the most relevant products given some input. In particular, as can be seen from Figure 2.3, RecSys take as

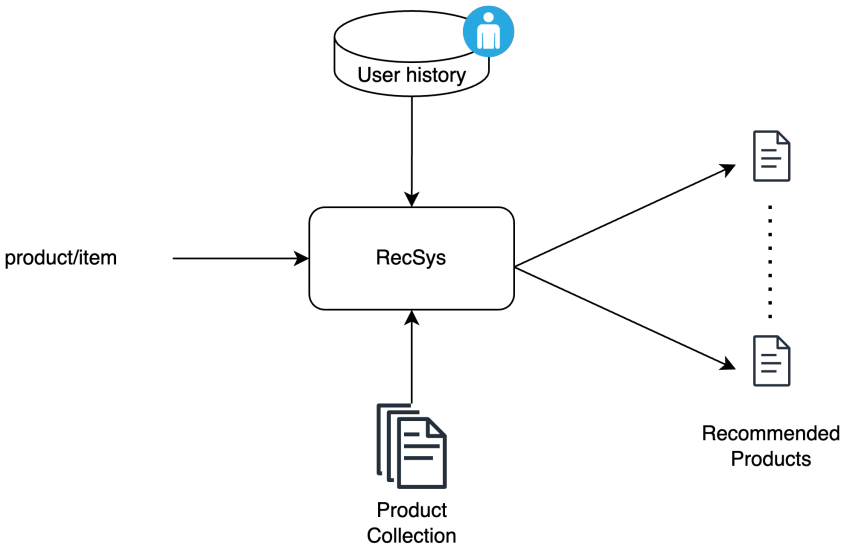


Figure 2.3: Base structure of RecSys.

input a product/item, some user history and a collection of products/items. The product/item usually represents an element that has been clicked/bought by the user and it is the item on the basis of which the system performs the recommendation. The user history represents a list of items/products with which the user had some interactions in the past (clicked or purchased items). This element plays an important role since it is used to personalize the recommendation based on the user previous interactions and this allows to improve the performance. The products/items collection represents a set of items available and in which the user might be interested. Furthermore, the product/item can be both an element with which the user had the interaction

that triggered the use of the recommender system or one of the products/items present in the user history, thus, in most of the cases, the recommendation is not performed because of an explicit user request but it is executed as a consequence of some user action. RecSys provide in output a list of ranked relevant products/items that contains the most relevant products/items in the collection with respect to the product/item and user history provided in input, ordered from the most to the least relevant.

Recommendation could have different purposes, in fact, for example, the relevant products provided as output by RecSys could be product similar to the item provided input (*e.g.* if we click on the product *"Iphone 15"* the system recommends also *"Iphone 15 Pro"*) or complementary to the item provided input (*e.g.* if we click on the product *"Iphone 15"* the system recommends also *"Iphone 15 charging cable"*), thus RecSys should be designed considering these different aspects of recommendation.

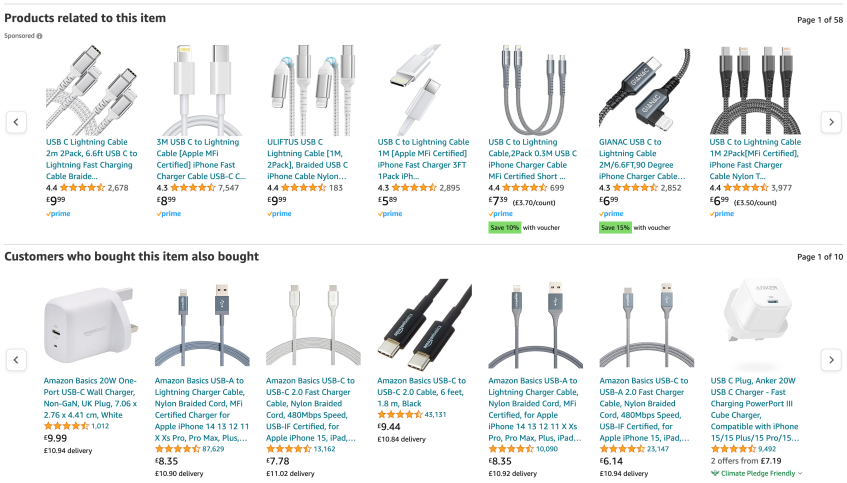


Figure 2.4: RecSys example.

An example of the output provided by RecSys is reported in Figure 2.4 which shows the results of the recommendation computed after we clicked on a "iPhone usb-c cable" product on Amazon. As can be seen from Figure 2.4, there are both similar (other cables) and complementary (power adapter) recommended products.

The products/items used in recommendation could be very heterogeneous depending in the domain of the application, for example, they can be films, songs, physical products, web sites, etc..

2.2.2 Recommender Systems Methodologies

RecSys retrieve the most relevant products/items according to a product/item and the user history (see Section 2.2.1 for more details). To do this, those system usually exploit neural networks to map the products into some vector representations. The vectors representing the products in the collection are normally stored using specifics data structures in order to create an index that allows to improve the accuracy and the performance. The products representing the user history are also converted into some vectors and stored in this format. When a product/item is provided in input to a recommender system it is mapped into the corresponding vector representation and the vector is then modified according to the user history. The most innovative techniques used for personalization (tuning the vector representing the input product according to the user history) make use of concepts like attention (see Section 2.4.6 for more details). To compute the relevance of the products/items in the collection with respect to the product/item provided in input, RecSys exploit the index containing the vector representations of the entire collection and the personalized version of the vector representation of the input item and, in particular, they usually compare vectors using metrics based on the spatial distance (*e.g.* dot product, cosine similarity). Usually, the products corresponding to the closest vectors to the one representing the input item are the most relevant.

2.3 Matching Models

As we explained in Sections 2.1 and 2.2 in the pipeline of IR and RecSys the requests (queries, items or products, queries in the following) representations need to be compared with the representations of each of the documents/products (documents in the following) in the collection, in order to compute the relevance scores (R), which are then used to order the documents by relevance. This process is called matching and can be done in several ways depending on the application and/or on the way in which we represent the requests and the elements in the collection. In Section 2.3.1 we report a traditional term weighting technique while in Sections 2.3.2 and 2.3.3 we report two types of matching procedures.

2.3.1 Term Weighting

Term frequency (tf) and inverse document frequency (idf) are two measures that are widely used to compute some weights for each term in a collection. These measures play a fundamental role because they are somehow related to the matching techniques that have been developed, especially for the keyword based approaches. The formulas used to compute tf and idf are reported in equations 2.1 and 2.2, respectively, where with i we denote the $i-th$ term in the vocabulary containing all possible terms in the collection, with k we refer to the $k-th$ item in the collection, with t we indicate the total number of unique terms in the collection, with $f_{i,k}$ we represent the frequency of term i in document k, with N we denote the total number of documents in the collection and with n_i we refer to the number of documents containing term i.

$$tf_{i,k} = \frac{\text{count of term } i \text{ in document } k}{\text{number of terms in document } k} = \frac{f_{i,k}}{\sum_{j=1}^{t} f_{j,k}} \tag{2.1}$$

$$idf_i = \log_2 \frac{N}{n_i} \tag{2.2}$$

As can be seen from equations 2.3 and 2.2 the term frequency represents the probability of finding term i in document k while inverse document frequency represents the inverse of the fraction of documents that contain at least one occurrence of term i. The equation used to compute the term weights is: $w_{i,k} = tf_{i,k} \cdot idf_i$ and since by using Equation 2.1 to compute tf a term appearing twice in the same document would be given twice the weight we define a new way of computing tf that is reported in Equation 2.3.

$$tf_{i,k} = \begin{cases} 1 + \log_2(f_{i,k}) & \text{if} \quad f_{i,k} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Thus, the final formula that will be used to compute the actual weight for term i in document k is reported in Equation 2.4.

$$w_{i,k} = \begin{cases} (1 + \log_2(f_{i,k})) \times \log_2(\frac{N}{n_i}) & \text{if} \quad f_{i,k} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

2.3.2 Vector Based Matching

When we represent the requests (queries) and the elements in the collection by means of vectors (ordered sequences of numbers), there are several techniques that can be used to compute the relevance score of a document in the collection for a certain queries. Independently from the matching method that is used, the requests representations must have the same size of the collection elements representations. For example, we can use vectors having size (V) equal to the number of possible terms in the vocabulary ($\mathcal{V}$), which contains all the terms that a system can encounter, so that each element of the vector corresponds to the number of times that each term appears in the query/document (normalized by the total number of terms in the query/document).

In vector based matching the cosine similarity is frequently used to match the vectors and Equation 2.5 shows how it can be computed. In the equation we denote with q the query (request), with d_k the $k-th$ document (item) of the collection, with $\vec{q}$ and $\vec{d_k}$ the corresponding vector representation, with q_j the $j-th$ element of the query vector and with $d_{j,k}$ the $j-th$ element of the vector of the $k-th$ document in the collection.

$$R(q, d_k) = \cos(\theta(\vec{q}, \vec{d_k})) = \frac{\vec{q} \cdot \vec{d_k}}{\| \vec{q} \| \, \| \vec{d_k} \|} = \frac{\sum_{j=1}^{V} q_j d_{j,k}}{(\sqrt{\sum_{j=1}^{V} q_j^2})(\sqrt{\sum_{j=1}^{V} d_{j,k}^2})} \tag{2.5}$$

Note that the cosine similarity can be used to compare vector of any size V, thus also if the vector representations of the requests and of the elements of the collection are obtained by using neural approaches or other methods. The cosine is used because vectors close in space form a smaller angle than vectors that are far away from each other and the cosine of a smaller angle is closer to 1 than the cosine of a larger angle.Therefore, a higher value of cosine similarity corresponds to a higher relevance.

The standard dot product represents another method that is widely used to match vectors and Equation 2.6 reports the formula that is used to compute it. As it can be noticed, if the two vectors are normalized (meaning that their 2-norm is equal to 1) then the equation corresponds to the one of the cosine similarity

and, in that case, it inherits all the properties of the cosine similarity.

$$\vec{q} \cdot \vec{d}_k = \sum_{j=1}^{V} q_j d_{j,k} \tag{2.6}$$

Another measure that is used to perform the matching is the squared distance (d^2) between the vectors, that is reported in Equation 2.7. Note that in this case, differently from cosine similarity, the smaller the distance, the higher the relevance.

$$d(q, d_k)^2 = \| \vec{q} - \vec{d}_k \|^2 = \sum_{j=1}^{V} (q_j - d_{j,k})^2 \tag{2.7}$$

2.3.3 Probabilistic Matching

Since IR and RecSys aim to retrieve the most relevant items (documents) in a collection according to some request (query) provided in input, it is quite natural to think that such systems should provide as answer to a request the ranking of the items in the collection in order of decreasing probability of relevance, where the probabilities are estimated on the basis of the data available to the system for this purpose (training data). This approach is the one that is used by the majority of the keyword based approaches in retrieval (see Section 2.1.2). In particular, if we define with q a query, with d a document from the collection, with Q a random variable which can have as values the set of all possible request, with D a random variable which can have as values the set of documents (items) in the collection and with R a random variable with values 1 or 0 that expresses relevance if it is set to 1 or irrelevance if it is set to 0, then we can denote with $\mathbb{P}(R = 1|D = d, Q = q)$ the probability that a document is relevant for a certain query and with $\mathbb{P}(R = 0|D = d, Q = q)$ the probability that a document is irrelevant for a certain query. This way, the relevance of a document to a query can be computed as reported in Equation 2.8 since the most relevant document should be the ones that are more likely to be relevant.

$$R(d, q) = \frac{\mathbb{P}(R = 1|D = d, Q = q)}{\mathbb{P}(R = 0|D = d, Q = q)} \tag{2.8}$$

If we define $\mathbb{P}(r|D, Q) = \mathbb{P}(R = 1|D = d, Q = q)$ and $\mathbb{P}(\bar{r}|D, Q) = \mathbb{P}(R = 0|D = d, Q = q)$ and we consider a document as relevant for the query if and only if

$\mathbb{P}(r|D,Q) > \mathbb{P}(\bar{r}|D,Q)$ then applying the Bayes rule we can obtain the equations 2.9 and 2.10 and the rule to consider a document as relevant is the one reported in Equation 2.11.

$$\mathbb{P}(r|D,Q) = \frac{\mathbb{P}(D|r,Q)\mathbb{P}(r,Q)}{\mathbb{P}(D)} \tag{2.9}$$

$$\mathbb{P}(\bar{r}|D,Q) = \frac{\mathbb{P}(D|\bar{r},Q)\mathbb{P}(\bar{r},Q)}{\mathbb{P}(D)} \tag{2.10}$$

$$\frac{\mathbb{P}(D|r,Q)}{\mathbb{P}(D|\bar{r},Q)} > \frac{\mathbb{P}(\bar{r},Q)}{\mathbb{P}(r,Q)} \tag{2.11}$$

Since the right part of Equation 2.11 does not depend on the document we define the relevance as: $R(d,q) = \frac{\mathbb{P}(D|r,Q)}{\mathbb{P}(D|\bar{r},Q)}$. To estimate the probabilities $\mathbb{P}(D|r,Q)$ and $\mathbb{P}(D|\bar{r},Q)$ there are several techniques that can be used and we report two of them.

Binary Independence Model

To estimate the probabilities according to the binary independence model we introduce the following simplifications:

1. Documents (items) are represented by a vector of binary random variables indicating the term occurrence .
2. Given relevance, terms are statistically independent.
3. The presence of a term in a document depends on relevance only if that term is present also in the query.

Given those simplifications we can estimate the probabilities as reported in equations 2.12 and 2.13 where D_i represents the $i-th$ random variable in the vector representing each document.

$$\mathbb{P}(D|r,Q) \overset{1,2}{=} \prod_i \mathbb{P}(D_i|r,Q) \overset{3}{=} \prod_{i \in Q} \mathbb{P}(D_i|r) \tag{2.12}$$

$$\mathbb{P}(D|\bar{r},Q) \overset{1,2}{=} \prod_i \mathbb{P}(D_i|\bar{r},Q) \overset{3}{=} \prod_{i \in Q} \mathbb{P}(D_i|\bar{r}) \tag{2.13}$$

Then if we define $p_i = \mathbb{P}(D_i = 1|r)$ and $s_i = \mathbb{P}(D_i = 1|\bar{r})$ we can obtain Equation 2.14.

$$\frac{\mathbb{P}(D|r,Q)}{\mathbb{P}(D|\bar{r},Q)} = \prod_{i \in Q \cap D} \frac{p_i}{s_i} \prod_{i \in Q \setminus D} \frac{1-p_i}{1-s_i} \tag{2.14}$$

With some manipulation from Equation 2.14 we can obtain Equation 2.15.

$$\frac{\mathbb{P}(D|r,Q)}{\mathbb{P}(D|\bar{r},Q)} = \prod_{i \in Q \cap D} \frac{p_i(1-s_i)}{s_i(1-p_i)} \prod_{i \in Q} \frac{1-p_i}{1-s_i}. \tag{2.15}$$

Since the last product in the Equation 2.15 does not depend on the document we can ignore it and we can apply the logarithm. Therefore, we obtain Equation 2.16 that is used to compute the relevance scores.

$$R(q,d) \sim \sum_{i \in Q \cap D} \log_2 \frac{p_i(1-S_i)}{s_i(1-p_i)} = \sum_{i \in Q \cap D} \log_2 \frac{p_i}{(1-p_i)} - \sum_{i \in Q \cap D} \log_2 \frac{s_i}{(1-s_i)}. \tag{2.16}$$

In the expanded version of Equation 2.16 the first sum represents the contribution of terms appearing in relevant documents while the second sum represents the contribution of terms appearing in non relevant documents. Furthermore, we define with n_i the number of documents containing the $i-th$ term, with r_i the number of relevant documents containing the $i-th$ term, with N the total number of documents and with R the total number of relevant documents, then p_i and s_i can be computed as reported in Equation 2.17 where 0.5 and 1 represent some smoothing factors and Equation 2.16 can be rewritten as Equation 2.18

$$\begin{cases} p_i = \frac{r_i+0.5}{R+1} \\ s_i = \frac{n_i-r_i+0.5}{N-R+1} \end{cases} \tag{2.17}$$

$$R(q,d) \sim \sum_{i \in Q \cap D} \log_2 \frac{(r_i+0.5)(N-n_i-R+r_i+0.5)}{(n_i-r_i+0.5)(R-r_i+0.5)} \tag{2.18}$$

BM25

Okapi Best Match attempt 25 (**BM25**) [13] represents a well known and widely used baseline in IR systems to compute the relevance score of a document with respect to a query (performing the matching). BM25 is a more sophisticated version of the binary independence model described above and the relevance is computed according to Equation 2.19.

$$R(q,d) \sim \sum_{i \in Q \cap D} \log_2 \frac{(r_i+0.5)(N-n_i-R+r_i+0.5)}{(n_i-r_i+0.5)(R-r_i+0.5)} \cdot \frac{(k_1)f_i}{k_1((1-b)+b\frac{dl}{avdl})+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i} \tag{2.19}$$

In Equation 2.19 the first fraction in the sum corresponds to the same weight computed by the binary independence model and it represents a idf-like component (see Section 2.3.1), the second fraction is a component that consider the term frequency and is a tf-like component (see Section 2.3.1), the third fraction considers the query term frequency (qf_i) and represents a tf-like component but for query terms. More in detail, the second component introduces two normalization factors (k_1 which is typically set to 1.2 and b that is typically set to 0.75) and the variable $\frac{dl}{avdl}$ that represent the ratio between the length of the considered document (dl) and the average length of the document in the collection ($avdl$). The third component also introduces a normalization variable (k_2 that typically has values in the range $[0, 1000]$) and a new variable qf_i that represents the number of times that term i appears in the query.

2.4 Neural Approaches & Dense Retrieval

Machine Learning (ML) and Deep Learning (DL) represent constantly growing fields of research and, in particular, Neural Networks (NNs) are the core element of these fields. Nowadays NNs are starting to be used for many applications and between those they are being applied also to IR and RecSys. For this reason, in Section 2.4.1 we will provide a brief overview of the neural networks, in Section 2.4.2 we will explain how neural networks are trained, in Section 2.4.3 we will talk about negative sampling, in Section 2.4.4 we will introduce the cross entropy loss, in Section 2.4.5 we will describe how NNs are applied to IR and RecSys and in Section 2.4.6 we will illustrate how the most modern DL approaches work and can be used in the fields of IR and RecSys.

2.4.1 Neural Networks

Neural Networks have been created trying to emulate the structure of the human brain [14]. In particular, NNs are composed by a set of interconnected neurons, in Figure 2.5 we report an example of neural network in which the circles represent the neurons and the lines correspond to the connections (links). Note that the links are oriented. Each neuron has some inputs, some weights and some outputs (as reported in Figure 2.6) and it is responsible for computing a value (corresponding to the output, also known as activation) based on the input data that is weighted according to the Equation 2.20 where j is an index

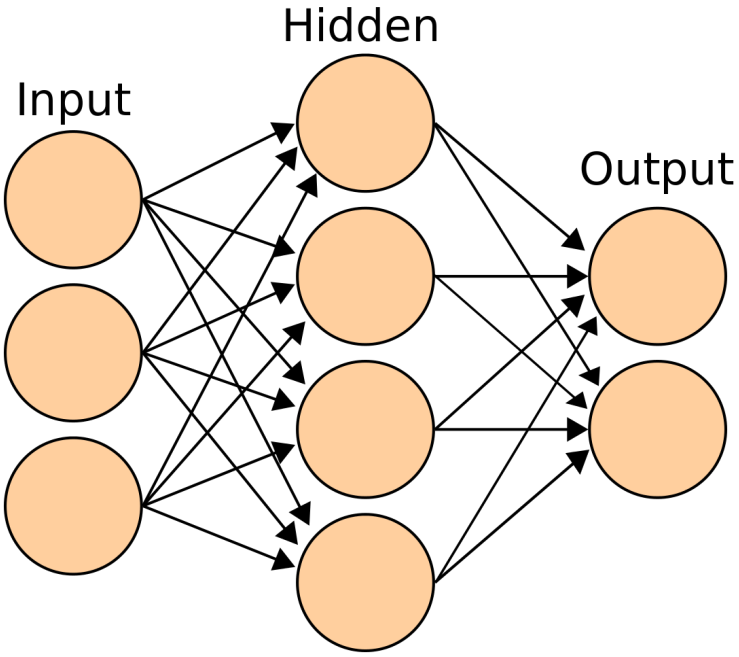


Figure 2.5: Neural Network. [19]

identifying the $j-th$ neuron in the network, w_{ij} represent the weight that the $j-th$ neuron gives to the input coming from the $i-th$ neuron, n is the total number of neurons providing their output as input to the neuron j, φ corresponds to the activation function (the most common activation functions are ReLu, tanh and Sigmoid) and o_j is the output of the neuron. The output is then forwarded to the other neurons in the network.

$$o_j = \varphi(\sum_{i=1}^{n} x_i w_{ij}) \tag{2.20}$$

Note that the weights (w_{ij}) are usually considered as labels of the links connecting the $i-th$ and the $j-th$ neuron. Furthermore, the weights of a neural network can be stored in a single matrix where the element having coordinates (i, j) corresponds to the weight of the link that goes from neuron i to neuron j.

Thanks to its interconnected structure if we provide some input (*i.e.* we set the value of the input neurons) to a neural network, it performs all the needed computations and provides some output (*i.e.* the value of the output neurons). The weights of NNs, usually, are initially set to random values, thus to use the network for a specific purpose some training is needed, otherwise the results

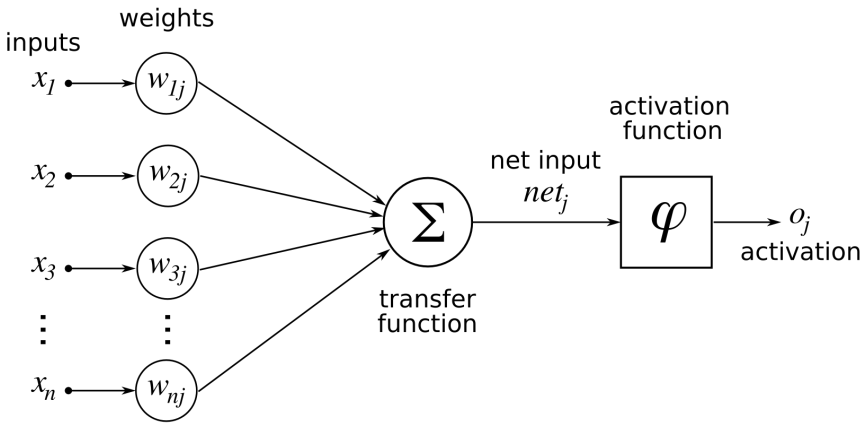


Figure 2.6: Single neuron. [19]

would be meaningless. The procedure used to train a neural network is reported in Section 2.4.2.

NNs may have different topologies, depending on the way in which the neurons are interconnected. As can be seen from Figure 2.5 the neurons can be organized in layers and, in particular, there is always one input, one output and at least one hidden layer. If the there is only one hidden layer the network is defined as "neural network" while if the hidden layers are more than one we talk about "deep neural network". Furthermore, if the connections between neurons go only from neurons of one layer to neurons of the next layer (there are no backward connections, cycles or loops) the network can be called feed-forward neural network.

2.4.2 Neural Networks Training

Training a NN corresponds to update all the weights so that given a certain input the network produces an output that is close to the desired one. Usually when performing the training of a neural network three sets of data are available: the training set, the validation set, and the test set. The training set is the one that is actually used to optimize the weights of the network, the validation set is the one that is used to perform a first evaluation (in the training phase) of the model in order to understand how to set external parameters (*e.g.* the number of neurons, the number of layers, activation functions etc.) that are normally called hyper-parameters, the test set is the one that is used to evaluate the final performance of the system when the training is ended.

To train a neural network it is common to use datasets in which for each

input example the desired output is also available (supervised learning). The actual training is performed feeding each example in the training set to the network, obtaining the corresponding output and comparing the obtained output with the desired output. The weights of the network are then updated based on the difference between the obtained and the desired output which is processed by using some function, usually called loss function, that is applied to turn the weight optimization problem into a minimization problem. The cross entropy function (described in Section 2.4.4) represents a very common loss function. To perform the actual weights update some backpropagation techniques, which make use of the derivatives in order to minimize the loss, are used. Well known backpropagation approaches are Stochastic Gradient Descent (SGD) [12, 7] and adaptive moment estimation (Adam) [8]. Thus, the end goal of a neural network is to minimize the loss.

If needed, to appropriately choose the hyper-parameters, it is common that multiple NNs are trained (also in parallel) using the same training set but a different hyper-parameters configuration. The trained NNs are then evaluated using the validation set and the best performing one is chosen. Eventually, the chosen NN is evaluated exploiting the test set to estimate its real performance.

Since neural networks need a huge amount of data to be trained, to avoid to update the weights (perform the backpropagation) for every training sample, it is possible to accumulate the loss for groups of samples that are commonly called batches. This is done because the backpropagation represents a time demanding operation, thus doing it less frequently allows to reduce the training time, and also because it helps to reduce overfitting, which is a phenomenon that happens when the networks learns too much from the training data and becomes too specific, losing the capability of generalizing.

Furthermore, when backpropagating the loss, an hyper-parameter called learning rate is exploited to control the impact of the weight update. In particular, a large learning rate allows the update to be more impactful and can fasten the training process of the network but it may lead to divergent behaviours due to the drastic updates. A small learning rate, instead, makes the updates less significant, avoiding divergent behaviours, but it can slow down the training process.

Eventually, it is possible to repeat the training of a NN multiple times using the same training set to try to increase its performance. We define as epoch each one of the times in which the training is repeated using the entire training set,

therefore in an epoch each sample in the training data is seen by the network once and only once. The number of epochs for which the training is repeated represents an hyper-parameter. This type of training procedure is useful when some hyper-parameters are changed across the epochs, for example the learning rate.

2.4.3 Negative Sampling

In Sections 2.4.1 and 2.4.2 we introduced how neural networks work and, in particular, we focused on the fact that for each training sample provided in input they try to learn to replicate the corresponding output. Nonetheless, it is important that NNs not only learn to produce outputs close to the correct ones but also acquire knowledge about how to produce outputs as different as possible from the incorrect ones. If we think of the outputs of NNs as multidimensional vectors we would like the NNs prediction to be close in space to the vectors representing the desired output and far away from the vectors representing wrong outputs. To favor this kind of behaviour negative sampling is used. Negative sampling consist in adding for each input sample in the training set one or more outputs that are wrong. Thus, the training set will be composed of both input-correct and input-wrong output pairs. By using appropriate loss functions, such as cross entropy (see Section 2.4.4) is then possible to properly train NNs in the same way as explained in Sections 2.4.1 and 2.4.2.

There are several techniques that can be used to sample negatives, some examples are: hard-negative sampling [17] and in-batch negative sampling. Hard negative sampling consists of selecting the negatives directly from a set of results. Think, for example, of a retrieval system: after a first training passage for each request we will obtain in output a list of ranked documents, then for each request we take one or more of the retrieved documents that are not actually relevant, we add them to the training dataset as negative samples and we repeat the training with this new dataset. The negatives can be sampled also from the results of some other system/network and not only from the output of the same system/network that we are trying to train. For the retrieval case, for example, we can select the negatives from the results obtained with traditional retrieval methods (*e.g.* BM25) and use the data to trained more sophisticated models. The in-batch negatives technique, instead, can be exploited when batches are used during the training and, according to it, for each sample in the batch the nega-

tives are sampled from the batch itself. More in detail, if we think of a batch as a set of input-correct output pairs this technique requires to select for each input one or more outputs that are present in pairs in the same batch of the input, creating new input-wrong output pairs that are then added to the batch itself.

2.4.4 Cross Entropy Loss

Suppose to have a binary classification problem in which one class represents the correct samples and another class represents the incorrect samples (similarly to what happens when we use negative sampling, explained in Section 2.4.3) and let y be a label that can take only values in $\{0, 1\}$ which will indicate if the *i-th* sample in a training set is correct ($y_i = 1$) or if it is incorrect ($y_i = 0$). The cross entropy loss (L_{CE}) function for binary classification is defined as reported in Equation 2.21 where with y_i we denote the true label of the *i-th* sample and with $\mathbb{P}(y_i)$ we refer to the probability that the sample belongs to the class y_i which is computed by the neural network (normally, in this case, the neural network has one neuron in the output layer and the probability that the sample is correct corresponds to its output value).

$$L_{CE} = -(y_i \log(\mathbb{P}(y_i)) + (1 - y_i) \log(\mathbb{P}(1 - y_i))) \tag{2.21}$$

Since we would like to have a model with a loss that is as small as possible, the goal of the neural network would be to minimize the it. As can be seen from the equation, because of the minus sign, if the label indicates that the sample is "correct" then we will keep only the first term and we would like the probability that the sample belongs to the correct class to be high, while if the label indicated that the sample is "incorrect" then we will keep only the second term and we would like the probability that the sample does not belong to the incorrect class to be high.

The cross entropy loss can be defined also for multiple classes (normally, the network has one neuron for each class in the output layer) and can be accumulated over multiple samples. Equation 2.22 reports this version of the loss where with C we indicate the total number of classes, with j we denote the $j-th$ class, with N we represent the total number of samples and with i we refer to the $i-th$ sample. Note that in this case the labels y_{ij} take values in $[0, 1]$ and indicate the

true probability that the $i-th$ sample belongs to the $j-th$ class.

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\mathbb{P}(y_{ij})) \tag{2.22}$$

The cross entropy loss can be used also for tasks different from classification, for example in the case of negative sampling (see Section 2.4.3) the label y_i will be used to specify if the corresponding sample is negative or positive, while the probability of belonging to the class ($\mathbb{P}(y_i)$) can be replaced with some error measure (for example the inverse of the distance between the obtained and the desired output, the dot product between the obtained and desired output etc.).

2.4.5 Dense Retrieval & Recommender Systems

Dense retrieval is the branch of IR that exploits NNs to represent the input data in form of vectors, called text embeddings, which are then used to perform the needed operations (vector based search presented in Section 2.1.2 is a clear and general example of dense retrieval). More specifically, IR is concerned with retrieving the most relevant documents according to some query provided in input (see Section 2.1). To do this, each document and the query are processed to be transformed into sequences of tokens (which can be single words or even subparts of words) and, after that, the tokens are provided as input to some neural network (usually the transformer architecture or other models that are based on the transformer architecture are used, see Section 2.4.6) that is able to convert the sequence of tokens into text embeddings (which are also able to capture contextual information). The main goal is to map the queries to some vectors that are close in space to the vectors representing the most relevant document for that query, so the neural network is trained accordingly.

RecSys, which aim to recommend the most relevant products/items in the collection according to some product/item and to the user history provided in input, exploit the NNs in a similarly to IR. RecSys use the NNs to encode the input product/item and each product/item in the collection into some text embedding and try to train the networks so that the input items and the corresponding relevant products are mapped to some vectors that are close in space. Differently from IR usually the embeddings of the input items are modified according to the user history (this process is normally called personalization, see Section

2.2 for more details). Frequently the personalization is performed exploiting the concept of attention (see Section 2.4.6).

2.4.6 Neural Models, Architectures and Attention

Most of the models related to IR and/or RecSys adopt architectures that are based on the transformer architecture (or part of it) which has been introduced in 2017 [18]. In Figure 2.7 we report the complete architecture of the transformer.

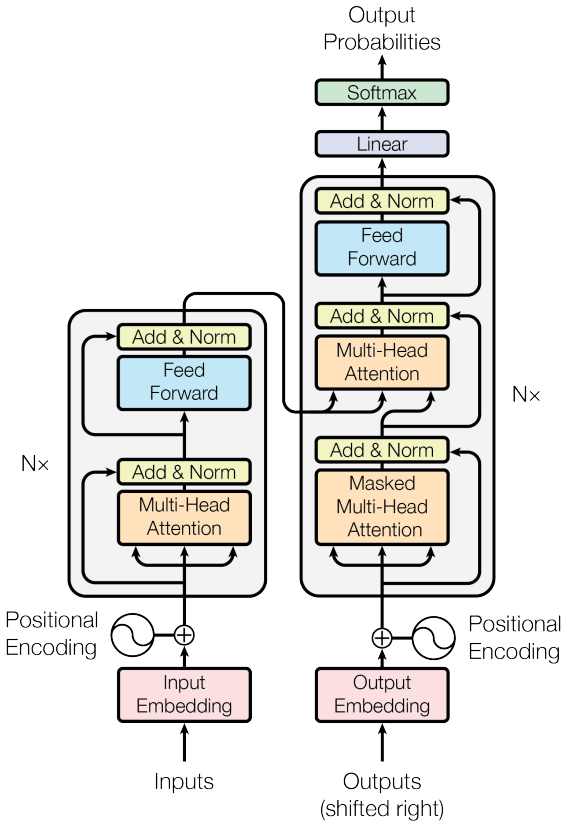


Figure 2.7: General transformer architecture. [18]

As can be seen, the transformer can be splitted into two sub-components that are the encoder (left box in the figure) and the decoder (right box in the figure). Frequently, the models used for IR and/or RecSys are based only on one of the two components, for example Bidirectional Encoder Representations from Transformer (BERT) exploits only the transformer encoder while Generative Pre-Trained Transformers (GPTs) usually use only the transformer decoder. Normally, transformers and in particular BERT are used in IR and RecSys to map the input tokens into some textual embeddings (see Section 2.4.5 for more details). BERT represents a model which is pre-trained to be able to perform sev-

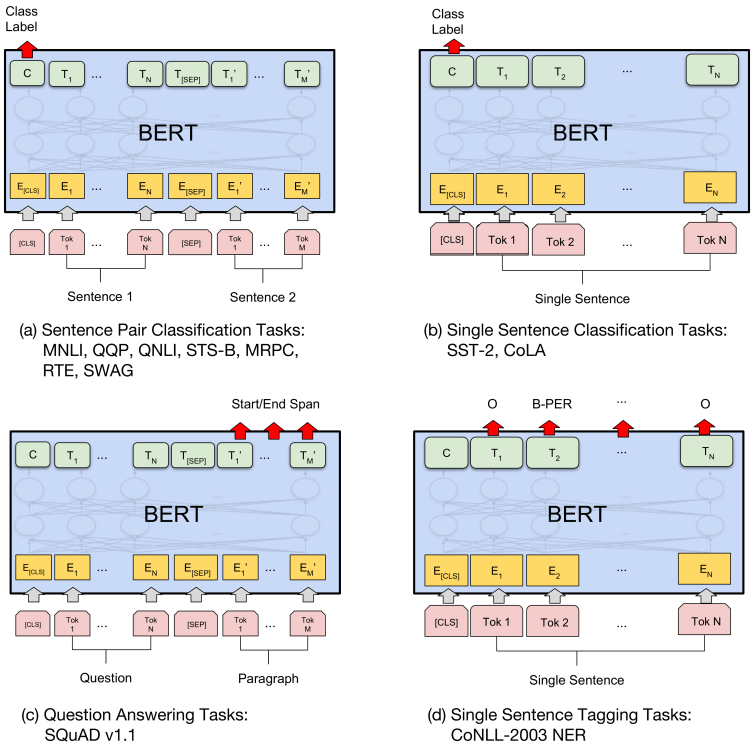


Figure 2.8: BERT use example. [2]

eral tasks that are reported in Figure 2.8 and it is widely used in dense retrieval and RecSys. As can be seen from the figure, BERT takes in input a sequence of tokens and it carries out the desired task using an architecture that corresponds to the transformer encoder. This model is frequently used to perform different tasks than the ones for which it has been pre-trained and, for doing this, usually the architecture is extended with some additional layers at the end and then the model is fine-tuned with appropriate data. As explained in Section 2.4.5, when we use NNs for IR and RecSys, the products, documents and queries are first turned into sequences of tokens and then the tokens are given as input to the neural network to obtain some text embeddings, thus BERT represents an ideal neural component to be used. In particular, for IR and RecSys, since it is not necessary to perform an end-to-end task (*e.g.* classification) but it is needed to transform some tokens into an embedding (*i.e.* a vector), in the majority of cases, the tokens are provided in input to BERT (or another appropriate model) which performs its computation and, after that, the value of the neurons of the last hidden layer of the BERT network are taken to form the embedding.

Normally, the models based on transformers take as input some tokens which

usually are pre-processed before performing other operations. In Figure 2.9 we

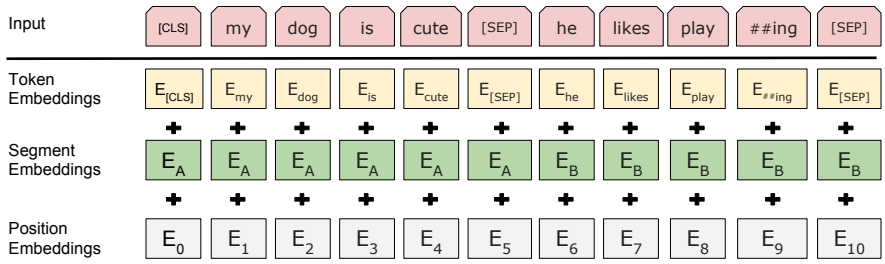


Figure 2.9: Input tokens pre-process. [2]

report an example representing the pre-processing that is done to the tokens by BERT. In particular, the tokens are transformed in some initial embedding (random or fixed) that are combined with a segment embedding (since BERT can take as input up to two sentences, these segment embeddings are used to allow the model to recognize to which sentence each token belongs) and then the obtained embeddings are further integrated with some positional embeddings (which are used so that the model is able to consider the order of the tokens in the sentence). The way in which BERT pre-process the input tokens is the same as the ones that nowadays is used by the majority of models/approaches.

Models based on transformers require an enormous amount of time and computational power to be trained from scratch, thus these models are usually pre-trained by large companies/institutions with some generic data and then the pre-trained models are provided to the developers that will perform a fine-tuning phase. Fine-tuning corresponds to re-train part of the model (starting from the pre-trained one), that commonly corresponds to the final layers, with data that is highly specific for the application for which the model is used. For some applications the models are also completely re-trained (always starting from the pre-trained version). Furthermore, before being fine-tuned the models can be slightly modified adding some layers (usually at the end of the network), if it is needed.

As can be seen from the structure of the transformer reported in Figure 2.7, the encoder, after obtaining some initial embedding of the inputs, modifies them with some positional encoding (these initial steps are performed in the same way as we described for BERT). The modified input embeddings are then handled in parallel using N identical encoder blocks. Each encoder block first processes the input token using a concept that is called attention (which is described below), then it combines the results of the attention block with the original input using a

residual connection and, after that, it normalizes the output (to achieve stability and avoid problems like exploding gradients). The normalized results are then fed to a feed-forward neural network (see Section 2.4.1) and eventually a residual connection and a normalization component are used in the same way as before. The transformer decoder works in the same manner as the encoder but it has two attention steps. While the first attention block is identical to the encoder's one, the second takes as input also the output of the encoder (which is used as keys and values). The output of the decoder, differently to the one of the encoder, is processed applying some functions to obtain some probabilities in output.

The most important element that was introduced with transformers is the concept of attention [18]. The aspect that led attention to be widely used is its ability to capture correlations and contextual information, since this is fundamental especially in natural language and text based applications. Attention has been deeply studied and modified and several variations of it have been defined like self-attention or multi-head attention. In the following we first introduce the general version of attention (also known as "Scaled Dot-Product Attention"), then we explain the self-attention and eventually we describe the multi-head attention which is also used in the transformer architecture.

Scaled Dot-Product Attention

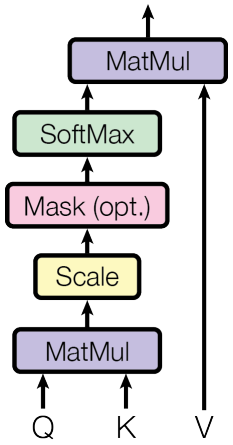


Figure 2.10: Attention structure. [18]

Figure 2.10 reports a scheme that represents the way in which attention is computed. In particular, Q is the query and corresponds to the current focus (what we are looking for), K stands for keys which represent the importance given to the query (what we can offer) and V means values which refer to the

contribution given to the query (what we actually offer). Informally, the query is "compared" with the keys to determine the query-keys correlation that is expressed in form of weights, then these parameters are used to weight the values, which are eventually combined to build a new representation. The understanding of the correlation between queries and keys is what allows to determine the context and how to extract information from it. Formally, the mathematical equation that is used to perform this computation is reported in Equation 2.23 where d_k is a normalization factor that depends on the dimensionality of the keys and $softmax$ represents a function that is used to normalize the values of vectors/matrices in order to obtain some probabilities (values in the range [0,1]).

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.23}$$

The mask component present in Figure 2.10 is not reported in Equation 2.23 because it is optional and it is used to compute the correlation of the query with only part of the keys. The mask is very useful when the attention is used with tokens (representing words or sub-words), in fact, imagine a situation in which attention is exploited in an application which aims to predict the next word in a sentence, in this case the candidate word is correlated only with the past words in the sentence and not with the future words so the mask can be used to mask the latter in order to avoid to compute some correlations that could be wrong or impossible to consider in a real word scenario. The masking operation consists of summing (before applying the $softmax$ function) to the matrix $\frac{QK^T}{\sqrt{d_k}}$ another matrix (the mask) which contains only zeros and $-\infty$ and, therefore, allows either to keep unchanged or set to $-\infty$ some values according to the desired behaviour (since $softmax$ is used, the values $-\infty$ will correspond to 0 after applying it, allowing to mask some tokens). Figure 2.11 reports a common mask that is used so that each token is correlated only with the tokens appearing before it in the sentence and not with the ones appearing after (note that each row of the mask corresponds to the mask of a single input token). Since the value $-\infty$ cannot to be represented in a computer, some implementations replace $-\infty$ with the smallest negative value possible while others, instead of the performing the sum, multiply the matrix $\frac{QK^T}{\sqrt{d_k}}$ with a matrix (the mask) that contains ones (instead of zeros, in correspondence the values that must be kept) or zeros (instead of $-\infty$, in correspondence the values that must not be considered).

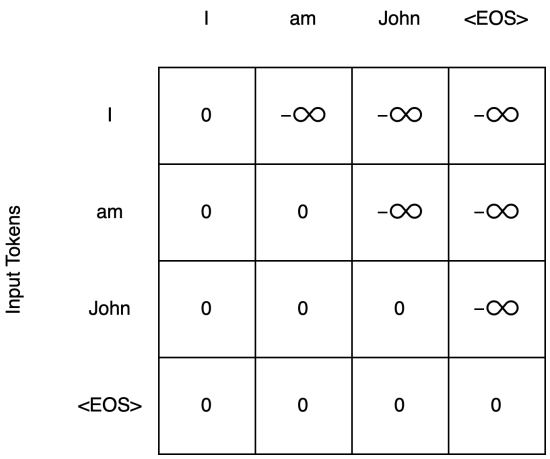


Figure 2.11: Example of attention mask.

Self-Attention

Self-attention is based on the scaled dot-product attention but the queries, the keys and the values are all generated from the data provided in input. Let X be the matrix representing the input and suppose that it contains in each row the initial embedding of the input tokens (already modified according to the token position, the sentence to which they belong etc.), then the queries, the keys and the values are obtained from the input by means of three linear projections: $Q = XW_Q$, $K = XW_K$ and $V = XW_V$, where W_Q, W_K and W_V are learnable parameter matrices. The queries, the keys and the values are then processed in the same of the scaled dot-product attention.

This version of attention is very useful because it allows to find correlations between the words of a sentence and, thus, to consider the syntactic and contextual structure of the sentence. This is important when the transformers are used to compute a vector representation (embedding) of a textual string.

Multi-Head Attention

Figure 2.12 reports a scheme that represents how multi-head attention is computed. In particular, instead of applying a single attention function to keys, values and queries which have the same dimensionality d_{model}, it has been discovered [18] that it is convenient to linearly project the queries, keys and values. The queries, keys and values are projected h times with separate, learned linear projections to d_k, d_k and d_v dimensions, respectively. The standard attention

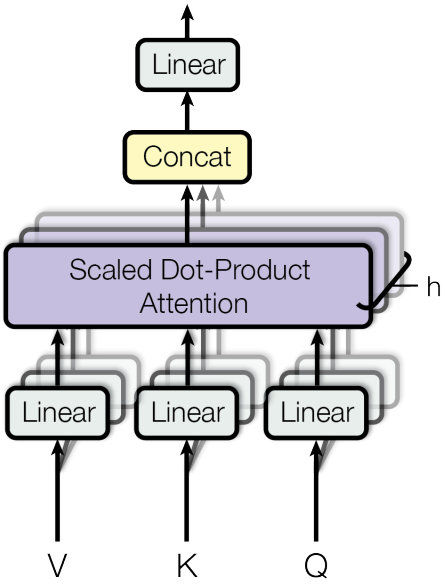


Figure 2.12: Multi-Head attention structure. [18]

function (reported in Equation 2.23) is then applied in parallel to each of the projected versions of the queries, keys and values. The attention function produces in output d_v dimensional values which are then concatenated and projected to obtain the final values.

Equation 2.24 reports the formula that is used to compute multi-head attention where the matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are the learnable parameter matrices that are used to perform the linear projections and Attention is the same function as the one reported in Equation 2.23.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \cdots, \text{head}_h) W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.24}$$

This version of attention is quite useful because it allows (differently to the standard one) to distill knowledge at different positions also from different subspaces. Note that multi-head attention has been introduced as an extended version of the self-attention.

2.5 Joint Information Retrieval and Recommender Systems

Since the nineties it has been clear that there were common aspects between IR and RecSys, in fact, they are considered as two sides of the same coin [1]. IR and RecSys can be thought as joint tasks because they both are concerned with retrieving the best documents/products given some data in input (textual query and/or user history, see Section 2.1 and 2.2) that represents the information need and, even though under the hood there are some variations, the end user does not perceive differences between those two technologies, also because the output of the two tasks is usually mixed in the same interface. Furthermore, in the case of e-commerce the queries usually represent short, keyword based descriptions of a product/item that someone is looking for, so the gap between IR and RecSys becomes even smaller. Additionally, normally a user seeks for information (by providing queries) that are related to its previous interactions. Nonetheless, for historical and industrial reasons these two fields have been developed independently but it would be more convenient to consider them jointly.

Recently the research community started to develop systems performing both the IR and the RecSys jointly, noticing promising results. In particular, Zamani and Croft [24, 25] have shown that developing such models allows to improve the performance thanks to the sharing of knowledge between IR and RecSys. However, most of the developed systems focus on refining RecSys capabilities by exploiting the search data [15] or on gaining the knowledge to carry out one of the two tasks based on the data of the other [25]. Only a few relevant results addressed the issue of joint IR and RecSys: a first proposed model based on Graph Neural Networks (GNNs) was SRJGraph [27] and the Unified Information Access (UIA) framework [26] recently outperformed it.

The main advantage of developing models performing both the retrieval and recommendation tasks jointly is derived from the fact that is possible to create a shared knowledge base between the two tasks. This allows to improve the performance of both tasks and also gives them the ability to support each other (imagine a situation in which we have a low amount of data for one task and a large amount of data for the other, in this case the task for which there is limited data can still exploit the knowledge coming from the other task).

2.6 Evaluation Measures

Models after being trained should also be properly evaluated. The evaluation is performed at the end of the testing phase and must use appropriate measures to define the performance of the models and make them comparable. When evaluating a model the ground truth corresponding to the test data must be available; in particular this ground truth contains for each of the request in the test data the corresponding relevant documents/products of the collection. In the majority of cases the ground truths are created by humans that manually mark as relevant or not relevant the documents/products of a sampled subset of the collection; this procedure is called relevance assessment.

Evaluation measures are categorized into two main families that are: set based evaluation measures and rank based evaluation measures. Set based evaluation measures are indicators related to the number of documents marked as relevant in the ground truth that have been retrieved/recommended by the system. Precision, recall and F-measure are some examples of metrics that belong to this family. Rank based measures instead consider also the rank of the relevant documents retrieved, where with rank we mean the position of the documents in the list that is provided in output by IR systems or RecSys. Normalized Discounted Cumulative Gain (nDCG) and Mean Reciprocal Rank (MRR) are some examples of metrics that belong to this family. Especially for set based measures, we define with TP the True Positives that represent the retrieved documents that are also relevant (based on the ground truth), with FP the False Positives that correspond to the documents that are retrieved but are not relevant, with TN the True Negatives that are the documents that are not retrieved and also not relevant and eventually with FN the False Negatives that refer to documents that are not retrieved but are relevant. Furthermore, with RB we define the total number of relevant documents (according to the ground truth) and with $\mathcal{R}$ the set of the positions in the output list (produced by some retrieval or recommendation model) of the retrieved documents that are also relevant.

Usually, to test the systems several queries are performed, thus it is common to compute the performance of the systems by considering all the queries in the test set. The most common technique to compute performances considering sets of queries is to perform the mean of the value of the measures computed for single queries. Some measure have an explicit definition for the case in which multiple queries are considered (this is the case of precision and the corresponding

mean average precision), others do not have an explicit definition for this specific situation thus the arithmetic mean is used.

Note that we always considered binary relevance, in which the documents are either relevant or not relevant with respect to some input. Nonetheless, in real scenarios that could also be multiple level of relevance (*e.g.* highly relevant, relevant, not relevant) and usually in this cases the level of relevance is expressed with an integer number (whose value ranges from 0 up to the number of relevance degrees); this is called multi-graded relevance. Normally, the relevance label for the $i-th$ document in the list provided in output by IR or RecSys models is denoted with r_i.

Precision (**P**) represents a significant measure for both IR and RecSys and reports the fraction of retrieved documents that are actually relevant (according to the ground truth). The equation used to compute precision is reported in Equation 2.25.

$$P = \frac{|TP|}{|TP| + |FP|} = \frac{|\text{relevant retrieved documents}|}{|\text{retrieved documents}|} \tag{2.25}$$

Precision at k (**P@k**) represents a variation of precision that is used when it is needed to consider only the first k positions in the output list of retrieved documents. The equation used to compute precision at k is reported in Equation 2.26.

$$P@k = P(k) = \frac{1}{k} \sum_{n=1}^{k} r_n \tag{2.26}$$

Average Precision (**AP**) represents the average of the precision at the position of the relevant retrieved documents and its one of the most used metrics to summarize the performances of the system. The equation used to compute average precision is reported in Equation 2.27.

$$AP = \frac{1}{RB} \sum_{k \in \mathcal{R}} P(k) \tag{2.27}$$

Mean Average Precision (**MAP**) represents the mean of AP across several queries (also called topics). It is quite frequent, in fact, that to test a system several queries are preformed and the mean average precision in used to evaluate the overall performance. The equation used to compute average precision is reported in Equation 2.28 where i refers to the $i-th$ query performed and Q

represents the total number of queries executed.

$$MAP = \frac{1}{Q} \sum_{i=1}^{Q} AP_i \tag{2.28}$$

Recall (**R**) represents a significant measure for both IR and RecSys and reports the fraction of relevant documents (according to the ground truth) that are actually retrieved. The equation used to compute recall is reported in Equation 2.29.

$$R = \frac{|TP|}{|TP| + |FN|} = \frac{|\text{relevant retrieved documents}|}{|\text{relevant documents}|} \tag{2.29}$$

Recall at k (**R@k**) represents a variation of recall that is used when it is needed to consider only the first k positions in the output list of retrieved documents. The equation used to compute recall at k is reported in Equation 2.30.

$$R@k = \frac{1}{RB} \sum_{n=1}^{k} r_n \tag{2.30}$$

F-measure (**F**) represents the harmonic mean between precision and recall and it is used to evaluate the trade-off between the two. The equation used to compute f-measure is reported in Equation 2.31.

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.31}$$

Discounted Cumulative Gain (**DCG**) represents a significant measure for both IR and RecSys and is used to evaluate the performance in terms of the rank of the relevant retrieved documents/items considering also a normalization factor that is related to the user behavior. The equation used to compute discounted cumulative gain is reported in Equation 2.32. In particular , as can be seen from the second part of the equation, DCG sums the rank of the relevant retrieved document, thus systems that are able to retrieve the relevant documents placing them in the top positions of the output list are preferred to systems that put the same documents in lower positions. Furthermore, each element of the sum is normalized according to a logarithmic factor. Since the logarithm is a monotonic non-decreasing function and the operand of the logarithm is the position of the retrieved document in the output list, this factor allows to give more importance to the relevant retrieved documents that are in the top positions. The

base b of the logarithm represents the user patience, in fact, a smaller b reflects an impatient user (since the logarithm grows more rapidly) while a larger b denotes a patient user. The $\max$ function is used to avoid having a negative value as denominator, since if some relevant documents are placed in a position with index lower than the base b of the logarithm we end up in this situation. Thus, DCG represents a fundamental measure because is able also to capture the user behavior. Furthermore, other advantages of DCG are that it does not depend on the recall base (RB) and that it is also able to handle multi-graded relevance. Note that with DCG@k (or DCG(k)) we denote the DCG that is computed considering only the top k documents retrieved by the system.

$$DCG(k) = \begin{cases} \sum_{n=1}^{k} r_n & if \quad k < b \\ DCG(k-1) + \frac{r_k}{\log_b(k)} & if \quad k >= b \end{cases} = \sum_{n=1}^{k} \frac{r_n}{\max(1, \log_b(n))} \tag{2.32}$$

Normalized Discounted Cumulative Gain (**nDCG**) represents the normalized version of DCG, which allows to restrict the value of DCG between 0 and 1 (because since DCG is a sum of independent elements it can grow above one). The equation used to compute discounted cumulative gain is reported in Equation 2.33, where iDCG represents the "ideal DCG" that corresponds to the DCG that the system would have if it worked perfectly (ranking the most relevant documents all at the top of the list, ordered by relevance).Note that with nDCG@k (or nDCG(k)) we denote the nDCG that is computed considering only the top k documents retrieved by the system.

$$nDCG@k = nDCG(k) = \frac{DCG(k)}{iDCG(k)} \tag{2.33}$$

Mean Reciprocal Rank (**MRR**) represents a measure that is mostly used for RecSys but it can be meaningful also for IR. This measure is useful to understand the position of the first relevant retrieved document for each query. The equation used to compute MRR is reported in Equation 2.34 where i refers to the $i-th$ query performed and Q represents the total number of queries executed. MRR, in fact, is computed by summing the reciprocal of the rank of the first relevant retrieved document ($rank_i$) of a set of queries, providing an indication of how good is the systems in terms of ranking. Note that with MRR@k we denote the MRR that is computed considering only the top k documents retrieved by the

system. If there is no relevant document in the output list or between the top k documents the reciprocal rank for that query will be considered zero.

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i} \tag{2.34}$$

3

UIA Framework

SE and RecSys nowadays are used more and more by everyone. Frequently, the results of these two systems must be combined in order to provide to the user a complete and satisfying answer to its needs. This is quite common, for example, in e-commerce platforms. As we discussed in Section 2.5 systems performing retrieval and recommendation have been developed independently but creating models able to carry out both IR and RecSys related tasks jointly can have a promising impact in terms of performances. The Unified Information Access (UIA) framework [26] represents a novel and extensible framework that can handle several (personalized) information access requests, exploiting the flexibility of dense retrieval models. It was presented by Hansi Zeng, Surya Kallumadi, Zaid Alibadi, Rodrigo Nogueira and Hamed Zamani at SIGIR 2023 (the major conference in the field of IR) and it is the current state-of-the-art in the field of joint IR and RecSys.

In this chapter we will delineate the main functionalities that the UIA framework supports, we will introduce the adopted notation, we will describe the general architecture of the framework, we will explain how the model is optimized (trained), we will present the datasets used in the original paper to train and evaluate the framework, we will delineate some practical details regarding the settings of the parameters of the model and, eventually, we will evaluate the performance of the model as proposed by the original authors. In the following we will refer to the paper [26] that introduced UIA as "original paper".

3.1 Functionalities

The UIA framework implements both IR and RecSys tasks, supporting various functionalities:

1. Keyword Search (KS): it corresponds to the traditional IR task that retrieves documents/items in response to a short textual query, *e.g.* retrieving *'Nike Air Force 1'* for query *'Nike shoes'*.

2. Query By Example (QBE): it corresponds to the similar item search RecSys task that retrieves items similar to another item provided in input, *e.g.* retrieving *'Nike Air Jordan 1'* for a user interested in *'Nike Air Force 1'*.

3. Complementary Item Recommendation (CIR): it corresponds to the complementary item search RecSys task that retrieves items complementary to another item provided in input, *e.g.* retrieving *'Nike shoes laces'* for a user interested in *'Nike Air Force 1'*.

As can be noticed the QBE and CIR functionalities have identical inputs but produce different outputs. Furthermore while QBE and CIR have the same type of inputs and outputs (both items) the KS functionality has a different type of inputs (queries instead of items). Therefore, developing a framework supporting those three functionalities allows to demonstrate that the framework has the ability to learn how to treat various types of information access functionalities.

3.2 Information Access

Users can access information in many different ways, for example in traditional IR the user provides a query to the system and expects an ordered list of documents (or products) as a result, while in RecSys the user uses the output of the system (a set of items/products) to explore and fulfill their necessities instead of formulating a textual representation of their information need (see Sections 2.1 and 2.2). The UIA framework aims to support the user in all the different information access modalities.

We define the following element that will be considered when treating an information access request:

1. *Information Access Request* ($\mathcal{R}$): it represents the actual request, such as a textual query and the related context (time, location, session data, etc.). This request can also be empty in the case of zero-query retrieval.

2. *User History* ($\mathcal{H}$): it represents information related to the previous user behaviour (*e.g.* previous clicks, purchased items, etc.), also long-term.

3. *Candidate Item Information* ($\mathcal{I}$): it represents the candidate item to be retrieved (it may include some side information such as the item content, the author and the source).

4. *Access Functionality* ($\mathcal{F}$): it represents the required functionality (needed to distinguish the IR and RecSys tasks, see section 3.1 for more details).

The UIA framework focuses on the three information access functionalities ($\mathcal{F}$) reported in Section 3.1 but it can be extended to support also other functionalities, for example including multi-modal retrieval (*e.g.* textual queries and image items), contextual requests (exploiting session data) and/or zero-query retrieval/recommendation (retrieving/recommending items based only on the user history or session data).

Defining with t the index of the considered request (usually the last) of a user u for which the framework is expected to provide some output and given the variables that we just introduced, we can formulate a unified information access model, parameterized by θ for user u, as $f(\mathcal{F}_t^u, \mathcal{R}_t^u, \mathcal{H}_t^u, \mathcal{I}_i; \theta)$, where $\mathcal{F}_t^u$ represents a textual description of the functionality relate to the request, $\mathcal{R}_t^u$ refers to the information access request performed (a textual query or the textual content of an item in the case recommendation), $\mathcal{H}_t^u$ denotes the set of interactions that the user u had prior to triggering the *t-th* request and $\mathcal{I}_i$ represents the textual content of the *i-th* candidate item.

The history of user u prior to the *t-th* request is a set containing all the interactions that the user u had prior to performing the *t-th* request, ordered by time (the smaller index, the earlier the instant in which the interaction happened): $\mathcal{H}_t^u = \{(\mathcal{F}_1^u, \mathcal{R}_1^u, \mathcal{I}_1^u), (\mathcal{F}_2^u, \mathcal{R}_2^u, \mathcal{I}_2^u), \cdots, (\mathcal{F}_{t-1}^u, \mathcal{R}_{t-1}^u, \mathcal{I}_{t-1}^u)\}$. Each interaction is represented by a triplet composed by the past user's request, the information access functionality that has been used, and the item that the user interacted with (*e.g.* bought or clicked items).

Note that in the original paper they used two different datasets but since one of them is not publicly available, we limited ourselves to the use of the Amazon

ESCI dataset which does not have user history related data (see Chapter 3.5 for more details). Thus, we can reformulate the previously defined unified information access model, parameterized by θ for the interaction t, as $f(\mathcal{F}_t, \mathcal{R}_t, \mathcal{I}_i; \theta)$, where $\mathcal{F}_t$ represents a textual description of the functionality, $\mathcal{R}_t$ represents a textual query or the textual content of an item (for recommendation) and $\mathcal{I}_i$ represents the textual content of the *i-th* candidate item.

3.3 Framework Architecture

The UIA framework follows a bi-encoder architecture in which the first encoder is intended to be used to encode the requests, while the second encoder should encode the items. An Attentive Personalization Network (APN) is then used to enhance the representation of the request (provided by the first encoder) using the user historical interaction data in order to be able to personalize the results. The item encodings (produced by the second encoder) instead are fed to a feed-forward neural network in order to adjust the items representations based on the personalized request vectors. This way, the final representation of the requests and of the items should have the same dimensionality.

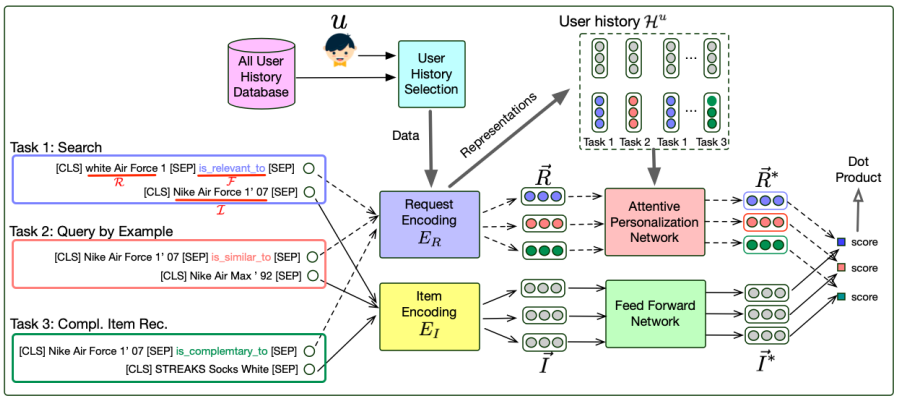


Figure 3.1: UIA framework architecture with user history. [26]

Figure 3.1 represent the actual structure of the system as depicted in the original paper [26]. The four major components of the architecture are the:

1. Request Encoding ($\mathbf{E}_{\mathcal{R}}$).
2. Item Encoding ($\mathbf{E}_{\mathcal{I}}$).
3. User History Selection & Encoding.

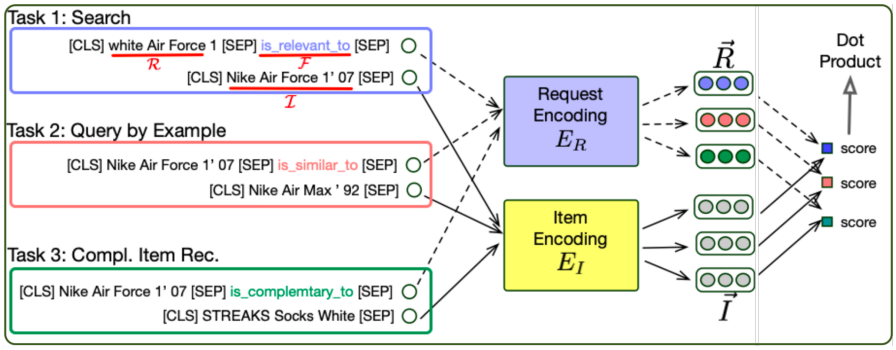


Figure 3.2: UIA framework architecture without user history.

4. Attentive Personalization Network.

The components are described in Sections 3.3.1, 3.3.2, 3.3.3 and 3.3.4 respectively.

Since we use the Amazon ESCI dataset that has no user information (see Section 3.5), we can reduce the structure of the framework to the one reported in Figure 3.2. In this case, the major components of the system are only two: the Request Encoding ($\mathbf{E}_{\mathcal{R}}$) and the Item Encoding ($\mathbf{E}_{\mathcal{I}}$).

3.3.1 Request Encoding

To implement the request encoder we exploit a pre-trained language model ($\mathbf{E}$) to obtain a dense vector representation of the requests (the queries, which are turned into sequences of tokens in order to be fed to the encoder) . In particular, we use the BERT-base [2] model to encode each request $\mathcal{R}_t^u$ and information access functionality $\mathcal{F}_t^u$ into $\vec{R}_t^u$ as reported in Equation 3.1 (see Section 2.4.6 to understand how BERT works).

$$\vec{R}_t^u = \mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}_t^u\ \texttt{[SEP]}\ \mathcal{F}_t^u\ \texttt{[SEP]}) \tag{3.1}$$

If we do not consider the user history, as it happens when the Amazon ESCI dataset is used (see Chapter 3.5), the equation can be simplified as reported in Equation 3.2.

$$\vec{R}_t = \mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}_t\ \texttt{[SEP]}\ \mathcal{F}_t\ \texttt{[SEP]}) \tag{3.2}$$

3.3.2 Item Encoding

To realize the item encoder we start form a pre-trained language model ($\mathbf{E}$) to obtain a dense vector representation of the collection items (the products which

are turned into sequences of tokens in order to be fed to the encoder). In particular, we use the BERT-base [2] model to encode each item $\mathcal{I}_i$ as reported in Equation 3.3 (see Section 2.4.6 to understand how BERT works).

$$\vec{I}_i = \mathbf{E}_{\mathcal{I}}(\texttt{[CLS]}\ \mathcal{I}_i\ \texttt{[SEP]}) \tag{3.3}$$

3.3.3 User History Selection & Encoding

To personalize the request encoding $\vec{R}^u_t$ the original paper considers only the last N interactions of the user, *i.e.* $\{(\mathcal{F}^u_{t-N}, \mathcal{R}^u_{t-N}, \mathcal{I}^u_{t-N}), \cdots, (\mathcal{F}^u_{t-1}, \mathcal{R}^u_{t-1}, \mathcal{I}^u_{t-1})\}$. For each user past interaction we obtain two encodings: one for the request and the related information access functionality ($\mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}^u_{t'}\ \texttt{[SEP]}\ \mathcal{F}^u_{t'}\ \texttt{[SEP]}) : \forall t - N \leq t' \leq t - 1$) and the other for the item with which the user interacted after submitting the request ($\mathbf{E}_{\mathcal{I}}(\texttt{[CLS]}\ \mathcal{I}^u_{t'}\ \texttt{[SEP]}) : \forall t - N \leq t' \leq t - 1$). We are sure that all the elements of the user history are represented in the same space as the one of the requests and the items since the encoding of the user history is performed using the same encoders $\mathbf{E}_{\mathcal{R}}$ and $\mathbf{E}_{\mathcal{I}}$ as the ones used for the requests and items encoding. Furthermore, since all the parameters of the framework are trained end-to-end (see Section 3.4), the parameters of the encoders are also updated during training. The encoded version of the user history is a set of $2 \times N$ vectors: $\{(\vec{R}^u_{t-N}, \vec{I}^u_{t-N}), (\vec{R}^u_{t-N+1}, \vec{I}^u_{t-N+1}), \cdots, (\vec{R}^u_{t-1}, \vec{I}^u_{t-1})\}$.

3.3.4 Attentive Personalization Network

The personalization of the request encoding $\vec{R}^u_t$ is performed exploiting a novel Attentive Personalization Network (APN) that allows to carry out both content-based and collaborative personalization. The APN is based on the concept of attention (see Section 2.4.6 for more details) and in particular it exploits the multi-head attention. The content-based personalization is done learning the attention weights from the encodings of the user's past N interactions to the current request. Note that for each information access functionality, UIA is able to use the user's past interactions with all functionalities (independently from the required one). Collaborative personalization, instead, is done by learning a latent representation for each user and information access functionality based on all the past interactions.

Figure 3.3 reports an overview of the architecture of APN. For content-based

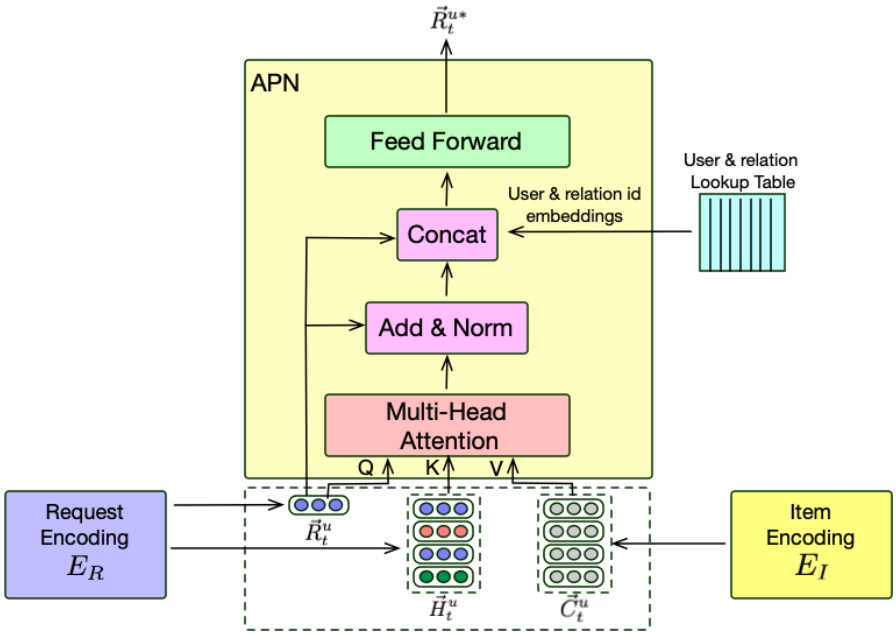


Figure 3.3: Attentive Personalization Network architecture. [26]

personalization, given the request encoding $\vec{R}_t^u$ of the *t-th* request performed by user u, APN considers the last N user history encodings $\{(\vec{R}_{t-N}^u, \vec{I}_{t-N}^u), \cdots, (\vec{R}_{t-1}^u, \vec{I}_{t-1}^u)\}$ and it splits them into to two matrices: $H_t^u \in \mathbb{R}^{N\times d}$, whose rows are equal to the past requests encodings, and $C_t^u \in \mathbb{R}^{N\times d}$, whose rows are equal to the past interacted items (clicked) encodings.

APN, following the definition of multi-head attention, exploits N_h attention heads, where the *j-th* attention function contains three parameter matrices $\theta_j^Q \in \mathbb{R}^{d\times l}$, $\theta_j^K \in \mathbb{R}^{d\times l}$, and $\theta_j^V \in \mathbb{R}^{d\times l_v}$, and it learns attention weights from the user's interaction history (which represents the keys and the values) to their current request (which represents the query). Therefore, each APN layer obtains the query $Q_j \in \mathbb{R}^{1\times l}$ (request), the keys $K_j \in \mathbb{R}^{N\times l}$ (past requests) and the values $V_j \in \mathbb{R}^{N\times l_v}$ (past interacted items) matrices performing the following operations:

$$Q_j = \vec{R}_t^u.\theta_j^Q, \quad K_j = H_t^u.\theta_j^K, \quad V_j = C_t^u.\theta_j^V.$$

APN exploits the computed matrices as reported in Equation 3.4 following the concept of attention (see Section 2.4.6).

$$\text{Attn}(Q_j, K_j, V_j) = \text{softmax}(\frac{Q_j K_j^T}{\sqrt{l}})V_j \tag{3.4}$$

The results of all the attention functions are then concatenated (as reported in Equation 3.5) and the concatenation is fed to an "Add & Norm" layer which exploits a residual connection and works in the same way as the corresponding layers in the transformer architecture (see Section 2.4.6).

$$\text{concat}(\{\text{Attn}(Q_j, K_j, V_j)\}_{j=1}^{N_h}) \in \mathbb{R}^{1 \times N_h l_v} \tag{3.5}$$

For collaborative personalization, instead, some user and functionality embeddings are needed. Let us define with $|\mathcal{U}|$ the number of users, with l_u the dimensionality of the user embedding, with $|\mathcal{F}|$ the number of functionalities (three in our case) and with l_f the dimensionality of the functionality embedding. APN, to be able to perform the collaborative personalization, learns a user embedding matrix $\mathbf{E}_{\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}| \times l_u}$, where each row represents the embedding of a user. Furthermore, to be able to distinguish between the different behaviours that a user can have when dealing with different information access functionalities, APN learns also an embedding matrix for the information access functionalities $\mathbf{E}_{\mathcal{F}} \in \mathbb{R}^{|\mathcal{F}| \times l_f}$, where each row represents the embedding of one functionality. To perform the collaborative personalization, APN selects the user embedding vector $\vec{u}$ for user u (that performed the request) from $\mathbf{E}_{\mathcal{U}}$ (*i.e.*, user embedding lookup) and the functionality embedding $\vec{f}$ for the functionality f (associated with the request) from $\mathbf{E}_{\mathcal{F}}$. The two embeddings are then concatenated to the output of the "Add & Norm" layer and, eventually, the concatenation is fed to a feed-forward neural network layer which uses a non-linear activation function (ReLU). The output of APN is a personalized version of the request embedding, which is denoted by $\vec{R}_t^{u*}$.

The item representations, instead, are fed to a feed-forward neural network to be adjusted, since APN is used only for the request personalization and not for the items. The output of the feed-forward neural network is denoted by $\vec{I}_i^*$.

3.4 Framework Optimization

The optimization (training) of the UIA framework follows a two stage process:

1. Non-personalized pre-training
2. Personalized fine-tuning

As we already mentioned, the original paper trained some models on a private dataset which contained also user information and some other on the publicly available Amazon ESCI dataset (see Chapter 3.5) which does not contain user information. The second stage of the optimization process is used to optimize the personalization components of the framework, thus this stage is not needed in the cases in which the Amazon ESCI dataset is used since these components are removed (see Section 3.3).

In Section 3.4.1 and 3.4.2 we describe the first and second stage of the optimization process, respectively.

3.4.1 Non-Personalized Pre-Training

To perform the non-personalized pre-training, if a dataset with user information is used, we construct a non-personalized training set by aggregating the training data across all users. Therefore, in any case, the UIA framework is trained starting from a dataset in which the *k-th* training instance has the form $(\mathcal{F}_k, \mathcal{R}_k, \mathcal{I}_k, \mathcal{Y}_k)$, where $\mathcal{Y}_k$ represents the ground truth label. We denote with $\vec{R}_k$ and $\vec{I}_k$ the output of the request and item encoders respectively, that are computed for the *k-th* instance of the training data.

Since the training data contains only positive instances (*i.e.* user interactions) we define the following two-phase negative sampling and training strategy (see Section 2.4.3 for more details about negative sampling):

- *Phase 1*: For each training instance k a negative item is sampled from the top 200 items retrieved for the request $\mathcal{R}_k$ by BM25 (that is a keyword based, traditional retrieval method, see Sections 2.1.2 and 2.3.3) and it is used to create a new,negative training instance (that has the same functionality and request as the one in the generating instance but the sampled negative item as item and zero as ground truth label) which is added to the dataset. After that, the model is trained using a cross entropy loss function (see Section 2.4.4). Note that, in addition to BM25 negatives, in-batch negatives (see Section 2.4.3) are also used.

- *Phase 2*: After the model has been trained a first time with the data processed according to *Phase 1* the negatives are removed from the dataset and new negatives are sampled according to the following procedure. The Item Encoder $\mathbf{E}_I$ (that has been trained) is used to encode all the items in

the collection and then an Approximate Nearest Neighbor (ANN) index is created using the Faiss library [4]. After that, the index is exploited to retrieve items for each request in the training data. To add the negative instances to the dataset, *Phase 2* follows the same path as *Phase 1* but the negative sample for each training instance k is randomly sampled from the top 200 items retrieved for the request $\mathcal{R}_k$ by using the ANN index, instead of BM25. Eventually, the model is re-trained, starting from the one trained in *Phase 1* (which has already been partially optimized), using a cross-entropy loss function (see Section 2.4.4) and in-batch negatives.

The actual training phases use batches and happen as follows: first we provide to the request encoder $\mathbf{E}_{\mathcal{R}}$ the request $\mathcal{R}_k$ obtaining the vector $\vec{R}_k$, then we provide to the Item Encoder $\mathbf{E}_{\mathcal{I}}$ the item $\mathcal{I}_k$ (that could be positive or negative) obtaining the vector $\vec{I}_k$, after that we compute the dot product between the two dense vectors produced by the encoders ($\vec{R}_k \cdot \vec{I}_k$) obtaining the matching score (relevance value) and, eventually, (at the end of each batch) we backpropagate the loss based on cross entropy loss function, which is created to handle both positive and negative samples (see Section 2.4.4). Note that in all the phases the ratio between positive and negative samples for each training instance is set to one.

Since the non-personalized pre-training considers only the encoders $\mathbf{E}_{\mathcal{R}}$ and $\mathbf{E}_{\mathcal{I}}$ and backpropagates the error only to those two components, then the only parameters that are optimized are the ones of $\mathbf{E}_{\mathcal{R}}$ and $\mathbf{E}_{\mathcal{I}}$.

3.4.2 Personalized Fine-Tuning

Personalized fine-tuning is performed to optimize also the personalization part of the network, when the dataset used contains user related information, since the non-personalized pre-training stage optimizes only $\mathbf{E}_{\mathcal{R}}$ and $\mathbf{E}_{\mathcal{I}}$. In personalized fine-tuning, we re-create the training data to include the user information and their past interactions (because in non-personalized pre-training the dataset does not contain them) . For negative sampling, in this case, we use BM25 results in addition to in-batch negatives (analogously to *Phase 1* in non-personalized pre-training). To train the framework we provide each training instance to the model and, this time, exploiting also the personalization components, we obtain the personalized representation of each request and candidate item in the training data, i.e., $\vec{R}^*_k$ and $\vec{I}^*_k$ respectively (see Figure 3.1). The dot product is then used to compute the matching score (relevance value): $\vec{R}^*_k \cdot \vec{I}^*_k$.

As for the non-personalized pre-training the framework is trained using batches and the cross entropy loss function (see Section 2.4.4), therefore the loss is accumulated over all the samples in a batch before the backpropagation takes place.

3.5 Datasets

In the original paper [26], the UIA framework has been optimized and evaluated using two different datasets:

1. Lowe's dataset.
2. Amazon ESCI dataset [11].

Lowe's dataset represents a large scale e-commerce dataset containing over 5.3 million user interactions obtained from more than 890K unique users. The item collection in this dataset includes over 2.2 million products. Unfortunately, this dataset is private, thus only the authors of the paper were able to use it. The Amazon ESCI dataset, instead, is publicly available and it corresponds to the datasets that we used to reproduce the work of the paper and to perform all the needed analyses. In the following sections we first provide an overview of the original Amazon ESCI dataset and then we describe how we obtained several datasets (one for each of the tasks: KS, QBE, CIR, see Section 3.1 for the task definition) from it. Eventually, in Section 3.5.5 we focus on the splitting of the datasets obtained from the original Amazon ESCI dataset into training, validation and test sets, on how these are used to train and test the model and on the potential problem derived from the splitting process.

3.5.1 Original Dataset

The Amazon ESCI dataset [11] has been originally published by the organizers of the KDD Cup 2022[1] Amazon ESCI challenge and it is a large, multilingual dataset of difficult Amazon search queries and results, containing around 130 thousand unique queries and 2.6 million manually labeled (query, product) relevance judgements. Following the work of the original paper, we considered the dataset related to the Task 2 of the Amazon ESCI challenge and, in particular,

[1]KDD Cup 2022: `https://amazonkddcup.github.io/`

we exploited the product catalogue and the training data made available. Note that in the product catalogue each line correspond to a product and contains the product id, the product title and some other product information (such as the description, bullet list etc.) that were not used, while the training data contains, for each line, a query identifier, the query text, the query language, the id of a product and the ESCI label (which represents the relevance judgement) that states if the product is an exact (E), similar (S), complementary (C) or irrelevant (I) match for the query. From these two sources of data we created 3 separated datasets, one for the IR task (KS) and the other two for the RecSys tasks (QBE, CIR) (see Sections 3.5.2, 3.5.3, 3.5.4) considering only queries and products in English language. We defined as relevant products for a query the products marked as exact (E) in the dataset, as similar products the ones marked as similar (S) and as complementary products the ones marked as complementary (C). Furthermore, because of the nature of the dataset, we considered the products as the documents to retrieve and the product title as the document body. Therefore, when using the Amazon ESCI dataset, if we talk about encoding a query we mean encoding the string that represents the query itself, while if we talk about encoding a product/item we refer to encoding the title of the product/item, which is also represented by a textual string.

3.5.2 Keyword Search Dataset

To create the elements (samples) for the KS dataset, that are pairs of the type [query, relevant item], we took the queries of the original dataset (see Section 3.5.1) for which there was at least one relevant item and (one at the time) the corresponding items marked as exact (E). At this point, to obtain the queries for the KS task we simply get the unique queries of the KS dataset. Furthermore, to produce the ground truth corresponding to those information needs we consider for each query the items appearing in the corresponding pairs.

The KS dataset is then split in training, validation and test sub-datasets. To produce these three sets we randomly select queries in order to create three separate groups, the first containing 80 percent of them, the second holding 10 percent of them and the third including the remaining 10 percent of them. The pairs related to the queries in the first group are then used to create the KS training set, the ones related to the second group are used to generate the KS validation set and the ones related to the third group are used to form the KS test set.

3.5.3 Query By Example Dataset

The elements (samples) of the QBE dataset are pairs of the type [relevant item, similar item]. To obtain these pairs we consider only the queries of the original dataset (see Section 3.5.1) for which there are both exact (E) and similar (S) matches. For each of these queries we combine each of the relevant products with all of the similar products (one at the time), creating the desired pairs. At this point, to obtain the queries for the QBE task we simply get the unique relevant items of the QBE dataset. Furthermore, to produce the ground truth corresponding to those information needs we consider for each relevant item the similar items appearing in the corresponding pairs.

The QBE dataset is then split in training,validation and test sub-datasets. To produce these three sets we randomly select relevant items in order to create three separate groups, the first containing 80 percent of them, the second holding 10 percent of them and the third including the remaining 10 percent of them. The pairs related to the relevant items in the first group are then used to create the QBE training set, the ones related to the second group are used to generate the QBE validation set and the ones related to the third group are used to form the QBE test set.

3.5.4 Complementary Item Recommendation Dataset

The elements (samples) of the CIR dataset are pairs of the type [relevant item, complementary item]. To obtain these pairs we consider only the queries of the original dataset (see Section 3.5.1) for which there are both exact (E) and complementary (C) matches. For each of these queries we combine each of the relevant products with all of the complementary products (one at the time), creating the desired pairs. At this point, to obtain the queries for the CIR task we simply get the unique relevant items of the CIR dataset. Furthermore, to produce the ground truth corresponding to those information needs we consider for each relevant item the complementary items appearing in the corresponding pairs.

The CIR dataset is then split in training,validation and test sub-datasets. To produce these three sets we randomly select relevant items in order to create three separate groups, the first containing 80 percent of them, the second holding 10 percent of them and the third including the remaining 10 percent of them. The pairs related to the relevant items in the first group are then used to create the CIR training set, the ones related to the second group are used to generate

the CIR validation set and the ones related to the third group are used to form the CIR test set.

3.5.5 Training, Validation & Testing

As explained in Sections 3.5.2,3.5.3 and 3.5.4, the original dataset is processed to generate three separate datasets (one for KS, one for QBE and one for CIR) and each of them is then splitted into training, validation and test sets. The splitting is performed independently for each one of the new datasets and at the end of this operation 80% of the samples of each dataset end up in the corresponding training set, 10% of them end up in the corresponding validation set and 10% of them end up in the test set.

The framework is trained following the procedure described in Section 3.4 and by providing in input the training set of each of the tasks (KS,QBE and CIR) one at the time, allowing to consider the tasks jointly. Since the datasets are all composed by samples that have the form of pairs, according to the notation defined in Section 3.4.1 the functionality $\mathcal{F}_k$ takes one of the values among *"is_relevant_to"*, *"is_similar_to"* and *"is_complementary_to"* based on which of the three dataset the sample comes from (KS dataset, QBE dataset or CIR dataset, respectively), the request $\mathcal{R}_k$ corresponds to the first element of the pair (query text or item title) and the item $\mathcal{I}_k$ corresponds to the second item in the pair (title of an item). Furthermore, since in this case we use binary relevance, $\mathcal{Y}_k$ is set to one (indicating relevance) for all the samples provided in input with the three datasets and it is set to zero (indicating irrelevance) for all the samples produced by the negative sampling procedures (see Section 3.4.1). The evaluation is then performed separately for each task exploiting the corresponding test set.

Since the framework is jointly trained, the generation and splitting operations can be the cause of some issues that we address as "data leakage". In particular, since the three datasets for the three tasks (KS,QBE and CIR) are generated starting from the same original Amazon ESCI dataset and splitted into training, testing and validation only later it is possible that the same entry in the original dataset is used to generate a sample that ends up in the training set for one task and another sample that ends up in the test data of another task.

3.6 Implementation Details

The architecture of the framework exploits some encoders that, as we explained in Section 3.3, are based on a pre-trained language model. The model that is used in the original paper is BERT-base [2] (an overview of BERT is provided in Section 2.4.6) available on HuggingFace [20], and, specifically, in the case in which the Lowe's dataset is exploited the pre-trained weights are loaded from the checkpoint *"bert-base-uncased"*[2], while for the Amazon ESCI dataset (which is smaller) the pre-trained weights are loaded from the checkpoint *"msmarco-bert-base-dot-v5"*[3].

The hyper-parameters are set based on the framework performance in terms of nDCG on the validation set. The number of training epochs is chosen from $[8, 12, 16, 24, 48]$, the number of the considered user's historical interactions (N) is set to 5, the batch size is empirically set to 384 and the learning rate for non-personalized pre-training and personalized fine-tuning is empirically set to $7e^{-6}$ and $7e^{-5}$. For the case in which the model is trained on the Lowe's dataset, for personalized fine-tuning, only users with at least 10 interactions in the KS and QBE tasks and at least 5 interactions in the CIR task are kept. Furthermore, in the Attentive Personalization Network the hidden dimension d is 768, the number of heads N_h is 12, the hidden dimension of key and value in each head are $l = l_v = 64$, the dimension of the user embeddings is $l_u = 128$ and the dimension of the functionality embeddings is $l_f = 64$. For all the neural components Adam [8] is used as the optimizer.

The learning rate used to train the BERT based encoders is not static but it is modified each time a batch is completed. In particular, if we define as the total number of steps the number of epochs multiplied by the number of batches for each epoch, the learning rate is updated a number of times that corresponds to the total number of steps. The learning rate starts from 0 and in a certain number of warmup steps (4000) reaches the specified value $(7e^{-6})$, after that the learning rate is reduced at each step reaching zero at the end.

[2] `https://huggingface.co/bert-base-uncased`
[3] `https://huggingface.co/sentence-transformers/msmarco-bert-base-dot-v5`

3.7 Framework Evaluation

UIA represents a framework used for joint IR and RecSys and (as explained in Sections 2.1 and 2.2) IR and RecSys must produce in output a list of relevant items ordered by relevance. In order to do this, when a request is performed by a user (both implicitly or explicitly) all the items in the collection and the request itself are encoded into some vector representation exploiting the trained version of UIA. After that, the request vector is compared with the items vectors, using the Faiss library, in order to create an ordered list of items which contains the best matches (the items corresponding to the closest vectors to the one of the request) at the top positions.

To evaluate the performances of UIA several requests (belonging to a test set) are submitted to the framework which produces for each of them a list of ranked items. For each request the obtained list is then compared with the ground truth and some metrics are used to obtain a numerical representation of the performance. Note that for UIA only the top most 1000 items of the ranked items list of each request have been considered.

The evaluation of the framework is performed exploiting three metrics: MRR@10 which is used to understand how good is the framework in ranking the most relevant retrieved document, nDCG@10 which is used to understand the overall performance of UIA and Recall@50 which is used to understand how good is the system in retrieving the relevant documents (the specific definition of these evaluation metrics is reported in Section 2.6).

The authors of the original paper, together with the framework, evaluated also several baselines on the same datasets in order to be able to compare UIA with some other traditional and/or modern approaches. The baselines used range from term matching models to dense retrieval models and correspond to:

- **BM25** [13]: This is a traditional and effective bag-of-word retrieval model which has been described in Section 2.3.3.

- **NCF** [3]: This is a recommendation (collaborative filtering) model, which combines generalized matrix factorization and a multi-layer perceptron approach for recommendation. It only learns from item-item interactions and cannot be applied to KS task.

- **DPR** [6]: This is a dense retrieval model that performs negative sampling

exploiting BM25 results. In addition to that it employs also in-batch negative sampling. DPR only uses the last request (query text or query item) and does not perform personalization.

- **Context-Aware DPR**: This model extends DPR to include personalization based on the user history. In order to perform the personalization, the requests are concatenated with the past interactions of the corresponding user (separated by a `[SEP]` token) and the concatenation is then fed to the query encoder.

- **ANCE** [21]: This is an effective dense retrieval model that uses the model itself to mine hard negative samples. Analogously to DPR, ANCE is not capable performing the personalization, therefore to consider also the user history the authors defined **Context-Aware ANCE** which uses a similar approach to the one exploited by Context-Aware DPR.

- **RocketQA** [10]: This is a state-of-the-art dense retrieval model which exploits large batch sizes and denoised negative samples to achieve more robust contrastive learning. RocketQA, similarly to ANCE and DPR, is not capable of performing the personalization, thus the authors followed a path similar to DPR and ANCE to define **Context-Aware RocketQA** which is able to handle the user history.

- **BERT4Rec++**: BERT4Rec [16] is a sequential recommendation model which uses BERT to represent the user history for predicting the next item that should be recommended to the user. The original BERT4Rec model exploits only the item IDs (it takes in input item IDs and predict the next item ID in the sequence) but the authors improved BERT4Rec by encoding also the content of the items. BERT is used for content embedding. This approach is called BERT4Rec++.

- **SASRec++**: SASRec [5] is a sequential recommendation model which uses the self-attention mechanism (see Section 2.4.6) to identify the items that are "relevant" to the user interaction history for the next item prediction. This approach cannot be used for the IR tasks (KS task). Analogously to BERT4Rec++, SASRec is modified in order to be able to exploit also the content of the items, using BERT for content embedding. This new model is called SASRec++.

- **JSR** [24]: This is a neural framework that jointly learns the search and recommendation tasks. In this model, each task has a task-specific layer over the base shared network.

- **JSR + BERT4Rec++**: The original version of JSR uses the user ID to encode user information. The authors of the paper improved the JSR performance by using the representation from BERT4Rec++ to encode the user content.

- **SRJGraph** [27]: This is a recent framework based on neural graph convolution that jointly models the search and recommendation tasks.

While for BM25, NCF, DPR, ANCE, RocketQA, SASRec++ and BERT4Rec the code is publicly available, for JSR and SRJGraph it is not, therefore the authors of the paper implemented them.

The training of models is classified into two categories (task-specific training and joint training) based on whether the model is optimized on a single task (one between KS, QBE, CIR) or on all the tasks jointly (like UIA). Furthermore, the authors used the same initial pre-trained weights for BERT to train all dense retrieval baselines which exploit it (DPR, ANCE, RocketQA, JSR+BERT). All the approaches that are able to handle the user history consume the historical data provided in input in reverse chronological order. For SASRec++ and BERT4Rec++, the number of additional transformer layers is chosen from [1, 2, 4], each transformer layer contains 12 heads and each head's hidden dimension is 64. The task-specific layer for JSR and SRJGraph is a single dense layer with hidden dimension 768 and their networks have been initiated based on the same BERT model as the dense retrieval baselines. The learning rate for all baselines is chosen from [1e-4, 7e-5, 1e-5, 7e-6] considering the value of the performances on the validation sets used.

Table 3.1 reports the performance of the UIA framework and the baselines on the Lowe's dataset while Table 3.2 reports the performance when the Amazon ESCI dataset is used. Since the Amazon ESCI dataset does not contain user information the baselines requiring user data have not been used. Note that for the systems that support only task-specific training different instances have been optimized (one for each task) even if the results for all the tasks have been reported on the same line. From both Table 3.1 and 3.2 it is possible to notice that jointly trained systems and, in particular, UIA, perform better that the ones optimized for single tasks. Furthermore, from Table 3.2 we can notice that when

Table 3.1: Experimental results on the Lowe's dataset.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
BM25	0.089	0.095	0.367	0.153	0.167	0.584	0.016	0.014	0.111
Task-Specific Training									
NCF	-	-	-	0.132	0.147	0.351	0.117	0.118	0.236
DPR	0.188	0.192	0.578	0.171	0.180	0.598	0.153	0.156	0.487
ANCE	0.193	0.199	0.582	0.176	0.188	0.601	0.159	0.158	0.494
RocketQA	0.201	0.207	0.595	0.189	0.204	0.613	0.174	0.176	0.507
Context-Aware DPR	0.324	0.377	0.848	0.311	0.356	0.860	0.278	0.283	0.707
Context-Aware ANCE	0.332	0.385	0.856	0.317	0.361	0.866	0.289	0.292	0.714
Context-Aware RocketQA	0.335	0.389	0.861	0.326	0.369	0.874	0.300	0.304	0.723
SASRec++	-	-	-	0.305	0.347	0.836	0.271	0.264	0.695
BERT4Rec++	-	-	-	0.314	0.354	0.851	0.283	0.279	0.703
Joint Training									
JSR	0.324	0.379	0.853	0.349	0.380	0.878	0.325	0.317	0.760
JSR+BERT4Rec++	0.337	0.394	0.871	0.415	0.479	0.919	0.421	0.419	0.820
SRJGraph	0.336	0.392	0.874	0.416	0.478	0.921	0.423	0.420	0.822
UIA	**0.340**	**0.399**	**0.880**	**0.433**	**0.495**	**0.945**	**0.438**	**0.432**	**0.836**

Table 3.2: Experimental results on the Amazon ESCI dataset.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
BM25	0.513	0.351	0.494	0.017	0.011	0.084	0.030	0.032	0.165
Task-Specific Training									
DPR	0.505	0.347	0.511	0.235	0.174	0.527	0.434	0.450	0.838
ANCE	0.522	0.354	0.519	0.237	0.178	0.531	0.431	0.443	0.825
RocketQA	0.526	0.357	0.525	0.244	0.185	0.538	0.445	0.458	0.847
Joint Training									
JSR	0.528	0.355	0.527	0.243	0.192	0.536	0.477	0.484	0.853
SRJGraph	0.526	0.351	0.522	0.241	0.187	0.540	0.479	0.488	0.855
UIA	**0.532**	**0.360**	**0.533**	**0.251**	**0.199**	**0.543**	**0.490**	**0.493**	**0.868**

the Amazon ESCI dataset is used, the task that gains the most in terms of performance when the systems are jointly trained is the CIR task. This may occur because the CIR dataset is the smallest one and thus it benefits the most from the shared knowledge between the different tasks.

4

Reproducing UIA

In this chapter we will explain how we were able to reproduce the work of the original paper that introduced UIA [26] by exploiting the code that was made available by the authors[1], we will highlight some problems and differences with respect to what is stated in the publication and we will discuss the achieved results.

In Section 4.1 we will delineate the experimental the values of the hyper-parameters that we used in the experiments, in Section 4.2 we will report some discrepancies between the paper and the actual operations performed on the datasets, in Section 4.3 we will explain how we tried to emulate the performance results of some basic retrieval/recommendation methods and, eventually, in Section 4.4 we will describe how we reproduced the performance of UIA presented in the original paper and the problems that we encountered in doing that.

We recall that, as explained in Section 3.5, in all our reproducibility work and experiments we only used the Amazon ESCI dataset, therefore all the considerations of this chapter will be based on that dataset (as well as on the ones derived from it) and on the portion of the code dedicated to the training and testing of the UIA framework when that dataset is used. Moreover, as introduced in Chapter 3, no user information is available in the Amazon ESCI dataset and the personalization component are removed from the architecture, thus the training is carried out by performing only the non-personalized pre-training stage.

[1] `https://github.com/HansiZeng/UIA`

4.1 Experimental Setup

All the experiments and tests were carried out by using the datasets derived from the Amazon ESCI dataset as explained in Section 3.5, modified according to what reported in Section 4.2.

The training of the framework requires the use of multiple GPUs (Graphics Processing Units) due to the high demand of memory. In our experiments we used three different setups:

- *Setup 1*: three NVIDIA GeForce GTX 1080 Ti (11 GB of memory each).
- *Setup 2*: two NVIDIA GeForce RTX 3090 (24 GB of memory each).
- *Setup 3*: two NVIDIA A40 (48 GB of memory each).

Using the first setup a single phase of the training of UIA lasts approximately 6 days, while using the second or third setup it takes about 4 days. Furthermore, because of the limited amount of resources available, we needed to change the value of some hyper-parameters as reported in Section 4.1.1.

The measures used to evaluate the performance of the framework are MRR@10, nDCG@10 and recall@50 (see Section 3.7 for more details). For all the experiments/tests for which we reported the value of these metrics we expressed them as percentages (values between 0 and 1), thus, the higher the value the better the framework performs.

4.1.1 Hyper-parameters Settings

As explained in Section 3.6 the number of epochs is chosen from a set of numbers based on the performance of the system on a validation set. From the code shared by the authors of the papers we were able to understand that the better performing number of epochs would have been 48 but, due to our limited computational resources, performing 48 epochs resulted too time demanding, thus we carried out all the tests by using 24 as number of epochs. Furthermore, due to the limited memory of our GPUs we were not able to handle batches of 384 samples, therefore in all our experiments we used batches containing only 48 samples. These changes can have an impact on the performance of the framework.

4.2 Dataset Preprocessing Discrepancies

The first thing that is not stated in the paper [26] but is actually performed by the code is that, after the generating the datasets for the KS, QBE, CIR tasks from the Amazon ESCI dataset (see Section 3.5) and splitting them independently into training, validation and test sets, the training sets are further processed. In fact, for each unique relevant item in the training set of the QBE task which is paired with more than 5 similar items, only 5, randomly sampled, similar items are kept. The same happens for the unique relevant items in the training set of the CIR task, in which, at the end of this process, each relevant item appears in at most 5 pairs. For the training set of the KS task there is a slight difference, in fact, for each unique query in this dataset which is paired with more than 10 relevant items, only 10, randomly sampled, relevant items are kept. Actually, several version of the training set for the KS task are produced, keeping each time a different number of relevant items for each query, but the version that we reported is the one that has been used. In our experiments we used the just introduced versions of the training sets since from the original code of the framework we understood that these were the ones that also the authors exploited.

Other discrepancies that we discovered are related to the negative sampling procedure performed in the *Phase 1* of the optimization process (see Section 3.4.1). In particular, while for the training set of the CIR task the negatives are correctly sampled from the BM25 results, for the training set of the QBE task the negatives were sampled randomly between the items in the collection. We solved this problem by sampling the negatives also for the QBE task from the BM25 results. Furthermore, the negative samples of the training set of the KS task are not directly sampled from BM25, in fact, for each sample in the set, which has the form [query, relevant_item], a negative sample is created by taking the relevant item and, if it has some similar items (based on the QBE task dataset), the negative is sampled from those, else if it has some complementary items (based on the CIR task dataset) the negative is sampled from those, else the negative is sampled exploiting the results of BM25. We decided to keep this negative sampling procedure for the KS dataset because in this case, differently from the one regarding the QBE dataset, the differences seemed to be intentionally introduced by the authors.

4.3 Reproducing Baselines

Before replicating the UIA framework itself we tried to reproduce the performance obtained with the baseline models and, specifically, we decided to focus on BM25 since it is used in the *Phase 1* of the UIA framework optimization and also because it is a traditional, well known and widely used information retrieval model (see Section 2.3). Following the work of the original paper [26] we decided to exploit Pyserini[2] [9] which is a Python toolkit for reproducible information retrieval research with traditional models and models based on sparse/dense representations. To provide this features Pyserini makes use of the Faiss library [4] and of Anserini[3] [22, 23], which is another IR toolkit that is built on Lucene[4] (an open source library for IR).

In order to replicate the BM25 results it is needed to first exploit Pyserini to create an index of all the items in the collection and then it is possible to perform the search and the evalutation (see Section 2.1). To execute the indexing of all the items it is sufficient to provide in input to Pyserini the item collection and the model to be used (BM25 in our case). To perform the search we provided in input to Pyserini the created index, the model to be used (BM25), the list of the queries (a file that contains the queries/items that represent the requests) and the number of documents to be retrieved for each request (200 in our case). The evaluation is eventually performed with Pyserini which exploits trec_eval (a tool for the evaluation of retrieval results originally developed for the Text REtrieval Conference) and the results of the search. We recall that BM25 does not consider KS, QBE and CIR tasks jointly, therefore to perform the search it is necessary to provide to Pyserini the files corresponding to the training, validation and test sets (depending on the goal) of each task one at the time.

Table 4.1 reports the performance of BM25 appearing in the original paper and results that we obtained trying to reproduce it. As can be noticed for the KS and CIR tasks we were able to reproduce the behaviour almost perfectly while for the QBE task this is not the case. In particular, for QBE our results are quite better than the original ones, in fact our MRR and nDCG are almost four times larger than the original ones while our recall is more than twice the original

[2]Pyserini: `https://github.com/castorini/pyserini`
[3]Anserini: `https://github.com/castorini/anserini`
[4]Lucene: `https://lucene.apache.org/`

recall. Considering that the QBE datasets happens to be the largest one (it is almost twice as big as the KS dataset and five times larger than the CIR), a possible explanation of this strange behavior (considering also the results obtained reproducing UIA, see Section 4.4) can be the fact that the authors of the original paper might not have used the complete ground truth for the QBE task (the entire QBE dataset) but only a sampled version of it, without highlighting this fact.

Table 4.1: Original and reproduced BM25 performance.

	Keyword Search			Query By Example			Complementary Item Rec.		
Model	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
original BM25	0.513	0.351	0.494	0.017	0.011	0.084	0.030	0.032	0.165
our BM25	0.511	0.351	0.490	0.066	0.042	0.203	0.038	0.032	0.193

4.4 Reproducing UIA

The UIA framework was implemented by the original authors exploiting Python and the most common Python libraries, including PyTorch. In particular, since the BERT models used to implement the encoders of the framework require a large amount of computational resources (especially of memory), the framework has been trained on multiple NVIDIA GPUs in a distributed way, therefore the PyTorch version that we used is the one that provides support also for CUDA (Compute Unified Device Architecture, the architecture of the NVIDIA GPUs).

The UIA framework considers the KS, QBE and CIR tasks jointly and the Amazon ESCI dataset needs to be properly handled for this purpose. Therefore, in Section 4.4.1 we explain how we used the dataset to train the model and in Section 4.4.2 we discussed the achieved results.

4.4.1 Dataset Management

The Amazon ESCI dataset, as explained in Section 3.5, is processed in order to create three separate datasets (one for each task: KS, QBE, CIR) and then each of these new datasets is splitted into training, validation and test sets independently. Since the UIA framework considers the three tasks jointly it must be trained exploiting the training sets of both KS, QBE an CIR tasks together and to do this the three training sets are provided in input sequentially to the model.

As explained in Section 3.4, the framework is optimized following a two stage approach and each of them requires to perform some negative sampling. While for the *Phase 2* the negative sampling procedure employs the UIA item encoder, the *Phase 1* makes use of some results obtained with BM25. In our case to carry out the negative sampling required by *Phase 1* we exploited the output of the search that we obtained using Pyserini while reproducing the performance of the paper on the BM25 baseline (see Section 4.3).

Note that the all this processing and, consequently, the training of the framework is executed starting from the datasets modified according to what is explained in Section 4.2.

4.4.2 Reproducibility Results

The results that we were able to achieve in terms of performance of the framework are reported in Table 4.2. As can be seen from the table, analogously to

Table 4.2: Original and reproduced UIA performance.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
original UIA	0.532	0.360	0.533	0.251	0.199	0.543	0.490	0.493	0.868
our UIA	0.491	0.327	0.484	0.442	0.374	0.673	0.463	0.459	0.833

what happened with the baselines (see Section 4.3), we were able to replicate the performance of the framework for the KS and CIR tasks while we performed better on the QBE task. Noticing that in the code of UIA made available by the authors two different versions of the training set for the QBE task were generated and used, the first corresponding to the complete set and the second containing only half of the training data, we decided to try to train the framework using the second version to understand if this was the origin of the strange behaviour. The performance obtained using half of the data for the QBE training set are reported in Table 4.3 and, based on those, we can conclude that probably the second version of the dataset has been used only for testing purposes by the authors. Therefore, even in this case (as for BM25, see Section 4.3), the possible explanation of this strange behaviour can be the fact that the authors might have used a sampled version of the ground truth for the QBE task, given also the fact that the QBE dataset is the largest.

There are several reasons that might be the cause of the difference in performance that obtained: we trained the framework only for 24 epochs (instead

Table 4.3: Original and reproduced UIA (with half of the data for QBE) performance.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
original UIA	0.532	0.360	0.533	0.251	0.199	0.543	0.490	0.493	0.868
our UIA (half QBE)	0.510	0.341	0.504	0.316	0.250	0.561	0.455	0.452	0.838

of 48), we used smaller batches (48 samples for each batch instead of 384), the authors of the paper may have applied some further processing to the datasets without reporting it.

Table 4.4 shows the performance of the framework (with complete training set for QBE) after the *Phase 1*. We report also these results because *Phase 2* represents a step that is carried out just to increase the performance of UIA and does not have an impact on the data flow or on the structure of the framework, therefore the majority of our experiments were done training UIA just with *Phase 1* in order to save time (since each stage of training requires several days).

Table 4.4: Original and reproduced UIA (without *Phase 2*) performance.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
original UIA	0.532	0.360	0.533	0.251	0.199	0.543	0.490	0.493	0.868
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760

5

Potential Issues of UIA and Experiments

In this chapter we describe some potential issues that we found in the original paper [26] as well as some experiments that we performed to study and/or analyse UIA and its components. The problems found are both related to the datasets and to the architecture of the framework. Furthermore, for each issue/-experiment we will explain how we modified the framework and/or the dataset in order to perform further studies and eventually understand the magnitude of the problem.

In Sections 5.1 and we 5.2 describe two potential issues concerning how the dataset is processed; in Section 5.3 we report an experiment that we performed to better understand if the framework was actually learning how to distinguishing the three tasks; in Section 5.4 we depict the tests that we carried out to verify if UIA performed better when jointly trained on the Amazon ESCI dataset; eventually, in Section 5.5 we explain how we investigated whether combining the datasets of the different tasks allows to improve the performance.

For all the experiments reported in this chapter we used the same datasets (the Amazon ESCI dataset and the ones derived from it) and the same setup as those used to reproduce UIA, which are reported in Section 4.1. Furthermore, as explained in Sections 4.1 and 4.4 a single phase of the training with our experimental setups can take from 4 to 6 days, therefore given that the *Phase 2* of the optimization does not change the data flow or the structure of the framework but is used only to further improve the performance and considered that the ex-

periment that we executed are used only to perform comparisons, in all the tests reported in this chapter we trained the framework terminating after *Phase 1* to save time.

5.1 Random Sampling

The UIA framework is trained following a two stage approach, as explained in Section 3.4. In particular, by looking at the code made available by the authors, after the *Phase 1* is performed and prior to starting the *Phase 2* the datasets for the KS, QBE and CIR datasets are re-created and re-splitted into training, validation and test sets starting from the original Amazon ESCI dataset and following the same procedure described in Section 3.5. During the generation of the datasets the method *sample* of the *random* python module is called several times and to guarantee a deterministic behaviour across several executions, the random seed is properly set. Setting the random seed, however, does not ensure that the method *sample* returns always the same results at every call but it guarantees that the method returns the same results for the same call across multiple executions. Considering our case, the problem is that the Python scripts files used to generate the datasets before *Phase 1* are different from the ones used before *Phase 2* and, even if the algorithm implemented is identical, the number of times that the method *sample* is called before the calls used for the actual generation of the dataset is not the same, therefore the datasets created before *Phase 1* are slightly different from the ones produced before *Phase 2*. The code snippets 5.1 and 5.2 represent a simplified example of the problem. As can be seen, in both the snippets we declare a vector x that represents the dataset and contains the numbers 1, 2, 3, 4, 5, 6 which can be thought as the ids of some samples. Furthermore, we set the random seed and, tanks to that, in both the snippets the different calls of the method sample return always the same results in the same order. Anyway, in the snippet 5.1 we store the output of the method sample, which contains the ids of the samples selected to form a potential training set, after two calls while in snippet 5.2 we do the same thing but after three calls and this leads to have different training sets.

```
1 import random
2
3 x=[1,2,3,4,5]
4
```

```
random.seed(10)

y=random.sample(x,3) #sampling 3 items from array x
print(y) #Output: [3,1,5]

y=random.sample(x,3) #sampling 3 items from array x
print(y) #Output: [2,1,3]

training_set=random.sample(x,3) #sampling 3 items from array x
print(training_set) #Output: [4,2,5]
```

Code 5.1: Phase 1 sampling example.

```
import random

x=[1,2,3,4,5]

random.seed(10)

y=random.sample(x,3) #sampling 3 items from array x
print(y) #Output: [3,1,5]

training_set=random.sample(x,3) #sampling 3 items from array x
print(training_set) #Output: [2,1,3]

y=random.sample(x,3) #sampling 3 items from array x
print(y) #Output: [4,2,5]
```

Code 5.2: Phase 2 sampling example.

We decided keep this error and to not correct it because our main goal was to reproduce the work of the paper [26] and to find potential problems based on the reproduced version of the framework. Furthermore, since in all the tests we used the version of UIA trained only according to *Phase 1* (skipping *Phase 2*), in our case, the just introduced issue can be ignored.

5.2 Data Leakage

The way in which the training, validation and test sets for the KS, QBE and CIR tasks are derived from the Amazon ESCI dataset may be the cause of a phenomenon called "data leakage", as introduced in Section 3.5.5. In particular, we recall that the original dataset (Amazon ESCI) is made of queries and each query

is paired to a set of products which are marked as exact, substitute, complementary or irrelevant matches for it. Furthermore, these queries are used to generate the KS dataset which is composed of pairs of the type [query, relevant item], the QBE dataset which is composed of pairs of the type [relevant item, similar item] and the CIR dataset which is composed of pairs of the type [relevant item, complementary item]. The key is the way in which the training, validation and test sets for each task are derived, given the fact that the framework is jointly trained on all the tasks. Specifically, the three obtained datasets are splitted independently and this may lead to have the same pairs (or very similar pairs) in the training set of one task and on the test set of another one. Imagine a situation in which there are two different queries that have a common item marked as exact match (relevant item) but for one of them a second item is marked as substitute (similar item) while for the other the same second item is marked as complementary (complementary item); in this case, we end up having the same pair in both the datasets for the QBE and CIR tasks and since the splitting into training, validation and test sets is performed independently, this may cause the pair to appear in the training set of the QBE task and in the test set of the CIR task (or the other way around). Since the system is jointly trained (thus all the samples of the training set of each task are provided in input during training) and considered that the only thing that differentiates the data coming from different training sets is the functionality $\mathcal{F}$, it is clear that by creating the datasets according to this procedure we can introduce some biases. We now report a real case in which this occurs: both the queries "*1/4" retractable air hose reel without hose*" and "*100' air hose retractable reel without hose*" share the same relevant item "*Primefit HRM38100 Manual Air Hose Reel with 100ft Capacity using 3/8" ID Air Hose*" but the item "*GARDENA Retractable Hose Reel 82-Feet With Convenient Hose Guide*" is marked as complementary for the first query and as similar for the second one. Note that this may lead to some leakage only between the QBE and CIR datasets, because the pairs of the QBE and CIR datasets are made of two items, while in the pairs of the KS dataset the first element is a query of the original dataset (and it is quite unlikely that the text of a query is equal to the title of an item) . Furthermore, if we consider pairs that are not affected by the problem described above, because of the nature of the original dataset and for the way in which it is processed, some pairs belonging to the test set of one task can result to be very similar to pairs in the training set of another task, leading, also in this case, to some biases (for the same reasons as the ones of the previous

problem). For example, the relevant item "*Emraw Pre Sharpened No 2 HB Wood Cased Premium Pencils with Eraser Top, Bulk Pack of 24 Pencil - For Professionals, Artists, Designers, Teachers*", the complementary item "*Arteza HB Pencils #2, Pack of 48, Wood-Cased Graphite Pencils in Bulk, Pre-Sharpened, with Latex-Free Erasers, Office & School Supplies for Exams and Classrooms*" and the similar item "*Grading Checking Erasable Pencils, Pre-Sharpened #2 HB Red Pencils, With Eraser Tops - 12-Pack*" generate pairs leading to the issue just described. In particular, the items reported produce the pair [relevant item, similar item] that ends in the training set for the QBE task and the pair [relevant item, complementary item] that ends in the test set for the CIR task. In this case, we referred to the datasets of the QBE and CIR tasks that are composed by pairs made of two items. Nonetheless, even if in the dataset of the KS task the first element of each pair is a query of the original dataset, since the queries performed by the users are usually very close to the title of the product that they are looking for, this second problem may occur between all the datasets. Note that in the two cases that we just described the fact that the dataset of each task is splitted into training, validation and test sets independently represents the main issue. In fact, because of this, the same queries may be used to generate data that end up both in a training set of one task and on the test set of another task and this may lead to make wrong estimations of the performances of the framework (since the data in the training sets and in the test sets is strongly related).

We now report some further insights of the datasets to better understand that the way in which the splitting is performed may have a large impact on the performance. From our analyses we were able to understand that only 32.7% of the query items (relevant items, used as requests/queries to perform the recommendation) of the test set of the QBE task are never seen in the training sets of the CIR and KS tasks and only 15% of the query items of the test set of the CIR do not appear in the training sets of the QBE and KS tasks. Moreover, 83.43% of the relevant items in the test set of the QBE task are not seen in the training set of the CIR task, while only 38.83% of the relevant items in the test set of the CIR task are not seen in the training set of the QBE task. In addition to the fact that the dataset for the CIR task is the smallest one (as explained in Section 3.7), this last consideration and, specifically, the fact that the majority of the relevant items in the test set of the CIR task are already seen in training, may be a further reason why the increase in performance for CIR is grater than the increase in performance for QBE and KS when we compare the UIA framework with the

baselines.

In Sections 5.2.1 and 5.2.2 we first introduce a methodology to produce new, unseen pairs to test the performance of the framework without requiring a re-training and then we introduce an alternative way to derive the needed datasets from the Amazon ESCI dataset that avoids the majority of the leakage issues but requires to re-train UIA.

5.2.1 New Queries

We developed a new algorithm to derive, from the available datasets, some new unseen pairs that can be used to evaluate the framework and, thus, to obtain a better estimation of the performance. Since the queries for the KS dataset correspond to the queries of the original Amazon ESCI dataset, it is impossible to generate new pairs for the KS task because we cannot create new queries. Therefore, we produced new pairs only for the QBE and CIR tasks.

To create the new pairs for the QBE task we first select from the original dataset all the queries having only exact or irrelevant matches but not substitute or complementary matches. This is done to be sure that those queries were not already used to generate pairs for the QBE or CIR datasets. After that, for each of the selected queries we get all the relevant items (exact matches) which have some similar products (based on the QBE dataset) and we add the products similar to the these items to a unique list. Therefore, after this step, for each query we obtain a list of the items similar to at least one of the items that are relevant for that query. Eventually, for each of the selected queries, for each of the relevant items for that query, if the item is not appearing in pairs of the QBE or CIR datasets then it is paired with each of the similar items in the list of the corresponding query (creating the pairs). The procedure used to create the new pairs for the QBE task is summarized in Algorithm 1. The consideration that led to the definition of this algorithm is the following: if a given item is an exact match for a certain request and a second item is a substitute match for that request, then if the same given item is an exact match for another request we can conclude that it is very likely that the second item would be a substitute match also for the second request (even if not explicitly marked in the original dataset). An empirical check of the generated pairs demonstrated that this consideration is correct.

The procedure to create the new pairs for the CIR task is symmetric to the one

Algorithm 1 Build not seen QBE queries

for each query without similar/complementary products **do**
get all the relevant products that have at least 1 similar product (obtained from other queries)
add to a list all the products similar to at least one of the products selected at the previous step
for each product relevant for the query but not seen in the QBE or CIR task **do**
create pairs pairing the relevant product with each of the similar products in the generated list
end for
end for

used for the QBE task. We first select from the original dataset all the queries having only exact or irrelevant matches but not substitute or complementary matches. This is done to be sure that those queries were not already used to generate pairs for the QBE or CIR datasets. After that, for each of the selected queries we get all the relevant items (exact matches) which have some complementary products (based on the CIR dataset) and we add the products complementary to these items to a unique list. Therefore, after this step, for each query we obtain a list of the products complementary to at least one of the items that are relevant for that query. Eventually, for each of the selected queries, for each of the relevant items for that query, if the item is not appearing in pairs of the QBE or CIR datasets then it is paired with each of the complementary items in the list of the corresponding query (creating the pairs). The procedure used to create the new pairs for the CIR task is resumed in Algorithm 2.

Algorithm 2 Build not seen CIR queries

for each query without similar/complementary products **do**
get all the relevant products that have at least 1 complementary product (obtained from other queries)
add to a list all the product complementary to at least one of the products selected at the previous step
for each product relevant for the query but not seen in the QBE or CIR task **do**
create pairs pairing the relevant product with each of the complementary products in the generated list
end for
end for

Similarly to what happened for the datasets of the three tasks, as explained in Sections 3.5.2, 3.5.3, 3.5.4, also in this case, for both QBE and CIR, to obtain the queries from the new generated pairs we simply get the unique relevant items

(first elements of the pairs) and to produce the ground truth corresponding to those information needs we consider for each relevant item the similar/complementary items appearing in the corresponding pairs. Note that the new sets of pairs are only used as test sets to evaluate the performance of the framework.

In Table 5.1 we report the number of requests and the total number of pairs (which corresponds to the ground truth size) for the training and test sets of each task and for the new sets generated. As can be seen from the table if we compare the number of requests and the number of pairs of the test sets that were used before with the ones of the new generated data we can notice that, given the number of request, there are way more pairs in the new generated data. In fact, if we consider the QBE task, in the new generated data for each request there are 20.7 pairs on average while in the base QBE test set for each request there are 5.7 pairs on average. The same happens for the CIR task since in the new generated data there are 6.3 pairs (on average) for each request while in the base CIR test set there are 2.9 pairs (on average) for each request. This difference leads to having a larger ground truth for the new test sets (since it is created based on the pairs) and, therefore, for each request there are more items considered as correct matches. Since when evaluating the performance of the framework this may have an impact, we decided to perform the evaluation using both the new test sets and some rescaled versions of them, which have been obtained by randomly sampling for each request a certain number of pairs, to reduce the size of the new test sets to the one of the base test sets. In our case for QBE we kept up to 7 pairs for each request while for CIR we kept up to 5 pairs (these numbers have been chosen empirically).

Table 5.1: Size of the base datasets and of the new queries.

Dataset	Set	Task	# Requests	# Pairs
Base Dataset	Training set	KS	54511	452081
		QBE	315575	1067739
		CIR	82580	184361
	Test set	KS	6814	88767
		QBE	39447	223998
		CIR	10323	30259
New Queries	Test set	QBE	167293	3457531
		CIR	90379	571962

Exploiting the generated datasets and their rescaled versions we evaluated

the performance of the UIA framework, which are reported in Table 5.2 (we recall that we used the version of the framework which has been trained only according to *Phase 1*). As can be seen, when using the new data, the performance

Table 5.2: Performance of UIA (without *Phase 2*) on the new queries.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
new queries (*Phase 1*)	-	-	-	0.291	0.170	0.273	0.259	0.195	0.465
new queries rescaled (*Phase 1*)	-	-	-	0.159	0.095	0.273	0.203	0.168	0.465

decreases significantly, especially with the rescaled version. Looking at these results it is clear that the data leakage actually represents a problem and that the framework does not only take advantage of the shared knowledge derived from the joint training of the system but it also benefits from the leakage.

Since this approach does not require to re-train the framework, we decided to carry out the evaluation also on the version of UIA which has been trained performing both the two phases. The performance of this version of the framework is reported in Table 5.3 and reflects the one obtained training UIA only according to *Phase 1*, avoiding *Phase 2*.

Table 5.3: Performance of UIA (with *Phase 2*) on the new queries.

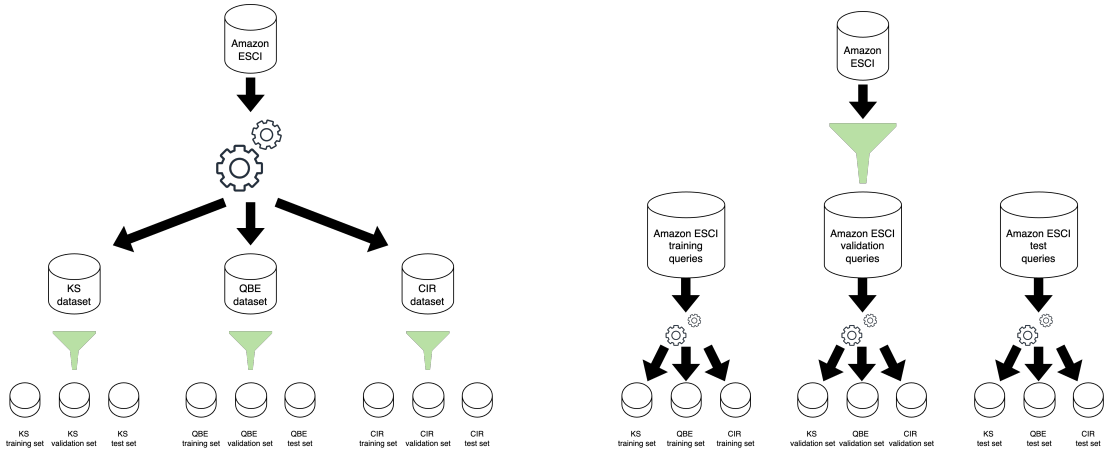
Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
original UIA	0.532	0.360	0.533	0.251	0.199	0.543	0.490	0.493	0.868
our UIA	0.491	0.327	0.484	0.442	0.374	0.673	0.463	0.459	0.833
new queries	-	-	-	0.404	0.262	0.357	0.317	0.247	0.522
new queries rescaled	-	-	-	0.238	0.153	0.357	0.255	0.216	0.522

5.2.2 Early Split

The main cause of the problem that we defined as data leakage is the way in which the original Amazon ESCI dataset is processed, for this reason we decided to start from the original dataset and manipulate it in a different way to avoid the leakage problem and simulate as much as possible the reality. Differently to the other approach (see Section 5.2.1), since we modify all the datasets, the previously trained framework cannot be used, therefore UIA must be re-trained.

The original processing procedures of the Amazon ESCI dataset required to first generate the datasets for each task and then split them independently

to obtain the corresponding training, validation and test sets. Our idea was to split the Amazon ESCI dataset queries into three sub-sets and then apply the procedure to generate the datasets for the tasks to each sub-set independently, to try to limit the problems deriving from the fact that before the same queries were used to generate data that could end up in the training set of one task and in the test set of another tasks. The three sub-sets in which the original dataset is splitted are: a training sub-set (training queries) which contains the 80% of the queries, a validation sub-set (validation queries) which contains 10% of the queries and a test sub-set (test queries) which contains the remaining 10% of the queries. Each of these three sub-sets is then considered independently and the same procedures described in Section 3.5 are applied to each of them to generate the pairs (without performing the splitting into training,validation and test set that was executed at the end of these algorithms). Using this method the training sets for the KS, QBE and CIR tasks are generated from the training sub-set, the validation sets for the tasks are created from the validation sub-set and the test sets for the tasks are derived from the test sub-set. This way we are sure that all the data ending up in the training set of a task cannot be generated from queries that contributed also to the generation of data for the test/validation set of another. Figure 5.1a reports a scheme representing the original processing pipeline while Figure 5.1b depicts the new pipeline.



(a) Original Split. **(b)** Early Split.

Figure 5.1: Amazon ESCI dataset processing.

Despite the fact that also if we exploit this new algorithm it may still happen that the same pair appear in the training set of one task and in the test set of another (fortunately in our case this problem can be ignored since a it happens

a very small number of times), an empirical analysis of the generated data revealed that the magnitude of the problem related to having similar pairs in the training set of one task and in the test set of the other is drastically reduced. Furthermore, another advantage derived from this processing procedure is that the performance of the framework can be estimated more accurately. This is possible because this way of manipulating the dataset is closer to the reality since it is more likely that both the retrieval and recommendation requests related to one topic (query of the original Amazon ESCI dataset) are carried out close in time and, given the fact that usually the training and test sets are built considering the temporal distribution of the requests (that in our case is not available), in a real scenario it is quite rare that the retrieval data related to one topic ends up in a training set while the recommendation data for the same topic ends up in a test set. Furthermore, a real user may repeat a (search) query over time expecting the same results or it may interact with the same item at different times considering as relevant the same recommendation output, for this reason we decided to keep the pairs that are shared among the training set of one task and the test set of another task.

Table 5.4 reports the performance of the framework, after the *Phase 1* of training, both using the original procedure ("our UIA" in the table) and the new procedure ("early split UIA" in the table) to generate the datasets. As can be seen, the performance with the new datasets drastically drops for the recommendation tasks but not for the retrieval task; this happens because, as explained before, the QBE and CIR datasets take more advantage from the leakage since they are actually generated from scratch exploiting the Amazon ESCI dataset while the KS dataset is obtained with minor modifications from the Amazon ESCI dataset and it results to be more realistic, since the original dataset has been created for retrieval purposes on the basis of real data. In Table 5.4 the last row ("mixed UIA") reports the performance of the version of UIA that is trained using the training set generated according to the just introduced procedure but is evaluated using the test sets created exploiting the original procedure. This way we are sure that there is some leakage between the training and the test sets used. We can notice how in this last case the framework performs better than the our reproduced version of the original UIA ("our UIA") but, especially for the recommendation tasks, the results are quite close to the ones of "our UIA", confirming that the leakage has a large impact. Based on all the considerations made above, we can conclude that the way in which we manipulate the dataset

plays a key role and, in this case, either or the leakage represents a real problem that cannot be neglected or the dataset is not appropriate for this framework.

Table 5.4: Performance of UIA (without *Phase 2*) on the new datasets.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
early split UIA (*Phase 1*)	0.467	0.306	0.451	0.053	0.034	0.129	0.041	0.037	0.147
mixed UIA (*Phase 1*)	0.577	0.401	0.585	0.356	0.283	0.594	0.400	0.391	0.740

5.3 No Functionality

The UIA framework makes use of only two encoders (as described in Section 3.3): the encoder $\mathbf{E}_{\mathcal{I}}$ which is used to map the items in the collection to some vector representation and the encoder $\mathbf{E}_{\mathcal{R}}$ which is used to map the requests to some vector representation. We recall that, since we use the Amazon ESCI dataset which has no user information, the *t-th* request submitted is encoded as $\vec{R}_t = \mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}_t\ \texttt{[SEP]}\ \mathcal{F}_t\ \texttt{[SEP]})$ while the *i-th* item in the collection is encoded as $\vec{I}_i = \mathbf{E}_{\mathcal{I}}(\texttt{[CLS]}\ \mathcal{I}_i\ \texttt{[SEP]})$. All the requests, independently from the task, are provided in input to the same encoder $\mathbf{E}_{\mathcal{R}}$, therefore the only thing that allows the framework to identify the task related to the requests is the functionality $\mathcal{F}$.

While studying the framework we realized that the amount of data composing the training set of each task was quite large and considering also the fact that the functionality $\mathcal{F}$ provided in input to the encoder the $\mathbf{E}_{\mathcal{R}}$ is the only thing allowing to distinguish among the different tasks, a doubt started growing: was UIA actually capable of learning from all the tasks jointly and then share the knowledge between them or was it only taking advantage from the fact that it was trained on a huge amount of data related to different tasks? In particular, the functionality $\mathcal{F}$ is simply a short textual string (in our case chosen between *"is_relevant_to"*, *"is_similar_to"* and *"is_complementary_to"* based on the task) that is concatenated to the request by means of a separator "[SEP]" and then the concatenation is provided in input to the encoder $\mathbf{E}_{\mathcal{R}}$ which is a BERT model. BERT is pre-trained to be able to recognize the separator and also to consider both the first and the second sentences (in our case the request and the functionality, respectively). Therefore, the functionality should be the element that guides BERT into turning the request in the best vector to match the most appropriate items based on the task. Nonetheless, since the amount of training data is very large

and considered that the training sets of the tasks have very different sizes (see Table 5.5, we recall that in the training phases the pairs are considered one at the time, so the size of the training set corresponds to the number of pairs in the set) and several similarities (see also the data leakage problem that we have reported in Section 5.2), there is the risk that BERT starts learning how to encode the request to match certain items without caring about the functionality. If this was the case, maybe it would not make sense to develop such a complex framework or it would be better to change its architecture (for example using different request encoders, one for each task and keeping a common item encoder to be able to share some knowledge).

Table 5.5: Size of the training and test sets for all the tasks.

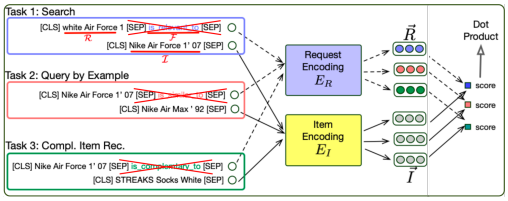
Set	Task	# Requests	# Pairs
Training set	KS	54511	452081
	QBE	315575	1067739
	CIR	82580	184361
Test set	KS	6814	88767
	QBE	39447	223998
	CIR	10323	30259

To answer to the question we decided to modify the framework by removing the functionality $\mathcal{F}$ and to re-train it considering all the data of the training sets of the different task as data belonging to a unique, large training set. In order to remove the impact of the functionality $\mathcal{F}$ we modified the input provided to the request encoder $\mathbf{E}_{\mathcal{R}}$ so that the *t-th* request is encoded as reported in Equation 5.1.

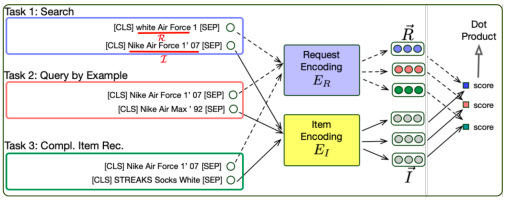
$$\vec{R}_t = \mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}_t\ \texttt{[SEP]}). \tag{5.1}$$

In Figure 5.2 we first show the original structure of the framework highlighting the portions that are removed (Figure 5.2a) and then we report the new architecture without the functionalities (Figure 5.2b).

The framework without the functionality $\mathcal{F}$ has been trained and evaluated using the same training and test sets used to reproduce the work of the paper [26]. As explained before, the training sets of the tasks have been joined together into a single training set while the test sets have been kept separated in order to be able to obtain the performance results for each tasks. Also in this case, as for the vast majority of the experiments that we carried out, the framework was re-trained only according to *Phase 1* (skipping *Phase 2*). The results of the evaluation



(a) Removing functionality from UIA architecure.



(b) UIA architecture without functionality.

Figure 5.2: UIA without the functionality $\mathcal{F}$.

are reported in Table 5.6. As can be seen from the table, in general, the perfor-

Table 5.6: Performance of UIA (without *Phase 2*) with and without the functionality.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
UIA w/o $\mathcal{F}$ (*Phase 1*)	0.441	0.280	0.419	0.247	0.185	0.472	0.181	0.175	0.524

mance of the framework without the functionality decrease but, while for the KS and QBE tasks the results are quite close to the ones of the framework with the complete architecture, for the CIR task the gap is bigger. From our studies we concluded that this could be happening because the KS and QBE tasks have larger training sets than CIR (see Table 5.5) and because end goal of the KS and QBE tasks are quite close if compared to the one of CIR . In general, in fact, KS and QBE try to retrieve the most similar things to some input provided, the first aiming to find the most relevant items to a user request and the second trying to recommend the most similar products to an item, while CIR attempts to retrieve items that are complementary to other items. Therefore, removing the functionality from UIA may affect more the CIR task because it has the smallest training set and it presents least commonalities with the other tasks. We can conclude that the functionality $\mathcal{F}$ actually plays a role in the framework, especially for the CIR task.

5.4 Isolated Tasks

Joint IR and RecSys, as explained in Section 2.5 of Chapter 2, is a promising research field and the UIA framework represents the currrent state-of-the-art. In particular, it seems that creating models that carry out both the IR and RecSys

tasks jointly allows to improve the performance by sharing knowledge between them.

Training and evaluating the UIA framework on the tasks in isolation corresponds to optimize and evaluate three separate instances of UIA, the first trained using only the training set of the KS task and evaluated using only the test set of the KS task , the second trained using only the training set of the QBE task and evaluated using only the test set of the QBE task and the third trained using only the training set of the CIR task and evaluated using only the test set of the CIR task. In Figures 5.3, 5.4, 5.5 we report the architecture of the three instances of the framework, each optimized on a specific task in isolation.

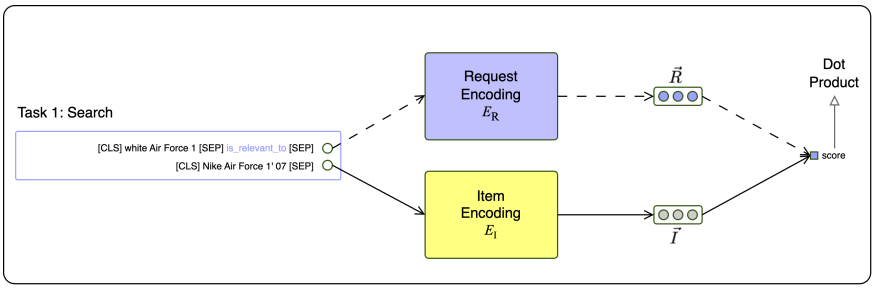


Figure 5.3: UIA framework for KS in isolation.

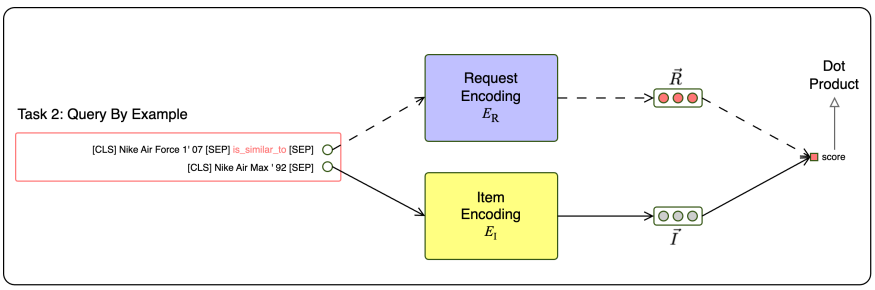


Figure 5.4: UIA framework for QBE in isolation.

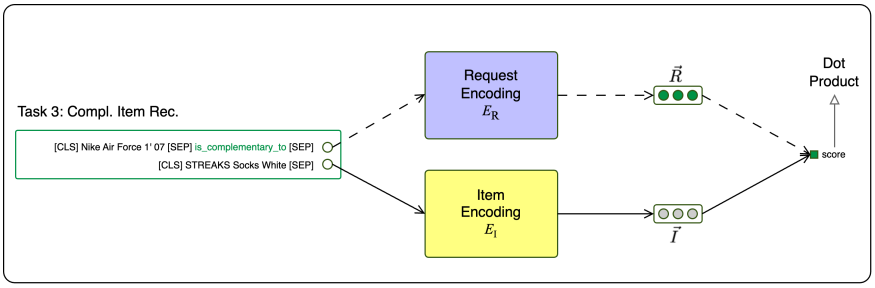


Figure 5.5: UIA framework for CIR in isolation.

The authors of the paper [26] that introduced UIA provided the performance of the framework on Lowe's dataset both when it is trained on all the tasks jointly and when is optimized on the different tasks one by one in isolation. We reported those results in Table 5.7 and, as can be seen, it seems that UIA performs better when jointly trained. Note that in the table "Lowe's UIA" refers to the version of UIA that is jointly optimized on the three tasks while "Lowe's UIA isolated tasks" encloses the three instances of the framework each trained on a single task. Furthermore, all the instances/versions of UIA whose results are reported in the table correspond to instances/versions of UIA that have been optimized performing both *Phase 1* and *Phase 2*. The authors of the paper, because of the limited amount of space available, for the instances of the framework trained on the tasks in isolation, reported only the value of nDCG.

Table 5.7: Original performance of UIA when jointly and not jointly trained on the Lowe's dataset.

	Keyword Search			**Query By Example**			**Complementary Item Rec.**		
Model	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
Lowe's UIA	0.340	0.399	0.880	0.433	0.495	0.945	0.438	0.432	0.836
Lowe's UIA isolated tasks	-	0.391	-	-	0.369	-	-	0.298	-

The potential issues that we discovered in the Amazon ESCI dataset, the way in which it is processed and used (discussed also in Section 5.2) and the fact that the authors of the paper only reported the value of nDCG on the Lowe's dataset for the framework trained on the tasks in isolation, without considering also the Amazon ESCI dataset, brought us to verify the performance of UIA when is optimized on the tasks one by one also for the Amazon ESCI dataset. Table 5.8, in fact, reports the results of the evaluation of both the reproduced version of the framework that is jointly trained on all the tasks ("our UIA (*Phase 1*)") and the three versions of UIA which are optimized on a single task (grouped as "our UIA isolated tasks (*Phase 1*)"). Note that also in this case, our experiments have been performed training the all the versions of the framework only according to *Phase 1* (skipping *Phase 2*).

As can be seen from Table 5.8, when the datasets derived from the Amazon ESCI dataset are used, the performance of the framework that is jointly optimized on all the tasks are from 3% to 5% lower than the ones of the versions of UIA trained on a single task. This behaviour is not consistent with the previous studies about joint IR and RecSys [24, 25, 27] and with the UIA results reported in the paper by the authors when the Lowe's dataset is used. We think that this

Table 5.8: Reproduced performance of UIA (without *Phase 2*) when jointly and not jointly trained on the Amazon ESCI dataset.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
our UIA isolated tasks (*Phase 1*)	0.506	0.340	0.493	0.324	0.257	0.561	0.414	0.412	0.779

may happen because in the architecture of the UIA framework the Attentive Personalization Network (APN) plays a fundamental role. When the Lowe's dataset is used and the framework is jointly trained, in fact, the personalization of a request related to a specific task is performed by using all the information contained in the user history, independently from the task. Therefore when UIA is trained on the tasks in isolation not only the knowledge sharing at the encoders level is missing but also the personalization is affected, since the user history contains only data from the same task on which the framework is optimized. For this reason, the jointly trained UIA is able to better exploit the shared knowledge between the tasks and, thus, perform better. When the Amazon ESCI dataset is used, instead, the APN is removed, reducing the complexity of the architecture of the framework but this eliminates one of the components that allows to take advantage from the knowledge sharing. We can conclude that the Amazon ESCI dataset does not seem to be the most appropriate dataset for UIA because additionally to the potential issues found performing other the experiments reported in Section 5.2, it also requires an oversimplification of the architecture. Note that the fact that on the Amazon ESCI dataset UIA performs better when not jointly trained does not affect the experiments, the studies and the potential issues discussed in this chapter.

5.5 Merged Tasks

While studying the UIA framework and performing some tests we notice some similarities and some differences between the various tasks, for this reason we decided to train multiple instances of the framework, each time merging two of the datasets of the KS, QBE and CIR tasks , in order to try to understand if some combination would have affected the performance in a positive way.

To merge the datasets of two tasks we need to combine their two training sets into a single training set, the two validation sets into a single validation set

and their two test sets into a single test set. Consider that a request $\mathcal{R}_t$ is encoded by the request encoder as $\vec{R}_t = \mathbf{E}_{\mathcal{R}}(\texttt{[CLS]}\ \mathcal{R}_t\ \texttt{[SEP]}\ \mathcal{F}_t\ \texttt{[SEP]})$ and, thus, the only thing that allows to determine the task to which the request is related is the functionality $\mathcal{F}$, since, when the Amazon ESCI dataset is used, according to the training procedure defined in Section 3.5.5, for each task we use a different functionality $\mathcal{F}$. Therefore, to merge two training/validation/test sets belonging to different tasks, it is sufficient to modify training procedure such that for the data coming from the datasets of the two tasks that must be combined the same functionality $\mathcal{F}$ is used.

In Table 5.9 we report the size of the training sets for all the tasks since it represents a key aspect to consider to understand the behaviour of the framework when it is trained after merging some tasks.

Table 5.9: Size of the training sets for all the tasks.

Task	# Requests	# Pairs
KS	54511	452081
QBE	315575	1067739
CIR	82580	184361

In Sections 5.5.1, 5.5.2 and 5.5.3 we report all the experiments that we carried out to test all the possible combinations of the datasets.

5.5.1 Merged KS and QBE

The first test that we performed is merging the datasets of the KS and QBE tasks. Both the KS and QBE tasks, in our case, are focused with retrieving the most similar items in a collection given a request. In text based retrieval (this is the case of the KS task), in fact, the request is represented by a short textual string that is provided by the user and it is used to retrieve the most relevant textual documents/items in a collection, which usually are the "most similar" documents/items to the given request. In QBE, instead, the main goal is still to retrieve the most similar items in the collection with respect to a request, which, this time, is represented by an item provided in input. Furthermore, since the CIR task focuses on retrieving the best complementary product to an item provided in input (which represents the request) combining KS and QBE could allow to better separate the tasks based on their goal. Considering all these aspects, merging the tasks this way could potentially increase the performance of

the framework.

Figure 5.6 reports the architecture of the framework when the KS and QBE tasks are combined. As explained before, to merge the tasks it is necessary to define the functionalities and use the same one for the tasks that must be combined, for this reason we decided to keep the functionality *"is_complementary_to"* for CIR while we used the functionality *"is_relevant_to"* for KS and QBE, since it seemed the most generic and appropriate for both.

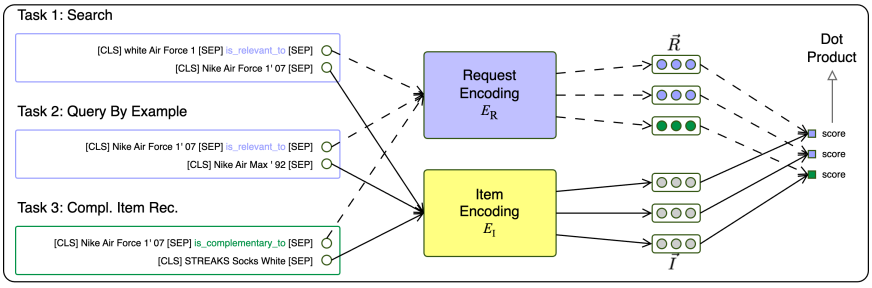


Figure 5.6: UIA framework when KS and QBE are merged.

Table 5.10 contains the results of the evaluation of the framework trained by considering the KS and QBE tasks as a single one. In the table we reported

Table 5.10: Performance of UIA (without *Phase 2*) when the KS and QBE tasks are merged.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
merged KS & QBE UIA (*Phase 1*)	0.466	0.303	0.447	0.272	0.208	0.510	0.347	0.338	0.740

both the performance of the version of UIA that we obtained reproducing the work of the paper [26] and the ones of the version of UIA for which the KS and QBE tasks have been merged. As can be seen, when the tasks are combined, for all the evaluation measures of all the tasks we obtain values that are from 1% to 2% lower than the case in which the tasks are not merged. This may be happening because even though the goal of the KS and QBE task is very similar, the type of the request that they expect in input is quite different: KS requests are human written queries while QBE requests are textual strings that correspond to the titles of some products. Furthermore, also CIR is negatively affected by the changes and this could be due to the fact that KS and QBE are the tasks with the largest training sets (see Table 5.9) . Their combination, in fact, leads to having

a single task that has an enormous training set if compared to the one of CIR, driving the system to give less importance to the CIR task.

5.5.2 Merged QBE and CIR

The second test that we performed is merging the datasets of the QBE and CIR tasks. Both the QBE and CIR tasks, in fact, represent recommendation tasks and take in input a request that is represented by an item, differently to KS which is a retrieval task and in which the requests are represented by short, human written textual strings. Despite that, QBE and CIR present also some differences, in fact, the first is concerned with selecting the items in the collection that are most similar to the item representing the request while the second one focuses on retrieving the items in the collection that are complementary to the item representing the request. Therefore, based on those considerations, merging the tasks this way makes it possible to better split the IR related tasks and the RecSys related tasks, still allowing to share knowledge between them and this could potentially increase the performance of the framework.

Figure 5.7 reports the architecture of the framework when the QBE and CIR tasks are combined. As explained before, to merge the tasks it is necessary to define the functionalities and use the same one for the tasks that must be combined, for this reason we decided to keep the functionality *"is_relevant_to"* for KS while we used the functionality *"is_similar_to"* for QBE and CIR, since it seemed the most generic and appropriate for both.

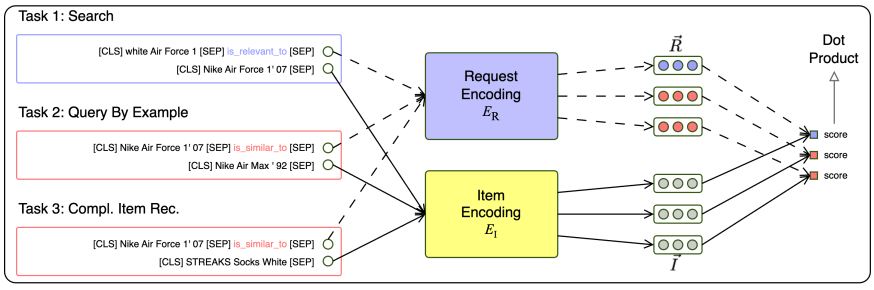


Figure 5.7: UIA framework when QBE and CIR are merged.

Table 5.11 contains the results of the evaluation of the framework trained by considering the QBE and CIR tasks as a single one. In the table we reported both the performance of the version of UIA that we obtained reproducing the work of the paper [26] and the ones of the version of UIA for which the QBE

Table 5.11: Performance of UIA (without *Phase 2*) when the QBE and CIR tasks are merged.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
merged QBE & CIR UIA (*Phase 1*)	0.471	0.308	0.455	0.268	0.207	0.522	0.248	0.239	0.633

and CIR tasks have been merged. As can be seen, when the tasks are combined, the framework does not perform better. In particular, if we compare the values of the evaluation metrics when the tasks are merged with the case in which they are not, we discover that for the QBE tasks the results are from 0.5% to 2.5% lower while for the CIR task the results are from 11% to 13% lower. This may be happening, because, as explained above, even though QBE and CIR are both recommendation tasks, their end goal is quite different and therefore in the merged training set, the data coming from one task may represent noise for the data coming from the other . Furthermore, we can notice that the performance of CIR are more affected than the ones of QBE and this may be due to the fact that the training set for QBE is almost six times larger than the one of CIR (see Table 5.9), meaning that the noise introduced by QBE towards CIR probably has a larger magnitude than the one introduced by CIR towards QBE. The results for the KS task, instead, are almost unaltered. In this case, it is possible that the slight decrease in the value of the measures derives from the fact that merging the QBE and CIR may also have an impact on the sharing of knowledge (since the framework learns less and/or wrong things).

5.5.3 Merged KS and CIR

The last test that we performed is merging the datasets of the KS and CIR tasks. This experiment was done for completeness and to verify that the framework was behaving properly. The KS and the CIR tasks, in fact, are the ones that have the least common aspects among all, since the requests for the KS tasks are represented by short, human written textual strings, while the requests for the CIR tasks correspond to the title of a product. Furthermore, while KS aims to retrieve the most relevant items in the collection according to a human written request, CIR tries to retrieve the items in the collection that are complementary to an item provided in input. Therefore, based on those considerations, merging the tasks this way should decrease the performance of the framework.

Figure 5.8 reports the architecture of the framework when the KS and CIR tasks are combined. As explained before, to merge the tasks it is necessary to define the functionalities and use the same one for the tasks that must be combined, for this reason we decided to keep the functionality *"is_similar_to"* for QBE while we used the functionality *"is_relevant_to"* for KS and CIR, since it seemed the most generic and appropriate for both (in this case we could have defined a new functionality for the KS and CIR tasks but we preferred to not do it to be more consistent with the other experiments discussed in Sections 5.5.1 and 5.5.2).

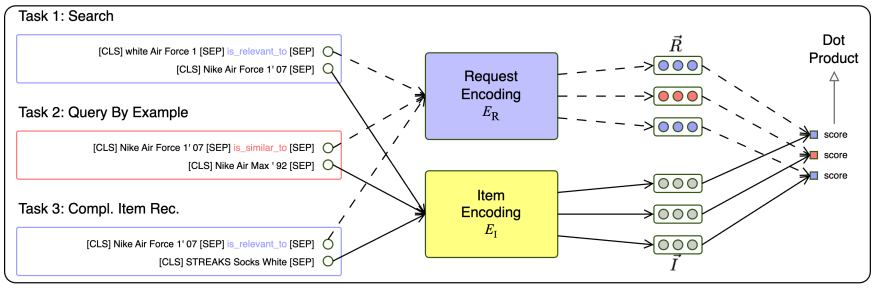


Figure 5.8: UIA framework when KS and CIR are merged.

Table 5.12 contains the results of the evaluation of the framework trained by considering the KS and CIR tasks as a single one. In the table we reported

Table 5.12: Performance of UIA (without *Phase 2*) when the KS and CIR tasks are merged.

Model	Keyword Search			Query By Example			Complementary Item Rec.		
	MRR	nDCG	Recall	MRR	nDCG	Recall	MRR	nDCG	Recall
our UIA (*Phase 1*)	0.477	0.313	0.461	0.294	0.227	0.531	0.361	0.353	0.760
merged KS & CIR UIA (*Phase 1*)	0.466	0.305	0.455	0.283	0.217	0.520	0.341	0.331	0.728

both the performance of the version of UIA that we obtained reproducing the work of the paper [26] and the ones of the version of UIA for which the KS and CIR tasks have been merged. As can be seen, when the tasks are combined, the behaviour of UIA confirms the expectations since the framework performs worse. In particular, if we compare the values of the evaluation metrics when the tasks are merged with the case in which they are not, we discover that for the KS tasks the results are about 1% lower while for the CIR task the results are from 2% to 3.5% lower. This may be happening, because, as explained above, the KS and CIR tasks are quite different from each other, therefore in the merged training set, the data coming from one task may represent noise for the data

coming from the other. Furthermore, we can notice that the performance of CIR are more affected than the ones of KS and this may be due to the fact that the training set for KS is larger than the one of CIR (see Table 5.9), meaning that the noise introduced by KS towards CIR probably has a larger magnitude than the one introduced by CIR towards KS. The results for the QBE task are also affected but, in this case, it is possible that the decrease in the value of the metrics (about 1%) derives from the fact that merging the KS and CIR may also have an impact on the sharing of knowledge (since the framework learns less and/or wrong things).

5.6 Final Considerations

Based on the experiments executed and on the potential issues found, which have been discussed in Sections 5.1, 5.2, 5.3, 5.4 and 5.5, we can make several considerations. The first thing that can be noticed is that there are various issues that are related to the way in which the Amazon ESCI dataset is processed to obtain the training, validation and test stets for the various tasks. In addition to that, when the Amazon ESCI dataset is used, the architecture of the framework must be simplified removing the personalization component, which seems to be a core part of UIA . All these things combined lead to think that the Amazon ESCI dataset is not appropriate to be used to test the performance of the framework and, thus, there is no known publicly available dataset that can be used for this purpose. Furthermore, the authors of the paper [26] performed some ablation studies, which are important to understand the behaviour of UIA (for example to evaluate if the framework works better when jointly trained), only using the Lowe's dataset which is a private dataset and, therefore, those tests cannot be reproduced. Eventually, the results that we obtained by training the framework after merging two of the tasks together (see Section 5.5) are consistent throughout all the experiments and, thanks to that, we were able to discover that, when the Amazon ESCI dataset is used, merging the tasks to try to exploit the similarities between them does not seem to improve the performance.

6

Conclusions and Future Work

In this work we analysed the field of joint Information Retrieval and Recommender Systems also highlighting the common and different aspects between IR and RecSys. We studied the architecture of the UIA framework and how the publicly available Amazon ESCI dataset has been processed and used to optimize the system. Furthermore, we discussed about the potential issue that there could be when using the Amazon ESCI dataset, which are related both to the processing of the dataset itself and to the architecture of the framework. We discovered, in fact, that the way in which the dataset is processed may lead to a major issue that we addressed as "data leakage". The "data leakage" causes to have the same/similar samples both in the training and in the test sets of the dataset and this alters the value of the metrics that are used to evaluate the performance of the framework. In addition, since the Amazon ESCI dataset, unfortunately, does not contain user information, the personalization component which is present in the architecture of UIA and strongly relies on users must be removed. This removal corresponds to erasing a core part of the framework and, thus, it makes the architecture too simple.

While analysing UIA and describing the experiments that we performed we gave also some hints on how the potentials issues found can be solved and how the framework can be enhanced in order to perform better.

Based on this work we will continue to study the field of joint IR and RecSys, trying to solve the potential issues found in UIA and developing new models. While doing this we will consider that:

1. Most of existing models are jointly optimized by simply aggregating data from retrieval and recommendation, without considering that user intents in IR and RecSys sometimes may be different.

2. Current models deeply focus on user history without considering that the search task (IR) could be performed by ”fresh” users or by external people/agents.

3. There are no appropriate, public datasets suitable for training and evaluating models performing both IR and RecSys tasks jointly.

For this reasons, we define our following future goals:

- Develop new models to carry out IR and RecSys tasks jointly that are also able to capture the differences (and not only the similarities) between the user intents. Moreover, aim at improving the current systems user management, in order to effectively exploit the user history while allowing fresh/external users to take advantage of the benefits of joint IR and RecSys. Focus also on realizing all the models in such a way that they can be implemented and used without requiring excessive computational resources.

- Create and make publicly available some new datasets necessary for an appropriate evaluation of the models.

Eventually, we hope also that the research community will start to increase its effort in this promising field in order to favor the development of new models and the sharing of knowledge between the researchers.

References

[1] Nicholas J. Belkin and W. Bruce Croft. "Information filtering and information retrieval: two sides of the same coin?" In: *Commun. ACM* 35.12 (Dec. 1992), pp. 29–38. ISSN: 0001-0782. DOI: 10.1145/138859.138861. URL: https://doi.org/10.1145/138859.138861.

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[3] Xiangnan He et al. "Neural Collaborative Filtering". In: *Proceedings of the 26th International Conference on World Wide Web* (2017).

[4] Jeff Johnson, Matthijs Douze, and Hervé Jégou. *Billion-scale similarity search with GPUs*. 2017. arXiv: 1702.08734 [cs.CV].

[5] Wang-Cheng Kang and Julian McAuley. "Self-Attentive Sequential Recommendation". In: *2018 IEEE International Conference on Data Mining (ICDM)* (2018), pp. 197–206.

[6] Vladimir Karpukhin et al. "Dense Passage Retrieval for Open-Domain Question Answering". In: *ArXiv* abs/2004.04906 (2020).

[7] J. Kiefer and Jacob Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *Annals of Mathematical Statistics* 23 (1952), pp. 462–466.

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[9] Jimmy Lin et al. "Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. , Virtual Event, Canada, Association for Computing Machinery, 2021, pp. 2356–2362. ISBN: 9781450380379.

DOI: 10.1145/3404835.3463238. URL: https://doi.org/10.1145/3404835.3463238.

[10] Yingqi Qu et al. "RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open Domain Question Answering". In: *NAACL*. 2021.

[11] Chandan K. Reddy et al. *Shopping Queries Dataset: A Large-Scale ESCI Benchmark for Improving Product Search*. 2022. arXiv: 2206.06588.

[12] Herbert E. Robbins. "A Stochastic Approximation Method". In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[13] Stephen E. Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Found. Trends Inf. Retr.* 3 (2009), pp. 333–389.

[14] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[15] Zihua Si et al. "When Search Meets Recommendation: Learning Disentangled Search Representation for Recommendation". In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '23. ACM, July 2023. DOI: 10.1145/3539618.3591786. URL: http://dx.doi.org/10.1145/3539618.3591786.

[16] Fei Sun et al. "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019).

[17] Afrina Tabassum et al. *Hard Negative Sampling Strategies for Contrastive Representation Learning*. 2022. arXiv: 2206.01197 [cs.LG]. URL: https://arxiv.org/abs/2206.01197.

[18] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[19] Wikibooks. *Artificial Neural Networks/Print Version — Wikibooks, The Free Textbook Project*. [Online]. 2013. URL: https://en.wikibooks.org/w/index.php?title=Artificial_Neural_Networks/Print_Version&oldid=2501560.

[20] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771 (2019).

[21] Lee Xiong et al. "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval". In: *ArXiv* abs/2007.00808 (2021).

[22] Peilin Yang, Hui Fang, and Jimmy Lin. "Anserini: Enabling the Use of Lucene for Information Retrieval Research". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 1253–1256. ISBN: 9781450350228. DOI: 10.1145/3077136.3080721. URL: https://doi.org/10.1145/3077136.3080721.

[23] Peilin Yang, Hui Fang, and Jimmy Lin. "Anserini: Reproducible Ranking Baselines Using Lucene". In: *J. Data and Information Quality* 10.4 (Oct. 2018). ISSN: 1936-1955. DOI: 10.1145/3239571. URL: https://doi.org/10.1145/3239571.

[24] Hamed Zamani and W. Bruce Croft. *Joint Modeling and Optimization of Search and Recommendation*. 2018. arXiv: 1807.05631 [cs.IR].

[25] Hamed Zamani and W. Bruce Croft. "Learning a Joint Search and Recommendation Model from User-Item Interactions". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM '20. Houston, TX, USA: Association for Computing Machinery, 2020, pp. 717–725. ISBN: 9781450368223. DOI: 10.1145/3336191.3371818. URL: https://doi.org/10.1145/3336191.3371818.

[26] Hansi Zeng et al. *A Personalized Dense Retrieval Framework for Unified Information Access*. 2023. arXiv: 2304.13654 [cs.IR].

[27] Kai Zhao et al. "Joint Learning of E-commerce Search and Recommendation with a Unified Graph Neural Network". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM '22. Virtual Event, AZ, USA: Association for Computing Machinery, 2022, pp. 1461–1469. ISBN: 9781450391320. DOI: 10.1145/3488560.3498414. URL: https://doi.org/10.1145/3488560.3498414.