information

MDPI

Article

Hybrid Stacking Approach for Biomedical Full-Text Classification: Combining ILP with Refinement Operators and Propositional Learners

Carlos Adriano Gonçalves [1,2,†], Rui Carlos Camacho [3,4,†], Eva Lorenzo Iglesias [5,†], Lourdes Borrajo Diz [5,†], Adrian Seara [5,†] and Célia Talma Gonçalves [6,7,*,†]

1 Informatics Engineering, ISEP, Polytechnic University of Porto, Rua Dr. António Bernardino de Almeida, 4249-015 Porto, Portugal; cag@isep.ipp.pt
2 SoftCPS–Software Technologies for Cyber–Physical Systems, Rua Dr. António Bernardino de Almeida, 4249-015 Porto, Portugal
3 Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal; rcamacho@fe.up.pt
4 LIAAD–Laboratory of Artificial Intelligence and Decision Support, INESC TEC-Institute for Systems and Computer Engineering, Technology and Science, Universidade do Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
5 Instituto de Física y Ciencias Aeroespaciales (IFCAE), Escuela Superior de Ingeniería Informática, Universidade de Vigo, 32004 Ourense, Spain; eva@uvigo.gal (E.L.I.); lborrajo@uvigo.gal (L.B.D.); adrseara@uvigo.gal (A.S.)
6 CEOS.PP–Centro de Estudos Organizacionais e Sociais do Politécnico do Porto, ISCAP–Instituto Superior de Contabilidade e Administração do Porto, Polytechnic University of Porto, Rua Jaime Lopes Amorim, s/n, 4465-004 S. Mamede de Infesta, Portugal
7 LIACC-Laboratório de Inteligência Artificial e Ciência de Computadores, Universidade do Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
* Correspondence: celia@iscap.ipp.pt
† These authors contributed equally to this work.

Abstract

This study addresses the problem of automatic classification of MEDLINE full-text biomedical documents and investigates whether relational learning can enhance performance within a stacking-based text classification framework. A major problem in using Inductive Logic Programming systems is their limited scalability in the presence of large search spaces and with many examples. The research addresses (i) the incorporation of an Inductive Logic Programming component into a traditional machine learning pipeline, namely WEKA platform, and (ii) a new methodology to enable a significant reduction in the redundancy of the hypothesis space in Inductive Logic Programming systems (ILP). This reduction enabled the use of Inductive Logic Programming in the very large and complex full-text classification problems. To accomplish the first objective, the Aleph system—an Inductive Logic Programming system—was integrated into the WEKA platform, and for that, we used a stacking architecture in which propositional learners and Inductive Logic Programming were applied to different document sections, with their outputs combined by a meta-learner. Experiments were conducted on an extended OHSUMED corpus comprising MEDLINE full-text documents mapped to MeSH disease categories. The achieved results indicate the feasibility of using propositional learners with relational learners, taking advantage of an integrated platform. As far as the available literature indicates, a hybrid solution that combines Inductive Logic Programming and propositional learners within a multi-view stacking architecture for MEDLINE full-text classification constitutes a novel approach. The results obtained by the proposed reduction in the redundancy of the hypothesis space were very promising, leading to substantial improvements for several disease classes, with some configurations achieving almost perfect agreement regarding the kappa statistics metric.

Academic Editors: Ebuka Ibeke and Chinedu Pascal Ezenkwu

Received: 14 August 2026
Revised: 25 August 2026
Accepted: 7 September 2026
Published: 9 September 2026

Keywords: text mining; machine learning; inductive logic programming; refinement operators; multirelational data mining

1. Introduction

Bioinformatics is a scientific area that takes advantage of computational developments to analyze very large amounts of data and to address biological questions [1,2]. Repositories of medical literature like the National Center for Biotechnology Information (NCBI) and PubMed Central (PMC) are currently available. Such repositories provide free access to full-text documents. The main goal of PubMed Central is to provide an electronic repository of peer-reviewed literature relevant to the Life Sciences [3]. The PMC repository is constantly increasing the number of documents publicly available. PMC not only provides the title and abstract of a document, but it also allows access to complete full-text documents. With this new facility, researchers can expand the ability to apply more sophisticated text analytic tools. The chances for better analyses arise, but the use of larger amounts of data requires higher processing capabilities.

Text mining is a research field that aims to extract high-quality information from text [4]. Text mining can be applied to several forms of text that include structured, semi-structured and unstructured texts. Automatic text classification is one of the main tasks that can be addressed by using text mining. Text classification tasks are commonly addressed with propositional learners; in contrast, the present study adopts multi-relational algorithms and employs an Inductive Logic Programming implementation called Aleph (A Learning Engine for Proposing Hypotheses) [5]. Inductive Logic Programming uses First-Order Logic (FOL) to encode examples and user-provided background knowledge [6,7]. Using Inductive Logic Programming provides advantages over propositional learners, not only by enabling the straightforward inclusion of background knowledge in the induction process but also by allowing the use of a powerful representation language to encode the induced models. The current study focuses on the application of Inductive Logic Programming for full-text biomedical document classification. Several studies compare results between methodologies that only use titles and abstracts versus full-text documents [8–10], highlighting the advantages of using a full-text approach.

To make the comparison of the two paradigms of algorithms, the WEKA platform was used. An interface was developed to incorporate the Aleph system into the WEKA platform. An advantage of using a unified platform is the possibility of using blends of algorithms to address the problem at hand. In this study, a stacking algorithm was used with the application of several algorithms (ILP included) to different sections of the dataset corpus. In this paper, the experiments were conducted using a well-known OHSUMED dataset [11] that was improved with the full-text of the documents and semantically enriched [10].

The study pursues two main objectives: (i) to integrate the Aleph Inductive Logical Programming system into the WEKA data mining platform and evaluate its use as a relational base learner in combination with propositional learners and (ii) to develop and evaluate a refinement operators methodology for reducing redundancy in the Inductive Logical Programming hypothesis space and, consequently, improving the computational efficiency of model construction without degrading classification performance. The effectiveness of the proposed approach is assessed by comparing the same stacking architecture with and without refinement operators using accuracy, F-measure, kappa statistics and CPU execution time.

The rest of the paper is organized as follows. Section 2 is divided into subsections. Section 2.1 presents the concepts related to text mining. Section 2.2 shows the Inductive

Logic Programming algorithm used in this study: the Aleph algorithm. Section 2.3 describes the new methodology to reduce redundancy in the clause construction process. In Section 2.4, the WEKA platform is described, putting the focus on how it can be extended to accommodate an Inductive Logic Programming algorithm. Section 3 presents the pipeline of architecture used, the characterization of the dataset used in the study, the ILP-related components and the WEKA experiment configurations. The results are presented in Section 4. Section 5 shows the main findings and future work.

2. Materials and Methods

2.1. Text Classification

A very relevant problem, especially in the biomedical literature, is the automatic classification of textual scientific documents, which is the main target of the present study.

Automatic text classification is a sub-field of text mining. The goal of text classification is the assignment of documents into a fixed number of predefined categories. Each document can be in multiple, exactly one, or no category at all [12]. The main issue of text classification is to build a set of models that can correctly predict the class of the different unseen text documents with a high kappa statistics measure.

The Medline documents used in the study are structured documents, and each document is divided into six sections: Title, Abstract, Introduction, Methods (Materials and Methods, Methods, Experimental Procedures), Results (Results, Discussion, Results and Discussion) and Conclusions.

The text classification process involves the following phases:

- Document representation uses the Bag-of-Words (BoW) approach, where the value of a term is given by the standard $TF \times IDF$ (Term Frequency–Inverse Document Frequency) [13].
- In preprocessing, the documents pass through the application of several preprocessing techniques like special character removal, stop-word removal, dictionary validation, synonym handling, stemming and feature selection [8].
- In semantic enrichment, the full-text documents are semantically enriched using the Semantic Repository (SemRep), which extracts semantic predications from biomedical text. SemRep predications identify relationships between entities in MEDLINE documents. SemRep can combine and link knowledge from several sources and discover connections that might otherwise go unnoticed [10,14].
- In classifier learning, the dataset containing the documents is divided into a training and testing set. The classifier learns through the classified examples present in the training dataset and then automatically assigns a class label to new unlabeled text documents of the test dataset.
- The models were evaluated using the F1-measure and kappa statistics measures.

2.2. Inductive Logic Programming

Inductive Logic Programming [6,7,15] is a machine learning research area characterized by the development and application of logic-based multi-relational algorithms. Inductive Logic Programming uses a learning-from-examples approach. These examples can be of two kinds: positive or negative. Inductive Logic Programming represents data (examples), background knowledge and hypotheses as logic programs. The learning examples are presented as a set of facts. A very relevant feature of an Inductive Logic Programming system is that it can use background knowledge from the problem domain where the examples belong. Moreover, due to the powerful representation of FOL, used by Inductive Logic Programming, it is usually simple to encode the domain knowledge into the background

knowledge of an Inductive Logic Programming systems. Since FOL is also used to encode the induced models, very complex and comprehensible models can be induced.

A common method in Inductive Logic Programming to construct hypotheses/models is to search a lattice that has the most general clause at the top and, for practical reasons, is bound at the bottom by the most specific one. An algorithm like Aleph [5] uses a Mode-Directed Inverse Entailment (MDIE) [16] to construct a bottom clause that acts as the bottom limit of the search space. Depending on the number and nature of the background predicates, the number of examples provided and the search constraints, it may take a very long time to search the hypothesis space. Moreover, there is usually a lot of redundancy in the hypotheses generated automatically. Just recall that, as formal logic is concerned, two clauses that have the same literals but in a different order are logically equivalent (they are the same clause in a logical perspective). However, since the hypothesis space (the lattice) is constructed automatically, it may contain all the combinations of "the same logical clause". As the hypothesis space clauses are traversed, their length is increased, which also expands the possible number of combinations of their set of literals. This study proposes a new methodology that avoids this type of redundancy.

Inductive Logic Programming has been widely used in classification, clustering and regression analysis where data have an intrinsic structure. Relevant areas and applications where Inductive Logic Programming has been successfully applied include chemoinformatics (ex: rational drug design) [17], bioinformatics (analysis of genomics datasets) [18], text mining [19] and others.

Inductive Logic Programming system's main purpose is to find a set of hypotheses that, on the one hand, explain (cover) the positive examples—completeness, and on the other hand are consistent with the negative examples—consistency. A more formal definition of Inductive Logic Programming by Muggleton [6] states that given positive and negative examples and background knowledge, the goal of an Inductive Logic Programming system is to induce (learn) a hypothesis (a logic program), which, with the background knowledge, covers as many positive examples as possible and few negative examples or none.

Essentially, learning in an Inductive Logic Programming system can be viewed as a search problem for the solution within the space of all hypotheses called the hypothesis space. The background knowledge is a set of relations that can be used by the learning algorithm for inferring a definition of the target relation.

One of the major problems in using Inductive Logic Programming systems is poor scalability to problems with large search spaces and many examples. One main advantage of using Inductive Logic Programming is the creation of rules that can be easily interpreted by humans. In contrast to most forms of machine learning, Inductive Logic Programming can learn human-readable hypotheses from small amounts of data.

The Aleph System

The present study uses the Aleph system (version 5). Aleph is an Inductive Logic Programming system completely written in Prolog, developed at Oxford University, that induces hypotheses from background knowledge and examples.

The Aleph system requires four input files to construct a theory:

- The background knowledge (BK) specified as a logic program;
- The hypotheses specification (H);
- An optional set of restrictions (I);
- A finite set of examples E: positive examples E^+ and negative examples E^-, where

$$E = E^+ + E^- \tag{1}$$

The Aleph induction algorithm can be described as follows [5]:

1. Take one positive example from the example set. If none exists, stop.
2. Build the most specific clause that entails the example selected.
3. Search for a clause that is more general than the current clause. A more general clause is defined as a subset of the current set of literals. Search is guided by an evaluation function.
4. Remove the redundant examples (all examples covered by the current clause).
5. Repeat from Step 1.

To find a rule, the Aleph system starts by building the most specific clause, which is called the "bottom clause", that entails a seed example. Then, it uses a branch-and-bound algorithm to perform a general-to-specific heuristic search for a subset of literals from the bottom clause to form a more general rule.

The Aleph system has a large number of alternative methods to perform the search the hypothesis space. One such method uses refinement operators (RO). RO were first introduced in the Inductive Logic Programming setting by Ehud Shapiro in his Model Inference System (MIS) [20]. An RO is a "function" that receives as input a clause and returns all immediate legal refinements of the input clause. The recursive application of RO generates the whole search space. Another advantage of using ROs is that the ROs may be defined by the user and therefore make a priority-based predicate ordering mechanism for search strategy. Section 2.3 explains how to take advantage of that facility to reduce redundancy in the hypothesis search space.

2.3. Redundancy Reduction in ILP

As stated previously, redundancy, when searching the hypothesis space, is one of the major causes of long running times. There have already been several approaches to address the redundancy problem in Inductive Logic Programming. In [21,22], a set of query transformations were identified that enabled a reduction in redundancy, especially when using mathematical relational operators. In [23], an incremental language-level partition of the search space helped avoid redundancy by defining, at each level of the language-space partition, the maximum number of repeated predicate symbols.

In this work, we developed and implemented a new methodology to avoid another type of redundancy taking advantage of refinement operators. The rationale of the method is based on the fact that different combinations of the same set of literals are logically equivalent and therefore redundant. To be able to implement the methodology, the refinement operators were used as described in Section 2.2. To be able to apply the methodology, each predicate symbol to be used in the background knowledge is assigned a different "priority" number. The usual information concerning the predicates in the background knowledge must then be provided, as in the Aleph, IndLog [24] or Progol [16] Inductive Logic Programming systems.

Apart from the determinations declarations (specifying which predicate symbols should be used to build the models) and mode declaration (the "types" associated to the arguments of the predicates) a number (that is called an order number) is required, associated with each predicate symbol of the background knowledge. The refinement operators are then encoded to generate new and more specific clauses from previous ones to use the following restriction. Each new literal added to each clause refinement must have a predicate symbol with an order number higher than any predicate symbol already belonging to the clause. This introduces a strict total order set in the predicate symbols of each clause that does not allow for repetition of predicate symbols.

This methodology is acceptable for a large number of problems where the model does not use repeated predicate symbols. In terms of Language Level Search (LLS) the search is restricted to level 1 [23].

Related Work

Bart [25] proposes a new method that allows one to generate a synthetic image from a set of first order logic expressions that are characterizing an image class. The authors conclude that the enhanced expressiveness of the description language improves the produced classification rules, by reducing their size.

The study [26] presents OntoILPER, which is a system for extracting entity and relation instances from unstructured texts using ontology and Inductive Logic Programming, a symbolic machine learning technique. OntoILPER uses the domain ontology and takes advantage of a higher expressive relational hypothesis space for representing examples whose structure is relevant to information extraction.

The authors [27] present an automatic image interpretation method based on Inductive Logic Programming. The work consists in inducing classification rules that explicitly take into account structural knowledge, using Aleph, an Inductive Logic Programming (ILP) system. The proposed methodology is applied to three land cover/use maps of the French Guiana littoral. One hundred and forty-six classification rules were induced for the 39 land-cover classes of the maps. The rules are expressed in first-order logic language, which makes them intelligible and interpretable by non-experts. A ten-fold cross-validation gave average values for classification accuracy, specificity and sensibility equal to, respectively, 98.82%, 99.65% and 70%. The authors claim that the proposed methodology could be valuable to exploit to automatically classify new objects and/or help operators using object-based classification procedures.

These works are related because they use an Inductive Logic Programming based solution, although it is for image classification. As far as the available literature indicates, a hybrid solution that uses both Inductive Logic Programming and a propositional learner embedded in a multi-view system is a novel approach.

2.4. The Data Mining Platform

The machine learning algorithms are available in various programming languages implementations and accept different input data formats, making the task of comparing techniques hard to implement. To overcome this limitation, in the early 1990s, Waikato university started the development of a Java environment that can accelerate the implementation of the machine learning use cases, allowing the utilization of several techniques.

The WEKA—Waikato Environment for Knowledge Analysis—is a well-known open-source and no code machine learning platform that delivers a collection of machine learning algorithms [28,29], and it also has capabilities that allow the creation of additional algorithms. The WEKA platform has been accepted by the scientific community as a workbench, and it is a widely used tool for research tasks. WEKA organizes the machine learning algorithms in three categories: classification, clustering and association [29].

Aside from the machine learning algorithms, the WEKA tool has also features to preprocess data, to make feature selection and for data visualization. The tool can also be used through the Graphical User Interface (GUI), as well as using a command line.

One important feature of WEKA is that the tool is not limited to the algorithms available in the downloaded package. It is possible to improve the tool using the package manager that allows the integration of foreign algorithms.

The extension of the WEKA tool by embedding a new algorithm, is exactly one of the objectives highlighted and implemented in the work presented in this paper.

Extending WEKA with Aleph

The WEKA platform allows the extension with new schemes [28] and has several active projects with the aim of enriching the platform [30]. These improvements allow

WEKA to be extended without modifying the core classes of the platform. Because WEKA is developed with Java language, the extension must be created using Java language. The Aleph system, is however, implemented in Prolog. It may run using the Yet Another Prolog (YAP) [31] or the SWI Prolog. For this kind of interface, a wrapper was built to encapsulate the execution of the Aleph and take advantage of the WEKA platform functionalities. The WEKA platform version 3.8.3 provides three standard methods that should be implemented for the purpose of creating a "new" AbstractClassifier:

- *buildClassifier(instances)* is a method to build the model based on the given training instances;
- *classifyInstance(instance)* is a method that takes a test instance and returns a single predicted class using the model created earlier in the buildClassifier method;
- *distributionForInstances(instance)*, assuming the class is nominal (the datasets used in this study, the documents are classified as Relevant and Not_Relevant), this returns a class probability distribution.

The *AbstractClassifier* requires the buildClassifier to build the model and one of the other two classes, the *classifyInstance* or the *distributionForInstances*, to validate the model. Algorithm 1 presents the implementation of the *buildClassifier* and the *distributionForInstances*. The *buildClassifier* method receives a training dataset as a parameter, and based on the dataset, the model is created. In this study, the implementation starts by creating an ARFF file of the train dataset. The ARFF file is required for the WEKA platform. After this operation, the ARFF file must be converted to an Inductive Logic Programming format; in this case, a Python version 3 script was used to convert the ARFF file into a set of files suitable for Inductive Logic Programming.

Algorithm 1 ILP extends the AbstractClassifier-Method

```
1:  procedure buildClassifier(TrainingData)
2:      CreateArff(TrainingData)
3:      TransformArff2ILP(TrainingData)
4:      CreateILPModel(TrainingData)
5:  end procedure

6:  procedure distributionForInstances(TestInstance)
7:      CreateArff(TestInstance)
8:      TransformArff2ILP(TestInstance)
9:      RunILP( )
10:     for all i ∈ TestInstance do
11:         if i.IsCovered then
12:             Unseen ← 1
13:         else
14:             Unseen ← 0
15:         end if
16:         distribution[i][Unseen] ← distribution[i][Unseen] + 1
17:     end for
18: end procedure
```

To build the model, ILP requires the following files, which are created by the Python script:

- *data.yap* contains auxiliary predicates of the background.
- *ds.b* contains the background knowledge.
- *ds.f* contains the positive instances of the dataset.
- *ds.n* contains the negative instances of the dataset.

These four files (*data.yap*, *ds.b*, *ds.f* and *ds.n*) were generated by the referred Python script having the ARFF file as the source. The bash script *runaleph* must be called for the execution of the Aleph algorithm to build the model, calling the *induce* predicate after

loading the dataset files. As a result, a *rule*'s file is generated with the theory. Algorithm 2 shows an example of rules induced by Aleph.

Algorithm 2 ILP Model Example

```
doc(A) :-
  bact_bacteria(A,method,B),
  gtbact_bacteria(B,0.171),
  inoculum(A,method,C),
  ltinoculum(C,0.415).

doc(A) :-
  sepsi(A,result,B),
  gtsepsi(B,0.675).
```

From the model example in Algorithm 2, it is possible to verify that, if the term *bact_bacteria* exists in the Method section of the document, it must have a value that is greater than or equal to 0.171 for the document to be covered by the rule (the model created by the Inductive Logic Programming).

To apply the model to unseen instances, the *distributionForInstances* method must be called, as can be seen in Algorithm 1. A method is called to create an ARFF file from the instance that we want to predict the class. Then, call a Python script to transform the ARFF file into a Prolog file (test instance in this case a test.uf), and finally, call a bash script to execute the Aleph using the previous model from the *theory constructed* file. The result is caught through the execution of the Test Unseen command to verify whether the instance is covered or not covered by the rules. The distribution of the instance is based on the previous result. Algorithm 1 presents the methods to accomplish the extension of the WEKA platform with Inductive Logic Programming algorithms as implemented in Aleph.

Figure 1 shows the configuration of the Inductive Logic Programming algorithm (Aleph) inside the WEKA platform. The current version of this extension allows the following parameters:

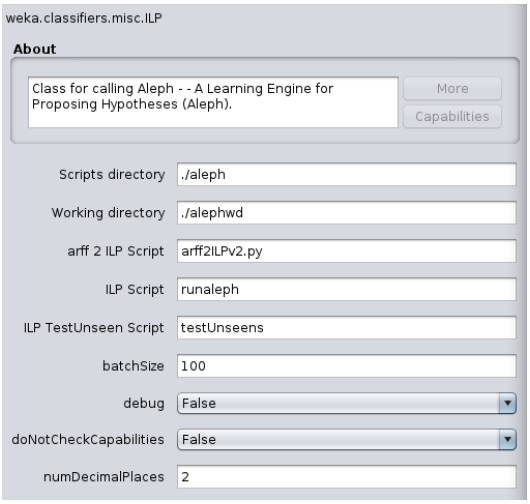

Figure 1. ILP as an algorithm of WEKA-GUI.

- *Scripts directory*—Aleph scripts location
- *Working directory*—Aleph working location
- *ARFF to ILP Script*—Script to transform an ARFF file into ILP suitable files. This script also creates, *on-the-fly*, the background knowledge
- *ILP Script*—Script to call Aleph with the training instances

- *ILP TestUnseen Script*—Script to call Aleph model with the test instance

3. Experimental Setup

To validate the approach, the proposed WEKA extension with Inductive Logic Programming was then applied to a specific case. Figure 2 presents the architecture pipeline used in the study. The architecture has four main components: data acquisition and storing, context awareness, feature selection and classification.

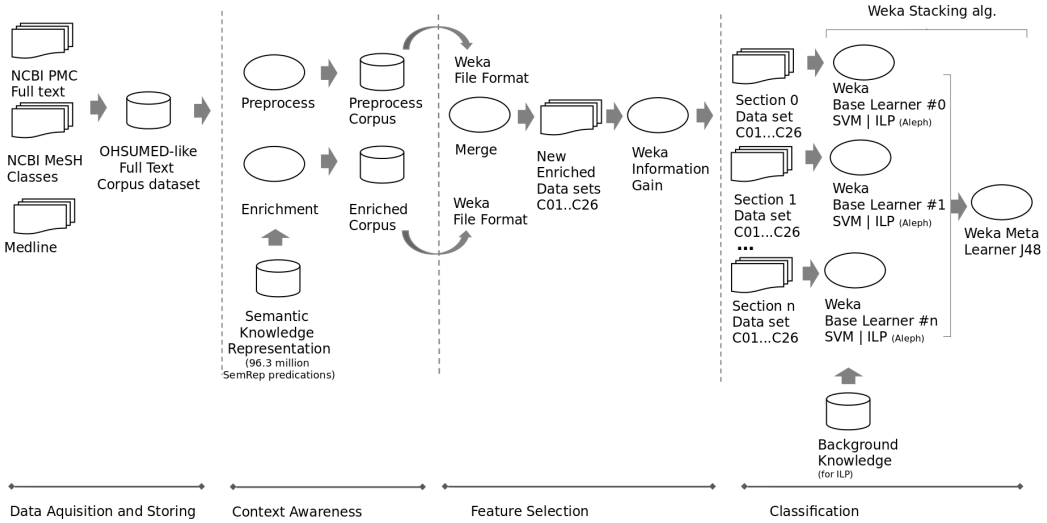

Figure 2. The architecture pipeline.

The data acquisition and storing stage encompassed the gathering of the MEDLINE full-text documents mapped with the MeSH classes, and an OHSUMED [11] based corpus was used. The initial document collection includes 50,216 titles and abstracts from 1987 to 1991. The corpus was implemented following the [32] study and was enriched with all sections of the papers (Title, Abstract, Introduction, Methods, Results and Conclusions) that were obtained through an interpolation process using the MeSH terms. The referred corpus contains 20 classes, each one associated with a particular disease.

The second stage, context awareness, includes the preprocessing technique and the semantic enrichment of the dataset using the semantic repository (SemRep) [14].

The third stage applies a feature selection algorithm to reduce the number of different terms introduced in the previous stage when enriching the dataset. The feature selection algorithm used was the *Information Gain*, which is part of the attribute selection in the WEKA platform. It used the Information Gain attribute selector (called Info Gain Attibute Eval) of WEKA [28,33]. A threshold cut of 1000 attributes was applied, which is justified in [34]. After the threshold cut, if a document has no terms, the document is removed from the dataset. The final number of documents for each of the 20 classes after the feature selection stage can be observed in Table 1, with the relevant and non-relevant documents.

The last stage of the pipeline, classification, uses the WEKA Experimenter to validate the approach. The system starts to divide the document sections of the dataset and applies a different algorithm (Support Vector Machine—SVM [35] or Inductive Logic Programming) to the different sections. The results of each of these algorithms are then used as the input to the stacking algorithm [36] that makes an ensemble of the results using the J48 (a C4.5 implementation [37]) as the meta-learner).

Table 1. OHSUMED dataset characterization.

Dataset	Description	Rel.	Non-Rel.
C01	Bacterial Infections and Mycoses	417	13,625
C02	Virus Diseases	1178	13,080
C03	Parasitic Diseases	51	13,884
C04	Neoplasms	5537	8789
C05	Musculoskeletal	51	13,884
C06	Digestive System	1662	12,484
C08	Respiratory Tract	857	13,184
C11	Eye Diseases	392	13,699
C12	Urologic and Male Genital Diseases	1196	12,985
C13	Female Genital Diseases and Pregnancy Complications	1136	12,954
C14	Cardiovascular Diseases	2532	11,792
C15	Hemic and Lymphatic	450	13,756
C16	Neonatal Diseases and Abnormalities	469	13,753
C17	Skin and Connective Tissue	1227	13,072
C18	Nutritional and Metabolic	1043	13,267
C19	Endocrine Diseases	772	13,415
C20	Immunologic Diseases	1721	12,536
C22	Animal Diseases	76	13,964
C25	Chemically-Induced Disorders	174	13,995
C26	Wounds and Injuries	247	13,949

The WEKA Experimenter was configured to run 10 times (the number of iterations) and using a 10-fold cross-validation procedure. With this configuration the system calls each algorithm 100 times using the training dataset and test dataset in order to be considered statistically meaningful [38].

3.1. DataSets Characterization

The dataset was based on the OSHUMED corpus and is characterized in Table 1, where each line shows the class name, the class description, the number of relevant documents and the number of non-relevant documents in the class.

From Table 1, it is possible to see the unbalanced datasets that were used in the experiments. Some of the datasets contain more relevant documents than other datasets presented on this corpus. Three of the datasets (C03, C05 and C22) have fewer than 100 relevant documents. One of the datasets (C04) is a nearly balanced dataset, containing 5537 relevant documents and 8789 non-relevant documents. The other datasets contain more than 100 relevant documents and more than 11,000 non-relevant documents.

3.2. Aleph Configuration

Aleph has a very large number of parameters that can be set to guide the search for good rules. We have changed the default values of the following parameters:

1. *clauselength* sets an upper bound on the number of literals in any constructed clause. We set *clauselength* to nine, which means that a clause can not have more than nine literals (including the head literal).
2. *minpos* sets the minimum number of positive examples that a rule is required to cover to be acceptable. We have set *minpos* to two, which avoids Aleph from accepting

clauses that "cover" less than two positive examples, e.g., to produce rules that generalize beyond a single case in the training set at minimum.

3. *noise* sets the maximum number of negative examples that a rule is allowed to cover to be acceptable. We have set *noise* to 20, which means that an acceptable clause may "cover" 20 negative examples.
4. *nodes* sets the upper bound on the number of clauses generated when searching for an acceptable clause. We set *nodes* to 10,000.
5. *searchtime* sets an upper bound on the time (in seconds) for searching the lattice associated with each bottom clause. We set the parameter to 600 s to limit the time taken to test an example.
6. *samplesize*: we set the *samplesize* to one, which means that only one example is used in the search for the next best clause to add to the theory.
7. *refine* allows the search in the space of hypotheses to be defined by the user, defining "refinement operators".

The experiments carried out used the parameters with the values presented in Algorithm 3.

Algorithm 3 Aleph Settings

```
:- set(noise, 20).
:- set(minpos, 2).
:- set(nodes, 10000).
:- set(samplesize, 1).
:- set(searchtime, 600).
:- set(clauselength, 9).
:- set(refine, user).
```

3.3. Background Knowledge

Background knowledge is the set of relevant pieces of knowledge that are useful to build models that solve the problem at hands. Difficult problems require, usually, a body of prior knowledge in order to allow the system to learn beyond the positive examples [39]. Since our original problem can be represented using an attribute–value representation (terms in sections with a $TF \times IDF$ value), the background knowledge can be expressed as relations. Algorithm 4 presents some examples of the background knowledge used. All the predicates can be expressed using mathematical relations such as *greater than*, *lower than* and *equal*.

The background knowledge uses determination declarations to list the predicate symbols that can be used in the construction of the model—predicate: *determination(targetPredicate, bodyPredicate)*. *Mode* declarations are used to "assign" types to predicate arguments. Although Prolog language is not a typed language, Inductive Logic Programming systems usually use mode declarations with types to avoid useless combinations of constants with the same value and a "different type" that would enormously increase the number of alternative instantiations.

The Aleph system allows the user to specify domain-specific refinement operators. The predicate *refine/2* was used with the aim to speed up the time taken to run our experiments by constructing only relevant clauses with a minimum of redundancy. Algorithm 5 presents the refinement operators used in this study. Apart from defining the refine/2 predicate, to specify our search procedure in detail, we had to develop two extra auxiliary predicates: *flatBody* and *lastPredicateNumber*.

Algorithm 4 Background Knowledge

```
...
gt(X, Type, Y) :-
        number(X),
        var(Y),
        findall(Val, (Call = ..[Type, _, _, Val], Call), Values),
        member(Y, Values),
        X > Y.
gt(X, _, Y) :-
        number(X),
        number(Y),
        X > Y.

eq(X, Type, Y) :-
        number(X),
        var(Y),
        findall(Val, (Call = ..[Type, _, _, Val], Call), Values),
        member(Y, Values),
        X = Y.

eq(X, _, Y) :-
        number(X),
        number(Y),
        X = Y.
...

:- determination(doc/1, infect/3).
:- determination(doc/1, ltinfect/2).
:- determination(doc/1, gtinfect/2).
:- determination(doc/1, eqinfect/2).
...
:- modeb(*, infect(+id, #section, infect)).
:- modeb(*, ltinfect(+infect, #infect)).
:- modeb(*, gtinfect(+infect, #infect)).
:- modeb(*, eqinfect(+infect, #infect)).
...
```

To define a domain-specific refinement, the operator *refine/2* statement must be used. Along with this predicate, a set of additional predicates have been incorporated that contribute to (1) assigning "a priority" –being an order number– to background predicates and (2) avoiding recombination of already used predicates. With this approach, a substantial reduction in the search space and thus in the execution time of the algorithm is achieved. The predicate priority *assigns* an order/priority to each predicate symbol. When refining a new clause, the highest priority number assigned to the existing literal clause must be fetched. This is accomplished by predicate *lastPredicateNumber/2*. When we have the value of the highest priority predicate symbol, we can only select a new predicate symbol with higher priority that will be added to the clause being refined at the end of its body. This last operation of adding a new literal at the end of a clause being refined is done by *flatBody/3*.

The background knowledge is created during the WEKA execution of the buildClassifier method, when the script to transform the ARFF file into an ILP format is called.

Algorithm 5 Background Knowledge–Refinement Operators

```
1:  refine(false, doc(_)).

2:  refine(doc(Id), (doc(Id) :- Literals)) :-
3:          sortedPred(_Order, Id, Literals).

4:  refine((doc(Id) :- Body), (doc(Id) :- NewBody)) :-
5:          lastPredicateNumber(Body, LastPredicateNumber),
6:          sortedPred(NewOrder, Id, NewLiterals),
7:          NewOrder > LastPredicateNumber,
8:          flatBody(Body, NewLiterals, NewBody).

9:  flatBody((Lit1, Rest), Last, (Lit1, More)) :-
10:         flatBody(Rest, Last, More).
11: flatBody(Lit1, (Lit2, Lit3), (Lit1, Lit2, Lit3)).
12: flatBody(Lit1, Lit2, (Lit1, Lit2)).

13: lastPredicateNumber((_, More), LastPredicateNumber) :-
14:         lastPredicateNumber(More, LastPredicateNumber).
15: lastPredicateNumber(LastPredicate, LastPredicateNumber) :-
16:         functor(LastPredicate, Symb, _Arity),
17:         predNumber(Symb, LastPredicateNumber).
```

3.4. Experimenter

The experiments were carried out using the WEKA Experimenter using the configurations in Table 2.

Table 2. Experimenter parameters used.

Parameters	Description
10 folds	Dataset divided in 10 parts with approximately equal size
10 iterations	The system runs 10 times each configuration
Datasets	C01 thru C26
Algorithms	• Stacking implements the ensemble method • J48 was the Meta Learner algorithm • ILP (Title, Abstract and Conclusions) and SVM (Introduction, Methods and Results) were the Base Learners
Measures	• Accuracy • F-Measure • Kappa Statistics

The Experimenter configurations are presented in Figure 3.

Table 3 shows the number of terms of the dataset after applying the Information Gain feature selection algorithm in order to reduce the number of terms. From the table, we see that the sections with more terms are the Introduction, Methods and Results (on average) with 213.2, 245.3 and 287.7 respectively.

Table 3. Number of terms by section.

	Title	Abs.	Introd.	Method	Result	Conc.
min	28	128	141	164	188	11
max	113	211	308	352	370	117
avg	46.4	165.7	213.2	245.3	287.7	41.5

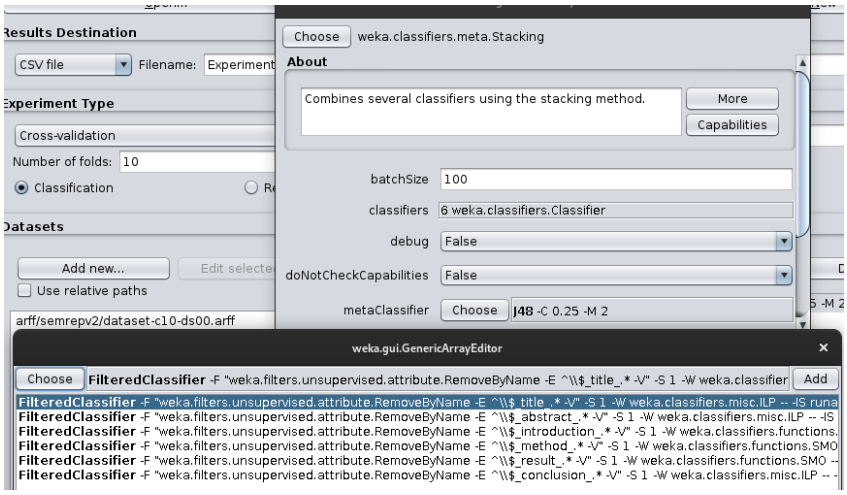

Figure 3. Experimenter configurations.

4. Results

This section presents the results obtained in the experiments. The WEKA platform has several metrics that can be used for the evaluation task. The study used the accuracy, F-measure and also thekKappa statistics [40], as they arestatistically robust [41] and widely used to compare the performance of algorithms [42]. In the experiments, the measures were selected since the two approaches (stacking algorithm with Inductive Logic Programming vs. stacking algorithm with Inductive Logic Programming plus refinement operators) were used with the same criterion for evaluating the results obtained.

To measure the effectiveness of the research, a set of quality evaluations are available. Most of the evaluation measures can be computed based on the confusion matrix (see Table 4) [43].

Table 4. Evaluation: confusion matrix.

	Classified as Positive RELEVANT	**Classified as Negative** NON-RELEVANT
Is Positive: RELEVANT	*True Positive* (TP)	False Negative (FN)
Is Negative: NON-RELEVANT	False Positive (FP)	*True Negative* (TN)

Each column of the matrix represents the classifier output (the predicted class), while each row represents the actual class of the instances. The *TP* (True Positive) is the number of instances correctly classified as positive in this study, "Relevant". The *TN* (True Negative) is the number of instances correctly classified as negative in this study "Non-Relevant". The *FP* (False Positive) is the number of instances incorrectly classified as positive, and the *FN* (False Negative) is the number of instances incorrectly classified as negative.

In this research the F-measure (2) was used—a measure of testing the accuracy. The F-measure combines precision and recall reflecting the relative importance of precision vs. recall.

$$F - Measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{2}$$

Precision or the *positive predictive value* is a measure that estimates the probability that a positive prediction is correct.

$$Precision = \frac{Number\ of\ Retrieved\ Relevant\ Documents}{Total\ Number\ of\ Relevant\ Documents} = \frac{TP}{TP + FP} \tag{3}$$

Recall or the *true positive rate* or *sensitivity*, is the proportion of documents belonging to the positive class (*Relevant*) that were correctly predicted as positive (*Relevant*).

$$Recall = \frac{Number\ of\ Retrieved\ Relevant\ Documents}{Total\ Number\ of\ retrieved\ Documents} = \frac{TP}{TP + FN} \quad (4)$$

The F-measure is computed as the harmonic average of the precision and recall. The best performance of a classifier on a classification task is when the F-measure has a value of 1 (perfect precision and recall), and its worst performance is when the F-measure is 0.

Kappa statistics represents the value of Cohen's kappa coefficients, a statistical measure that determines the degree of agreement between different classifiers. The jappa is a measure of differences of obtained agreement and the chance agreement. The values can be scaled from -1 to 1. When the obtained agreement is equal to the change agreement, then k is equal to 0. The upper limit is 1 and represents the perfect agreement. Negative values indicate agreement less than chance. Table 5 presents a common interpretation of the kappa agreement [44].

Table 5. Kappa statistics.

Kappa Agreement	
<0	Less than chance
0.01–0.20	Slight
0.21–0.40	Fair
0.41–0.60	Moderate
0.61–0.80	**Substantial**
0.81–0.99	**Almost perfect**

The analysis in this study focuses on kappa values that are greater than or equal to 61% a **substantial** or an **almost perfect** agreement.

The Corrected Resample *t*-Test [45] is used by the WEKA tool to determine the statistical significance of a pair-wise comparison of schemes. According to the study, this approach outperforms the standard *t*-Test, and this technique is used as the default by the WEKA tool used in this study.

4.1. Discussion

This subsection presents the results achieved when using the pipeline with the four stages presented in Figure 2. Table 6 highlights the comparison between the two approaches: (i) using Inductive Logic Programming in a stacking algorithm as a base learner without refinement operators and (ii) using refinement operators on Inductive Logic Programming inside the stacking algorithm. The stacking algorithm applies SVM and Inductive Logic Programming as base learners to sections of the document of the datasets and applies the C4.5 algorithm as a meta-learner, as described in Section 3. Due to the nature of the unbalanced datasets, the accuracy alone can provide an overly optimistic assessment.

Table 6 presents the results obtained in the experiments. It has the following columns:

- Dataset: the dataset used;
- Measures: accuracy, F-measure and kappa statistics;
- Baseline represents the results obtained using the dataset containing the sections: Title, Abstract, Introduction, Methods, Results and Conclusions. The stacking algorithm was used with Inductive Logic Programming applied as a base learner in the Title, Abstract and Conclusion sections, with SVM applied to the other sections (Introduction, Methods and Results). The baseline does not use refinement operators.

- Ref. Oper. uses the same configurations of the baseline, but it takes advantage of the refinement operators.

Table 6. Comparison of baseline and refined operators across datasets using a stacking algorithm with embedded ILP as a base learner, comparing a base line without refinement operators and with refinement operators. The (+) represents the best result. The **bold** results represent the statistical significance of the best result $\geq 61\%$.

	Accuracy (%)		F-Measure		Kappa	
Dataset	**Baseline**	**Ref. Oper.**	**Baseline**	**Ref. Oper.**	**Baseline**	**Ref. Oper.**
C01	97.789	97.988 (+)	0.496	0.548 (+)	0.486	0.539 (+)
C02	96.965	97.375 (+)	0.796	0.841 (+)	**0.780**	**0.827 (+)**
C03	99.643	99.643	0.000	0.402	0.000	0.048
C04	90.943	93.885 (+)	0.877	0.922 (+)	**0.806**	**0.872 (+)**
C05	99.634	99.638	0.167	0.341	0.003	0.031
C06	93.676	95.220 (+)	0.675	0.789 (+)	**0.642**	**0.762 (+)**
C08	96.296	96.296	0.636	0.636	**0.617**	**0.617**
C11	98.711	98.711	0.732	0.732	**0.725**	**0.725**
C12	95.788	95.788	0.706	0.706	**0.684**	**0.684**
C13	94.778	95.371 (+)	0.595	0.658 (+)	0.569	**0.635 (+)**
C14	91.621	92.119	0.728	0.755	**0.680**	**0.709**
C15	97.126	97.307 (+)	0.298	0.349	0.288	0.340 (+)
C16	97.063	97.084	0.288	0.292	0.278	0.282
C17	95.424	96.088 (+)	0.680	0.735 (+)	**0.657**	**0.714 (+)**
C18	94.623	94.817	0.519	0.539	0.493	0.514
C19	96.458	97.389 (+)	0.581	0.709 (+)	0.564	**0.696 (+)**
C20	93.969	94.730 (+)	0.699	0.754 (+)	**0.667**	**0.725 (+)**
C22	99.459	99.459	0.225	0.225	0.004	0.004
C25	98.942	99.025	0.329	0.393	0.309	0.389
C26	98.310	98.558 (+)	0.172	0.346 (+)	0.142	0.338 (+)

One interesting observation is that the refined operators generally improve the classification performance, particularly for the F-measure and kappa (except C08, C11, C12 and C22). Sixteen classes improve the F-measure and kappa results, 10 have statistical significance, and 11 present kappa results higher than 61% (substantial or almost perfect). Analysing the highest kappa results, it can be observed that seven results have substantial and almost perfect results (C02, C04, C06, C13, C17, C19 and C20). C13 improves from moderate 0.569 to substantial with 0.635, and C19 improves from 0.564 to 0.696.

The best results are obtained with datasets C02, C04, C06, C08, C11, C12, C13, C14, C17, C19 and C20. From Table 7, it is possible to highlight the substantial increase in the results when applying refinement operators with Inductive Logic Programming as a base learner (C02, C04, C06, C13, C17, C19 and C20). The performance disparity in highly unbalanced datasets, such as C03, C05, and C22, which contain fewer than 100 relevant documents out of nearly 14,000 total instances, highlights that a majority-class classifier can achieve over 99.4% accuracy simply by predicting 'non-relevant' for all instances, but with very low F-measure and kappa statistics. Classes C15, C16, C25, and C26 represent moderately unbalanced minority classes, with positive sample ratios ranging between 1.2% and 3.3%. While these datasets did not achieve the target 'substantial' agreement threshold (kappa $\geq$ 0.61), they exhibit a positive trend when evaluating the impact of refinement operators. The results also show that the refinement operators are not universally beneficial. For C08, C11, C12 and C22, all three metrics remain unchanged. This suggests that, for these datasets, the refinement operators do not alter the classification decisions sufficiently to produce measurable performance gains.

To calculate the percentage increase (presented in Table 7), the following Equation (5) was used:

$$\%Increase = \frac{RO - BL}{\|BL\|} \times 100 \quad (5)$$

BL baseline represents the results obtained using the dataset containing the sections: Title, Abstract, Introduction, Methods, Results and Conclusions. The stacking algorithm was used with Inductive Logic Programming applied as a base learner in the Title, Abstract and Conclusion sections, with SVM applied to the other sections (Introduction, Methods and Results). The baseline does not use refinement operators.

RO refinement operators uses the same configurations of the baseline but takes advantage of the refinement operators.

From Table 7, it is possible to see that the C02 and C04 obtained an **almost perfect** agreement with 82.7% and, 87.2% respectively. C02 has an improvement of (82.7 − 78.0)/78.0 = 6.03 (an increase of 6.03%). C04 has a percentage increase of (87.2 − 80.6)/80.6 = 8.19, representing an increase of 8.19%. C06, C13 and C19 have the highest improvements of 18.69%, 11.60% and 23.40% respectively. C13 and C19 improve to be considered in this approach of ≥61%.

Table 7. Comparison of baseline and refinement operators with kappa larger or equal to 61%.

	Kappa		
Dataset	**Baseline**	**Ref. Oper.**	**% Increase**
C02	0.780	0.827	6.03%
C04	0.806	0.872	8.19%
C06	0.642	0.762	18.69%
C08	0.617	0.617	0.00%
C11	0.725	0.725	0.00%
C12	0.684	0.684	0.00%
C13	0.569	0.635	11.60%
C14	0.680	0.709	4.26%
C17	0.657	0.714	8.68%
C19	0.564	0.696	23.40%
C20	0.667	0.725	8.70%

Table 7 shows that the refinement operators improve or preserve the kappa statistics for all 11 datasets. Kappa increases in eight datasets, while the remaining three datasets (C08, C11 and C12) maintain exactly the same value, with no observed degradation in kappa. Regarding the significant jump in kappa (e.g., 0.564 to 0.696 in C19), this can be attributed to the 600 s search limit and the use of the refinement operators. Without refinement operators, the system frequently exhausted its time budget evaluating logically equivalent redundant clauses. By enforcing a strict total order on predicates, the search becomes more efficient, allowing Aleph to traverse a wider variety of unique high-quality hypotheses within the same time slot. Thus, the performance gain is a direct result of the computational efficiency enabling the discovery of superior rules that were previously out of reach. The results presented in Table 8 also suggest that the refinement operators pipeline does not produce a uniform performance improvement during training, but it produces a much more consistent reduction in test execution time.

Table 8. Elapse execution CPU (time in milliseconds) results using a stacking algorithm with embedded ILP as a base learner, comparing a baseline without refinement operators and with refinement operators.

	Baseline		Ref. Oper.	
Dataset	**Train**	**Test**	**Train**	**Test**
C02	177849	4847	104148	2510
C04	63613	2968	126914	2571
C06	129152	2998	153458	4263
C08	70803	3072	164273	2907
C11	90994	5231	111598	2752
C12	107027	3142	236390	2941
C13	233220	5821	119216	2750
C14	89934	3271	122236	2871
C17	302198	5606	140817	4162
C19	182658	5313	120957	4025
C20	255294	5107	125123	2607

The refinement operators pipeline reduces the overall execution time by approximately 10.14%. The training results are considerably more heterogeneous. The refinement operators approach reduces the training time in five datasets: C02, C13, C17, C19 and C20. C13, C17 and C20 show substantial reductions. Overall, the results indicate that the proposed refinement operators improve the computational efficiency of the pipeline, particularly during the application of the resulting operators. Additionally, the time analysis supports the earlier accuracy/F-measure/kappa results: if the refinement operators reduce the overall execution time while simultaneously improving the predictive performance, it can be argued that both the computational efficiency and effectiveness are improved.

4.2. Research Limitations

While the proposed hybrid stacking framework and refinement operators demonstrate significant performance and efficiency advantages, several limitations must be acknowledged:

- Search space restriction. While highly effective at eliminating logically equivalent clauses, contributing to a substantial reduction in the search space, this constraint restricts the search space to level 1, which may limit applicability in domains requiring recursive or highly nested relational patterns.
- Class imbalance. This imbalance affects the interpretation of accuracy and is particularly relevant for classes which contain fewer positive instances, resulting in near-zero kappa statistics on these specific classes.
- Heterogeneity in computational overhead: Although the refinement operators pipeline achieves an overall 10.14% reduction in execution time and a highly consistent speedup during testing, the training time remains highly heterogeneous. This may indicate that when the search space is highly dense, the overhead of applying user-defined refinement operators during training can in some cases need more computational time.
- Hybrid stacking configuration. The proposed approach was evaluated within a specific stacking architecture using SVM and Inductive Logic Programming as base learners and J48 as the meta-learner. Alternative propositional learners, relational learners, meta-learners, and ensemble configurations were not systematically evaluated. Consequently, the robustness of the redundancy-reduction operators across different learning algorithms, parameter configurations, and alternative datasets remains an open research question that future studies should address.

5. Conclusions

This study had two main objectives to address. The first objective was to verify whether Inductive Logic Programming could be used in conjunction with propositional algorithms, how it behaved when used together, and also implement the pipeline defined in the proposed architecture.

The second objective was to improve the long running time taken to construct models with Inductive Logic Programming. The last goal was achieved by means of reducing redundancy.

Concerning the first objective, it was not only possible to demonstrate the joint use of multi-relational algorithms and propositional algorithms but also that this combination presents very promising results. It was demonstrated that it is possible to extend the WEKA platform with relational algorithms, namely the Aleph Inductive Logic Programming system. This approach has contributed to speed up the process of building solutions that use different algorithms, allowing the comparison of results in an aggregated manner.

For the second objective, a proposed methodology enables a significant reduction in the redundancy of the hypothesis space by using refinement operators with a controlled combination of the literals.

On top of these objectives, the results obtained are very promising, showing the ability to use an Inductive Logic Programming system within a data mining platform.

As future work, three lines of improvement were identified: (1) embedding the entire pipeline within the WEKA platform, namely the "Data Acquisition and Storing" and "Context Awareness" components and (2) making available Inductive Logic Programming/Aleph configuration parameters on the WEKA GUI platform. While the present study demonstrated the feasibility of the approach within a specific stacking architecture, (3) it is important to investigate the algorithmic robustness of the proposed methodology across different datasets, learning algorithms, parameter settings, and Inductive Logic Programming search strategies.

Author Contributions: Conceptualization, C.A.G. and R.C.C.; methodology, L.B.D. and E.L.I.; software, C.A.G. and R.C.C.; validation, C.A.G. and C.T.G.; formal analysis, R.C.C., C.A.G. and C.T.G.; investigation, R.C.C. and C.A.G.; resources, C.A.G. and R.C.C.; data curation, C.A.G. and C.T.G.; writing—original draft preparation, C.T.G. and C.A.G.; writing—review and editing, R.C.C., E.L.I., L.B.D. and A.S.; visualization, E.L.I.; supervision, L.B.D.; project administration, C.T.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work is financed by Portuguese national funds through FCT—Fundação para a Ciência e Tecnologia, under the project UID/05422/2025: Centre for Organisational and Social Studies of Polytechnic of Porto, with doi https://doi.org/10.54499/UID/05422/2025. This work was financially supported by UID/00027/2025 of the Artificial Intelligence and Computer Science Laboratory (LIACC), with doi https://doi.org/10.54499/UID/00027/2025, funded by Fundação para a Ciência e a Tecnologia, I.P./MECI through national funds.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The research data used can be found in the following url: https://bitbucket.org/coliveira1/datasets/src/master/dataset-c.zip (accessed on 14 August 2026).

Acknowledgments: This manuscript is based in part on research originally presented in the doctoral thesis *Multi-Relational Learning for Full Text Scientific Document Classification*, by Carlos Adriano de Oliveira Gonçalves, University of Vigo, Ourense, Spain, 2022. The authors gratefully acknowledge the *SoftCPS: Software Cyber-Physical Systems Group* at the School of Engineering of the Polytechnic of Porto (ISEP) for their valuable assistance and provision of research resources and infrastructure. https://www2.isep.ipp.pt/softcps/ (accessed on 14 August 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Baxevanis, A.D.; Ouellette, B.F. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*; Baxevanis, A.D., Francis Ouellette, B.F., Eds.; John Wiley & Sons: Hoboken, NJ, USA, 2005.
2. Lesk, A. *Introduction to Bioinformatics*; Oxford University Press: Oxford, UK, 2019.
3. Roberts, R.J. PubMed Central: The GenBank of the published literature. *Proc. Natl. Acad. Sci. USA* **2001**, *98*, 381–382. [CrossRef] [PubMed]
4. Allahyari, M.; Pouriyeh, S.; Assefi, M.; Safaei, S.; Trippe, E.D.; Gutierrez, J.B.; Kochut, K. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. *arXiv* **2017**, arXiv:1707.02919.
5. Srinivasan, A. The Aleph Manual, University of Oxford, Department of Computer Science, Oxford, U.K. [Online]. Available online: https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html (accessed on 14 August 2026).
6. Muggleton, S. Inductive logic programming. *New Gener. Comput.* **1991**, *8*, 295–318. [CrossRef]
7. Muggleton, S.; De Raedt, L. Inductive logic programming: Theory and methods. *J. Log. Program.* **1994**, *19*, 629–679. [CrossRef]
8. Gonçalves, C.; Camacho, R.; Gonçalves, C.T.; Seara, A.; B. Diz, L.; L. Iglesias, E. Classification of Full Text Biomedical Documents: Sections Importance Assessment. *Appl. Sci.* **2021**, *11*, 2674. [CrossRef]
9. Gonçalves, C.A.; Vieira, A.S.; Gonçalves, C.T.; Camacho, R.; Iglesias, E.L.; Diz, L.B. A Novel Multi-View Ensemble Learning Architecture to Improve the Structured Text Classification. *Information* **2022**, *13*, 283. [CrossRef]
10. Gonçalves, C.A.; Vieira, A.S.; Gonçalves, C.T.; Borrajo, M.L.; Camacho, R.; Iglesias, E.L. To Enhance Full-Text Biomedical Document Classification Through Semantic Enrichment. In *Hybrid Artificial Intelligence Systems (HAIS 2023)*; Lecture Notes in Computer Science (LNCS); Springer: Cham, Switzerland, 2023; pp. 554–565. [CrossRef]
11. Hersh, W.; Buckley, C.; Leone, T.; Hickam, D. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*; Springer: London, UK, 1994; pp. 192–201.
12. Joachims, T. Text categorization with Support Vector Machines: Learning with many relevant features. In *Proceedings of the Machine Learning: ECML-98*; Nédellec, C., Rouveirol, C., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 137–142.
13. Zhou, W.; Smalheiser, N.R.; Yu, C. A tutorial on information retrieval: Basic terms and concepts. *J. Biomed. Discov. Collab.* **2006**, *1*, 2. [CrossRef] [PubMed]
14. Kilicoglu, H.; Rosemblat, G.; Fiszman, M.; Shin, D. Broad-coverage biomedical relation extraction with SemRep. *BMC Bioinform.* **2020**, *21*, 188. [CrossRef] [PubMed]
15. Plotkin, G. Automatic Methods of Inductive Inference. Ph.D. Thesis, Edinburgh University, Edinburgh, UK, 1971.
16. Muggleton, S. Inverse Entailment and Progol. *New Gener. Comput.* **1995**, *13*, 245–286. [CrossRef]
17. Fonseca, N.A.; Pereira, M.; Santos Costa, V.; Camacho, R. Interactive discriminative mining of chemical fragments. In *International Conference on Inductive Logic Programming*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010 ; pp. 59–66.
18. Fertin, G.; Jean, G.; Tannier, E. Genome rearrangements on both gene order and intergenic regions. In *Algorithms in Bioinformatics*; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2016; pp. 162–173.
19. Lima, R.; Freitas, F.; Espinasse, B. Relation extraction from texts with symbolic rules induced by inductive logic programming. In *Proceedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), Vietri sul Mare, Italy, 9–11 November 2015*; IEEE: New York, NY, USA, 2016; pp. 194–201.
20. Shapiro, E.Y. *Algorithmic Program Debugging*; Yale University: New Haven, CT, USA, 1982.
21. Costa, V.S.; Srinivasan, A.; Camacho, R. A note on two simple transformations for improving the efficiency of an ILP system. In *Proceedings of the International Conference on Inductive Logic Programming*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 225–242.
22. Costa, V.S.; Srinivasan, A.; Camacho, R.; Blockeel, H.; Demoen, B.; Janssens, G.; Struyf, J.; Vandecasteele, H.; Laer, W.V. Query transformations for improving the efficiency of ILP systems. *J. Mach. Learn. Res.* **2003**, *4*, 465–491.
23. Camacho, R. Improving the efficiency of ilp systems using an incremental language level search. In *Proceedings of the Twelfth Belgian-Dutch Conference on Machine Learning (Benelearn 2002)*; Utrecht University: Utrecht, The Netherlands, 2002; pp. 16–22.
24. Camacho, R. IndLog—Induction in Logic. In *Proceedings of the Logics in Artificial Intelligence*; Alferes, J.J., Leite, J., Eds.; Springer Nature: Cham, Switzerland, 2004; pp. 718–721.
25. Lamiroy, B.; Ropers, J.P. Assessing Inductive Logic Programming Classification Quality by Image Synthesis. In *Proceedings of the Eighth IAPR International Workshop on Graphics Recognition-IAPR-GREC*; University of La Rochelle: La Rochelle, France, 2009; pp. 344–352.
26. Lima, R.; Espinasse, B.; de Freitas, F.L.G. OntoILPER: An ontology- and inductive logic programming-based system to extract entities and relations from text. *Knowl. Inf. Syst.* **2018**, *56*, 223–255. [CrossRef]

27. Bayoudh, M.; Roux, E.; Nock, R.; Richard, G. Automatic learning of structural knowledge from geographic information for updating land cover maps. In *Symposium of the Latin American Society for Remote Sensing and Spatial Information Systems (SELPER)*; Universidad Nacional de Luján: Luján, Argentina, 2012.
28. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA Data Mining Software: An Update. *ACM Sigkdd Explor. Newsl.* **2009**, *11*, 10–18. [CrossRef]
29. Witten, I.H.; Frank, E.; Hall, M.A. *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed.; Morgan Kaufmann Series in Data Management Systems; Morgan Kaufmann: Amsterdam, The Netherlands, 2011.
30. Gewehr, J.E.; Szugat, M.; Zimmer, R. BioWeka—extending the Weka framework for bioinformatics. *Bioinformatics* **2007**, *23*, 651–653. [CrossRef] [PubMed]
31. Costa, V.S.; Rocha, R.; Damas, L. The YAP prolog system. *Theory Pract. Log. Program.* **2012**, *12*, 5–34. [CrossRef]
32. Moschitti, A.; Basili, R. Complex linguistic features for text classification: A comprehensive study. In *Proceedings of the European Conference on Information Retrieval*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 181–196.
33. Witten, I.H.; Frank, E.; Trigg, L.E.; Hall, M.A.; Holmes, G.; Cunningham, S.J. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations*; University of Waikato: Hamilton, New Zealand, 1999.
34. Forman G. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* **2003**, *3*, 1289–1305.
35. Platt, J. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*; Technical Report; Microsoft: Redmond, WA, USA, 1998.
36. Wolpert, D.H. Stacked generalization. *Neural Netw.* **1992**, *5*, 241–259. [CrossRef]
37. Quinlan, J.R. *C4. 5: Programs for Machine Learning*; Elsevier: Amsterdam, The Netherlands, 2014.
38. Scuse, D.; Reutemann, P. *Weka Experimenter Tutorial for Version 3-5-5*; University of Waikato: Hamilton, New Zealand, 2007.
39. Lavrac, N.; Dzeroski, S. *Inductive Logic Programming: Techniques and Applications*; Routledge: New York, NY, USA, 1993.
40. Cohen, J. A coefficient of agreement for nominal scales. *Educ. Psychol. Meas.* **1960**, *20*, 37–46. [CrossRef]
41. Ben-David, A. Comparison of classification accuracy using Cohen's Weighted Kappa. *Expert Syst. Appl.* **2008**, *34*, 825–832. [CrossRef]
42. Vieira, S.; Kaymak, U.; Sousa, J. Cohen's kappa coefficient as a performance measure for feature selection. In *Proceedings of the International Conference on Fuzzy Systems, Barcelona, Spain, 18–23 July 2010*; IEEE: New York, NY, USA, 2010; pp. 1–8.
43. Costa, E.; Lorena, A.; Carvalho, A.C.P.L.F.; Freitas, A. A review of performance evaluation measures for hierarchical classifiers. In *Evaluation Methods for Machine Learning II: Papers from the AAAI-2007 Workshop*; AAAI Press: Menlo Park, CA, USA, 2007; pp. 1–6.
44. Viera, A.J.; Garrett, J.M. Understanding interobserver agreement: The kappa statistic. *Fam Med.* **2005**, *37*, 360–363. [PubMed]
45. Nadeau, C.; Bengio, Y. Inference for the Generalization Error. *Mach. Learn.* **2003**, *52*, 239–281. [CrossRef]