PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

BERT FOR SCIENTIFIC ARTICLES RECOMMENDATIONS USING OPEN SOURCE INFORMATION

BERNARDO BARÍAS COMPAGNONI

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
MARCELO ARENAS

Santiago de Chile, January 2023

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

BERT FOR SCIENTIFIC ARTICLES RECOMMENDATIONS USING OPEN SOURCE INFORMATION

BERNARDO BARÍAS COMPAGNONI

Members of the Committee:
MARCELO ARENAS
JUAN REUTTER
ANDREA RODRÍGUEZ
MANUEL CARPIO

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, January 2023

Gratefully to my family

ACKNOWLEDGEMENTS

Acknowledgements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the last few years, language models have made great advances in Natural Language Processing (NLP). In particular, the pre-training and fine-tuning of models such as BERT (Bidirectional Encoder Representations from Transformers) and its derivatives have become the state of the art in many language understanding tasks (Devlin et al., 2018). One interesting research field that uses NLP language models is the one that studies the similarity between texts (Shahmirzadi, Lugowski, & Younge, 2019; Wang & Dong, 2020). These texts can be anything from large documents or paragraphs, to sentences or short phrases. Much of the difficulty with this problem is that text, in general, is not well structured. Different techniques have been used to try to understand the context of texts and thus understand the semantics of the documents. Naturally, if the texts are longer, this becomes a more difficult task. Within the study of similarity between texts, recent attempts have been made to study the relationship between pairs of scientific articles (Knoth, Novotny, & Zdrahal, 2010; Knoth et al., 2017; Tarnavsky, Harpaz, & Perets, 2021). When comparing scientific articles, it is possible to take advantage of the fact that the document is divided in different parts, such as the title, abstract, conclusions, and other sections. Although the text is still unstructured data, this partition gives some structure to the input text. This property also allows to work with smaller pieces of text, and thus, getting a better understanding of the context. In this work we use BERT to propose a pipeline that, given a publication, delivers related publications: scientific articles that may be of interest to the reader. To achieve this we deal with two NLP problems applied to scientific articles: text classification and sentence-pair classification. The labels of these problems come from hierarchical strutured information provided by the authors. We use the base version of BERT to understand the semantic meaning of the publications using only the information from the abstracts and the title, and build models for each task. These models are evaluated in terms of precision, recall, and F_1 score.

Keywords: natural language processing, BERT, sentence pair classification, similarity measures.

RESUMEN

En los últimos años, los modelos de lenguaje han experimentado grandes avances en el Procesamiento del Lenguaje Natural (NLP). En concreto, el pre-entrenamiento y el desarrollo de modelos como BERT (Bidirectional Encoder Representations from Transformers) y sus derivados se han convertido en el estado del arte para muchas tareas de comprensión del lenguaje. Un campo de investigación interesante que utiliza modelos lingüísticos de PNL es el que estudia la similitud entre textos (Shahmirzadi et al., 2019; Wang & Dong, 2020). Estos textos pueden ser desde grandes documentos o párrafos, hasta oraciones o frases cortas. Gran parte de la dificultad de este problema radica en que los textos, en general, no están bien estructurados. Se han utilizado distintas técnicas para tratar de entender el contexto de los textos y comprender así la semántica de los documentos. Naturalmente, si los textos son más largos, esto se convierte en una tarea más difícil. Dentro del estudio de la similitud entre textos, recientemente se ha intentado estudiar la relación entre pares de artículos científicos (Knoth et al., 2010, 2017; Tarnavsky et al., 2021). Al comparar artículos científicos, aprovechamos que el documento está dividido en distintas partes, como el título, el resumen, las conclusiones y otras áreas. Aunque siguen siendo datos no estructurados, dan cierta estructura al texto de entrada. Esta propiedad también nos permite trabajar con fragmentos de texto más pequeños y, por tanto, comprender mejor el contexto. En este trabajo utilizamos BERT para proponer un pipeline que, dada una publicación, entregue publicaciones relacionadas: artículos científicos que puedan ser de interés para el lector. Para ello, abordamos dos problemas de NLP aplicados a artículos científicos: la clasificación de textos y la similitud entre pares de textos. Las etiquetas para los conjuntos de datos de estos problemas proceden de información jerárquica estructurada provista por los autores. Además, utilizamos la versión base de BERT para comprender el significado semántico de las publicaciones utilizando únicamente la información del

resumen y el título, mediante la construcción de modelos para cada tarea. Estos modelos fueron evaluados en términos de precisión, recall y puntuación F_1.

Conceptos Clave: procesamiento del lenguaje natural, BERT, similitud entre pares de textos, medidas de similitud.

1. INTRODUCTION

1.1. Motivation

The last decade has seen many advances in the study of natural language processing (NLP). This branch of artificial intelligence (AI) is about learning to process language like humans do. Specifically, language models such as BERT (Devlin et al., 2018) or its derivatives have had great achievements in tasks in this area, and have radically changed the paradigm of how language problems are treated: starting from understanding the context, one can begin to succesfully solve difficult problems.

In this work we are interested in comparing and relating scientific articles using NLP techniques. In particular, given an article submitted by a reader, we want to find articles related to it that are of interest to this reader. As the publications are long texts and the number of publications grows at a high rate, doing the work of comparing and finding relationships between scientific articles becomes a very difficult task for humans. This is why we have to take advantage of the new technologies and techniques that allow us to process large volumes of data, in order to learn how to make comparisons automatically. To narrow down our problem, we only focused on computer science scientific articles, since they are easy to access and their information is generally well structured.

We believe that we can use a small part of the documents to make comparisons between different publications and determine which ones are related under some predefined criteria, as in our case: that they are of interest to the reader. Moreover, we propose that using only the title and the abstract of the documents, which is in general open source information, we can find the relationships between the publications. We argue that the techniques that are based on making clusters or graphs using citations are not enough to solve the problem, since they carry the bias that authors present when referencing. This is due to the fact that authors can have citation niches, or self-citations (Hyland, 2003; Aksnes, 2003; Engqvist & Frommen, 2008; Teodorescu & Andrei, 2014). Although self-citations and citation

niches can strengthen the author's knowledge claims (Hyland, 2003), and thus can be positive, we believe that making recommendations based solely on this aspect leaves out other interesting articles that may be from other niches or authors. These methods also do not take into account that the main function of referencing other research is to provide relevant information for the understanding of the study, not to indicate which documents are related.

Our research involves using hierarchical structures provided by the authors to classify scientific publications. These structures are trees, and they are organized into nodes that represent the specific research fields to which the publication belongs. The level of specificity increases as we move further down the tree. In Figure 1.1 we can see an example

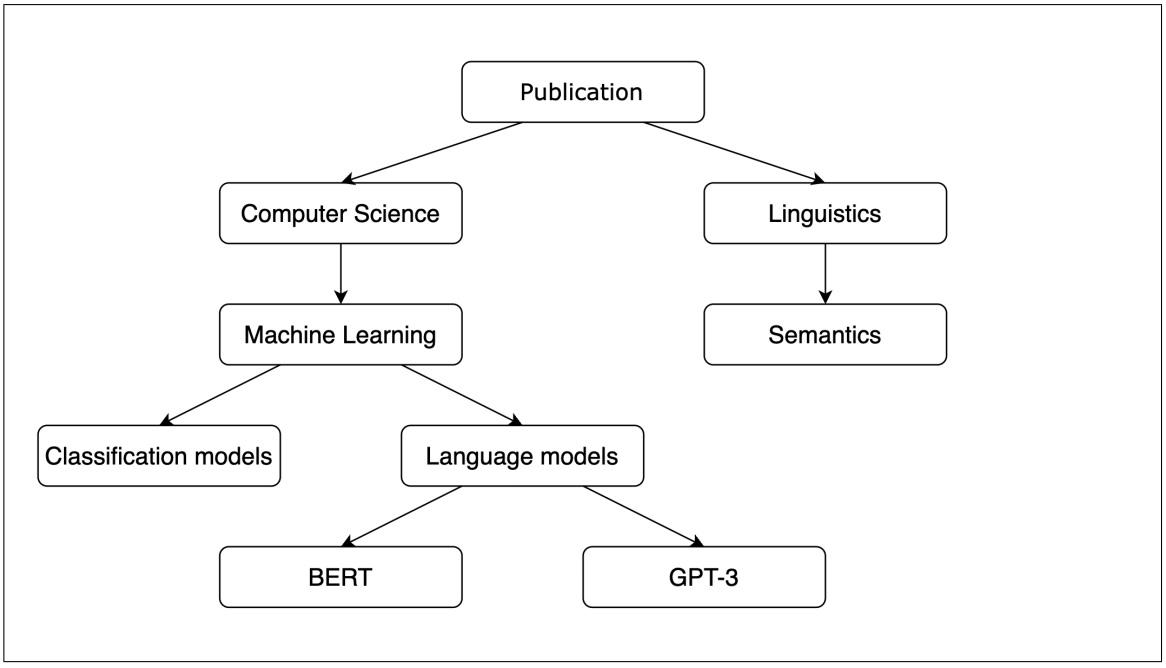


Figure 1.1. Example of a possible instance of a the hierarchical structure used in this research.

of this type of hierarchical structure. This tree corresponds to a scientific publication. As mentioned above, the nodes correspond to fields of study related to the scientific article, and the depth levels give more specificity about the subjects. If we look at the nodes of

depth 1 (“Computer Science” and “Linguistics”), we can deduce that, probably, this publication is about the study of language using computational techniques. If we go down a level, we have more detailed information. In addition to using computer science to deal with a language problem, we can see that the area of ”Machine Learning” is used, so we are probably talking about NLP. On the other hand, not only do we know that we are working with linguistics, but with semantics in language. This way we can keep going down levels, getting more specificity and clarification about what are the article main ideas. In this example, if we take all levels into account, we will probably come to the conclusion that this is a problem related to the liguistics-sematics field that is related to NLP, using classification models and linguistic models such as BERT and GPT-3. As one can see from the example of Figure 1.1, these kind of representation gives a lot of information about the publications. As the authors are the ones who provide this hierarchical structure representation for each document, we can be confident in its accuracy and quality. One of the main contributions of our research is using this hierarchical information to understand and classify the content in scientific articles, and because it comes directly from the authors, we can trust that it is accurate.

1.2. Main results

In this work we focus on the independent components that are necessary to build a recommender system that solves the problem stated above: given a publication, output a list of publications related to it, which may be of interest to the reader. Concretely, the main contributions we make with this research are the following:

- We propose using a hierarchical structure to understand the information in scientific articles, and consider it to be the ground truth of our experiments.

- We reintroduce a measure of similarity for trees that we call the Weighted Jaccard Index. This similarity measure takes two trees and, in relation to the number of shared nodes, non-shared nodes and the length of the branches, provides

a similarity value. To the best of our knowledge, we present the first modification to the Jaccard index that takes into account hierarchical structures, assigning different weights based on the depth of the tree.

- We build two independent classification models that are the main components of a recommender system pipeline. The first one consists of a model to classify a scientific article into a research area. The second consists of a model to determine how much two publications are related, using the criteria that they are of common interest to the reader. We do this by fine-tuning BERT using a dataset that has tree-shaped structures as labels, which were provided by the authors.
- To the best of our knowledge, we propose a new pipeline based on fine tuning the BERT language model to recommend articles to a reader given a publication.

1.3. Organization of the document

We start with the related work and explain the differences with the problem we try to solve in Section 2 and then present the main terminology used throughout this work in Section 3. We introduce and reformulate the definitions needed for the theory behind our research in Section 4. In Section 5 we present the pipeline of our solution. We then give in Section 6 the detailed design of our experiments, and in Section 7 we discuss the obtained results. Finally, we provide the summary of the research, some conclusions and mention possible lines of future work in Section 8.

2. RELATED WORK

There are different approaches that try to solve the problem of building relationships between scientific articles. We will briefly present these solutions and explain why we consider they are not sufficient for solving our problem.

One solution that can be observed in practice is the CORE recommender tool (Knoth et al., 2010, 2017). CORE is an aggregator of open access research papers from repositories and journals. One of the tools provided by the CORE platform is the CORE recommender, which is responsible for obtaining instant recommendations and discovering articles of interest given a user. CORE's initial approach in 2010 (Knoth et al., 2010) was to search for relationships between publications by vectorizing the documents using Tf-Idf (Jones, 1972) and then calculating cosine similarity. In this way they linked articles in pairs. To evaluate their performance they had human-made links, which they considered as ground truth. Then, CORE presented in 2017 a new solution based on vectorizing the title, abstract, authors and the year of publication, and then applying cosine similarity (Knoth et al., 2017) to compute the relation between publications. For this research, they evaluated the model using co-citation (Small, 1973) and citation networks as ground truth. Although this solution is focused on understanding the text, the language models used in our work have shown to have a considerably higher performance than the methods mentioned above (Devlin et al., 2018). In addition, we do not use the entire body of the article and focus on a very small part of the text corpus, which helps to reduce the noise generated by many sections of the articles.

Another tool that tries to solve the problem of relating scientific publications is the one provided by Connected Papers (Tarnavsky et al., 2021). They compute the similarity between texts using co-citation and bibliographic coupling (Kessler, 1963) over publications. We believe that this approach is not sufficient to solve the problem of relating two scientific articles, since we can often see patterns of self-citations and reference niches between

the authors (Hyland, 2003; Aksnes, 2003; Engqvist & Frommen, 2008; Teodorescu & Andrei, 2014). In this way, the relationship graph generated by Connected Papers is biased by these closed niches, which limits the circle of papers that the system can recommend.

Finally, the Toronto Paper Matching System (Charlin & Zemel, 2013) tries to solve a related problem: assignment of reviewers to articles at conferences. The relationship of this solution to our problem is that one step of the recommender system consists of comparing the articles written by the readers and the publications to be reviewed at the conference. The technique they use is word count representation to build vectors and then apply cosine similarity to get the relationship between the items. They evaluated their models using two conference reviewer-article datasets. Again, language models like BERT achieve much higher performance than the techniques used in this work.

The ideal scenario to evaluate the quality of the models that we build is to compare the performance metrics obtained in the experiments with those of other related works (for example, the mentioned above). However, to the best of our knowledge, there is no work in the literature that uses the ground truth we use, which are hierarchical structured labels called Index Terms (see Section 3.3). This is why we cannot use the evaluation datasets of the related works mentioned above, since their ground truth structures are different from the one we use.

3. PRELIMINARIES

In this section, we will describe the definitions and the notation we will use that is already defined in the literature.

3.1. Similarity measures

Similarity measures are something we work with frequently. For example, in graph theory we can use them to assign the weights of the edges between vertices. In this work, we repeatedly use similarity measures, which is why we think that readers should know well this concept. Definition 1 states the axioms of a similarity measure (S. Chen, Ma, & Zhang, 2009).

DEFINITION 1 (Symmetric similarity measure). *Given a set X, we say that a real-valued function $s(x, y)$ on the Cartesian product $X \times X$ is a symmetric similarity measure if, for every $x, y, z \in X$, it satisfies the following conditions:*

(i) $s(x, y) = s(y, x)$,

(ii) $s(x, x) \geq 0$,

(iii) $s(x, x) \geq s(x, y)$,

(iv) $s(x, y) + s(y, z) \leq s(x, z) + s(y, y)$,

(v) $s(x, x) = s(y, y) = s(x, y)$ *if and only if* $x = y$.

Furthermore, a normalized symmetric similarity measure is a symmetric similarity measure that satisfies the following property:

$$0 \leq s(x, y) \leq 1$$

For those readers who want a better understanding and a more detailed analysis of what these axioms mean and their relation to a distance metric, we strongly recommend further

reading the topic (S. Chen et al., 2009; Kosub, 2019; Rozinek & Mareš, 2021). Throughout this work we will omit the term symmetric, since we will only work with symmetric similarity measures. For example, instead of writing normalized symmetric similarity measure we will write normalized similarity measure. A typical example of a normalized similarity measure is the Jaccard index. The Jaccard index between two sets A and B is the total number of elements they have in common, divided by the total number of elements:

$$J(A, B) := \frac{|A \cap B|}{|A \cup B|}$$

We leave it as an exercise for the reader to prove that it is indeed a normalized similarity measure.

3.2. Subtrees of a fixed tree $\mathcal{T}$

Given a fixed tree $\mathcal{T}$, we are interested in the subtrees of $\mathcal{T}$. Since $\mathcal{T}$ is fixed we can represent each subtree of it as the set of nodes the subtree contains. Also, we are going to say that a subtree of $\mathcal{T}$ is a valid subtree of $\mathcal{T}$ if and only if the root of $\mathcal{T}$ is also in the subtree. We are going to use some set notations to refer to trees and subtrees and we are going to illustrate the main ideas along with some simple examples.

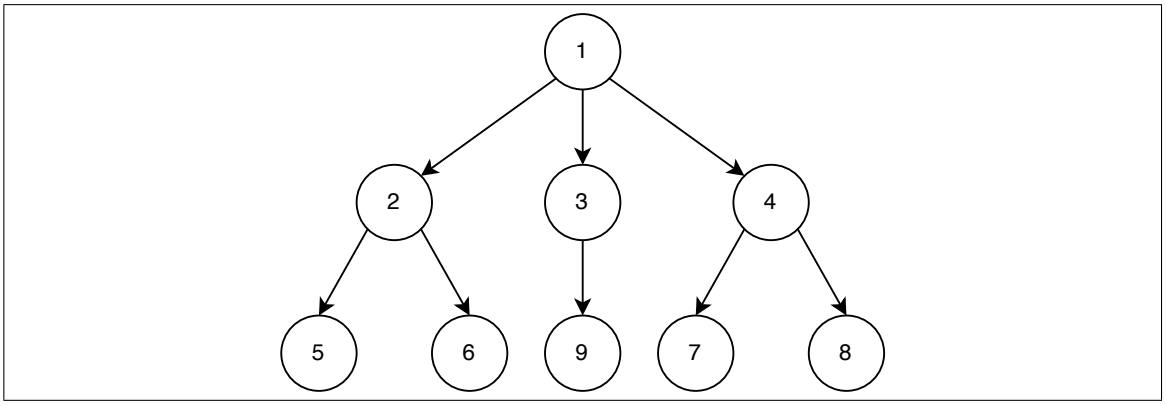


Figure 3.1. Example of a fixed tree $\mathcal{T} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, whose root is the node 1.

If we look at Figure 3.1, we can say that $1 \in \mathcal{T}$ since the node 1 belongs to the tree. Also, the tree defined as $T_1 = \{1, 2, 3, 4\}$ is a sub-tree of $\mathcal{T}$, and if $T_2 = \{1, 2, 3, 4, 7, 9\}$, then T_1 is a subtree of T_2.

However if $T_3 = \{2, 5, 6\}$, then we will say that T_3 is not a valid subtree of $\mathcal{T}$ because $1 \notin T_3$ (the root of $\mathcal{T}$ is 1, and $1 \notin T_3$). We can also use all the other operations over sets: for example, given two trees A and B, the intersection will be denoted as $A \cap B$ and the union $A \cup B$. The complement and the difference between trees will be written as usual. The symmetric difference between A and B, which is defined as $(A \setminus B) \cup (B \setminus A)$, will be denoted as $A \Delta B$. Note that unlike the intersection and the union of two trees, the complement, the difference and the symmetric difference can generate sets that are not trees (simply sets of nodes). Throughout this work, we will treat these operations as clearly as possible for the reader to understand it correctly.

On the other hand, we are also going to make use of some functions that are going to help us to formalize some properties that we are going to use. Given a subtree T of $\mathcal{T}$ with the same root as $\mathcal{T}$ and $N \in T$, $depth(N, T)$ will be the function that returns the depth of node N in the tree T (considering that the root of T has depth 0). In Figure 3.1, we have that $depth(1, \mathcal{T}) = 0$, $depth(5, \mathcal{T}) = 2$, and so on. The function $addChild(T, N)$ will return the tree T adding the node N, as long as the parent of N is in T. For example in Figure 3.1 if $T_1 = \{1, 2, 3\}$, then $addChild(T_1, 9) = \{1, 2, 3, 9\}$ but $addChild(T_1, 8)$ is not a valid use of this function, because $4 \notin T_1$. In any case, for the purpose of this work we will always use this operator in a valid way. As we mentioned before, we will use a fixed tree for this whole investigation. This will allow us to lighten the notation throughout this work by omitting the fixed tree in the functions. From now on, the reader will be able to deduce it from the context. For example, we will no longer need to write $depth(1, \mathcal{T})$, because if we know that we are working with $\mathcal{T}$, $depth(1)$ will be enough to know we are referring to the depth of node 1 of the tree $\mathcal{T}$.

Since we are working with sets, we are also going to make use of functions over sets. We are going to focus on two properties in particular. Let Ω be a finite domain and $f : 2^{\Omega} \to \mathbb{R}$ a set function. We say that the function f is an increasing function if $f(A) \leq f(B)$ holds for all $A \subseteq B \subseteq \Omega$. Furthermore, we say that f satisfies the submodularity property (Filmus, 2013) if $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ for all $A, B \subseteq \Omega$.

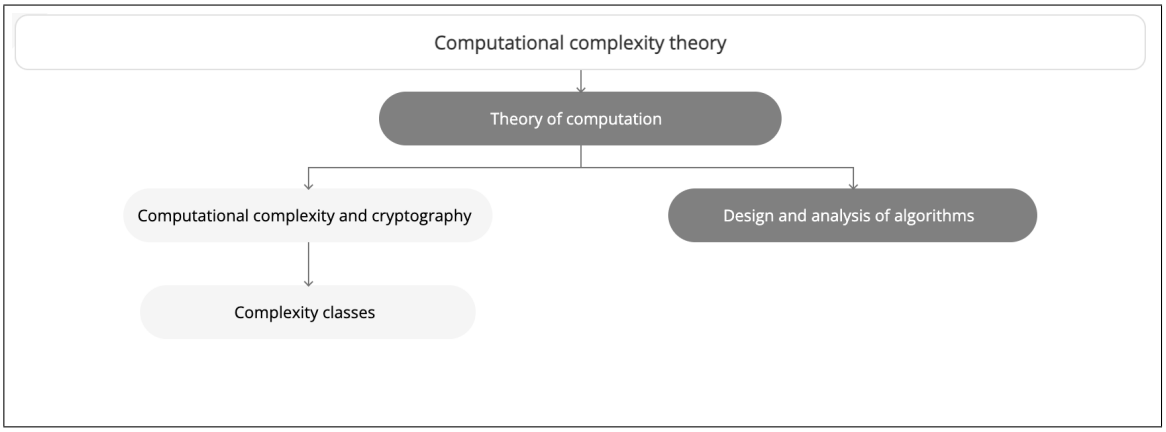


Figure 3.2. Example of the ACM Index Term for the article *Computational complexity theory* (Loui, 1996).

3.3. ACM Computer Classification System

The 2012 ACM Computer Classification System (CCS) is a standardized classification system for scientific publications. Although this system is adaptable to specific needs over time and can be modified, the ACM CCS provides a clear and standard classification structure. This system consists of a Directed Acyclic Graph (DAG) starting with 13 base categories defined in ACM (for more information you can see (for Computing Machinery, 2012)).

Every time a scientific article is submitted to ACM, the author must provide the corresponding indexing of the categories to which the publication belongs. The specific categorization of a publication is called the Index Terms. Such a specific categorization,

containing the knowledge of the authors themselves, leads to a very reliable classification system (obviously subject to the study areas provided by ACM). For example, the Index Term the article "Computational complexity theory" (Loui, 1996) is shown in Figure 3.2. We can see that Figure 3.2 shows "tree-structured" Index Terms, however we can also get other types of Index Terms structures. We can see in Figure 3.3 that the article "Algorithm-based recovery for iterative methods without checkpointing" (Z. Chen, 2011) has "DAG-structured" Index Terms: the sub-area "Parallel programming languages" has two different parents. Moreover, this node has diferent depths in the two paths shown in Figure 3.3.

If we connect all the labels provided by the ACM CCS (connecting the parents study areas with their children) we get a fixed DAG that we will call $\mathbf{DAG}_{\text{ACM}}$, which contains 1903 nodes. Thus, each publication in ACM can be viewed as a sub-DAG of $\mathbf{DAG}_{\text{ACM}}$. However, later we transform this DAG into a tree so that operations can be performed more conveniently. We will be left with an ACM fixed tree $\mathbf{T}_{\text{ACM}}$ that contains 1773 nodes and covers over $82,6\%$ of all the ACM publications we scraped. The entire transformation process can be seen in detail in Section 6.1.2.

3.4. BERT

In this NLP research we work with the BERT language model. BERT stands for Bidirectional Encoder Representations from Transformers which is presented in 2018 by Google (Devlin et al., 2018). Specifically, we work with both the base and large uncased versions of this language model, which were introduced in the original paper. BERT is designed in a way such that it comes with pre-trained bidirectional representations from unlabeled text. This method gives the model the "intuition" of what the language is, understanding its context. BERT was pre-trained using a book corpus (800 million words) and English Wikipedia (2.5 billion words) as input corpus in a self-supervised way. As a result of this pre-training, the BERT model can be adjusted to different NLP tasks, such as text classification and question answering, only by adding few simple layers and fine-tuning it (a

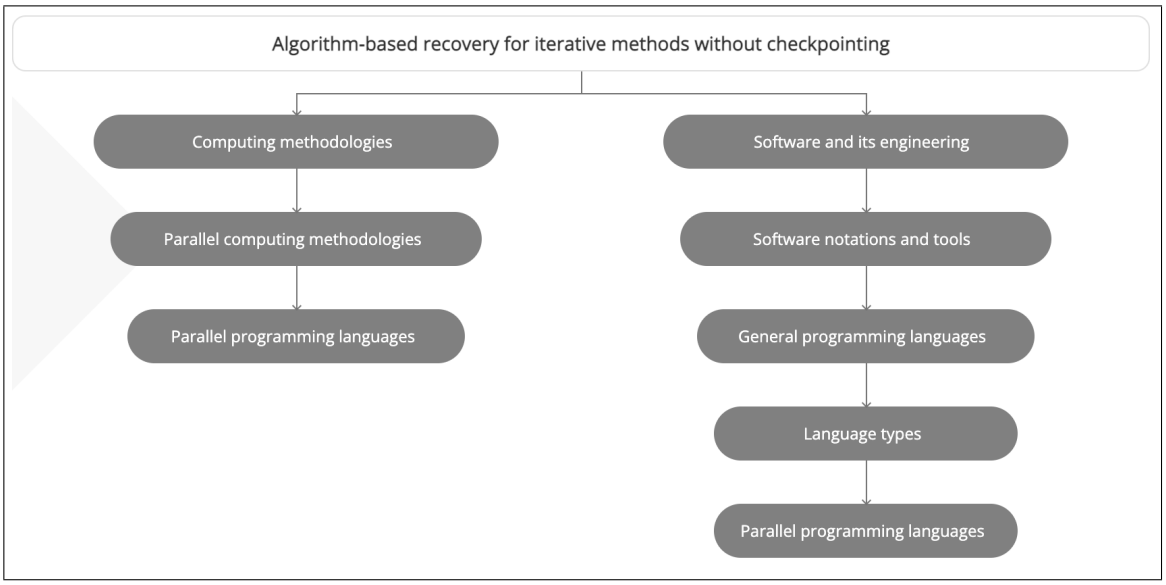


Figure 3.3. ACM Index Term for the article *Algorithm-based recovery for iterative methods without checkpointing* (Z. Chen, 2011).

specific way of training an existing model). We will briefly explain how these processes are carried out. If the reader wishes to understand further and delve in this model we recommend to read the original paper (Devlin et al., 2018).

The BERT pre-training process is done using two techniques: Mask Language Modeling and Next Sentence Prediction. In simple words, Mask Language Modeling consists of, given a phrase from the pre-training dataset, hiding some words from it and predicting them using the context given by the phrase. This action allows the model to learn how to contextualize words in sentences. On the other hand, Next Sentence Prediction is performed by taking two sentences, A and B, and solving the task of deciding if sentence B comes after A. This allows a better understanding of the continuity relationship between sentences. Note that BERT models and their derivatives come pre-trained with a corpus. However, you can continue pre-training with your own corpus.

On the other hand, the Bert fine-tuning process consists of, for each task, simply plugging the task-specific inputs and outputs into BERT and fine-tune a part or all the parameters

of the model. This is considerably faster than the pre-training part. The amount of data for this process is in the order of thousands of samples (it requires more data depending on the difficulty of the problem), which is considerably smaller than the amount of data used for the pre-training process. However, in most tasks we need good labeled datasets, which we know are difficult to obtain, so fine-tuning BERT can be very hard without the right data. Both pre-training and fine-tuning processes are depicted in Figure 3.4.

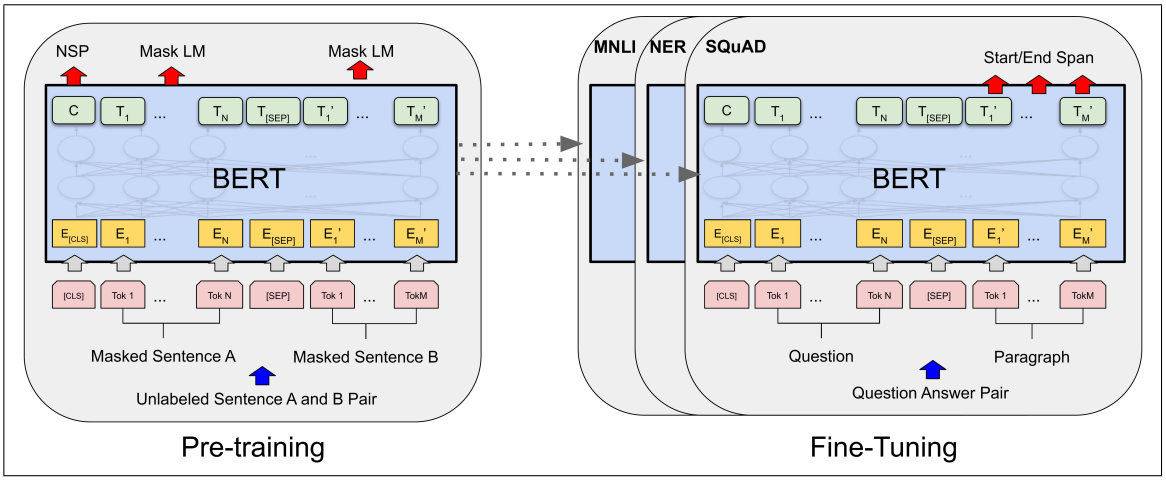


Figure 3.4. Pre-training and fine-tuning process ilustrations of BERT (Devlin et al., 2018).

As we mentioned before in the document, we plan on fine-tuning BERT for both components of the solution. We will explain this process in detail in Section 6.

4. METHODOLOGY

In this section we are going to delve deeper into what is the theory behind our research. We are going to introduce and reformulate some definitions that will be important to understand the experiments and results.

4.1. Similarity measure for ACM CCS

In section 3.2 we described the data structure that we use throughout this work. Much of this research has to do with calculating similarity between the sub-trees of $\mathbf{T}_{\text{ACM}}$ presented in section 3.3. Furthermore, we must measure similarity between many pairs of these sub-trees. We would like to use a similarity measure $s(\cdot,\cdot)$ that meets some conditions. Let T_1, T_2, T_3, T_4 be sub-trees of $\mathcal{T}$. Then:

C.1 If $T_1 \subseteq T_2$, $N_1, N_2 \in T_2$, $N_2 \notin T_1$, $depth(N_1) \leq depth(N_2)$ and T_1', T_1'' are the subtrees generated as follows

$$\begin{aligned} T_1' &:= T_1 \cup \{N_1\} \\ T_1'' &:= T_1 \cup \{N_2\}, \end{aligned}$$

then $s(T_1', T_2) \leq s(T_1'', T_2)$ and $s(T_1, T_1') \geq s(T_1, T_1'')$.

C.2 Assume that $T_1 \subseteq T_3$ and $T_2 \subseteq T_4$.

(a) If $T_1 \cap T_2 = T_3 \cap T_4$ then $s(T_1, T_2) \geq s(T_3, T_4)$.

(b) If $T_1 \Delta T_2 = T_3 \Delta T_4$ then $s(T_1, T_2) \leq s(T_3, T_4)$.

Property C.1 indicates that when calculating the similarity of two trees, the deeper a shared node is, the more it must contribute (positively) to the similarity, because the subtrees share more detailed information. For example, in Figure 4.1 we can see that T_1' and T_1'' have been created from the tree T_1. Since T_1' was built by adding node 5 (depth 2) and T_1'' by adding node 3 (depth 1) then $s(T_1', T_2) \geq s(T_1'', T_2)$. Property C.2 (a) indicates that a greater difference of nodes between the sub-trees imply a greater negative contribution to

the similarities. In order to visualize what C.2 (a) says, let us look at Figure 4.2a. One can observe that $T_1 \subseteq T_3$, $T_2 \subseteq T_4$ and $T_1 \cap T_2 = T_3 \cap T_4$. It would be reasonable for the similarity between T_1 and T_2 to be higher than the similarity between T_3 and T_4, since the only difference when comparing T_1 with T_2 vs. T_3 with T_2 is the set of nodes in B and E. These two sets make T_3 and T_4 "more" different. On the other hand, C.2 (b) states that a greater co-occurrence of nodes between the sub-trees imply a greater positive contribution to the similarities. To visualize what this condition is trying to say, let us look at Figure 4.2b. We see that $T_1 \subseteq T_3$, $T_2 \subseteq T_4$ and $T_1 \Delta T_2 = T_3 \Delta T_4$. Since the difference between T_1 and T_2 is the same as the difference between T_3 and T_4, then it makes sense that the pair of trees with the most parts in common is the most similar pair. Since T_3 and T_4, in addition to having the nodes of D in common, have the nodes of C in common, then T_3 with T_4 are more similar than T_1 with T_2.

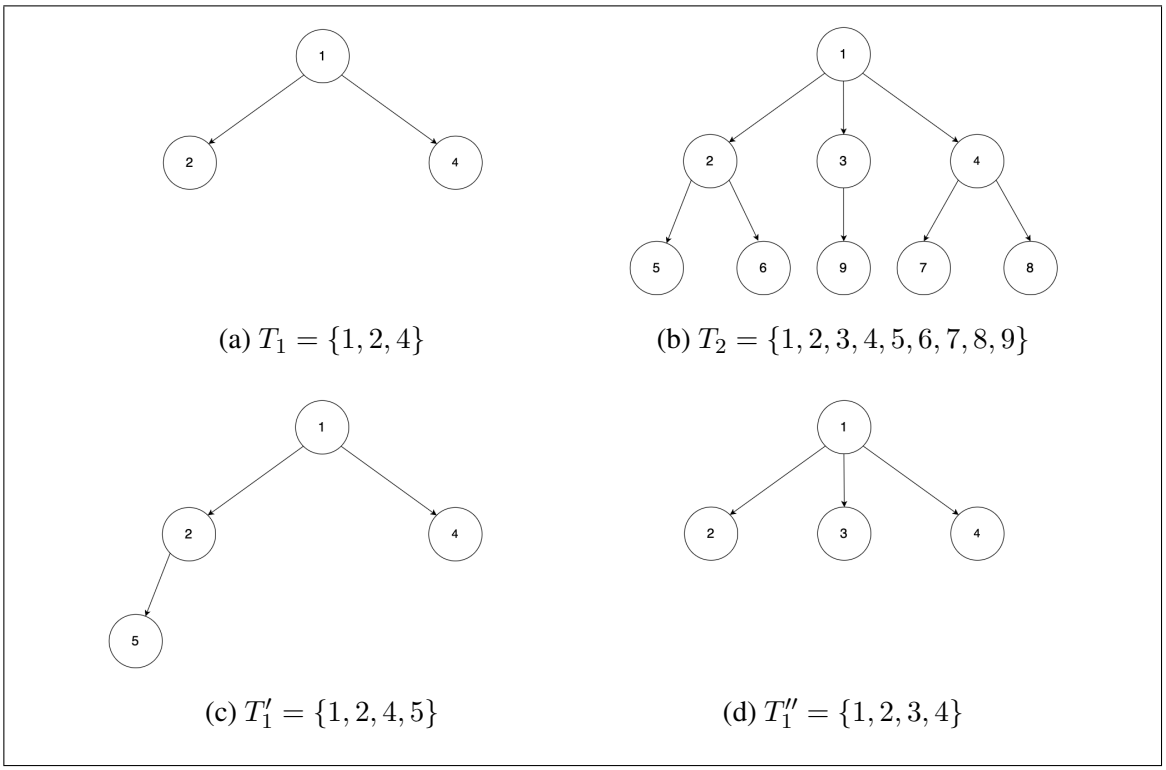
One property that can be derived from C.2 is the following one: if $T_1 \subseteq T_2 \subseteq T_3$ then $s(T_1, T_3) \leq s(T_2, T_3)$ and $s(T_1, T_2) \geq s(T_1, T_3)$. The proof of this implication is left as an exercise for the reader.

In the following section we are going to introduce a normalized similarity measure that satisfies the properties stated above.

4.2. Weighted Jaccard Index for tree similarity

Given two trees T_1, T_2 subtrees of $\mathcal{T}$ the Jaccard index $J(T_1, T_2)$ is well defined and can be calculated as usual (because we see every tree as a set). However, this similarity measure does not meet our requierements since we want it to fulfill the conditions C.1 and C.2. This is why we define the Weighted Jaccard Index for this data structure as follows.

DEFINITION 2 (Weighted Jaccard Index). *Let $\mathcal{T}$ be a fixed tree and T_1, T_2 subtrees of it. Let f be a monotonically increasing function. We define the sum transformed by f over a*



(a) $T_1 = \{1, 2, 4\}$ (b) $T_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

(c) $T_1' = \{1, 2, 4, 5\}$ (d) $T_1'' = \{1, 2, 3, 4\}$

Figure 4.1. Examples of valid sub-trees of the fixed tree presented in Figure 3.1.

set of nodes $S \subseteq \mathcal{T}$ as

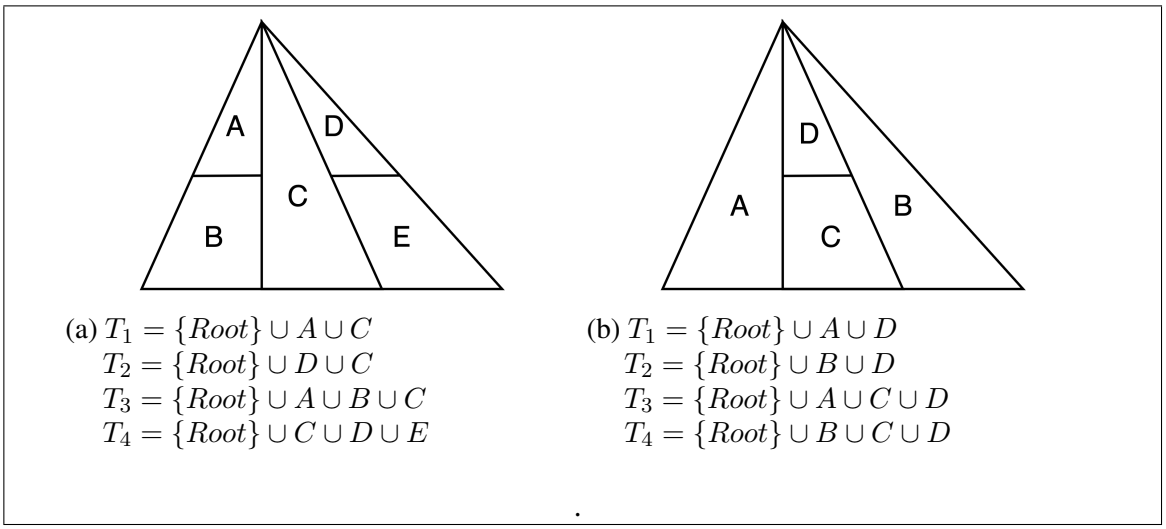
$$Sum_f(S) := \sum_{N \in S} f(depth(N))$$

Then the Weighted Jaccard Index for a function f is defined as

$$J_f(T_1, T_2) := \frac{Sum_f(T_1 \cap T_2)}{Sum_f(T_1 \cup T_2)}$$

Notice that every formula in Definition 2 is constructed based on $\mathcal{T}$. Having said this, note that by construction, the function Sum_f is a non-negative increasing set function. Moreover, it is easy to see that it satisfies the submodular property introduced in Section 3.2:

$$Sum_f(S \cup T) = \sum_{x \in S \cup T} f(depth(x))$$



(a) $T_1 = \{Root\} \cup A \cup C$
$T_2 = \{Root\} \cup D \cup C$
$T_3 = \{Root\} \cup A \cup B \cup C$
$T_4 = \{Root\} \cup C \cup D \cup E$

(b) $T_1 = \{Root\} \cup A \cup D$
$T_2 = \{Root\} \cup B \cup D$
$T_3 = \{Root\} \cup A \cup C \cup D$
$T_4 = \{Root\} \cup B \cup C \cup D$

.

Figure 4.2. Instances of trees created to illustrate Conditions C.1 and C.2

$$\begin{aligned} &= \sum_{x \in S} f(depth(x)) + \sum_{x \in T} f(depth(x)) - \sum_{x \in S \cap T} f(depth(x)) \\ &= Sum_f(S) + Sum_f(T) - Sum_f(S \cap T) \end{aligned}$$

THEOREM 1. *The Weighted Jaccard Index is a normalized measure of similarity. Moreover, it satisfies conditions C.1 and C.2.*

PROOF. (Theorem 1) We will first show that it is a normalized similarity measure. For this we must show the five axioms of Definition 1 and then show that the values of the function fluctuate between 0 and 1. The axioms 1, 2, 3 and 5 can be easily proven. We leave them as exercise to the reader. We also know that it is normalized since

$$0 \leq \frac{Sum_f(T_1 \cap T_2)}{Sum_f(T_1 \cup T_2)} = J_f(T_1, T_2) \leq 1$$

because $Sum_f(T_1 \cap T_2) \leq Sum_f(T_1 \cup T_2)$. The more interesting and slightly complicated axiom to prove is axiom 4. Since $J_f(A, A) = 1$ for every sub-tree A, it is enough to show that $J_f(A, C) + J_f(C, B) \leq J_f(A, B) + 1$. If A, B or C are the empty set then the

inequality trivially holds. Let us suppose then that $A, B, C \neq \emptyset$. We will use the following lemma and corollary directly in this proof.

LEMMA 1 ((Kosub, 2019)). *Let $f : 2^{\Omega} \to \mathbb{R}$ be a non-negative increasing set function that satisfies the submodular property and let $A, B, C \subseteq \Omega$. Then,*

$$f(A \cap C)f(B \cup C) + f(A \cup C)f(B \cap C) \leq f(C)(f(A) + f(B))$$

COROLLARY 1 ((Kosub, 2019)). *Let $f : 2^{\Omega} \to \mathbb{R}$ be a non-negative increasing set function that satisfies the submodular property and let $S, T \subseteq \Omega$. Then,*

$$f(S \cap T)f(S \cup T) \leq f(S)f(T)$$

As we stated previously, f is a non-negative, increasing function that satisfies the submodular property. We can now finish proving that J_f is a normalized similarity measure by using these tools.

$$\begin{aligned}
& J_f(A, C) + J_f(C, B) \\
= \; & \frac{Sum_f(A \cap C)}{Sum_f(A \cup C)} + \frac{Sum_f(B \cap C)}{Sum_f(B \cup C)} \\
= \; & \frac{Sum_f(A \cap C) \cdot Sum_f(B \cup C) + Sum_f(A \cup C) \cdot Sum_f(B \cap C)}{Sum_f(A \cup C) \cdot Sum_f(B \cup C)} \\
\leq \; & \frac{Sum_f(C)(Sum_f(A) + Sum_f(B))}{Sum_f(A \cup C) \cdot Sum_f(B \cup C)} \qquad \text{(by Lemma 1)}
\end{aligned}$$

Using Corollary 1 and taking $S := A \cup C$ and $T := B \cup C$ we have that

$$\begin{aligned}
& \frac{Sum_f(C)(Sum_f(A) + Sum_f(B))}{Sum_f(A \cup C) \cdot Sum_f(B \cup C)} \\
\leq \; & \frac{Sum_f(C)(Sum_f(A) + Sum_f(B))}{Sum_f((A \cup C) \cap (B \cup C)) \cdot Sum_f(A \cup B \cup C)} \\
\leq \; & \frac{Sum_f(C)(Sum_f(A) + Sum_f(B))}{Sum_f((A \cup C) \cap (B \cup C)) \cdot Sum_f(A \cup B)} \qquad (Sum_f(A \cup B) \leq Sum_f(A \cup B \cup C))
\end{aligned}$$

$$
\begin{aligned}
&= \frac{Sum_f(C)}{Sum_f((A \cup C) \cap (B \cup C))} \cdot \frac{Sum_f(A) + Sum_f(B)}{Sum_f(A \cup B)} \\
&= \frac{Sum_f(C)}{Sum_f((A \cap B) \cup C))} \cdot \frac{Sum_f(A) + Sum_f(B)}{Sum_f(A \cup B)} \\
&\leq \frac{Sum_f(C)}{Sum_f(C)} \cdot \frac{Sum_f(A) + Sum_f(B)}{Sum_f(A \cup B)} \qquad (Sum_f(C) \leq Sum_f((A \cap B) \cup C)) \\
&= \frac{Sum_f(A \cup B) + Sum_f(A \cap B)}{Sum_f(A \cup B)} \\
&= \frac{Sum_f(A \cap B)}{Sum_f(A \cup B)} + 1 \\
&= J_f(A, B) + 1
\end{aligned}
$$

So far we have shown that J_f is a normalized similarity measure. It remains for us to show that it satisfies the conditions C.1 and C.2.

To prove C.1 let $T_1 \subseteq T_2$, $N_1, N_2 \in T_2$, $N_2 \notin T_1$, $depth(N_1) \leq depth(N_2)$ and T_1', T_1'' be

$$
\begin{aligned}
T_1' &:= T_1 \cup \{N_1\} \\
T_1'' &:= T_1 \cup \{N_2\}
\end{aligned}
$$

Then, assuming that $N_1 \notin T_1$ we have that

$$
\begin{aligned}
J_f(T_1', T_2) &= \frac{Sum_f(T_1' \cap T_2)}{Sum_f(T_1' \cup T_2)} \\
&= \frac{Sum_f(T_1 \cap T_2) + f(depth(N_1)}{Sum_f(T_2)} \\
J_f(T_1'', T_2) &= \frac{Sum_f(T_1'' \cap T_2)}{Sum_f(T_1'' \cup T_2)} \\
&= \frac{Sum_f(T_1 \cap T_2) + f(depth(N_2)}{Sum_f(T_2)} \\
J_f(T_1, T_1') &= \frac{Sum_f(T_1 \cap T_1')}{Sum_f(T_1 \cup T_1')} \\
&= \frac{Sum_f(T_1)}{Sum_f(T_1')}
\end{aligned}
$$

$$
\begin{aligned}
& = \frac{Sum_f(T_1)}{Sum_f(T_1) + f(depth(N_1)} \\
J_f(T_1, T_1'') & = \frac{Sum_f(T_1 \cap T_1'')}{Sum_f(T_1 \cup T_1'')} \\
& = \frac{Sum_f(T_1)}{Sum_f(T_1'')} \\
& = \frac{Sum_f(T_1)}{Sum_f(T_1) + f(depth(N_2)}
\end{aligned}
$$

Since f is monotonically inscreasing and $depth(N_1) \leq depth(N_2)$, we have $f(depth(N_1)) \leq f(depth(N_2))$. This gives us $J_f(T_1', T_2) \leq J_f(T_1'', T_2)$ and $J_f(T_1, T_1') \geq J_f(T_1, T_1'')$. If $N_1 \in T_2$ then we do the same process and get the same results. This proves Condition C.1.

Finally, to prove Condition C.2 let us suppose that $T_1 \subseteq T_3$ and $T_2 \subseteq T_4$. First, if $T_1 \cap T_2 = T_3 \cap T_4$ then

$$
\begin{aligned}
J_f(T_1, T_2) & = \frac{Sum_f(T_1 \cap T_2)}{Sum_f(T_1 \cup T_2)} \\
& = \frac{Sum_f(T_3 \cap T_4)}{Sum_f(T_1 \cup T_2)} \\
& \geq \frac{Sum_f(T_3 \cap T_4)}{Sum_f(T_3 \cup T_4)}, \qquad \text{since } T_1 \cup T_2 \subseteq T_3 \cup T_4 \\
& = J_f(T_3, T_4)
\end{aligned}
$$

On the other hand, we want to prove that if $T_1 \Delta T_2 = T_3 \Delta T_4$ then $J_f(T_1, T_2) \leq J_f(T_3, T_4)$. We will expand this inequality to see what conditions must be satisfied for it to be true.

$$
\begin{aligned}
& J_f(T_1, T_2) \leq J_f(T_3, T_4) \\
\Leftrightarrow \quad & \frac{Sum_f(T_1 \cap T_2)}{Sum_f(T_1 \cup T_2)} \leq \frac{Sum_f(T_3 \cap T_4)}{Sum_f(T_3 \cup T_4)}, \qquad Sum_f \geq 0 \text{ so we can cross multiply} \\
\Leftrightarrow \quad & Sum_f(T_1 \cap T_2) \cdot Sum_f(T_3 \cup T_4) \leq Sum_f(T_3 \cap T_4) \cdot Sum_f(T_1 \cup T_2) \qquad (4.1)
\end{aligned}
$$

To simplify the notation we will define the following values:

$$\begin{aligned} A &= Sum_f(T_1 \cap T_2) \\ B &= Sum_f(T_3 \,\Delta\, T_4) \\ C &= Sum_f((T_3 \cap T_4) \setminus (T_1 \cap T_2)) \end{aligned}$$

If we rearrange A, B and C in equation 4.1 we have that

$$\begin{aligned} & Sum_f(T_1 \cap T_2) \cdot Sum_f(T_3 \cup T_4) \;\leq\; Sum_f(T_3 \cap T_4) \cdot Sum_f(T_1 \cup T_2) \\ \Leftrightarrow\;\; & A \cdot (A + B + C) \;\leq\; (A + C) \cdot (A + B) \\ \Leftrightarrow\;\; & A \cdot (A + B) + AC \;\leq\; A \cdot (A + B) + C \cdot (A + B) \\ \Leftrightarrow\;\; & AC \;\leq\; C \cdot (A + B) \\ \Leftrightarrow\;\; & AC \;\leq\; AC + BC \\ \Leftrightarrow\;\; & 0 \;\leq\; BC \end{aligned}$$

Since $A, B, C \geq 0$, the last inequality is always true. Therefore, $J_f(T_1, T_2) \leq J_f(T_3, T_4)$ is always satisfied under these conditions. This concludes the proof of Theorem 1. $\square$

5. ROADMAP

In this section we are going to explain how we are going to deal with the problem that we set out to solve. We will also present the process that will take place from the arrival of an abstract (input) to the generation of a list of related publications (output). Recall that while we present a complete outline of how the recommender system will work, in this research we focus on the training and testing of the two main components it must have.

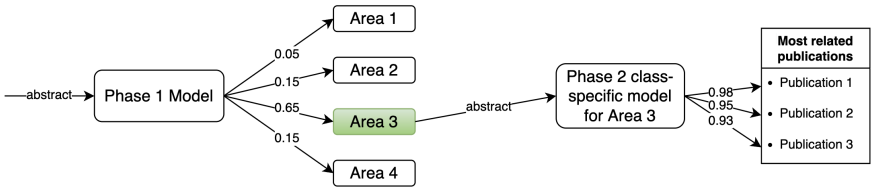
In order to solve our problem, we propose a pipeline that, given an abstract, returns related publications in a way that are of interest for the reader. To achieve this, we divided the problem into two phases. The first consists in determining, given an abstract, to which area of research it belongs. The second phase consists of determine how related two abstracts are. Within the same area of study, there may be publications that are more related than others. With this second phase we want to be able to determine which publications are strongly related. From now on we will refer to these phases as Phase 1 and Phase 2.

In general terms, the final pipeline consists of the concatenations of the resulting models for Phase 1 and Phase 2. Given a publication, the procedure to obtain the most related publications will be carried out as follows. First, it must be determined to which area of study the abstract belongs. This will be done by training a ML model to learn how to classify text into different classes. If the model resulting from Phase 1 predicts the research area of the input publication with high probability then we will take all the publications of that study area and compute the similarity with the model of that particular area. We will refer to these models as "class-specific models" If, on the contrary, the model of Phase 1 fails to determine with certainty the research area of the input abstract, then those that are most related to the input abstract will be searched among all the publications through a model who was trained using all classes. These models will be refered as "general models". Although we have not yet defined what does "high probability" stands for, we will after we extensively experiment on the pipeline. The reason behind this decision is that the values returned by a prediction model depend on the training of the model and can

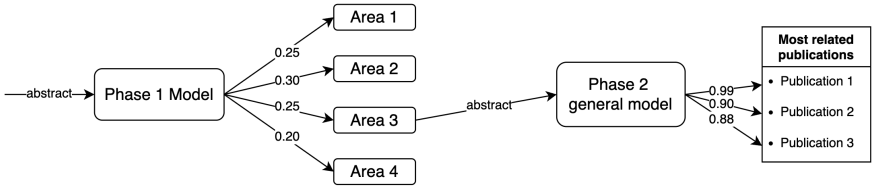
change between different models. Defining the "high probability" value will not be a part of this work.

In order to better visualize and understand this pipeline, let us look at Figure 5.1. Suppose that, given an input abstract, we want to look for the top 3 most related articles. Let us also suppose we set the "high probability" value to 0.50. This means that if the Phase 1 classification model manages to classify the input abstract into some class with probability greater than 0.5, then we trust the model's decision and believe that the publication really belongs to the area the model predicted. The first possible scenario is illustrated in Figure 5.1a. The Phase 1 model predicted that the publication belonged to Area 3 with probability $0.65 > 0.5$. Since we have exceeded the "high probability" value , we proceed to search for the most similar publications within all publications in Area 3. We do this with the class-specific model that we train using only publications from Area 3 and we obtain the top 3 articles most related to the input. The other possible scenario is illustrated in Figure 5.1b. Although the Phase 1 model predicted that the input corresponded to an instance of Area 2, it did not exceed the "high probability" value . Therefore we are not sure if the classification of the model is correct. So, we use the general model that was trained with all publications areas and we get the top 3 articles most related to the input.

The reasoning behind this pipeline is the following: in case the model is sure of its decision, the items most related to the input have to belong to the research area of the input. But if it is not sure of the decision it makes, then we cannot afford to blindly dismiss publications from the other research areas, because the input could belong to one of them.



(a) An abstract which was classified by the Phase 1 model as a publication of area 3 with probability 0.65.



(b) An abstract which was classified by the Phase 1 model as a publication of area 2 with probability 0.30.

Figure 5.1. Two possible pipeline flow scenarios. In this examples the "high probability" value is set to be 0.5.

6. EXPERIMENTS

In this section we will explain in detail the experiments that we carried out. We are also going to show where we got the data from and the preprocessing we had to do in order to feed our ML models. Along with this we will show the technology we use to train our models and the performance metrics we chose to evaluate the results of the models.

6.1. Data description

In this section, we will provide an overview of the data source and the pre-processing steps required to generate the input data for the machine learning models.

6.1.1. Data collection

The first step in building our dataset was to download the XML file containing raw data from the computer science bibliography website DBLP (DBLP, 2019). This file contains the basic metadata of the publications in DBLP, such as the name, authors, electronic edition links, among other information. In particular, we worked with the electronic edition link of each publication, which we will call e.e. link. Readers who are interested in finding out more about what information is available from DBLP can check out DBLP - Some Lessons Learned (Ley, 2009). The e.e. link refers each publication to the journal or conference page in which it was published. This page contains more specific metadata about the article, such as abstracts and bibliographic references. Each journals has different information structured in different ways. The majority of the records contain fields for the authors, abstract, and references of publications found in DBLP, in addition to the publication date and DOI. However, as we stated before, ACM has a mandatory field called the Index Terms of the ACM CCS (which is presented in Section 3.3). As we explained in Section 3.3, these are DAGs that can be seen as a label for each publication. We can build labeled datasets by comparing these graphs. It is this feature of ACM publications and the fact that ACM covers hundreds of thousand of papers that motivated us to focus only the publications provided by this association. In this way we proceeded to fetch

the information provided by ACM (open source information) on all the e.e. links in the XML document available in DBLP. First, we scraped web data from e.e. links. Next, we processed the scraped data and built a dataset containing four columns: title, e.e. links, abstracts, and the article's Index Terms. We call this dataset the ACM dataset.

6.1.2. Data preprocessing

The ACM dataset has 336,439 different publications. As previously mentioned, each entry in the table includes the title, the e.e. link, the abstract, and the Index Terms DAG. This DAG is given as a set of paths from an artificial root to each leaf over a total of 1,903 possible nodes. For example, the publication in Figure 3.2 would be given as the following set of lists:

```
{[Artificial root, Theory of computation,
  Computational complexity and cryptography,
  Complexity classes],
 [Artificial root, Theory of computation,
  Design and analysis of algorithms]}
```

We noticed some convenient properties that hold for most articles when we did an exploratory analysis of the ACM dataset's Index Terms. First, we observed that there are 116,737 different DAGs (a DAG can be used by more than one publication). This allows us to work with the different graphs instead of all of them, which reduces considerably the amount of time consumed by the algorithms applied to the dataset. Second, we noticed that exactly 326,049 of the DAGs were actually trees. This corresponds to more than 96.9% of all the DAGs in the dataset. Working with trees is easier than working with DAGs, which is convenient. Third, we found that most of the tree nodes had unique ancestors. We say that a node has unique ancestors if, across all trees that mention the node, the node has the same father, grandfather, and so on up to the artificial root. Out of 1,903 nodes, 1,773 have unique ancestors, which consists of more than 93.1% of all Index Terms trees. For example, Figure 3.3 shows that the node "Parallel programming languages" has at least

two whole different lines of ancestors, so in the complete $\mathbf{DAG}_{\text{ACM}}$ this node must have at least two different lines of ancestors. Having said the above, we kept the 1,773 nodes that had unique ancestors and the trees that only mentioned these nodes. We ended up with a dataset with 278,061 publications, which corresponds to the 82.64% of the total number of collected documents, containing 84,244 different Index Terms trees. Even though we reduced the size of our dataset, the deletion process resulted in labels with more structure. Each Index Terms tree of the new dataframe is a subtree of big fixed tree $\mathbf{T}_{\text{ACM}}$ (obtained by removing nodes that have no unique ancestors from $\mathbf{DAG}_{\text{ACM}}$). In addition to this pre-processing, we also performed other small transformations to the dataset to simplify some parts of the experiments, such as creating a column defining the label trees only by the edges (since our new tree structure allows it), lowercasing all the abstracts, maping the name of the nodes (research areas) to a unique ID, among other small adjustments. To keep the notation simple, we will now refer to the tree Index Terms as Index Terms or labels of the publications.

6.2. Experiments datasets construction

As we have stated before, in order to achieve our goal we we want to solve two problems: to classify articles into some kind of grouping, and to determine, given an article, which articles may be of interest to the reader. After we solve these tasks, we will join the resulting models into one pipeline. For this reason we need to create two new datasets from the one built in Section 6.1.2 one for each phase. In this section we describe them. In each phase the data is divided into train and test (80% and 20% of the data respectively). The train set is used throughout the experiments for the model to learn (it is further divided into smaller sets to validate the training process), while the test set remains unseen until the final evaluation.

6.2.1. Publication classification

Given the large number of Index Terms we have, we decided to use (i) the first and (ii) the second depth levels of $\mathbf{T}_{\text{ACM}}$ for this task: we cannot expect good performance on a

model that trains by classifying in more than 80,000 different labels. We have thirteen possible study areas for (i), which are shown in Table 6.1. We also decided to use the publications whose Index Terms only contained a single class, as each Index Terms tree could have more than one tag in the first level of depth. For example, the Index Terms tree of Figure 3.2 only has one class at depth one: "Theory of computation" so we would consider this publication, but the Index Terms tree of Figure 3.3 has two research areas at depth one: "Computing methodologies" and "Software and its engineering", so we do not include this publication in our dataset. This resulted in a dataset containing 151,316 entries, with the same columns as before. Looking at the $\mathbf{T}_{\text{ACM}}$, one can see that, although the first labels are accurate classifications, they are very broad and general. This is why we decided to also try and solve the classification task by going down one level of depth, which leads us to (ii). In the second level of $\mathbf{T}_{\text{ACM}}$ there are 67 different areas. Because we cannot have so many classes, we built clusters of these areas, according to our knowledge of computer science, forming eight classification classes for our data. These groups can be seen in Table 6.2. As we did earlier, we decided to work with one-label classification, so we kept only those entries that belonged to a single class. This resulted in a dataset of 126,600 entries.

Table 6.1. List of research areas in the first level of depth of $\mathbf{T}_{\text{ACM}}$.

Research areas		
General and reference	Hardware	Computing methodologies
Computer systems organization	Networks	Applied computing
Software and its engineering	Theory of computation	Social and professional topics
Mathematics of computing	Information systems	
Security and privacy	Human-center computing	

6.2.2. Publications of shared interest

For this task we refer to the problem of sentence-pair classification, a sub-area of sentence-pair modeling (Yin, Schütze, Xiang, & Zhou, 2016; W. Lan & Xu, 2018), among others. In simple terms, this consists of, given two sentences, deciding if they belong to the same

Table 6.2. List of research areas in the second level of depth of $\mathbf{T}_{\text{ACM}}$ with our proposed clusters.

Research areas and their clusters			
Printed circuit boards	1	Intrusion/anomaly det. and malware mitig.	4
Comm. hardware, interfaces and stor.	1	Models of computation	5
Integrated circuits	1	Formal languages and automata theory	5
Very large scale integration design	1	Comput. complexity and cryptography	5
Embedded and cyber-physical systems	1	Logic	5
Power and energy	1	Design and analysis of algorithms	5
Electronic design automation	1	Randomn., geom. and discrete structures	5
Hardware validation	1	Theory and algor. for application domains	5
Hardware test	1	Semantics and reasoning	5
Robustness	1	Discrete mathematics	5
Security in hardware	1	Probability and statistics	5
Architectures	2	Information theory	5
Real-time systems	2	Cryptography	5
Parallel computing methodologies	2	Formal methods and theory of security	5
Distributed computing methodologies	2	Data management systems	6
Concurrent computing methodologies	2	Information storage systems	6
Network architectures	3	Information systems applications	6
Network protocols	3	Information retrieval	6
Network components	3	Database and storage security	6
Network algorithms	3	Document management and text processing	6
Network performance evaluation	3	Artificial intelligence	7
Network properties	3	Machine learning	7
Ubiquitous and mobile computing	3	Modeling and simulation	7
Network services	3	Electronic commerce	8
Network types	3	Enterprise computing	8
World wide web	3	Physical sciences and engineering	8
Systems security	3	Life and medical sciences	8
Network security	3	Law, social and behavioral sciences	8
Accessibility	3	Computer forensics	8
Software organization and properties	4	Arts and humanities	8
Software notations and tools	4	Computers in other domains	8
Software creation and management	4	Operations research	8
Security services	4	Education	8
Software and application security	4		

class. A subtask of this problem is given two sentences, classify whether they are equivalent or not. Here, the word "equivalent" can mean different things depending on what

one considers to be similar. In our task, we will talk about "Index Terms equivalent" (IT-equivalent), which will mean that the Index Terms are identical. In this work we consider two articles to be of common interest for the reader when their Index Terms are similar (IT-similar). As we explained in Section 4 we will use the Weighted Jaccard Index introduced in Definition 2 to measure similarities between publications.

To use this similarity measure we had to choose a monotonically increasing function f. Naturally, the first step of this process was to use the identity function. Figure 6.1a shows a graphic view of the distribution using $f(x) = x$ and the nine clusters of the first level labels (see Table 7.1 for the research areas). We realized that we needed a function f that would try to distribute the values as evenly as possible (or at least try to depolarize the values) through the interval $[0, 1]$, and not concentrate only in the extreme points, since we planned to sample positive and negative samples to feed the BERT model. We tried common increasing functions such as power function and the root function. This test are shown in Figure 6.1. We decided to use the root square function, since it distributed the data in a more even way than the other functions we tested. Our Weighted Jaccard Index was defined as

$$J_f \ := \ J_{sqrt}$$

Note that if we take a function f that satisfies the properties stated in Definition 2, used it in J_f, and then apply it to every possible pair of trees, the order of the similarity scores may change from the choice of f. In fact, even if we decided to add more constraints to the function f, for example forcing it to be strictly monotone or to be continuous, it is not clear that the order of the similarity values is maintained. To clarify this statement, let us observe Figure 6.2. Using different valid functions f_1 and f_2 in J_f we could be changing the order of the similarity values of the pairs (x_1, y_1) and (x_2, y_2).

For training the models we cosidered that two publications would be IT-equivalent if their similarity score was greater or equal than 0.9.

For each of the class-specific classifiers, a dataset containing 8,000 IT-equivalent pairs and 8,000 non-IT-equivalent pairs was constructed. For the general area classifier, a dataset was built by joining all the class-specific datasets. As we mentioned before, these datasets were divided into train (which was further divided to get validation data) and test, so that the test set was not seen until the model and its parameters were chosen and fixed.

6.3. Adopted technology

As we mentioned before, we chose to use the BERT language model to solve the proposed tasks (explained in Section 3.4). In most of the experiments we use the base-uncased version of BERT, and the large-uncased version few times. The reason behind mainly using the base-uncased model is because it has performed well in other applications and on similar tasks (Geetha & Renuka, 2021; Yao, Mao, & Luo, 2019). Although other more complex models that have proven to be state of the art in different language comprehension tasks, such as BERT's large-uncased version (Devlin et al., 2018) or its derivatives like AlBert (Z. Lan et al., 2019), RoBertA (Liu et al., 2019), among others, we decided to start with the base version for a few practical reasons. First, the model has 110 million fine-tuneable parameters, which is almost a third of the parameters of the large version. This allowed us to train the model with tools that do not require as much disk memory. The second reason is that, being the base model, it makes sense to try this model before trying more complex or new ones, and then scale it depending on the quality of the results.

6.4. Performance metrics

When planning the experiments, the following question naturally arises: How do we get the most accurate model? This brings us to the next question: What problem are we attempting to solve?

Before the two experiments are carried out, performance metrics must be determined in order to compare the different resulting models and try to understand the scope of this work. The most common performance metrics in machine learning are accuracy, precision

and recall. In binary classification these metrics can be defined counting the true positives (TP), true negatives are (TN), false positives (FP) and false negatives (FN) of the outcome of the model. These are defined as

$$\begin{aligned} \text{TP} &:= \text{instances correctly predicted to be in the positive class.} \\ \text{TN} &:= \text{instances correctly predicted to be in the negative class.} \\ \text{FP} &:= \text{instances incorrectly predicted to be in the positive class.} \\ \text{FN} &:= \text{instances incorrectly predicted to be in the negative class.} \end{aligned}$$

Accuracy is the number of correctly predicted outcomes divided by the total number of predicted outcomes, precision is the ratio of correctly-predicted positive instances to all instances predicted as positive, and recall is the fraction of positive instances that were retrieved. Formally,

$$\begin{aligned} accuracy &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \\ precision &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ recall &= \frac{\text{TP}}{\text{TP} + \text{FN}} \end{aligned}$$

Although our first problem is not binary, we are going to analyze which of these metrics make sense for our experiments. We will then extrapolate to the non-binary case. For the first experiment we are trying to correctly classify the publications in their respective areas, so accuracy is an important metric. However, since the classes are unbalanced we care about the accuracy on a balanced sample as well. On the other hand, taking precision and recall individually as a performance metric does not provide an accurate model evaluation, since all classes are relevant. This is why we would like to take both metrics into account. We can do this with the F_1 score, which is defined as the harmonic

mean between precision and recall:

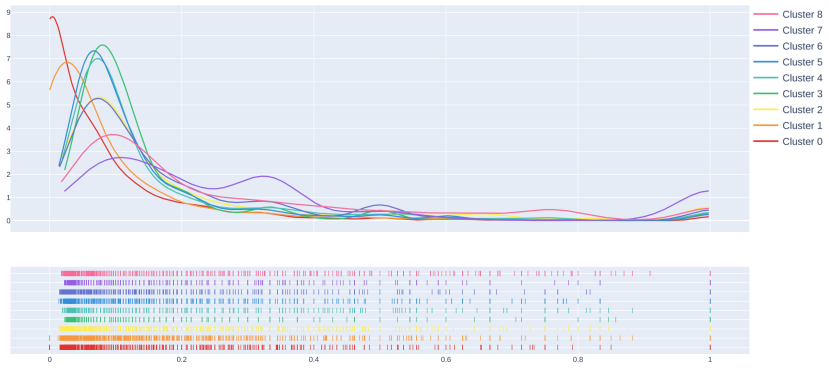
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In multiclass classification precision, recall and F_1 score are not well defined, since there is no "positive" class. However, the definitions of these metrics can be extended to multiclass classification by taking a weighted average over all classes, where the weight of each value (or metric) is determined by the percentage of the class in the test data.

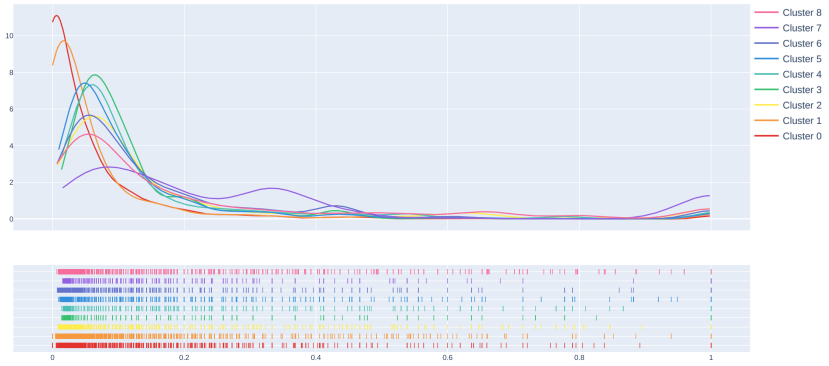
The second experiment is a binary classification problem, since we need to decide, given a pair of publications, whether or not they are IT-equivalent. We want the model to have a high degree of certainty when it predicts that two publications are similar. The cost of the model saying that two articles are not similar but in fact they were is much lower than the cost of being wrong in saying that they are similar (since it is a type of recommender system). This is why we have our main focus on the precision score, since we want false positives to be as few as possible. However, we do not want our model to have recall score 0, because we want it to retrieve something. That is why we are also interested in observing the recall score (mainly getting a non-zero recall score).

Finally, let us recall that the model outputs the probability that the instance belongs to the positive class. This is why we must set a decision threshold that will make all instances with a probability greater than this threshold to be considered positive. Since we want a high precision score, we must set a high threshold. For the class-specific models we define this threshold as 0.7, while for the general model we set it to 0.95. This difference is due to the fact that the general model is tested with much more data, so the precision must be higher. Although we could have defined the threshold for the class-specific models to be a higher value, we could not do this because there is a class that its recall score decreases sharply when we increase this value (see Section 7.2). This threshold difference has to be taken into account when analyzing the values of the performance metrics' tables.

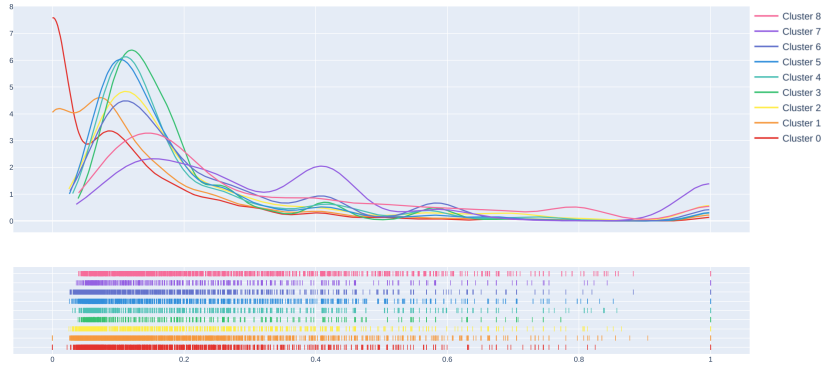
The performance values of class-specific models are not comparable with those of general models if we use different thresholds.



(a) Distribution plot using $f(x) = x$ in J_f.



(b) Distribution plot using $f(x) = 2^x$ in J_f.



(c) Distribution plot using $f(x) = \sqrt{x}$ in J_f.

Figure 6.1. Distribution of the Weighted Jaccard Index for a sample of 10,000 publications pairs using different functions $f(x)$ in J_f.

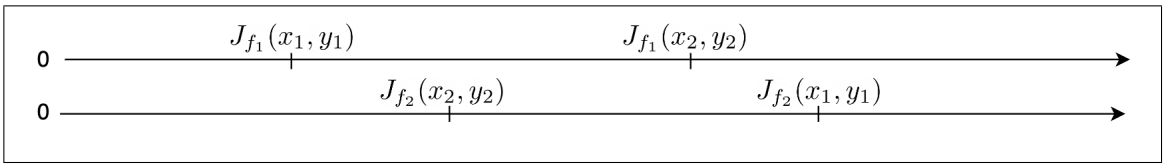


Figure 6.2. Possible similarity values and their relative position $\mathbb{R}$ axis for pairs (x_1, y_1) and (x_2, y_2) using valid functions f_1, f_2 in J_f.

7. RESULTS AND DISCUSSION

As explained in the previous section, we divided our dataset into three sets: training, validation, and final testing. The construction phase of the final model was carried out iteratively, trying to improve the models as the performance metrics were reported on the validation set. The final test set was not used until the final model was fully constructed: the idea is to never have seen this data until the end to really see how well the model performes. The timeline of how the models were built in the experiments is described below. In each task we trained different models with little modifications until we got the best result. We will refer to these little modifications as sub-experiments of the task. The main features used in the iterations were the confusion matrix and the accuracy of each sub-experiment.

7.1. Phase 1 - Publication classification

Recall that $\mathbf{T}_{\text{ACM}}$ is a tree whose nodes represent different computer science research areas. Let us also recall that a node in a deeper level of $\mathbf{T}_{\text{ACM}}$ gives a more in-depth understanding of the research field it represents (see Tables 6.1 and 6.2 for more detail). We approached this classification task using two levels of depth of the $\mathbf{T}_{\text{ACM}}$. Each model was validated over two datasets: balanced and unbalanced validation sets. The final model was chosen by comparing the performance metrics obtained using these datasets. It was then retrain using both the train and validation data, and tested on a unseen test set. Figures 7.1 and 7.2 show the confusion matrices of each sub-experiment over the balanced and unbalanced validation set, and Table 7.2 shows in detail the results of each model on the validation data. The final models' performance can be found in Table 7.4. In the following sub-sections we give details on how the sub-experiments were carried out, the analisis that led us to the different models and the performance metrics that we obtained.

7.1.1. First level classification

The first sub-experiment consisted on fine-tuning BERT's base model using the thirteen labels presented in Table 6.1. Looking at the data and the confusion matrix presented in Figure 7.1a we can see that the model almost never classifies a sample with label 2, which corresponds to the "General and Reference" tag. Also, if we look at Figure 7.2a we can confirm that this class introduces a lot of noise in the data (many false negatives distributed in the other twelve labels). We think that the reason why the model was not able to discriminate this class is because it covered many topics that could be related to more specific areas and it was a small class. This is why the second sub-experiment consisted on not changing any of the variables of the previous sub-experiment except for the elimination of the articles that corresponded to the "General and Reference" label, that is, classifying twelve classes instead of thirteen. The results of this sub-experiment confirmed the noise hypothesis generated by this label: although the accuracy and the F1-score increased only approximately 1% on the unbalanced validation set, they increased approximately 4% one the balanced validation set. However, looking at Figures 7.1b and 7.2b, we could see that there were classes that kept confusing the model. After examining the data, we came to the realization that there were classes with little data that include similar areas. This is why we proceeded to manually group some classes that include similar areas, and the same procedure was carried out but with nine new manually grouped labels. These new tags can be seen in Table 7.1. This led to a more balanced training set and thus a more accurate model. Figures 7.1c and 7.2c show the confussion matrix over the unbalanced validation sets. We can see that although the model still made mistakes, it had improved with respect to the last one. Specifically, on the balanced validation set, the model increased over 8% in the F1-score and almost 9% on the accuracy (not to say that it also performed better on the unbalanced validation set). This shows that the model could discriminate much better between classes and not just "guessing" that the sample was one of the bigger classes. Continuing with the small modifications that we believed could improve the model, we proceeded to add the title of the document at the beginning of the

abstract: `Title.Abstract`. This was done because we thought that it could be helpful for the model (usually when a person reads the abstract of a publications the title gives a lot of contextual information). This sub-experiment obtained the best overall performance of all the previous ones. With exception of the balanced F1-score, all the other metrics were approximately 0.5% better. We can see the change between the confussion matrices from Figure 7.2c to Figure 7.2d: the false negatives decreased. As an example, in Figure 7.2c there are a lot of publications of the class "Network" that were classified as instances of the class "Information systems", but in Figure 7.2d this phenomenon happens less. We associate this behaviour to an increase of understanding of the text by the model, since it was given relevant information. Finally, the last sub-experiment was performed by only changing the fine tuned BERT model: bert-large (Devlin et al., 2018). Although the number of parameters tripled, this model performed worse than the previous model. This entire process can be seen in detail in Table 7.2.

Table 7.1. Proposed clustering of the first level research areas in tree$\mathbf{T}_{\text{ACM}}$.

Clusters			
Theory of computation	1	Applied computing	4
Computing methodologies	1	Security and privacy	5
Mathematics of computing	1	Hardware	6
Software and its engineering	2	Information systems	7
Computer systems organizations	2	Human-centered computing	8
Networks	3	Social and professional topics	9

7.1.2. Task 2 - Second level classification

This sub-experiment was started right after the second sub-experiment mentioned in Section 7.1.1. From then on it was done in parallel with the sub-experiments in that section. The first thing that we did was to study the 67 areas of the second level of the $\mathbf{T}_{\text{ACM}}$ and manually group these areas into eight clusters. As we mentioned before, these clusters and the specific areas that are in the second level of $\mathbf{T}_{\text{ACM}}$ can be seen in Table 6.2. The BERT base model was trained in order to learn how to classify these clusters. In the same way as in the sub-experiment in Section 7.1.1, the title of the publication was added to the

abstract of the paper, in order to give the model more insight. This led to a growth in the performance metrics (on both balanced and unbalanced validation set) by approximately 1%. Finally, the fine-tuned model was changed from bert-base to bert-large and retrained. Again, no major changes were observed in relation to the increase in the number of model parameters. The results and performance of these sub-experiment can be seen in Table 7.3. One last thing to notice about these results is that although the performance metrics of the best classification model using the second level research areas is lower than the best one using the first level research areas, there are instances where one can find more convenient to use the second level for study areas clusters because it is more specific. For example, one can define new clusters based on more complex and personal equivalent rules: knowledge of different areas, team-expertise areas, or just want-to-learn study areas. In these cases and many more, having the flexibility to choose in a more detailed way is a strong component of the experiment to have in consideration. This can be also applied to deeper levels of $\mathbf{T}_{\text{ACM}}$.

Table 7.2. Performance metrics of the experiments using the first level research areas of $\mathbf{T}_{\text{ACM}}$.

N°	BERT	Classes	Title	F1	Accuracy	Balanced F1	Balanced Acc.
1	base	13	No	0.7055	0.6986	0.6191	0.5930
2	base	12	No	0.7180	0.7086	0.6515	0.6377
3	base	9	No	0.7428	0.7370	0.7343	0.7274
4	base	9	Sí	0.7491	0.7448	0.7321	0.7317
5	large	9	Sí	0.7345	0.7355	0.7245	0.7268

Table 7.3. Performance metrics of the experiments using the second level research areas of $\mathbf{T}_{\text{ACM}}$.

N°	BERT	Classes	Title	F1	Accuracy	Balanced F1	Balanced Acc.
6	base	7	No	0.7076	0.7057	0.6788	0.6825
7	base	7	Sí	0.7161	0.7151	0.6957	0.6980
8	large	7	Sí	0.7192	0.7183	0.7134	0.7134

7.1.3. Best models retrain and evaluation

Finally, the model with the best performance in each level were models number 4 and 7, which used the title and the abstract as input instances. The performance of these models on the test data can be seen in Table 7.4, along with the confussion matrices in Figure 7.3 (over the balanced and unbalanced test sets).

Table 7.4. Best models' performance metrics on the test data.

N°	BERT	Classes	Title	F1	Accuracy	Balanced F1	Balanced Acc.
4	base	9	Yes	0.7471	0.7427	0.7229	0.7209
7	base	7	Yes	0.7248	0.7260	0.7174	0.7188

7.2. Phase 2 - Common interest publications

In this section we present the results of Phase 2 of this work. As stated above, the construction of the models in this phase are directly linked to the results of phase 1. In the results of Section 7.1.3 we chose two models: one classify into seven classes ($\mathbf{T}_{\text{ACM}}$ second level clusters) and the other that classifies the input in nine classes ($\mathbf{T}_{\text{ACM}}$ first level clusters). In the following sections we show and explain the results we obtained. As we said in Section 6.4, we would like to evaluate our results mainly using the precision score but keeping in mind that the recall score cannot be zero. This is why for each experiment we have to set a threshold to determine when the model predicts that a pair of publications are IT-equivalent. For the class-specific models we used 0.7 as the classification threshold, while for the general model we used 0.95 as the classification threshold. We could not use a higher value for the class-specific models because there is a class that its recall score decreases sharply as the threshold increases above 0.7 (see class 8 in Table 7.6). This restriction does not affect our analysis mainly because we need to compare the models that try to learn the same tasks (specific-models with specific models and general models with general models). Also, we will not be using these thresholds in the final pipeline, since their purpose are only to evaluate the models and give us insights about them.

7.2.1. First level clusters

Recall that in Section 7.1.3 we chose a model that classifies an abstract into nine study areas of the first level of $\mathbf{T}_{\text{ACM}}$. Then, each sub-experiment in this phase must have nine class-specific models and one model trained with all the study areas together. We created two types of sub-experiments: the first consisted of simply comparing pairs of abstracts from publications. The second experiment consisted of comparing not only the abstract, but also adding the title of the publication before the abstracts. Tables 7.5 and 7.6 show the precision and recall by study area of the class-specific models, while the performance metrics of the general model can be seen in Table 7.7.

Table 7.5. Precision score of the class-specific models with a threshold of 0.7 to be classified as a positive instance for the first level of $\mathbf{T}_{\text{ACM}}$.

N°	**Tit.**	**C.1**	**C.2**	**C.3**	**C.4**	**C.5**	**C.6**	**C.7**	**C.8**	**C.9**
1	No	0.8782	0.8447	0.7612	0.8482	0.8684	0.7642	0.7686	0.8000	0.8121
2	Yes	0.8563	0.8533	0.7684	0.8289	0.8525	0.7910	0.8300	0.7888	0.8084

Table 7.6. Recall score of the class-specific models with a threshold of 0.7 to be classified as a positive instance for the first level of $\mathbf{T}_{\text{ACM}}$.

N°	**Tit.**	**C.1**	**C.2**	**C.3**	**C.4**	**C.5**	**C.6**	**C.7**	**C.8**	**C.9**
1	No	0.8324	0.7592	0.4573	0.7884	0.6731	0.6320	0.5636	0.0054	0.8132
2	Yes	0.8928	0.7482	0.5209	0.8406	0.7108	0.6127	0.4610	0.0973	0.8091

Table 7.7. Precision and recall scores of the general model with a threshold of 0.95 to be classified as a positive instance for the first level of $\mathbf{T}_{\text{ACM}}$.

N°	**Title**	**Precision**	**Recall**
1	No	0.9406	0.1361
2	Sí	0.9287	0.1559

We can see that although precision score is not dominant in any class-specific model, since there are classes that the first sub-experiment slightly dominates, while in others the second sub-experiment slightly dominates, large differences in recall score can be seen. This tells us that our model that includes the title of the publication manages to return more positive instances. On the other hand, if we look at the performance of the general

models, we can see that the precision score of the model that did not use the title is higher. Although the recall is slightly lower in this model, it is not zero.
This is why we chose the sub-experiment that makes use of the title of the publication for the class-specific models and the one that does not use the title for the general model. This models were retrained with the training and validation data and evaluated with the test data. The results of these final models can be seen in Tables 7.8 and 7.10.

Table 7.8. Performance metrics over the unseen test set of the first level class-specific models of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.7.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Prec.	0.8553	0.8688	0.7599	0.8648	0.8385	0.7618	0.7484	0.6858	0.8510
Rec.	0.8557	0.7570	0.5616	0.7838	0.8114	0.7132	0.6753	0.4462	0.8001

Table 7.9. Performance metrics over the unseen test set of the first level general model of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.95.

	Avg.
Prec.	0.8038
Rec.	0.7115

Table 7.10. Performance metrics over the unseen test set of the first level general model of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.95.

	General
Prec.	0.9485
Rec.	0.1795

7.2.2. Second level clusters

For the second level clusters, in phase 1 we chose a model that classified the abstracts into 7 different areas of study. Thus, each sub-experiment in this phase must have seven class-specific models and one model trained with all the research areas. Analogous to Section 7.2.1 we create two types of sub-experiments: one comparing only the abstracts and the other comparing the title plus the abstract. Table 7.11 and Table 7.12 show the precision

and recall by study area of the class-specific models, while the performance metrics of the general model can be seen in Table 7.13.

Table 7.11. Precision score of the class-specific models with a threshold of 0.7 to be classified as a positive instance for the second level of $\mathbf{T}_{\text{ACM}}$.

N°	**Tit.**	**C.1**	**C.2**	**C.3**	**C.4**	**C.5**	**C.6**	**C.7**
1	No	0.8002	0.8257	0.8925	0.8642	0.8211	0.9043	0.8294
2	Yes	0.8106	0.8039	0.9096	0.8639	0.8035	0.8910	0.9140

Table 7.12. Recall of the class-specific models with a threshold of 0.7 to be classified as a positive instance for the second level of $\mathbf{T}_{\text{ACM}}$.

N°	**Tit.**	**C.1**	**C.2**	**C.3**	**C.4**	**C.5**	**C.6**	**C.7**
1	No	0.7386	0.7548	0.7600	0.8939	0.5059	0.7172	0.8328
2	Yes	0.7716	0.8407	0.7961	0.9097	0.7875	0.8234	0.8522

Table 7.13. Precision and recall scores of the general model with a threshold of 0.95 to be classified as a positive instance for the second level of $\mathbf{T}_{\text{ACM}}$.

N°	**Title**	**Precision**	**Recall**
1	No	0.9716	0.0874
2	Yes	0.9416	0.4413

Again, the precision score was not decisive in the experiment, since neither class-specific model significantly outperformed the other. However, it can be seen that the recall scores of the titled class-specific models are much higher than those without titles. On the other hand, in the general models, the model without the titles outperforms the one that includes the title in the precision score. Although the recall score drops sharply in the model that did not use the title, it is still above 0, so it can be used to achieve our task.

This is why, just like in section 7.2.1, we chose the sub-experiment that makes use of the title of the publication for the class-specific models, and the one without titles for the general model. These models were retrained with the training and validation data and

evaluated with the test data. The results of this final model can be seen in Tables 7.14 and 7.16.

Table 7.14. Performance metrics over the unseen test set of the second level class-specific models of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.7.

	1	**2**	**3**	**4**	**5**	**6**	**7**
Prec.	0.8851	0.8333	0.9242	0.8738	0.7986	0.8921	0.9011
Rec.	0.6402	0.8188	0.8227	0.9177	0.7970	0.7502	0.8916

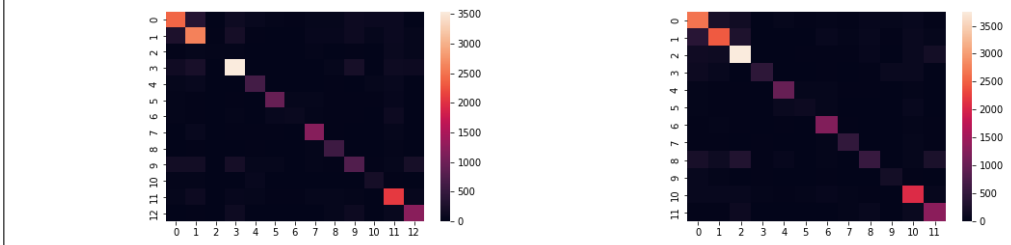
Table 7.15. Performance metrics over the unseen test set of the second level general model of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.95.

	General
Prec.	0.9401
Rec.	0.3022

Table 7.16. Performance metrics over the unseen test set of the second level general model of $\mathbf{T}_{\text{ACM}}$ evaluated using a classification threshold of 0.95.

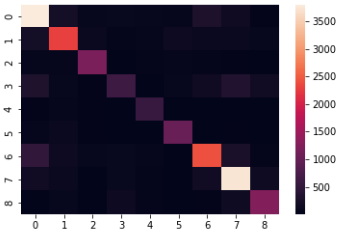
	Avg.
Prec.	0.8726
Rec.	0.8054

We can observe that the results on the test data were consistent with the results of the experiments. In addition, we can observe that the model that uses the study areas of the second level of the $\mathbf{T}_{\text{ACM}}$ achieves better performance metrics, which makes sense since the clusters at this level were built with much more detail than those at level 1. Again, it is worth to mention that the only reason we use thresholds is because we need to calculate performance metrics. In practice, given a scientific article, the predicted similarity value is used to obtain the most IT-similars.

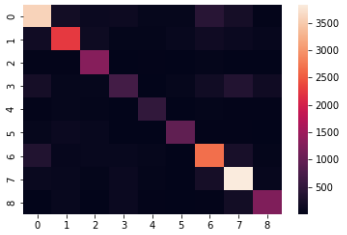


(a) Model N°. 1

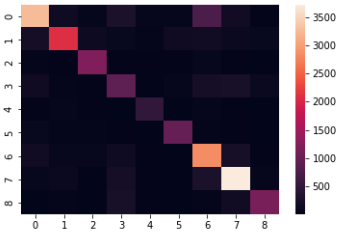
(b) Model N°. 2



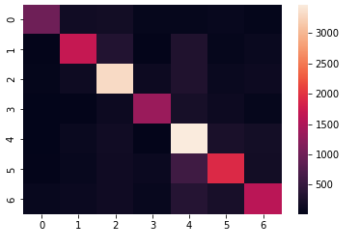
(c) Model N°. 3



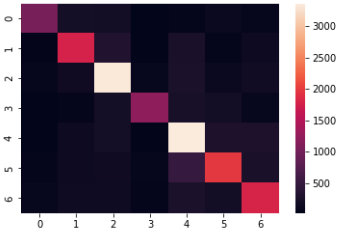
(d) Model N°. 4



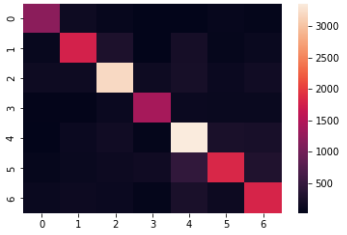
(e) Model N°. 5



(f) Model N°. 6

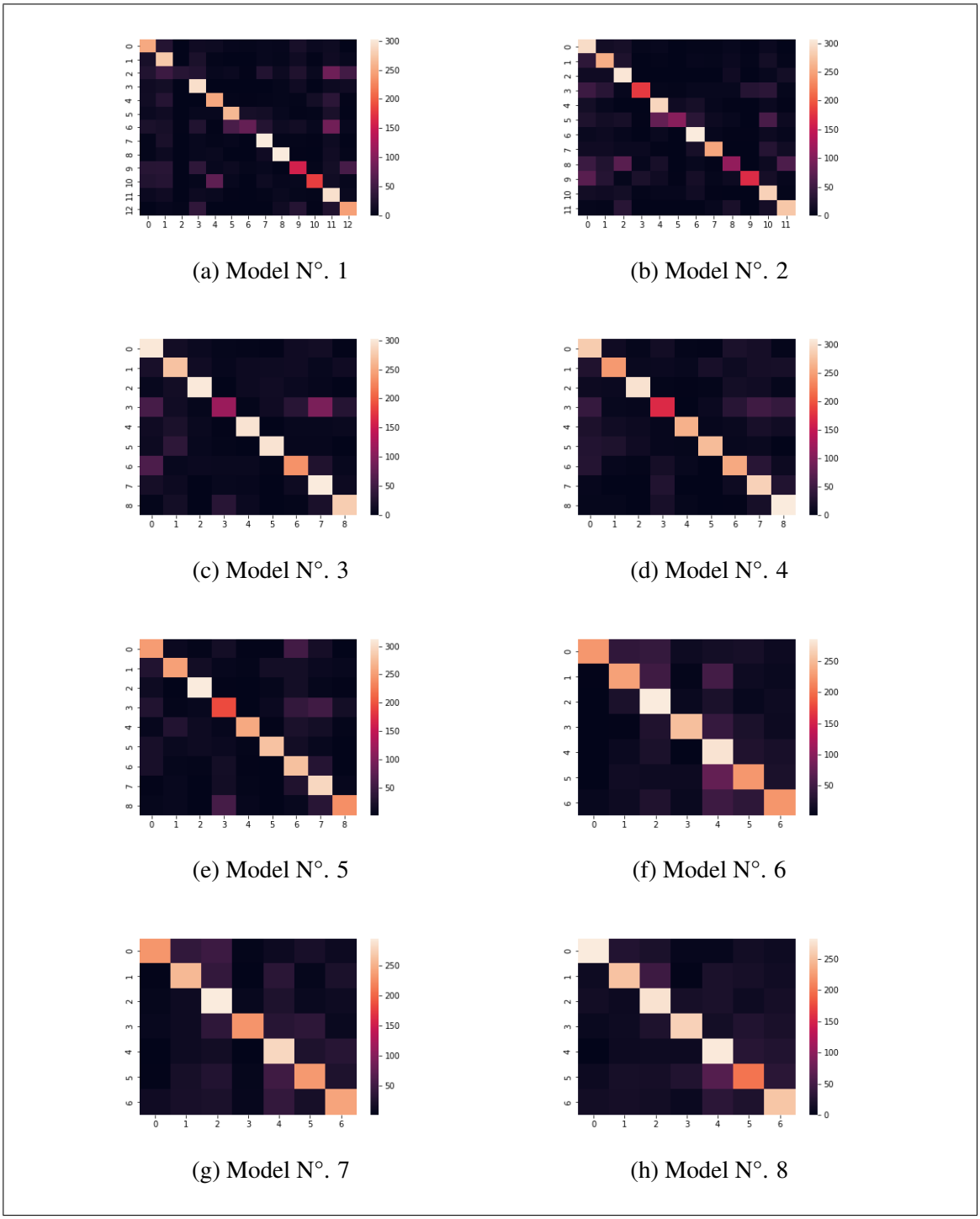


(g) Model N°. 7



(h) Model N°. 8

Figure 7.1. Confussion matrices over the balanced validation data of the publications classification problem.



(a) Model N°. 1

(b) Model N°. 2

(c) Model N°. 3

(d) Model N°. 4

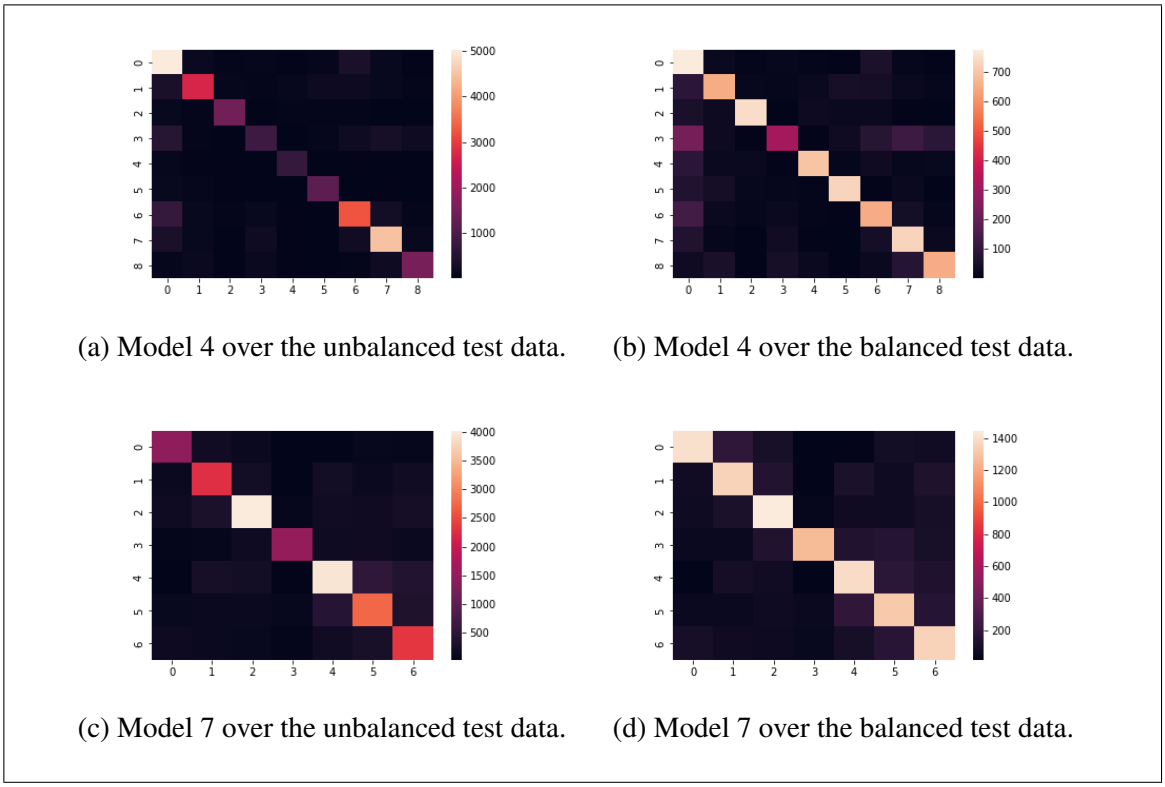
(e) Model N°. 5

(f) Model N°. 6

(g) Model N°. 7

(h) Model N°. 8

Figure 7.2. Confussion matrices over the unbalanced validation data of the publications classification problem.



(a) Model 4 over the unbalanced test data.

(b) Model 4 over the balanced test data.

(c) Model 7 over the unbalanced test data.

(d) Model 7 over the balanced test data.

Figure 7.3. Confussion matrices of the final models over balanced and unbalanced test data of the publications classification problem.

8. CONCLUSIONS

8.1. Summary of the results

In this work we have worked extensively with similarity measures. We have introduced a normalized similarity measure called the Weighted Jaccard Index for trees. This similarity measure takes two trees and assigns greater similarity weight to deeper nodes. This similarity measure ensures that trees of different sizes can be treated equally: add similarity when the intersection is large and subtract similarity when the symmetric difference is large.

We have built two classification models that are the result of fine tuning the BERT-base language model. The first consists of, given a scientific publication, classifying the article in the corresponding research area using only the title and abstract of the publication. We did this by using the tags at one and two levels of depth in $\mathbf{T}_{\text{ACM}}$ and using the Index Terms provided by ACM. In this way, we created labeled datasets where the labels had a tree structure. We obtained a F_1 score of 74.71% for the trained model with the first level labels of the tree $\mathbf{T}_{\text{ACM}}$ and 72.48% for the trained model with the second level labels on the unbalanced test data. On the balanced test data, 72.29% and 71.74% F_1 scores were obtained, respectively. The second model we built solves the task of determining, given a pair of publications, a similarity score (or IT-similarity as we call it in this work) using only the title and the abstract. We did this with a dataset containing pairs of publications and the similarity value using the Jaccard Weighted Index with the square root function within the similarity function, applied to the Index Terms of each post. Most of the class-specific models' precision scores were over 75% using the first level of the $\mathbf{T}_{\text{ACM}}$ tree, while using the second level of the tree we got over 80% in almost all classes. The performance in the general models was very similar, exceeding 94% at both depth levels of the $\mathbf{T}_{\text{ACM}}$ tree. With these two models in hand, the design of experiments to develop the pipeline presented in Section 5 can be carried out and properly evaluated, using recommender system techniques and appropriate performance metrics.

We also present a flexible solution to other problems. In our case we made clusters using the criterion that two publications are related if they are from areas of common interest. Clusters can also be made using other criteria: related areas of expertise, study areas that are presented in an academic curriculum, among many more that are small modifications to the original problem.

8.2. Future work

We believe that the possible lines of future work fall into two types: the creation and implementation of the recommender system, and possible extensions or improvements of the two components we have built and presented in this work. We will elaborate on both of them.

For the full implementation of the recommender system, which is discussed in detail in Section 5, we have to carry out two tasks. First, we need to define the "high probability" value that we will use to decide when to use a class-specific model and when to use a general one. Recall that in the recommender system we propose, given an abstract, the first thing we do is try to predict to which study area it belongs. Next, if the model is able to classify the input publication into a research area with a probability higher than our "high probability" value, we use the class-specific models we implemented in Phase 2. If it does not exceed the "high probability" value, then we must use the general model. The definition of this "high probability" value must be carried out experimentally since it depends on the ranking performance metrics we decide to use to evaluate the system. Once the pipeline is assembled and tested, we want to develop a web interface so that the model remains open to the community. A direct consequence of this step is the possibility to test and evaluate the recommender system in real scenarios using user feedback, and to iterate on the model in order to improve it.

As for the lines of work that consist of extending the capabilities of the recommender system, either by integrating more techniques for Phase 1 and Phase 2 models or by interfering the training data, we propose the following:

- Experiment both Phases 1 and 2 with other state-of-the-art language models, specially those implemented after BERT, such as RoBERTa (Liu et al., 2019), ALBERT (Z. Lan et al., 2019) and Big Bird (Zaheer et al., 2020), among others.
- Use deeper levels of $\mathbf{T}_{\text{ACM}}$ in Phase 1 to make more specific research field clusters.
- Build and perform experiments using other research disciplines.
- Add other variables to the input to complement the input we already use. For example, we could try and use the abstracts, titles and the bibliographic references to train the model.
- Extend the problem definition such that the input of the problem is not a single publication, but a set of publications. Thus, solve the following task: given a set of scientific articles, obtain a list of related publications.

REFERENCES

Aksnes, D. W. (2003). A macro study of self-citation. *Scientometrics*, *56*(2), 235–246.

Charlin, L., & Zemel, R. (2013). The toronto paper matching system: an automated paper-reviewer assignment system.

Chen, S., Ma, B., & Zhang, K. (2009). On the similarity metric and the distance metric. *Theoretical Computer Science*, *410*(24-25), 2365–2376.

Chen, Z. (2011). Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th international symposium on high performance distributed computing* (pp. 73–84).

DBLP. (2019). *The dblp computer science bibliography.* Retrieved 2022-09-09, from `https://dblp.org/xml/release/`

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Engqvist, L., & Frommen, J. G. (2008). The h-index and self-citations. *Trends in ecology & evolution*, *23*(5), 250–252.

Filmus, Y. (2013). Inequalities on submodular functions via term rewriting. *Information Processing Letters*, *113*(13), 457–464.

for Computing Machinery, A. (2012). *Acm computing classification system.* Retrieved 2022-08-18, from `https://dl.acm.org/ccs`

Geetha, M., & Renuka, D. K. (2021). Improving the performance of aspect based sentiment analysis using fine-tuned bert base uncased model. *International Journal of Intelligent Networks*, *2*, 64–69.

Hyland, K. (2003). Self-citation and self-reference: Credibility and promotion in academic publication. *Journal of the American Society for Information Science and technology*, *54*(3), 251–259.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Kessler, M. M. (1963). Bibliographic coupling between scientific papers. *American documentation*, *14*(1), 10–25.

Knoth, P., Anastasiou, L., Charalampous, A., Cancellieri, M., Pearce, S., Pontika, N., & Bayer, V. (2017). Towards effective research recommender systems for repositories. *arXiv preprint arXiv:1705.00578*.

Knoth, P., Novotny, J., & Zdrahal, Z. (2010). Automatic generation of inter-passage links based on semantic similarity. In *Proceedings of the 23rd international conference on computational linguistics (coling 2010)* (pp. 590–598).

Kosub, S. (2019). A note on the triangle inequality for the jaccard distance. *Pattern Recognition Letters*, *120*, 36–38.

Lan, W., & Xu, W. (2018). Character-based neural networks for sentence pair modeling. *arXiv preprint arXiv:1805.08297*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Ley, M. (2009). Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, *2*(2), 1493–1500.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Loui, M. C. (1996). Computational complexity theory. *ACM Computing Surveys (CSUR)*, *28*(1), 47–49.

Rozinek, O., & Mareš, J. (2021). The duality of similarity and metric spaces. *Applied Sciences*, *11*(4), 1910.

Shahmirzadi, O., Lugowski, A., & Younge, K. (2019). Text similarity in vector space models: a comparative study. In *2019 18th ieee international conference on machine learning and applications (icmla)* (pp. 659–666).

Small, H. (1973). Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, *24*(4), 265–269.

Tarnavsky, E., Harpaz, I. K., & Perets, S. (2021). *Connected papers.*

Teodorescu, D., & Andrei, T. (2014). An examination of "citation circles" for social sciences journals in eastern european countries. *Scientometrics*, *99*(2), 209–231.

Wang, J., & Dong, Y. (2020). Measurement of text similarity: a survey. *Information*, *11*(9), 421.

Yao, L., Mao, C., & Luo, Y. (2019). Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*.

Yin, W., Schütze, H., Xiang, B., & Zhou, B. (2016). Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, *4*, 259–272.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., ... others (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, *33*, 17283–17297.