Instituto Tecnológico
y de Estudios Superiores de Occidente

Recognition of official validity of higher level studies according to secretarial agreement 15018, published in the Official Gazette of the Federation of November 29, 1976.

Departament of Electronic, Systems and Computing

Master's Degree in Computational Systems

IMPROVED PREDICTION OF HEURISTIC CONFIGURATION FOR QUEUE PRIORITY ON FACT++ SEMANTIC REASONER

ACCEPTANCE WORK to obtain the **DEGREE** of **MASTER IN COMPUTATIONAL SYSTEMS**

Presents: Juan José González Álvarez

Supervisor: Dr. Luis Miguel Escobar Vega

Tlaquepaque, Jalisco. July 13, 2021

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales

MEJORA EN LA PREDICCIÓN DE LA CONFIGURACIÓN HEURÍSTICA PARA LA COLA DE PRIORIDAD EN EL RAZONADOR SEMÁNTICO FACT++

TRABAJO RECEPCIONAL que para obtener el **GRADO** de **MAESTRO EN SISTEMAS COMPUTACIONALES**

Presenta: Juan José González Álvarez

Asesor: Dr. Luis Miguel Escobar Vega

Tlaquepaque, Jalisco. 13 de Julio, 2021

ACKNOWLEDGMENTS

I would like to thank my family, partner and friends, who have been with me in difficult and happy times.

Thank Luis, my work supervisor, who showed me an interesting topic and guided me through the rigorous road of researching. To my professors whose passion on the subject was determinant on my success as a student.

I would like to thank CONACYT for the grant with number 501629, I received and made possible to take this master program.

AGRADECIMIENTOS

Quisiera agradecer a mi familia, mi pareja y amigos, quienes han estado conmigo en los momentos díficiles y los momentos felices.

Agradecer a Luis, mi supervisor de este trabajo, quien me mostró un tema interesante y me guió a través del riguroso camino de la investigación. A mis profesores cuya pasión en el tema fue determinante para mi éxito como estudiante.

Me gustaría agradecer a CONACYT por la beca número 501629 que recibí, y que hizo posible cursar este programa de maestría.

A mi madre Rosa y mi padre Juan

ABSTRACT

With the adoption of the semantic web, interest in technologies and theory about formalizing the representation of knowledge, and automated reasoning has increased. Ontologies are concrete instances of knowledge representation and logical reasoners play an important role during the creation of ontologies, since they can find logical errors during design.

One of these logical reasoners is FaCt++, which implements an optimized analytical tableaux algorithm. The specific implementation of tableaux in FaCt++ includes a set of priority queues to handle and expand the different operators that can be found during reasoning, these queues have an order for applying the different operators, which in this work will be referred as priority configurations.

It was proved by the authors of FaCt++ that there is not a single priority configuration that has the best performance for all types of ontologies. Recently it was suggested that machine learning models can be successfully applied to find the best heuristic for every specific ontology.

In this work is presented the process to build a machine learning model to find the best priority configuration for every ontology in detail. The model proposed in this work uses fewer features than the one shown previously and creates a simpler model with similar or better accuracy on the resulting classification.

RESUMEN

Con la adopción de la web semántica, el interés por las tecnologías y teoría sobre la formalización de la representación del conocimiento y el razonamiento automático ha incrementado. Las ontologías son instancias concretas de la representación del conocimiento y los razonadores lógicos juegan un papel importante en su creación, porque pueden encontrar errores lógicos durante el diseño.

Uno de estos razonadores lógicos es FaCt++, el cual implementa un algoritmo tableaux optimizado. La implementación específica de tableaux en FaCt++ incluye un conjunto de colas de prioridad que manejan y expanden los diferentes operadores que pueden ser encontrados durante el razonamiento, estas colas tienen un orden para aplicar los diferentes operadores, a las cuales en este trabajo se les referirá como configuraciones de prioridad.

Los autores de FaCt++ demostraron que no hay una única configuración de prioridad que tenga el mejor desempeño para todos los tipos de ontologías. Recientemente se sugirió que los modelos de aprendizaje automático pueden ser aplicados exitosamente para encontrar la mejor heurística para cada ontología específica.

En este trabajo se presenta un proceso a detalle para construir una serie de modelos de aprendizaje automático y encontrar una mejora en la predicción de la configuración de prioridad para cada ontología. El modelo propuesto en este trabajo utiliza menos características que el mostrado previamente y crea un modelo más sencillo con una precisión similar o mejor en la clasificación resultante.

Contents

List of Figures

List of Tables

1. INTRODUCTION

This work shows the implementation of a series of machine learning models that predict the best heuristic for reasoning a given ontology with the *ToDo List architecture* implemented in FaCt++. This introductory chapter explains the problem and importance of selecting the most suitable heuristic when reasoning an ontology with FaCt++ and how the proposal of this work can help to solve it.

1.1 Justification

Tsarkov and Horrocks introduced the *ToDo List architecture* for FaCt++, which it is implemented as a set of queues where every queue is dedicated to a type of expansion rule of the tableaux algorithm (e.g. $\sqcap$ and $\exists$) and a priority is assigned to the queue [1]. The tableaux algorithm takes an entry of a non-empty queue with the highest priority and handles it according to its expansion rules. Tsarkov and Horrocks stated in their work two relevant aspects of this architecture:

- Selecting a correct set of priorities for the queues is paramount on the reasoner's performance.

- There is not a single set of priorities that works the better for all ontologies.

Later Tsarkov and Horrocks showed a series of heuristics that lead them to set the default priority configuration in FaCt++ but stated that it might not be the best heuristic for all ontologies [2]. They concluded that choosing a suitable heuristic becomes of critical importance for the performance of the reasoning task.

In a later dissertation, Mehri *et al.* [3] found that by applying a machine learning model for selecting a heuristic order, reasoning tasks in JFact speed up by two orders of magnitude when compared to the worst order. JFact is a Java implementation of the FaCt++ reasoner, and they share the ToDo List architecture [4].

The work by Mehri *et al.* presents the techniques used to build the machine learning model and what considerations they took to collect and process the data from a corpus of ontologies. They concluded the paper showing a benchmark for a sample of the ontologies, with the default priority configuration and the one proposed by their machine learning model.

Based on the conclusions showed in the previous work it is proposed that implementing a machine learning model to predict the better heuristic order for FaCt++ can improve the performance of reasoning tasks and prevent timeouts. The proposal of this work is to implement a machine learning model to predict a suitable priority configuration for FaCt++ reasoner to improve the performance of reasoning tasks and prevent timeouts.

1.2 Problem

The semantic reasoner FaCt++ uses the TODO architecture [2] to process the different operators that can be found during reasoning, the configuration of the TODO architecture, i.e. the order on which the operators are handled, impacts the performance of the reasoner directly.

Tsarkov and Horrocks proved that there is not a unique priority configuration that ensure a good performance for all the ontologies [2], based on heuristics and an empirical experiment the authors set a priority configuration by default that worked well with most of the analyzed ontologies.

However there are some ontologies that when reasoned using the default priority configuration ends with timeout. An evidence of this scenario can be seen in the results of the Ontology reasoner evaluation competition (ORE) [5], where 76% of the reasoning errors were caused by timeouts, see table 1.1.

Track	Correctly evaluated	Total Errors	Timeouts
OWL DL consistency	276 of 306	30	16
OWL DL classification	200 of 306	106	87
OWL DL realization	172 of 264	92	58
OWL EL consistency	270 of 298	28	22
OWL EL classification	244 of 298	54	51
OWL EL realization	79 of 109	30	27

Table 1.1: Errors and timeouts by FaCt++ in ORE 2015.

The main cause of the poor performance for certain ontologies is the order on which the operators are processed by the tableaux algorithm, that's why the TODO architecture was introduced in FaCt++. Although FaCt++ has the ability to use a different priority configuration, there is no an easy way to know which priority configuration to use instead.

Even with the possibility to change the priority configuration, it is not guaranteed that editors of ontologies had the tools or knowledge to find a suitable replacement.

1.3 Hypothesis

Knowing how the structure of an ontology impacts the performance of the tableaux algorithm is a complicated task. To solve this problem it is suggest to apply a machine learning model that takes into consideration the main characteristics of ontologies and propose suitable priority configurations.

The proposal is to use the ORE ontology corpus [6] to train a machine learning classifier that recommends a priority configuration for a given ontology.

To test the improvement in performance, a sample of the ORE corpus and a snapshot of ontologies from Bioportal [7] will be used to gather an updated benchmark on the current version of FaCt++ [2]. After building the machine learning model that predicts a prioritization configuration, a new benchmark will be constructed for comparison.

It is expected that predicting prioritization configurations based on the ontology to be reasoned will show improved results in the benchmark and completion of reasoning tasks will be increased.

1.4 Objective

After building an initial benchmark of the performance of FaCt++, the objective of this work is to increase the number of correctly evaluated ontologies by means of proposing suitable priority expansion configurations for the FaCt++ reasoner.

The specific objectives this work achieves are:

- Extract and report common features of OWL ontologies useful for training machine learning models.
- Build a machine learning model to choose suitable priority expansion configuration for the tasks queues in FaCt++.

1.5 Contribution

Create machine learning models that can suggest a suitable priority configuration for FaCt++ to reason a given ontology, with the intention of preventing long reasoning times that can be classified as timeouts. By preventing reasoning timeouts, the experience of ontology editors can be improved.

Applying machine learning models and the work done with the data to prepare it for classification, will help to find the ontology characteristics that influence the performance of the tableaux algorithm, and therefore the performance of FaCt++. Knowing these relevant characteristics will help for future work on this topic.

2. PRELIMINARIES AND THEORETICAL BACKGROUND

This chapter is an overview of the relevant theory and technologies that will help to understand this work. This background starts with the concept of formal logic and the search of efficient automated reasoning. Following the concepts of ontology and semantic reasoner will be introduced.

This chapter will also explore the common use cases for ontologies at the time of writing this work and it will expose the importance of having a reasoner with good performance.

Finally it will present an overview of machine learning concepts and the specific techniques that are used during this work.

2.1 Ontologies and automated reasoning

Ontologies are a formal representation of knowledge that have been adopted recently by different areas of research including biology and health sciences [8]. The adoption of these technologies by the W3C for the implementation of the semantic web [9] has caused a new interest in ontologies and automated reasoning.

Ontologies have a focus on modeling tractable reasoning. For instance, satisfiability allows one to determine whether the definition of a concept is not contradictory within the ontology. Automated reasoning studies algorithms that can infer implicit knowledge from the explicitly represented knowledge in an ontology [10].

2.1.1 Formal logic and reasoning

Logic is the study of the rules of inference, that allows to follow a conclusion from a set of propositions. Although the scope and limits of logic are inexact and sometimes controversial, some examples might help to understand what logic is.

It is attributed to Aristotle the introduction of syllogism [11], which is a logical argument that consist of inferring a conclusion from exactly two propositions which are assumed to be true. see Fig. 2.1.

Another scenario where logic is involved is in the study of fallacies. A fallacy [12] is the incorrect use of reasoning, where the logical argument is incorrectly constructed following to invalid conclusions. It is common the use of fallacies during debates and discussions, in this case fallacies have clear classifications.

This work will focus on symbolic logic, a set of logical systems which are subject of study in the present. This logical systems make use of operations and quantified variables to create a more general use of logical reasoning.

All men are mortal.

Socrates was a man.

Therefore, Socrates was mortal.

Fig. 2.1: Valid syllogism.

It is important to make a distinction between a valid argument and a sound argument. The first one means that a logical arguments follows the rules of inference in a valid way, but it does not mean that the propositions are true. A sound argument means that is a valid argument and that propositions are true.

The previous argument about Socrates being a mortal man, is a valid and sound argument. But the syllogism in Fig. 2.2, though a valid argument is not sound, since the second proposition is not true.

All bats can fly.

Socrates is a bat.

Therefore, Socrates can fly.

Fig. 2.2: Valid but not sound syllogism.

It is trivial for a person to determine if the previous syllogisms are valid, both consist of two propositions. But arguments of hundreds or thousands of propositions can be formed in an intricate way, which is non-viable for a human to validate.

With that problem in mind and the introduction of computational systems, the interest of building an automatic reasoning system surged. Several approaches have been tested for achieving this purpose, from focusing in algorithms to data structures, and from networks to description logic.

This work will focus on description logic [13] as the way to achieve automated reasoning.

2.1.2 Ontologies and reasoners

An ontology is an explicit specification of a conceptualisation [14], the usual way to represent an ontology is using an ontology language such as web ontology language (OWL), then an ontology takes the form of an extensible markup language (XML) document that explicitly states the concepts, their properties and their relationships of a domain of knowledge.

Ontologies as well as formal logic support operations, e.g. conjunction, disjunction, and others. The more operations an ontology supports, it is said that it has greater expressiveness. Expressiveness is directly related to the complexity of reasoning on an ontology, the more expressive the language, the harder the reasoning [15].

Reasoning is what it has been shown with the previous syllogism about Socrates: find the validity of a logic argument. The basic reasoning task in an ontology is sumbsuption. Determining subsumption is the problem of checking whether the concept denoted by D (the subsumer) is considered more general than the one denoted by C (the subsumee)[13], in other words subsumption checks if the first concept is a subset of the second one.

A key reasoning task is satisfiability. When an ontology is being constructed, concepts and relations are introduced, during the process is important to see if the new concept makes sense. Logically a concept makes sense if there is an interpretation of the ontology that denotes a non-empty set for the concept. E.g. An ontology declares that pizza is food with dough, tomato sauce and cheese, a new concept claiming that a pizza is also food with dough, cheese but another kind of sauce will not be considered a pizza according to this ontology, that concept is not satisfiable.

Finding satisfiability with such a simple ontology is trivial, but ontologies are formed by thousands of propositions sometimes described in an intricate manner and with additional restrictions. For that case automated reasoning is necessary to reason over an ontology.

2.1.3 Ontologies use cases

In recent years the semantic web has been proposed as a way to enrich the content on the web with description and semantic relations that allow automated agents to understand and process in a broader way the public content of the web [9]. A clear example is to give semantic meaning to hours and dates of events, such a concert, agents can process unequivocally what the hours mean considering different formats and time zones and use the data in a more accurate manner.

The semantic web makes use of ontologies to describe concepts and relations and establish a formal system to let the agents process semantic information.

Ontologies are also used in biology and other health sciences, ontologies are being used to classify experimental data, centralize concepts and reason about organs, tissues and cells [8].

The complexity of these knowledge domains is such that a human can not process all the concepts at once. Using an ontology is paramount to achieve the formalization of this domain. Reasoners play a big role on formalizing the domains of biology, with the complexity of the ontology reasoning is important to check the validity of the ontology while working on it.

2.1.4 Relevance of reasoner performance

It is fair to state that the role of a reasoner is important while working with ontologies. Ontologies are created collaboratively, one contributor could introduce a logical contradiction without noticing, reasoners are used to test the validity of the whole ontology even after minor changes.

Reasoners are a working tool, that are constantly used during the creation and modification of an ontology, in the same way a compiler is used during coding to check for syntax errors.

Therefore it can be concluded the importance of reasoners being as fast as possible, since they are part of the daily process and not simply a tool that is run at the end of the whole edition effort.

2.1.5 Priority queues architecture in FaCt++

The most widely used technique to reason over ontologies is the method of analytic tableaux [16]. Tableaux is an algorithm that describes how to form a tree to detect logical contradictions using a set of rules that applies to logical operators [17].

Tableaux builds a tree where each node is a logical proposition. The set of rules showed in formulas (2.2, 2.3, 2.4, 2.5, 2.6 and 2.7) explains how to process a node and split it into simpler components. The tree becomes larger by splitting nodes. The idea is to find two nodes that causes an obvious logical contradiction, which marks the end of the algorithm.

There is not an explicit order to apply the different rules and can be easily intuited that the order affects performance, e.g. a common heuristic is to apply rules that creates branches once all other rules have been applied. Let proposition 2.1 be a logical proposition with P and Q as premises, to prove if it is true or false, the tableaux algorithm can be used.

$$\forall x Px \vee \forall x Qx \Rightarrow \forall x(Px \vee Qx) \tag{2.1}$$

Proposition (2.1) includes a non-deterministic operator $\vee$, i.e. causes branching. The tableaux algorithm could assign a low priority to $\vee$ and branch later, like in formula 2.3, or it could assign a high priority and branch sooner, like in formula 2.4. Note that branching sooner causes a more extensive tree.

$$\frac{\neg\neg X}{X} \tag{2.2}$$

$$\frac{X \wedge Y}{\begin{array}{c} X \\ Y \end{array}} \qquad \frac{\neg(X \wedge Y)}{\begin{array}{cc} \diagup & \diagdown \\ \neg X & \neg Y \end{array}} \tag{2.3}$$

$$\frac{X \vee Y}{\begin{array}{cc} \diagup & \diagdown \\ X & Y \end{array}} \qquad \frac{\neg(X \vee Y)}{\begin{array}{c} \neg X \\ \neg Y \end{array}} \tag{2.4}$$

$$\frac{X \Rightarrow Y}{\begin{array}{cc} \diagup & \diagdown \\ \neg X & Y \end{array}} \qquad \frac{\neg(X \Rightarrow Y)}{\begin{array}{c} X \\ Y \end{array}} \tag{2.5}$$

$$\frac{(\forall x)Px}{Pa} \qquad \frac{\neg(\forall x)Px}{\neg Pa} \quad \text{for fresh a} \tag{2.6}$$

$$\frac{(\exists x)Px}{Pa} \quad \text{for fresh a} \qquad \frac{\neg(\exists x)Px}{\neg Pa} \tag{2.7}$$

Ontologies are a subset of first order logic [16] therefore can be expressed and treated using the rules showed in formulas (2.2 - 2.7) and tableaux trees will look like a much more complex tree than formula 2.3 but using the same rules.

It is important to note that Tsarkov and Horrocks proved in [1] that there is not a single priority configuration that is suitable to reason all kind of ontologies.

The order of priorities in FaCt++ is expressed as a set of 7 digits that define the order of the following operators: id, and, or, exists, for all, less than or equal, greater than or equal. The string is known as *IAOEFLG* as shown in table 2.1 and the possible values are strings of 7 digits from 0 to 6 specifying which operator has a higher priority.

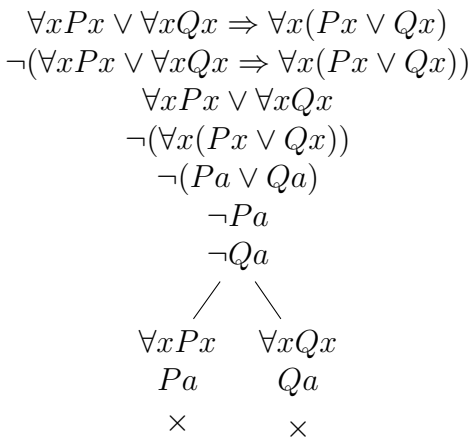


Fig. 2.3: Apply branching at the end.

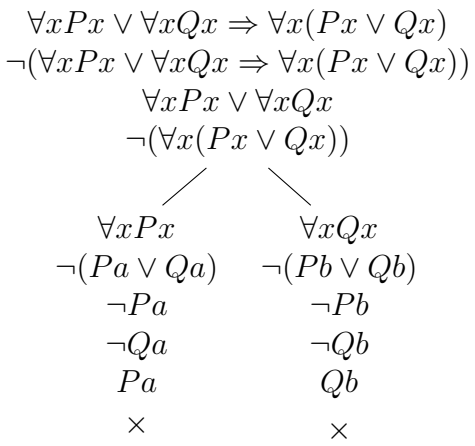


Fig. 2.4: Apply branching sooner.

Serve as an example the default configuration set by Tsarkov and Horrocks for FaCt++: `1263005`. `For all` and `less than or equal` have the highest priority and `or` has the lowest priority, see table 2.2 for a better visualization.

Since there are seven different positions and they can share priorities, it can be 7^7 possible priority configurations. With that number of label classifications is not viable to build models for everything, to make it more feasible some adjustment are suggested in [3]. Basically, operators are grouped according to its behavior in the tableaux algorithm. Id and $\sqcap$ are grouped together since they do not cause non-determinism or expansion (creation of new nodes). Operators $\geq$ and $\exists$ are grouped together because they cause expansion, $\leq$ and $\sqcup$ are grouped together since they cause non determinism, and finally $\forall$ is by itself since it is deterministic and it must be applied to all its successors.

During the development of this work it was found that the priority configuration where every operator has the same priority configuration (`0000000`) is recommended for a neg-

I	Id
A	$\sqcap$
O	$\sqcup$
E	$\exists$
F	$\forall$
L	$\leq$
G	$\geq$

Table 2.1: Different priority queues in FaCt++.

F, L	$\forall$, $\leq$	0
I	Id	1
A	$\sqcap$	2
E	$\exists$	3
G	$\geq$	5
O	$\sqcup$	6

Table 2.2: Ordered default configuration for FaCt++.

ligible number on ontologies, therefore that priority configuration is not considered. Also the default priority configuration suggested by Tsarkov and Horrocks [1] is included.

After these considerations, the seven priority configurations that are going to be evaluated in this work are shown in table 2.3.

Priority configuration	Order of operators			
0012312	$\cup \leq$	$\exists \geq$	$\forall$	
0013213	$\cup \leq$	$\forall$	$\exists \geq$	
0021321	$\exists \geq$	$\cup \leq$	$\forall$	
0023123	$\forall$	$\cup \leq$	$\exists \geq$	
0031231	$\exists \geq$	$\forall$	$\cup \leq$	
0032132	$\forall$	$\exists \geq$	$\cup \leq$	
1263005	$\forall \leq$	$\exists$	$\geq$	$\cup$

Table 2.3: Priority configurations in the benchmark.

In this work those candidate priority configurations will serve as labels in a series of binary classification models. The models will be used to suggest the best priority configuration for a given ontology.

2.2 Supervised classification

Supervised learning is the machine learning task of learning a function that maps an input to an output, based on example input-output pairs [18]. This process needs a set of data

which includes a label or class for every sample of the input, it is called supervised in the sense that the expected output has been previously defined.

Classification is the task of finding a function to map input to a specific discrete label, e.g. this is a cat, or a dog or a bird. Supervised classification needs a training data set that includes n number of samples with x features or predictors, and a label with the possible values previously assigned.

In this work three kind of classification models will be tested: Support vector machines K-nearest neighbors and Decision trees.

The support vector machine is a generalization of a simple and intuitive classifier called the maximal margin classifier. Though maximal margin classifier is an elegant classifier it can not be applied to most of the data sets since the classes must be separable by a linear boundary. Support Vector Machines is a further extension in order to accommodate no linear boundaries. Support vector machines are intended for binary classification and the basic idea is to separate the feature p-dimensional space with a p-1 dimensional hyperplane [19], e.g. a simple feature plane of two features can be represented by the cartesian plane and the corresponding hyperplane is a one dimensional line, samples on one side of the plane correspond to label α and on the other side samples labeled β. For non linear separable classes Support vector machines enlarge the feature space using kernels, in this work linear and radial basis function kernels are considered.

K-nearest neighbors are memory-based classifiers and require no model to be fit. Given a query point k nearest points closest in distance are considered as training points, the classification is the majority vote among the k neighbors [20]. K-nearest neighbors is a non-parametric method, hence it does not make assumptions about the form of the target function [19]. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

Decision trees segments the feature space into non-overlapping regions. The response of the observation is the mode of the samples in every region [19]. Trees are easy to explain to people, even non-technical people. Trees can be easily displayed graphically but unfortunately they tend to lack the accuracy of other models. Random forest are a model based on decision trees that have shown an improvement in accuracy [19], the idea is to prevent some features to be considered in deeper levels of the tree causing a decorrelation of predictors and increasing the chance of create trees with a higher variance.

2.2.1 Feature engineering

With the intention of increasing the performance of certain classification models, mainly support vector machines, some feature engineering techniques were applied to the training data set. The theory to understand the basics of each technique is treated in this section.

Models such as k-nearest neighbors and support vector machines are susceptible to problems found in continuous features, for example a skewed distribution and outliers. To prevent a skewed distribution, the training data set will be transformed using Box-Cox transformation [21]. Box-Cos transformation originally intended as a transformation of a model's outcome uses maximum likelihood estimation to find a transformation parameter λ in the equation 2.5.

$$x^* = \begin{cases} \frac{x^\lambda - 1}{\lambda \tilde{x}^{\lambda-1}}, & \lambda \neq 0 \\ \tilde{x} \log x, & \lambda = 0 \end{cases}$$

Fig. 2.5: Box-Cox transformation equation.

where $\tilde{x}$ is the geometric mean of the predictor data. Because the parameter of interest is in the exponent, this type of transformation is called power transformation [21].

Decision trees have an implicit feature selection technique built in the way a tree works. As it was explained before decision trees look for the most useful features that can help to predict an outcome, therefore features that are not relevant are implicitly discarded, or not taken into consideration. It can be said that there is an implicit feature selection. This kind of selection is faster as is part of the model itself.

A different feature selection technique is based on the calculation of variance for each feature in the training data set. A threshold is selected and features whose variance is not over that threshold are discarded [21], intuitively features with similar values and that do not offer a clear distinction between classes are discarded.

Although trees discard irrelevant features, variance feature selection is applied to the training data set to improve the performance of support vector machine models.

Oversampling is a technique to adjust the distribution of label in a data set. Using statistic methods as SMOTE new samples are created to balance a otherwise unbalanced set [21]. A unbalanced set is one that have significantly more samples of a given label that another one, e.g. a set with two labels with 100 samples for label α and 30 samples for label β. An unbalanced data set affects a classification model, favoring the majority label.

2.3 Conclusions

In this chapter it was exposed how the tableaux algorithm works in essence, and how the different logical operators impact in the performance of reasoning tasks. It is important to understand what the reasoner is doing during reasoning, to effectively improve its performance.

With this in mind it was also shown how the reasoner is a significant part of the day to day workflow of an ontology editor, and how poor reasoning performance impacts daily work.

The TODO list architecture implemented by FaCt++ provides a way to improve performance but it is dependant of selecting a proper priority configuration. There is where machine learning can help to recommend a suitable priority configuration by doing an analysis on existing ontologies and their performance using candidate configurations.

3. METHODOLOGICAL DEVELOPMENT

3.1 Data procurement

With the adoption of web technologies for the development and distribution of ontologies, there have been an increase in the number of ontologies. The OWL format has allowed the emergence of tools that makes easy to create and test ontologies, moreover the standardization of the format has made possible team collaboration.

The very nature of ontologies to express relationships between concepts encourage ontology editors to search for existing ontologies than can be reused as a base for a new ontology. Many ontologies depends on others to be defined.

Consequently there has been a need to facilitate publishing and sharing ontologies. None of this is a coincidence, Berners-Lee had it in mind since the conception of a semantic web, where ontologies and semantic markup could easily be edited by everyone [9], such as is done with current web technologies.

The need of a repository of library of ontologies is consequence of these previously exposed scenarios. Ontology libraries are the systems that collect ontologies from different sources and facilitate the tasks of finding, exploring, and using these ontologies [22].

For testing the performance of FaCt++ a corpora of ontologies that challenge the reasoner capabilities is desired. It is not enough to have simple ontologies that FaCt++ will process quickly no matter the priority configuration set, to have meaningful results in a benchmark for different priority configurations, big complex ontologies are needed.

The biggest consulted library was the corpora of the OWL reasoner evaluation competition. The OWL Reasoner Evaluation is an annual competition that tests the performance of different OWL2 reasoners [5][6], among them is FaCt++. The event consists of a competition among reasoners completing three reasoning tasks on a corpora of OWL2 ontologies. The corpora consist of 1,920 ontologies, it is a sampling of 3 sources:

- A January 2015 snapshot of Bioportal, with 330 biomedical ontologies.
- The Oxford Ontology Library, with 792 ontologies.
- A MOWLCorp 2014 snapshot of a web crawl.

For more up to date ontologies a June 2021 snapshot of Bioportal [7] is used as well, 627 ontologies were collected from this site. Bioportal has a collection of ontologies related to biomedical topics, it can be found ontologies about allergies, infectious diseases, anatomy, proteins and others. Ontologies in Bioportal have different sizes, some ontologies have half a million classes meanwhile others have just a couple dozen of classes.

The total 2,547 ontologies will be processed, measured and evaluated.

3.2 Data preparation

For building the classification models, features and labels are needed. Features describe the individuals, in this case ontologies. Labels in the case of this work is if it is recommended or not to use a given priority configuration. The features extracted from ontologies are based on the ones extracted in [3].

In this work features can be classified in two categories: counters and ratios. Counters count the number of appearances of a given OWL tag, ratios are calculated using a simple formula that will be discussed later.

The possible tags for OWL are well documented in [23], to ease the extraction of the tags a script was written as part of this work. Tags outside the OWL specification are out of scope, since it is not viable to know how all of them impact the tableaux algorithm and anyway not many custom tags are defined in the corpus.

Class	Datatype	DatatypeProperty
NamedIndividual	ObjectProperty	Restriction
allValuesFrom	assertionProperty	cardinality
complementOf	disjointUnionOf	disjointWith
distinctMembers	domain	intersectionOf
inverseOf	maxCardinality	maxQualifiedCardinality
members	minCardinality	minQualifiedCardinality
oneOf	propertyChainAxiom	propertyDisjointWith
qualifiedCardinality	range	someValuesFrom
subClassOf	subPropertyOf	unionOf
withRestrictions		

Table 3.1: List of counter features.

axioms	classes_with_existential
classes_with_universal	classes_with_subclasses

Table 3.2: Additional counter features.

OWL ontologies are parsed as XML documents and tags are counted. With this first step counter features are extracted, see table 3.1. Four additional counter features were extracted, total number of axioms, number of classes with existential, universal and subclasses axioms inside them, table 3.2.

Ratio features represent a proportion of certain operators over others. The operators are not arbitrary, in the TODO architecture of FaCt++ introduced in [2], these operators represent the available priority queues, see table 2.1.

The ratio features are calculated using the formulas 3.1, 3.2, 3.3 and 3.4, e.g. formula 3.3 for `ratio_disjunction`, is the ratio between the number of axioms that apply disjunc-

$$ratio\_min\_universal = \frac{N_{\leq} + N_{\forall}}{N_{\leq} + N_{\forall} + N_{\geq} + N_{\exists} + N_{\cup} + N_{\cap}} \tag{3.1}$$

$$ratio\_max\_existential = \frac{N_{\geq} + N_{\exists}}{N_{\leq} + N_{\forall} + N_{\geq} + N_{\exists} + N_{\cup} + N_{\cap}} \tag{3.2}$$

$$ratio\_disjunction = \frac{N_{\cup}}{N_{\leq} + N_{\forall} + N_{\geq} + N_{\exists} + N_{\cup} + N_{\cap}} \tag{3.3}$$

$$ratio\_conjunction = \frac{N_{\cap}}{N_{\leq} + N_{\forall} + N_{\geq} + N_{\exists} + N_{\cup} + N_{\cap}} \tag{3.4}$$

tion, and the sum of all axioms that apply minimum cardinality, universal qualifications, maximal cardinality, existential qualifications, union and disjunction. These ratios are included to represent the proportion between the operators that have an impact in the Tableaux algorithm.

With these 4 ratio features there is a total of 39 features, these features are extracted for all ontologies in the corpus and exported as a csv file.

To create the classification labels it was needed to measure the reasoning time for every ontology in the corpora. With that purpose it was used a machine with the following specifications:

- Kernel: 5.4.0-48-generic x86_64.
- Compiler: gcc v: 9.3.0.
- Linux Distribution: Ubuntu 20.04 LTS (Focal Fossa).
- CPU topology: Dual Core model Intel Core i5-2450M.
- CPU speed: 2190 MHz min/max: 800/3100 MHz Core speeds (MHz): 1: 1570 2: 1629 3: 1398 4: 2737.

The library to load a OWL ontology into the reasoner was owl_cpp v.0.3.3 [24] with some minor changes for working with the used version of FaCt++, these changes do not affect performance, but are for adjust the syntax to a more recent C++ standard. Since owl_cpp only supports RDF/XML it was needed to convert the ontologies to the RDF/XML syntax to process them, for this purpose a utility named robot was used, which is a tool that helps with common tasks when working with ontologies [25]. Both owl_cpp and FaCt++ were compiled in the same machine described above.

Every ontology was reasoned with FaCt++ in satisfiability service. And for every configuration of a set of 7 priority configurations, see table 3.3. Every priority configuration was

Priority configuration	Order of operators			
0012312	$\cup \leq$	$\exists \geq$	$\forall$	
0013213	$\cup \leq$	$\forall$	$\exists \geq$	
0021321	$\exists \geq$	$\cup \leq$	$\forall$	
0023123	$\forall$	$\cup \leq$	$\exists \geq$	
0031231	$\exists \geq$	$\forall$	$\cup \leq$	
0032132	$\forall$	$\exists \geq$	$\cup \leq$	
1263005	$\forall \leq$	$\exists$	$\geq$	$\cup$

Table 3.3: Priority configurations in the benchmark.

run 5 times using a utility called `hyperfine` [26], the machine was not running anything else but the benchmark.

A table with the times per priority configuration was obtained. For training purposes only ontologies which presents at least one time out in one of the configurations were considered. Most of the ontologies do not have large differences among their times with different priority configurations, for those ontologies is not required to propose an optimal priority configuration since any of them would work.

After processing the results of the benchmark, classification labels were obtained. The priority configuration is considered as not recommended if it takes 2 or more seconds than the fastest priority configuration or if after 10 minutes reasoning has not finished, what it is called a timeout. The ontology is considered as recommended otherwise. These timeouts thresholds are based on what has been observed by Mehri as convenient and published in their work [3].

3.3 Building the classification models

Due to the small number of samples for training, 171 ontologies, it was decided that it is a better approach to build multiple binary classifiers instead of a single multi label classifier, as it is suggested in the work published by Mehri [3].

Ontologies have a label for each priority configuration, each of these labels have a binary value `1` if the priority configuration is recommended to use it to reason the ontology, or `0` otherwise. Labels are based on the results of the performance benchmark done as part of this work.

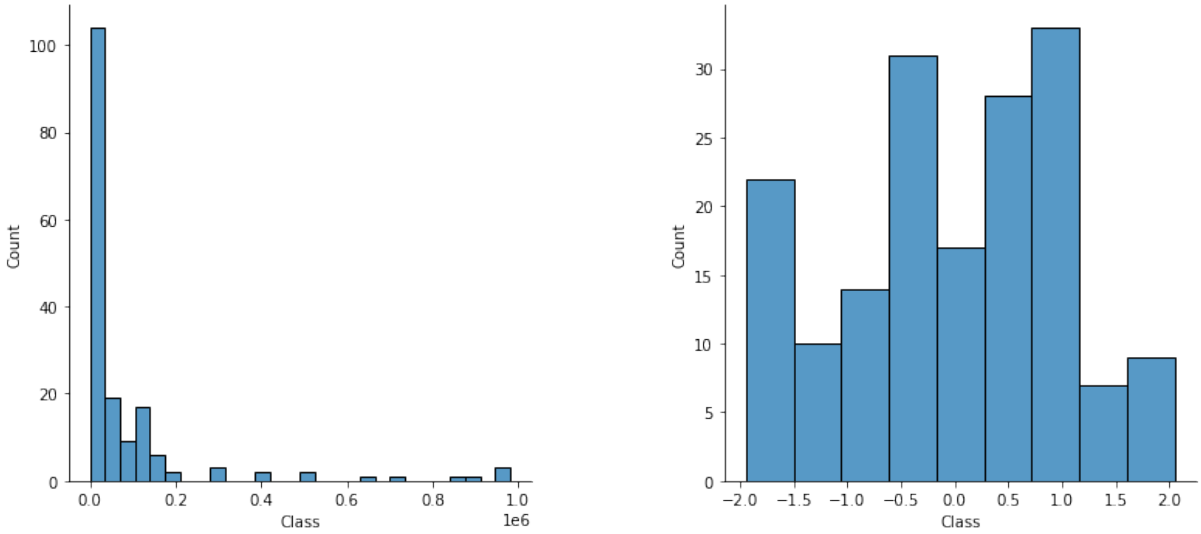
There are 7 priority configurations being evaluated, hence 7 classifications models will be trained. First for each class we build a data set with all the features extracted from the ontologies plus the label that represents the priority configuration.

Features can be cataloged in two kinds:

1. Counters are the number of occurrences of certain OWL tags. The scale is unknown, and

2. Ratios are calculated with a simple division. It is known the scale goes from 0 to 1.

Features that are on a continuous scale are subject to certain problems due to be on different scales, skewed distributions or outliers. Some models like trees are not affected by these problems but others like k-nearest neighbors and support vector machines are much more sensitive to predictors with skewed distributions or outliers [21].



(a) Not transformed.

(b) With Cox-Box transformation.

Fig. 3.1: Distribution of number of classes.

To help the classification algorithms counter features are transformed using the Box-Cox transformation to have a more gaussian distribution. Although is not guaranteed the improvement of the model after applying the transformation, a skewed distribution has harmful effect on models that utilize polynomial calculations such as linear models and support vector machines [21]. Figure 3.1 shows an example of distribution before and after Cox-Box transformation.

Counter features are OWL tags and, as previously noted, it is intended to extract all of the possible OWL tags based on standard documentation. This lead to the case that some tags are not present for any of the ontologies in the training data set. It is helpful to remove these features that do not contribute to the training of the models and decrease the number of features to train with.

A simple way to remove these features is to discard features with low variance. With this purpose a feature selection technique to remove features with low variance is applied. Again only counters features are considered for this selection technique, since the range of ratio features is [0, 1] by definition all of them would be discarded, this is not desired.

Priority configuration	Recommended	Not recommended
0012312	78	93
0013213	90	81
0021321	17	154
0023123	95	76
0031231	29	142
0032132	41	130
1263005	55	116

Table 3.4: Ratio of number of recommended to recommended.

For some priority configurations there is a bigger number of ontologies for which the configuration is not recommended, with the worst mismatch of 1:9 for priority configuration `0021321`, see table 3.4. This scenario impacts the classification models, obtaining a high accuracy but a low f1-score that takes precision and recall into account.

This phenomenon is known as class imbalance learning [27] and affects classification, intuitively the classification model favors the class with the larger number of samples. To fight imbalance learning new samples can be created. In this work new samples are created using the SMOTE technique [21].

Once the data has been prepared, four models are trained using the data set. The considered models are: support vector classification with linear and radial basis function kernels and decision tree and random forest classifiers. In [3] it is proposed that support vector machine classifiers are a very accurate model for this classification, for this reason both support vector classifiers are considered. Finally with the premise than most of the classification can be made using only ratio features with a simple threshold validation, both decision tree and random forest are tested.

Another advantage of using decision tree classifier is that the simplicity of the model allows to see what decisions are taken, e.g. a tree can be visualized with the considerations the model took to classify. This visualization can also help to further study the relevance of ontology features.

The results of training with the described data set and which models were finally selected as the most accurate is shown in the following chapter.

3.4 Conclusions

The accuracy of the classification models was broadly influenced by the quality and presentation of the training data. It was important to dedicate most of the effort to study and adapt the available data to prevent known issues caused by the characteristics of the data, e.g. a skewed distribution.

It was important to build mechanisms that could be easily used in an iterative experimentation, with this in mind preprocessing of data was split in reusable procedures, without side effects and easily repeatable. Another relevant aspect was the application of oversampling techniques, proving the need of a larger corpus of data.

Lastly, although no new classification models were implemented, it was significant to take time to understand the inner mechanism of the models to improve the training data for its use.

4. RESULTS AND DISCUSSION

4.1 Results

To build the binary classifiers the corpus from ORE 2015 was used. Not all the ontologies in the corpus were used, after performing the benchmark it was found that most of the ontologies did not show a relevant difference between times when using different priority configurations, it is then considered that the order of logical operators does not have an impact on performance.

Ontologies that presents at least one timeout in any priority configuration are considered to be part of the training set. The intention is to recommend a suitable priority configuration that do not cause a timeout.

Finally, ontologies with timeouts for all of the evaluated priority configurations are also discarded. The timeout used for this work is 15 minutes.

Class	Best model	Accuracy	F1-score
0012312	Random Forest	0.885	0.840
0013213	SVC RBF	0.678	0.696
0021321	SVC RBF	0.639	0.302
0023123	Random Forest	0.684	0.719
0031231	Random Forest	0.894	0.733
0032132	Random Forest	0.905	0.799
1263005	Random Forest	0.888	0.845

Table 4.1: Best model for every configuration.

After these considerations the training data set consists of 171 ontologies. This training data set was used to build seven binary classifiers. Models are cross validated using k-fold with 5 partitions, and two metrics: accuracy and f1-score are gathered. The results of training the models are presented in table 4.1.

To evaluate the accuracy of the classifiers, 15 ontologies from bioportal are used. These ontologies are evaluated using the same benchmark that was used for the training data set and the same considerations are observed to obtain the final ontologies used for evaluation.

Algorithm 1 Recommend a priority configuration.

1: **procedure** RECOMMENDPC
2: Order binary classifiers based on F1-Score.
3: *select*:
4: Select the classifier with highest F1-Score that has not been marked as *used*.
5: Predict the recommendation using the selected classifier.
6: Report if the priority configuration is recommended.
7: Mark as *used* the selected classifier.
8: **goto** *select*.

Recommendation of the priority configuration is done using the procedure described in Algorithm 1. It is important to remember that only one priority configuration is required to work with the ontology, hence it is relevant which is the first priority configuration recommended by the classifiers.

ore_ont_16444.owl	Real	Predicted
0012312	True	True
0013213	True	False
0021321	False	False
0023123	True	True
0031231	False	False
0032132	False	False
1263005	False	False

Table 4.2: Predicting order for ontology.

For example for ontology `ore_ont_16444.owl` from ORE2015 corpus [6], the prediction process will start with the classifier with highest f1score which will suggest to use the priority configuration or do not use it for reasoning. See table 4.2.

With binary classification four scenarios are possible, first the best scenarios:

1. The priority configuration is suitable and the classifier marks it as suitable as well. (True Positive).
2. The priority configuration is not suitable and the classifier marks it as not suitable as well. (True Negative).

The following cases are classification errors:

1. The priority classification is suitable but the classifier does not recommend it. This classification is still an error but it can be tolerable because it is expected than the following classifier can suggest a suitable configuration. (False Negative).
2. The priority classification is not suitable but the classifier recommends it. This is the worst case scenario since causes the reasoner to take the most time reasoning even causing a timeout. (False Positive).

The recommendation results for the 15 ontologies from bioportal are shown in table 4.3. First recommended PC is the first priority configuration that is recommended to be used to evaluate the given ontology, second recommended PC is the second recommended priority configuration by the classifiers. For the 73% of the evaluation ontologies the classifiers recommended a suitable priority configuration. Since the algorithm can recommend several priority configurations if after a bad prediction it is used a second recommendation, 93% of the evaluation ontologies are successfully evaluated.

Ontology	Real PC	First recommended PC	Result	Second recommendation
aero	1263005	1263005	Good	
allergydetector	1263005	1263005	Good	
bipom-ecmet-int	1263005	1263005	Good	
bt	1263005	0031231	Bad	Good
cmpo	0031231	0031231	Good	
fb-cu	0031231	0031231	Good	
flopo	0031231	0012312	Bad	Good
hoipo	0013213	None	Bad	Bad
ico	0031231	0031231	Good	
mhc	1263005	1263005	Good	
neurofma	0012312	0012312	Good	
obi-fged	0031231	0031231	Good	
ontokbfc	0031231	0031231	Good	
plantso	0031231	0031231	Good	
ppo	0012312	1263005	Bad	Good
		Accuracy	73%	93%

Table 4.3: Results of recommended a priority configuration.

4.1.1 Decreasing timeouts

A sample from ORE 2015 corpus [6] of 20 ontologies excluded from training the resulting models was used to measure the effectiveness of decreasing timeouts. The criteria to select the ontologies for this evaluation was that the default configuration resulted in a timeout, hence requiring a different priority configuration to reason over given ontology in an acceptable time.

Using algorithm 1 a priority configuration was suggested for every one of the 20 ontologies, resulting in correctly predicting a suitable priority configuration other than the default for 70% of the ontologies.

Meaning that 70% of the timeouts could be prevented by using the models presented on this work.

4.2 Conclusions

Applying the recommendations to select a suitable priority configuration to reason over a given ontology can prevent timeouts. It is important to remember that reasoning over an ontology is done periodically during the editing process, which can lead to a unusable creation environment.

During the development of this work it was observed that there is no many cases where ontology evaluation lead to a timeout, which in a way hinders statistical analysis like the one presented in this work. With proliferation of semantic web it is expected that new ontologies will be created which could help to enrich the ecosystem of tools and statistical analysis around this topic.

5. CONCLUSIONS

During the development of this work it was found that the ontologies that ends with a timeout for a particular priority configuration are a special case. The most popular ontologies used in the semantic web, such as foaf and goodrelations, can be reasoned with any priority configuration without noticing a penalty in performance.

The heuristic that there is a set of ontology features that help to predict what priority configuration is the most suitable to reason it, it is correct in most of the cases. However, there are still some particular ontologies that are a problem to predict, such as the cellular microscopy phenotype ontology (CMPO) and the informed consent ontology (ICO) found in bioportal.

With the final utility created on this work to predict the most suitable priority configuration to be used by FaCt++, reasoning timeouts can be avoided. Avoiding timeouts will result in a better experience while designing ontologies.

5.1 Future work

Exploring other features that impacts reasoning performance is suggested, during the development of this work two options were noted:

- The order of the axioms itself can impact performance. It can be useful to explore the relation between the order on which the axioms are handled by the tableaux algorithm and the performance, and if there is an optimal order to reason an ontology.
- How the structure of the axioms impacts the performance of the reasoner, e.g. in this work was intended to capture how many classes are impacted by the use of existential quantifiers.

BIBLIOGRAPHY

[1] D. Tsarkov and I. Horrocks, "Ordering heuristics for description logic reasoning," in *IJCAI*, 2005, pp. 609–614.

[2] ——, "Fact++ description logic reasoner: System description," in *International joint conference on automated reasoning*, Springer, 2006, pp. 292–297.

[3] R. Mehri, V. Haarslev, and H. Chinaei, "Optimizing heuristics for tableau-based OWL reasoners," *arXiv:1810.06617 [cs]*, Oct. 24, 2018. arXiv: `1810.06617`. [Online]. Available: `http://arxiv.org/abs/1810.06617` (visited on 05/15/2020).

[4] I. Palmisano, *Owlcs/jfact*, original-date: 2013-09-06T21:17:34Z, Oct. 27, 2020. [Online]. Available: `https://github.com/owlcs/jfact` (visited on 02/09/2021).

[5] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, and A. Steigmiller, "The owl reasoner evaluation (ore) 2015 competition report," *Journal of Automated Reasoning*, vol. 59, no. 4, pp. 455–482, 2017.

[6] B. Parsia and N. Matentzoglu, *Ore 2015 reasoner competition corpus*, Jun. 15, 2015. [Online]. Available: `https://zenodo.org/record/18578` (visited on 02/09/2021).

[7] W. PL, N. NF, S. NH, A. PR, N. C, T. T, and M. MA, *Bioportal: Enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications.* 2011.

[8] J. Hastings, "Primer on ontologies," in *The gene ontology handbook*, Humana Press, New York, NY, 2017, pp. 3–13.

[9] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[10] F. Baader and U. Sattler, "An overview of tableau algorithms for description logics," *Studia Logica*, vol. 69, no. 1, pp. 5–40, 2001.

[11] G. Striker *et al.*, *Aristotle's Prior Analytics Book I: Translated with an Introduction and Commentary*. Oxford University Press, 2009.

[12] F. H. Van Eemeren, B. Garssen, and B. Meuffels, *Fallacies and judgments of reasonableness: Empirical research concerning the pragma-dialectical discussion rules*. Springer Science & Business Media, 2009, vol. 16.

[13] F. Baader, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, "THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications," p. 500,

[14] T. R. Gruber, "A translation approach to portable ontology specifications.," in *Knowledge Acquisition*, 1993, pp. 199–220.

[15] R. J. Brachman and H. J. Levesque, "The tractability of subsumption in frame-based description languages," in *AAAI*, vol. 84, 1984, pp. 34–37.

[16] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *An Introduction to Description Logic*. Cambridge University Press, 2017.

[17] R. M. Smullyan, *A Beginner's Guide to Mathematical Logic*. Dover Publications, 2014.

[18] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2002.

[19] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[20] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[21] M. Kuhn and K. Johnson, *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019.

[22] M. d'Aquin and N. F. Noy, "Where to publish and find ontologies? a survey of ontology libraries," *Journal of Web Semantics*, vol. 11, pp. 96–111, 2012.

[23] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, *et al.*, "Owl 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation*, vol. 27, no. 65, p. 159, 2009.

[24] M. K. Levin, A. Ruttenberg, A. M. Masci, and L. G. Cowell, "Owl-cpp, a c++ library for working with owl ontologies," in *2nd International Conference on Biomedical Ontology, ICBO 2011*, 2011, pp. 255–257.

[25] R. C. Jackson, J. P. Balhoff, E. Douglass, N. L. Harris, C. J. Mungall, and J. A. Overton, "Robot: A tool for automating ontology workflows," *BMC bioinformatics*, vol. 20, no. 1, pp. 1–10, 2019.

[26] D. Peter, *Hyperfine*. [Online]. Available: `https://github.com/sharkdp/hyperfine` (visited on 09/20/2021).

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.