An Efficient Framework for Automatic Algorithm Selection using Meta-Learning

by

Mohammed Elmahgiubi

A thesis
presented to
the University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Applied Science
in
Engineering

Guelph, Ontario, Canada

© Mohammed Elmahgiubi, August, 2016

ABSTRACT

An Efficient Framework for Automatic Algorithm Selection using Meta-Learning

Mohammed Elmahgiubi
University of Guelph, 2016

Advisor:
Dr Shawki Areibi, Dr Gary Grewal

With the unprecedented growth of Information and communication technology (ICT) industry, core networking devices become highly stringent elements of the network due to the increase of packet classification (PC) requirements. Although many PC algorithms with variable performances and capabilities are available, no single algorithm is guaranteed to outperform every other one in every case. This research provides a generic and efficient framework for algorithm selection using Meta-Learning and Artificial Neural Networks (ANN). The developed framework was tested in different scenarios comprising different PC algorithms with different performance measures. Using ANN as the learning model and 10-fold cross validation as the evaluation criteria, the framework was able to achieve an average accuracy of 92.5% on predicting the most suitable algorithm that maximizes classification speed for an unseen ruleset, and 88% when minimizing memory footprint on a larger set of algorithms using the same evaluation criteria.

Acknowledgements

I would like to express my gratitude to the main reasons that made this thesis possible, first and formost, Allah for giving me the will and the blessings needed. Second, to my parents, for without their support and love I would not be here and I know that I can never repay them for all their efforts. This thesis would not have gotten here without the constant guidance and patience of my advisors Dr Shawki Areibi and Dr Gary Grewal, their academic and administrative excellence and supervision made this research possible. Last but not least, I would like to pay my sincere gratitude to the University Of Guelph members for their invaluable support and blessings, especially Dr Omar Ahmed for his continuous help and guidance throughout my degree, and to to my fellow ISLAB labmates who made my stay pleasent and provided comfort when needed.

To

my family

whose love and encouragement helped accomplish this

thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The problem of algorithm selection has received wide attention in different research areas as it represents a generic challenge for different industries. The need for automatic algorithm selection exists in industries where different algorithms are available and present to solve a set of various tasks, producing varying results depending on the selection of the proper algorithm. In such environments, the choice among all available algorithms on a particular task becomes a challenging decision to make.

Meta-Learning [1] is a subfield of *Machine Learning (ML)* [2] [3] research that automates and enhances such selection *decision* based on progressive adaptive learning. This thesis focuses on utilizing Meta-Learning to address the automatic selection of algorithms via features extracted from the set of problems to be solved. Unlike traditional basic ML that uses accumulating experience on specific learning

task, in Meta-Learning, the learning experience is extended to include information and knowledge gathered on a *meta-level* of a specific task, and across multiple tasks, such that, it gathers knowledge on the general performance of algorithms throughout the whole learning process.

In this research, we used Meta-Learning and *Artificial Neural Networks (ANN)* [4] to develop a generic framework to address the algorithm selection challenge with application to packet classification in the area of computer networking. Given a set of existing algorithms, our approach is based on first evaluating the performance of these algorithms on a large set of problem instances *(datasets)*. In each case, the performance metrics for each *algorithm/dataset* pair are stored in a database. Next,

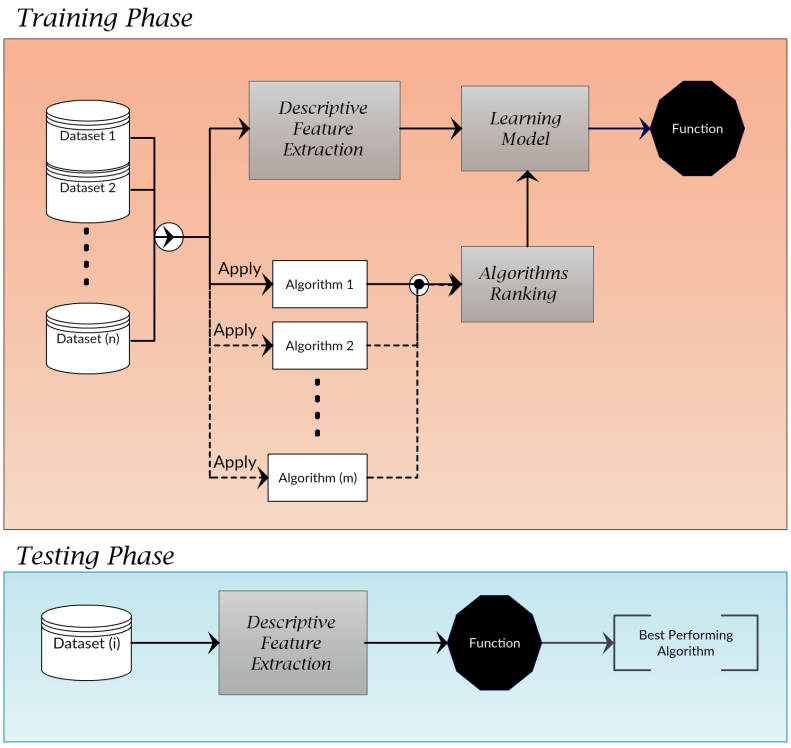

Figure 1.1: Overview of the framework

features related to each algorithm and its performance on the data are automatically generated and used to train an ANN-based classification model. Given a new dataset, this model (function) can then be used to efficiently predict the most appropriate algorithm that should be used to achieve the best performance. The reader can refer to Figure 1.1 for a high-level overview of the proposed framework.

1.2 Objectives and Motivation

In today's widely interconnected world, networking has become an essential technological backbone for most of our *information and communications technology (ICT)* applications. With the growing demand and ever ending advances in core networking devices to match user and application needs, more research is needed to introduce smarter intelligence in these devices to match this demand.

Networking devices have to make decisions in routing different data streams (packets) as per their required destination, application, and required networking resources. The automated selection of the proper algorithm to perform the task of packet classification to the proper flow within the networking performance constraints is an example of Meta-Learning challenge that motivated this research.

Given the above motivation for the research, the objectives of this thesis are to:

- Develop a generic Meta-Learning framework for automatic algorithm selection. The framework aims to aid the user in the process of selecting the most appropriate algorithm given a new unseen *dataset* instance.
- Apply and evaluate the generic framework to solve the *packet classification algorithm selection problem.*

- Develop insight of how different metrics and different evaluation criterion affect the performance of the framework.

The developed framework was tested in different scenarios comprising combinations of different packet classification performance measures and different algorithms. Promising results were obtained when applied to solve the problem of algorithm selection for packet classification using a collection of 108 datasets. Employing an ANN as the learning model and 10-fold cross validation as the evaluation criteria, the framework was able to achieve an average accuracy of 92.5% (100/108) on predicting the most suitable algorithm that maximizes packet classification speed for an unseen dataset. The framework also showed similar results when the packet classification objective changed to minimizing memory footprint and the number of algorithms possibilities increased, achieving an average accuracy of 88% (95/108) using the same learning model and the same evaluation criteria. The framework was extended to investigate a scenario where algorithm selection by itself is not sufficient. We extended the purpose of the framework to tackle the problem of algorithm's performance metrics estimation. We estimated the required memory consumption of two of the tested algorithms on any given dataset (ruleset). We were able to achieve acceptable initial results using the Root Mean Square Error (RMSE) measure, which will lead to further developments in this direction.

1.3 Contributions

The main contributions of this work are outlined as follows:

- The development of a generic, and modular framework for automatic algo-

rithm selection.

- Accurate algorithm selection predictions were observed when the framework was applied to selecting between different packet classification algorithms.
- Extending the framework objective beyond algorithm selection to estimate the real values of the performance (memory consumption) of an algorithm.

1.4 Thesis Outline

The remainder of the thesis is outlined as follows, in Chapter 2 of this thesis, we provide the necessary background needed to understand the proposed framework further. In Chapter 3 a literature review for the area Meta-Learning, and algorithm selection. Chapter 4 details the proposed framework that was developed during this research and the algorithm selection methodology that was used. We showcase the application of the framework to the packet classification problem, using different algorithms and different performance metrics in Chapter 5. In Chapter 6, we draw our concluding remarks and provide some suggestion for future work on both enhancing the developed framework, along with investigating further scenarios.

Chapter 2

Background

This chapter introduces the necessary background information required to further understand the work presented in this thesis. We first explain the notion of Machine Learning and its main categories; we then delve into the subarea of Meta-Learning and Algorithm Selection, explaining its goals and methodologies. The problem of packet classification, its challenges, and the algorithms that were developed to tackle it will also be discussed in this chapter.

2.1 Machine Learning

Machine learning [2] [3] is an inter-disciplinary field of research that combines ideas developed in computer science, mathematics, statistics, operation research, cognitive science and engineering, to provide intelligence to machines. Machine learning is a field of research devoted to automated learning of systems. Automated learning systems range in their applications and complexity. They may be represented by a

very simple learning system based on memorization [5] such as the one that filters unwanted emails (spams) based on memorized table of unwanted senders, to those that uses inductive reasoning in dynamic environments to perform more complex tasks [6]. Lately, machine learning has become the tool of choice when useful information must be extracted from large, and complex datasets. In this section, we will shed some light on the area of machine learning with the focus on the different available approaches.

Machine Learning can be classified in many different ways. It can be classified by the learning process or by the model (engine) it uses to make its decision. One way to classify a learning system is based on the learning principle it follows, another way is based on the level of interaction that a learning system employs with its input. An interactive learner will interact with its environment (input) by performing an experiment, for example, to get more information on the input, while a passive learner will simply rely on observing the information provided by the input. Machine learning research is broadly classified into three different strands (refer to Figure 2.1). Supervised [7], Unsupervised [8] and Reinforcement Learning [9].

2.1.1 Supervised Learning

In *supervised* machine learning, the machine is given a desired set of outputs for given inputs (learning phase). The machine is then asked to produce an output for the newly coming input. The desired output could be either a class label in machine learning systems that are used for *classification* [7], or a real number in those used for regression [10]. In regression problems, the desired output is to predict or

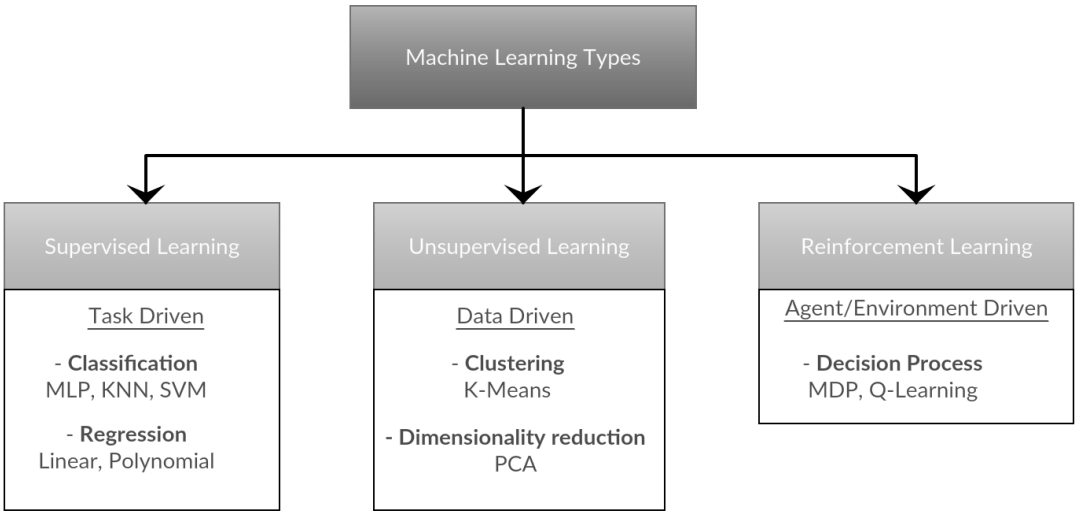

Figure 2.1: Types of Machine Learning

compute a new value for a dependent variable using the values measured from data attributes based on learning from the past training sets. However, in classification problems, the desired output is to classify the input data into predefined labels or classes using a training set of previously labeled data.

Supervised machine learning is characterized by the following:

- A training phase that includes both the input data and the desired output.
- The desired output is already known and usually given during the learning phase to the prediction model.
- The model should be able to generalize the result when it is given input without the target result.

2.1.2 Unsupervised Learning

Unlike supervised machine-learning systems, in *unsupervised* machine-learning systems, the machine is not provided with the desired result during the training phase. It is the machine's (model's) task to learn to build a function that clusters the input data based on their statistical characteristics. Such clustering is not given by any supervising mechanism during the learning phase or by any user; rather it is performed based on the discovered relationship between the different features of the input and the modeled clusters.

It is an essential job of the unsupervised learning systems to decide how to group inputs into clusters that share common properties. Clustering is the process of dividing datasets into subsets that share common characteristics. An example technique for unsupervised learning is the *K-means* clustering algorithm [11]. Another example of unsupervised learning is the one used broadly for dimensionality reduction known as *Principle Component Analysis* (PCA) [12]. PCA is used in our framework, and will be explained in a later section.

2.1.3 Reinforcement Learning

Reinforcement Learning [9] is an example of an interactive learning system. In reinforced machine-learning systems, the machine (agent) produces some actions on the environment. These actions will result in either a scalar reward signal from the input or punishment. The agent should be able to find the best interaction with an environment by itself. This is ensured by providing rewards signals for good actions and punishment signals for bad ones. The agent aims to maximize

the total rewards at the end of its interaction with the environment. Some of the challenges of reinforcement learning involve relating present actions with later rewards (delayed rewards). Such challenge requires the agent to try multiple different scenarios. An effective way to characterize a reinforcement learning problem is to use a probabilistic transition function known as *Markov Decision Process* MDP [13] [14].

2.1.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) [4] can be described as a combination of simple interconnected processing elements (neurons). Neurons can process information and provide output in response to external inputs. The terminology artificial-neural-networks comes as an analogue to the human brain which is made of billions of neurons (nodes) with all neurons interconnected in a massive network.

ANNs are typically organized and constructed in layers of nodes. The first layer is the input layer, this layer receives input from the outside world and introduces it into the network. The last layer in an ANN is the output layer; it produces output to the outside world. The number of layers between the input and output layers are called hidden layers, and can vary in size and number based on the needed complexity of the network. Some types of neural networks, such as the ones used for processing images [15], incorporate other types of layers, such as filtering layers and image pooling layers [16].

The interconnection between the layers consists of weighted fully-interconnected links between individual nodes going from one layer to the other, with the output of the first layer representing the input to the next one.

ANNs provide an output when presented with inputs. The desired output is achieved by adjusting the weighted interconnection between the nodes following the error between the current output versus the desired one. This dynamic cycle of weights adjustments represents the learning cycle for the ANN. There are different rules by which an ANN can learn, including those that propagate back marginal errors from the output to adjust the weighted inputs to the nodes known as *Back-propagation* [17]. Unlike traditional computing networks, ANNs are not sequential computing networks. In an ANN, there is no central computing processor; rather there are many simple processing nodes which take the weighted sum of their inputs from other nodes. The knowledge contained or learned in an ANN is thus represented by the weights in the network itself, which is quite literally more than the sum of its individual components.

ANNs outperform traditional analytical alternative using conventional techniques which are categorized by strict assumptions of normality, and linearity. An ANN allows users to quickly and easily model problems which otherwise may have been very difficult or impossible to explain, which makes ANN a preferable choice of model to use.

An ANN can take different forms and can be trained in multiple ways, for this work, a supervised learning variation of an ANN known as the *Multi-Layer Perceptron* MLP is used. The MLP uses the backpropagation algorithm to train a relatively simple type of architecture of ANN (three types of layers, an input layer, hidden layers, and an output layer). A simple one-hidden-layer MLP is shown in Figure 2.2.

The number of hidden layers in an MLP varies based on the needed complexity

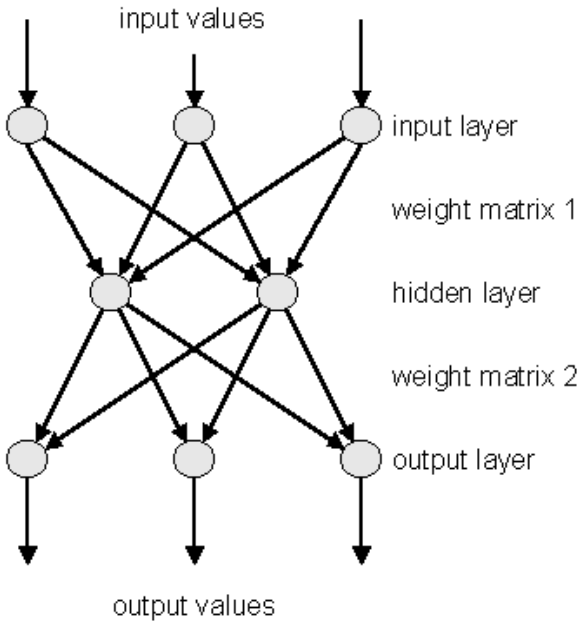

Figure 2.2: A simple MLP Architecture

and the amount of data examples and data features that are given. More hidden layers allow the MLP to learn more sophisticated representations of the data, however, overcomplicating the network results in overfitting. Overfitting the training data results in poor performance on the real world problem. Such tradeoff is taken into consideration (experimentally) when deciding on the number of layers to use in a network.

The MLP network uses nonlinear activation functions in its hiddem-layer neurons. The activation functions for any given input value (x) can be either of two main types, a *(Sigmoid)* function (Equation 2.1) [18] or a *(Tanh)* funnction (Equation 2.2) [19]. Both functions have advantages and disadvantages, however, for this work, only the sigmoid function was considered. The reader can refer to this paper [19] by Yann LeCun for further information and comaprisons.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

For the MLP output neurons, the type of function varies based on the task. For classsification tasks, each output neuron acts as a certain problem class. For such purpose, the *softmax* function (an extension to the *sigmoid* function) can be used. These functions output values for each neurons that are relative to all the other output neurons, forcing them to sum to "1". This produces a probability distribution over neurons of mutually exclusive discrete classes, which comes as a natural representation to the classifcation problem. Given a neural network with (n) output units (refer to Figure 2.3), the softmax function can be calculated using Equation 2.3. Moreover, for regression tasks, a regular *sigmoid* function is usually used as the output unit type.

$$y_i = \frac{e^{Z_i}}{\sum_{j=1}^{n} e^{Z_j}} \tag{2.3}$$

2.1.5 Principle Component Analysis

Principal Component Analysis (PCA) [12] is a statistical technique which falls under the unsupervised learning umbrella. The technique uses data variation maximization to transform a set of correlated features into a set of linearly uncorrelated ones. The transformation results in new features known as principle components. The

transformation is performed in such a way that the first new dimension (principle component) has the largest variance in the data (maximum information). Each succeeding component has the next highest variance possible. The new PCA dimensions are all orthogonal to each other. PCA is used for making predictive models. It is a widely used tool in data visualization, dimensionality reduction [20], trend analysis, factor analysis [12] and noise reduction. For this work, PCA is used as a *dimensionality reduction* and a feature enhancement tool. An example of PCA usage can be seen in Figure 2.4. The figure shows that PCA components have a better grasp on the information within the data. The new dimensions (PCA components) are centered in a way capturing as much of the variation in the data as possible.

2.2 Meta Learning

Meta-learning [1] is a key research direction in Machine Learning (ML) and Data Mining (DM). At the highest level, it studies the concept of "*Learning to Learn*" [21]. The main goals of Meta-learning are twofold. The first is learning to choose between algorithms to tackle a certain problem, maximizing the performance (known as *Algorithm Selection*). The second is learning to transform learned knowledge from one domain to a similar one (known as *Domain Knowledge Transfer* [22]). In this thesis, we focus on *Algorithm Selection* with an application to the problem of packet classification.

2.2.1 Algorithm Selection

Algorithm Selection can be seen vital in situations where the choice of an algorithm is not trivial and the cost of applying each algorithm is prohibitively costly.

According to the *"No Free Lunch Theorem"* (NFL) [23], there is no one algorithm that performs well on all problem instances and across all performance measures. Generally, Algorithm Selection in Meta-Learning can be described as a continuous process that constructs *"Meta-data"* database for the performance of every algorithm on any given dataset, and applies a *"Meta-feature"* extraction technique to extract features from datasets that best relate to algorithm performance. It then uses the pairing of Meta-features and algorithms' performance to infer the most suitable algorithm for new datasets.

2.2.2 Meta Features

Meta-learning systems, rely on the existence of three main components:
(i) Datasets/algorithms which make up the system inputs. (ii) Meta-data, which is comprised of algorithm performance information on datasets, and datasets *descriptive* features (referred to as "Meta-features"). (iii) Meta-learner, which is the model that learns useful information from the meta-data.

Meta-learning researchers attempted to simplify the task of algorithm selection from the datasets' side. One of the successful efforts was the development of specifically designed features that can accurately describe datasets, which made it easier to deal with large, and complex datasets. It also made it possible to perform tasks related to direct comparison of datasets from different domains - i.e., *Transfer*

Learning - [24]. These features are referred to as *Meta-features* [21].

Meta-features are carefully designed descriptive features that can describe a complete dataset rather than regular features which are only used to describe data records (data examples). Such *Meta-features* are very beneficial in algorithm selection when comparing different datasets, with different sizes, and different data distributions. However, the choice of meta-features is so far in the research very dependent on the domain of applicability. Most of the developed frameworks so far rely on hand engineered meta-features designed to handle datasets from a single domain.

Many *Meta-features* were developed and used in the literature. These meta-features can be categorized as follows:

1. ***Simple Meta-features***

   Simple Meta-features comprise of simple measures that are extracted directly from the datasets [25]. Such features can include the number of data instances, number of data attributes, the number of data classes, and the ratio of categorical attributes to numerical attributes. These feature, are simple to compute. However, they don't describe datasets accurately. Therefore, other meta-features were developed.

2. ***Statistical Meta-features***

   Statistical Meta-features describe statistical measures that are calculated from each dataset [25], and include but not limited to, the geometric mean, mean absolute correlation of attributes, first canonical correlation, the Skewness, and the Kurtosis.

3. ***Information-Theoretic Meta-features***

   These features include information theoretic datasets' measures [26], such as the class entropy, mean entropy of attributes, and the mean mutual information of class and attributes. Such features capture randomness in datasets. Randomness and redundancy in a dataset can be used as a strong measure of complexity. Information-theoretic meta-features are most useful when used on discrete categorical datasets.

4. ***Land-marking based Meta-features***

   Land-marking [27] uses a set of simple models (algorithms), different from the pool of algorithms to be chosen from, to characterize datasets. The technique uses the performance of these algorithms (*Land-markers*) on the respective datasets as characteristics to infer the performance of more complex algorithms. Examples of Land-markers can be linear models or pruned decision trees [27]. Landmarking is mostly useful when the cost of applying regular meta-features is prohibitive.

5. ***Other Meta-features***

   Other types of features that don't fall under the mentioned categories were developed. Some are based on dataset complexity, and others are based on dataset structure [28]. The work in [28] illustrates a meta-feature extraction technique that combines dataset binarization with summary statistics to represent dataset structure, using what they call *itemsets*. This technique is generic (works across different domains) yet effective. It can be used outside of the machine-learning domain, describing different types of datasets than

the ones used for supervised classification.

Meta-feature should have two important characteristics. They should first, be easy to calculate. Secondly, they should capture helpful information in finding algorithm performance. Meta-features should also capture information relating to dataset configuration, and dataset complexity.

2.3 Packet Classification

Packet Classification [29] is a core functionality in network routers. It is a fundamental part of any computer network. To facilitate the exchange of data between network nodes, data need to be divided into packets. Packets are then streamed throughout the network. In large networks, especially ones with exposure to the internet, to ensure the data exchange reliability and prevent attacks from outside sources, core routers and firewalls are used. Such vital network elements need guidance and governance on how to act when packets pass through them. Therefore, packet classification is used.

Packet classification enables the network's routers and firewalls to perform critical actions on packets. Computer networks use sets of rules to define the ways by which a router or a firewall behaves against incoming packets. Every packet header has F fields that are considered in the classification process. In IPv4 based networks, the packet classification fields reside in the IP layer and the transport layer [30] [31]. The set of rules governing the behavior of the routers and firewalls are referred to as *rulesets*.

2.3.1 Packet Classification Rulesets

In a packet classification ruleset, each ruleset RS has R rules, typically ranging from 100 rules to 10,000 rules. Each rule $r \in R$ has F components which identify all possible combinations of packet headers that match that rule. Accordingly, a packet will match the rule if, and only if, all the fields in that packet belong to all the corresponding fields in the rule.

A field $f \in F$, may contain a single value (e.g., the protocol), or a range of values (e.g. the Source IP range). Table 2.1 shows an example ruleset RS with $F = 5$ fields, $R = 5$ rules, and two possible actions.

Source Address	Dest. Address	Source Port	Dest. Port	Protocol	Action
240.118.164.224/28	250.13.215.160/28	88 : 88	53 : 53	0x01/0xFF	Permit
240.118.164.224/28	250.13.215.160/28	88 : 88	80 : 80	0x01/0xFF	Deny
209.67.92.32/28	55.186.163.16/28	88 : 88	80 : 80	0x11/0xFF	Deny
240.118.164.224/28	55.186.163.16/28	69 : 69	0 : 65535	0x06/0xFF	Deny
0.0.0.0/0	125.0.0.0/8	0 : 65535	0 : 65535	0x00/0x00	Permit

Table 2.1: An example of a 5 Fields Firewall ruleset

The number of fields F in an IPv4 packet classification ruleset is variable, and varies with different network applications. To measure the performance of different algorithms, researchers rely on a set of standard benchmarks that are generated by a sophisticated tool [32]. Although the data generated is synthetic, yet it still mimics the complexity of real rulesets which are not accessible due to privacy and security concerns. The benchmarks that were used in the literature can be categorized (based on [32]) into the following classes:

- **Access Control List** (ACL).
- **Firewall** (FW).

- **IP Chain** (*IPC*).

The *Access Control List* (ACL) format is a simple filter format used to define packet filters and forwarding rules for packets traversing the network elements, resulting either in a permit or deny action based on criteria match. Standard ACL uses IP source and IP destination address for the deny/drop action; however, extended ACL uses more detailed information such as source port and destination, application type, etc., to perform an action. ACLs are sometimes employed in Quality of Service (QoS) to prioritize packets and provide different treatment to a certain stream of packets based on different applications.

Unlike ACL, the *IP-Chain* (IPC) filter format utilizes IP Tables that define a traverse path for incoming packets that leads up to a final action. In IPC incoming packets are tested against a specific field of the ruleset (filter) and each result will lead to a further testing and decision branch until a final action on the packet is taken.

The *Firewall* (FW) filter format is the proprietary format used by each vendor for their specific firewall design and implementation. These rulesets are designed and implemented by each vendor to exhibit their strength and users need to use their tools to reconfigure the filters. FW also exhibits more complex challenging structure than the previous two, making it more challenging to tackle from an algorithm's point of view.

For the proposed work in this thesis, we generated more rulesets than the ones typically used in the literature; we used the twelve seed files that came with the tool in [32]. The generated rulesets are still within the same three categories. However, they have more distribution variation which allows us to explore a wider spectrum.

Label	***Source IP*** range
IP range 1	10 - 50
IP range 2	60 - 100
IP range 3	120 - 190
IP range 4	0 - 200

Table 2.2: Example 1-field ruleset (Ruleset # 1)

Five of the twelve seed files belong to ACL, five files for FW, and two files for IPC. We noticed that in previous work, only one of each seed types were used to generate the benchmarking rulesets.

2.3.2 Problem Complexity

The main challenging characteristic of the problem of packet classification is the overlapping of rules in a ruleset. To explain the overlap, let's take into consideration the following example, for simplicity purposes, the *SourceIP* will be used as the rule component by which the overlap is measured.

Consider a case where two rulesets each with four rules, *Ruleset # 1* in Figure 2.2 and *Ruleset # 2* in Figure 2.3, in each ruleset there is a header field that contains the SourceIP range for every rule. To simplify the visualization, IPs are denoted by real numbers, giving the *wildcard* IP range the values 1 - 200 covering the entire IP range spectrum, where any other IP range lies within those boundaries. Typically, a wildcard IP range is 0.0.0.0/0 which translates to addresses starting from 1 to 4294967294.

Taking the first ruleset (Figure 2.2) into consideration, the overlap (Figure 2.5) between the ranges is minimal. Each range (aside from the wildcard) has only one overlap, and in this case, each range only overlaps with the wildcard, which results

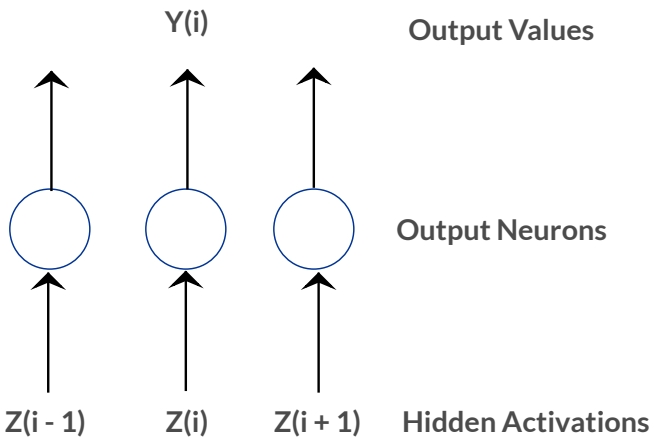

Figure 2.3: Softmax function illustration

Label	***Source IP*** range
IP range 1	10 - 60
IP range 2	100 - 180
IP range 3	40 - 120
IP range 4	0 - 200

Table 2.3: Example 1-field ruleset (Ruleset # 2)

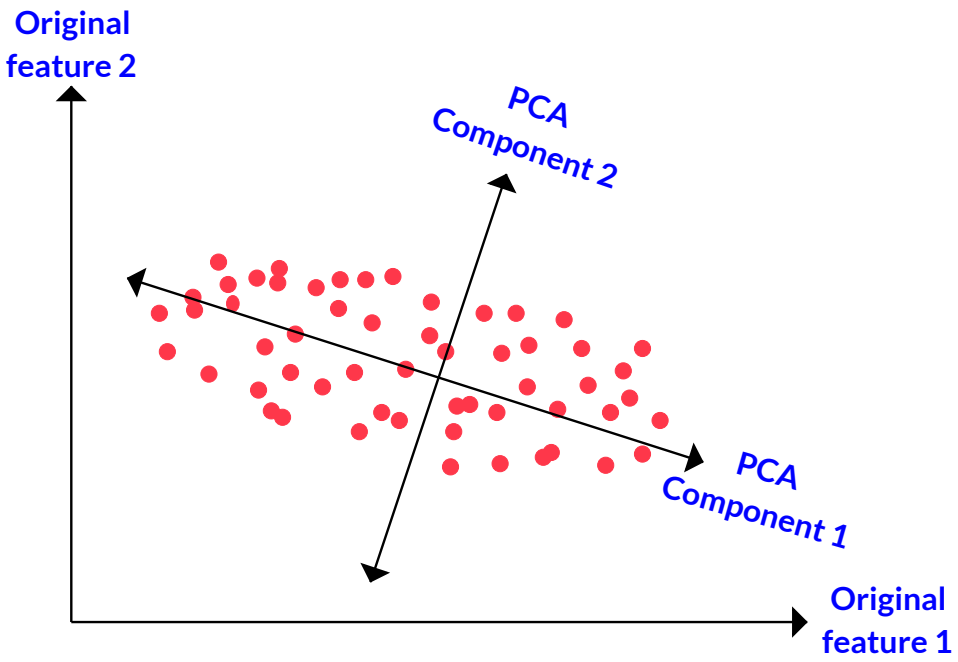

Figure 2.4: PCA dimension transformation example

in only four search possibilities, (1) an IP is in range # 4 but not in any other range. (2) An IP is in range # 2 and # 4. (3) an IP is in range # 3 and # 4. (4) an IP is in range # 1 and # 4. However, as shown in Figure 2.6, the overlap is more complicated in *Ruleset # 2*. The IP ranges intersect more often which yields additional search possibility regions than *Ruleset # 1*.

It can be seen in Figure 2.6 that the search have more possibilities. (1) An IP is in range # 4 but not in any other range. (2) An IP is in range # 4 and # 1. (3) An IP is in range # 4 and # 3 and # 1. (4) An IP is in range # 4 and # 3. (5) An Ip is in range # 4 and # 3 and # 2. (6) An IP is in range # 4 and # 2. Although the two rulesets have the same number of rules. However, their search possibilities are different. This added search complexity is what makes the problem more challenging from an algorithmic perspective, and depending on the type of

algorithm, this issue can yield either worse or better results based on the search technique that is used.

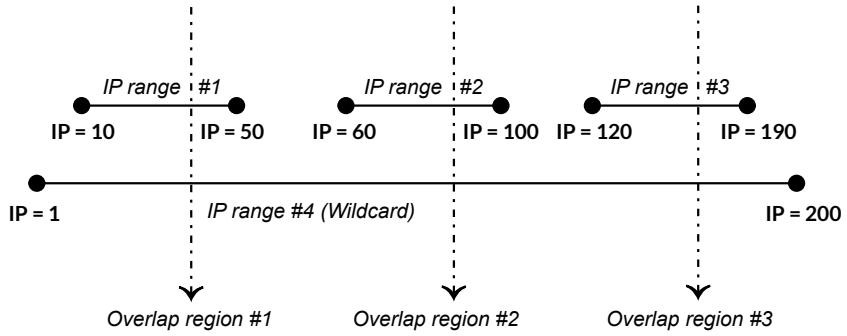

Figure 2.5: Light overlapping for *Ruleset # 1*

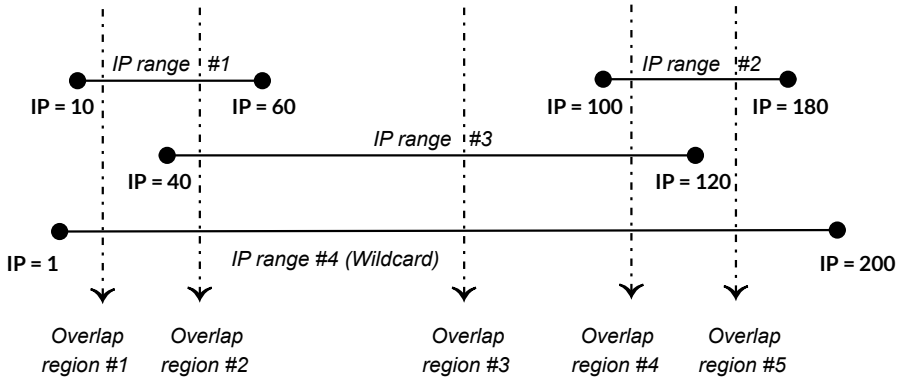

Figure 2.6: Heavy overlapping for *Ruleset # 2*

Additionally, depending on the type of ruleset, the overlap varies. For Access Control List (ACL) rulesets, the overlap is very minimal, while for other types of rulesets, such as Firewall, the overlap is much more complicated - e.g., the case in Figure 2.6. Furthermore, the problem of rules overlap is usually multidimensional (beyond 1-field), which makes it even more complicated.

2.3.3 Packet Classification Performance metrics

To measure the applicability and the robustness of a packet classification algorithm on a *ruleset* certain performance metrics were suggested in the literature.

- **Search Speed (Time)**

  This metric measures the time it takes an algorithm to process a ruleset and classify a certain number of incoming packets. It is measured in milliseconds (ms).

  - Pre-processing time is the time taken to pre-process a ruleset before performing any classification. Preprocessing time is measured in seconds.
  - Classification time is the time taken to classify packets into flows based on the rules in the ruleset. Classification time is measured in seconds.

  Some algorithms do not have pre-processing time, such as brute-force based techniques [33, 34], since they search the entire ruleset to pick an appropriate rule. While in other algorithms, such as decision-tree based algorithms [35, 36], where the pre-processing step involves building a rules tree based on the ruleset.

- **Storage requirements (Memory usage)**

  This metric measures the storage requirements based on the algorithm's required memory resources and it is usually measured in bytes.

Other metrics to measure performance are available, including algorithm *scalability* which is concerned with the algorithm's ability to handle a different number of header fields. Another metric is the algorithm's *update capability* which outlines

the algorithm's ability to update it's built data structure when rulesets are modified or changed.

2.4 Algorithm Selection for Packet Classification

Selecting the proper algorithm in a complex networking environment with a complex multitude of different rulesets and different performance metrics in real time is clearly a challenging problem. Such problems usually exhibit enough characteristics that make them qualify as algorithm selection problems.

2.4.1 Problem characteristics

The problem features can be summarized as follows:

- ***Performance Variation***

  Algorithms have varying packet classification performance on the respective rulesets (datasets). Many algorithms tend to perform well on some rulesets while performing poorly on others.

- ***Challenging Characteristics***

  Packet classification ruleset possesses intrinsic features such as rules overlapping; that correlates with the performance of different algorithms. This correlation can be exploited to infer algorithm performance from only the rulesets.

- ***Choice Ambiguity***

  In most cases, the choice of the most suitable algorithm on a given rule-set is

not a clear cut choice (an exhaustive brute force approach has to be taken) - e.g., an "IF-Then" approach is not suitable - Additionally, the random selection approach is not guaranteed to yield acceptable results.

- ***Resources Restriction***

  Due to the embedded nature of some networking applications, and the need for embedded processing, there is a huge restriction on resources, therefore, to choose between algorithms by applying all combinations of algorithms and rulesets is infeasible and impractical.

2.4.2 Problem formulation

We now provide a formal mathematical formulation of the problem of packet classification algorithm selection based on the original definition of the algorithm selection problem outlined by Rice in [37].

Assume:

- A ***Ruleset*** space $R \implies$ a set of packet classification rulesets.
- An ***Algorithm*** Space $A \implies$ a set of packet classification algorithms.
- A ***Feature*** Space $F \implies$ a combination of features that are measurable from the rulesets.
- A ***Performance*** space $P \implies$ a number of performance measures measurable from the algorithms with respect to rulesets.

The packet-classification algorithm selection problem can then be defined as follows: for a given ***ruleset*** $r \in R$, with ***features*** $f(r) \in F$, find the selection ***mapping*** $S(f(r))$ onto the algorithm space A, such that, the selected ***algorithm*** $a \in A$ maximizes the performance $p \in P$, Figure 2.7 shows a graphical representation of the problem.

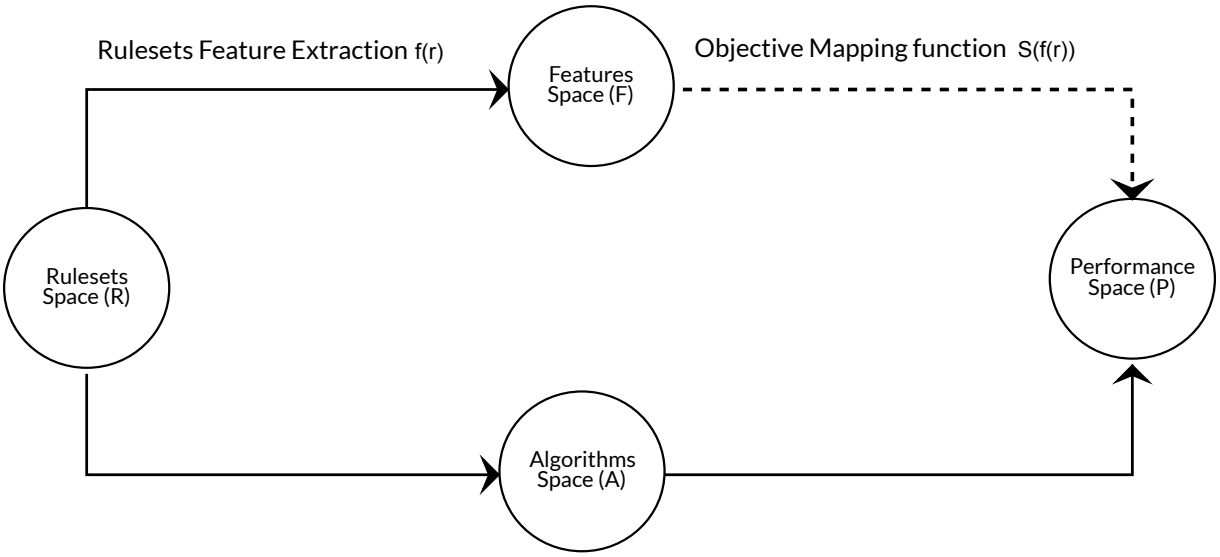

Figure 2.7: Graphical space representation of the problem

2.5 Summary

In this chapter, we introduced concepts related to machine learning with its broad classification and research strands. We also discussed ML algorithms with a detailed focus on ANN, as it is the algorithm of choice in this research. We also discussed Meta-Learning and how it is different from basic learning in ML. We explained the different categories of meta-features and their use as well as the added knowledge

they provide to basic ML. We later introduced the packet classification problem with its definitions as a challenge in the networking industry with the current and foreseen networking requirements. We concluded this chapter by matching the characteristics that packet classification algorithm selection exhibit and we showed that it lends itself very well to the class of problems that could be addressed by Meta-Learning.

Chapter 3

Literature Review

In Chapter 2 the problem of Algorithm Selection for packet classification was introduced. We argued that it could be tackled using the merits that Meta-Learning provides. In this chapter, we outline research work in Meta-Learning that we see relevant to the research work outlined in this thesis. Meta-Learning and Algorithm Selection research contributions are presented. The main contributions of the research and the proposed solutions to the packet classification problem are also outlined.

3.1 Packet Classification Algorithms

Packet-classification algorithms are categorized into 4 main categories [30]. Brute force algorithms can be found in [34]. These algorithms work by performing an exhaustive search on the ruleset. Accordingly, these algorithms tend to exhibit enormous classification times, especially as the size of the rulesets grow. Therefore,

a significant amount of research has been devoted to reducing classification time by using decision-tree techniques [38, 35, 36], decomposition based or divide and conquer based methods [39] [40], and tuple space based techniques [41, 42]. A taxonomy of packet classification algorithms is shown in Figure 3.1. These algorithms vary in their preprocessing time, classification time, memory requirements, etc.

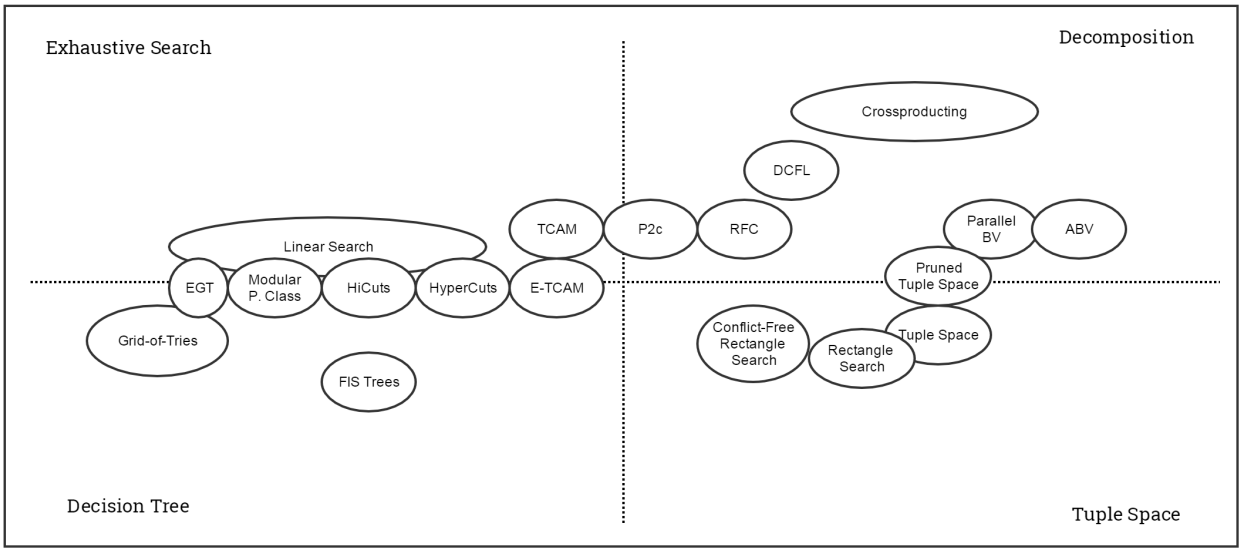

Figure 3.1: Taxonomy of Packet Classification algorithms

1. **Brute Force Techniques** tend to examine all entries sequentially in the ruleset. As with all exhaustive sequential searches, brute force techniques can easily deal with updated rulesets, require minimum memory usage and need no preprocessing. However, with the increase in the size of the ruleset, brute force methods [33, 34] tend to become impractical because of the time required to perform the search which is of order $O(R)$ where R is the number of entries in the ruleset.

2. **Decision Tree Based Techniques** reorganize the ruleset entries into a decision tree and use the incoming packet fields to navigate the ruleset tree. As with any decision tree construction, memory is required to construct and store the tree, and this will eventually consume some pre-processing time as well. Depending on the incoming packet fields and the path to be traversed in the decision tree, decision time can vary [35, 36].

3. **Tuple Space Based Techniques** is a bit different than other approaches in the sense that it relys on mapping the filters to a different space known as the tuple space. The advantage of having a tuple space is that good speedup over regular methods in classification speed is obtainable even with a naive search approach. This technique uses the least memory. However, it requires a significant preprocessing time [41, 42]..

4. **Decomposition/Divide-and-Conquer Techniques** split the full packet search involving all the header fields into sub-searches considering one field at a time [29, 43, 44, 45]. They also perform single field search and later combine the result into a single decision. These techniques reduce the search time but require more memory and longer pre-processing time.

   An example of such algorithms is *Group Based Search Algorithm* (GBSA) [40]. The main idea behind GBSA is to escape the linear search in large rulesets by reducing the number of search attempts on the ruleset to clusters-of-rules based search, rather than individual rules search. It is clear that such clustering will reduce the exhaustive linear search on large rule sets and reduce that to cluster search. However, a significant amount of preprocessing

time is required to form the cluster lookup table, making the GBSA perform poorly on smaller size rulesets. GBSA avoids complex data structures needed by other algorithms and uses a simple IP source and IP destination as an address for a lookup table of the generated clusters.

The clustering approach used in GBSA utilizes the starting bits from both IP source and IP destination to generate clusters of rules. GBSA is not limited to the starting bits of both the IP source and IP destination addresses; it can accommodate any number of bits from the source and destination IP to apply to the cluster formation rules.

(PCIU) *a Packet Classification Algorithm with Incremental Updates* [39] is yet another divide and conquer algorithm. It is composed of three stages, the preprocessing stage, the classification stage and the incremental update stage. It is due to the later stage or feature that PCIU obtained its name, where rules are either added or removed from the rule set to provide an incremental update to the ruleset. The preprocessing stage categorizes the algorithm as a decomposition class. It is during this stage where rules in the rule set are converted to range representation using the five fields in the rule header. The generated groups utilize overlapping between rules to produce groups that compose similar features. Binary vectors then represent the generated groups, that carry the information used in the classification phase.

The performance variation and pros and cons of an algorithm over another raise the question: *"How to know when to use Algorithm A over Algorithm B for Problem X? and vice versa"*. According to the *"No Free Lunch Theorem* (NFL)" [23], no

one algorithm performs well on all problem instances and across all performance measures. All the mentioned challenges provide impetus to explore the development of an automated system that can predict the most suitable algorithm for a given instance of a problem.

3.2 Algorithm Selection

The problem of *Algorithm Selection* was first introduced and formulated by Rice in [37]. Rice didn't provide a solution to the problem. He, however, identified the key ingredient to solving it. Rice acknowledged that for each problem or a domain of problems, there exist a set of problem characteristics (later defined as *meta-features*) that correlate with the performance of algorithms.

3.2.1 Meta Learning

Machine Learning researchers, devoted a community by the name of *Meta Learning* to find solutions to the Algorithm Selection problem. The Meta-Learning community, however, only focused their work on a certain domain. Meta-Learning systems were mainly developed to solve the Algorithm Selection problem for the machine-learning classification algorithms and datasets. Researchers from fields other than Machine Learning recently adopted Meta-Learning based systems to be applied in their domains.

3.2.1.1 Meta Learning for Machine Learning problems

Meta-Learning was introduced by the ML community and was successfully implemented in Algorithm Selection tasks related to supervised learning such as classification and regression Algorithm Selection. Successful contributions can be seen in StatLog [46], METAL [47], and NOEMON [48] projects, which were large-scale European funded projects.

Some successful implementations of Meta-Learning systems on supervised Machine Learning problems are addressed in the literature. The *Variable-Bias Management System* (VBMS) [49] is viewed as the first simple Meta-Learning system that learns to choose between three symbolic algorithms as a function of training and feature sets. The *StatLog* project [46] was a long-term (three years) heavily-funded and large scale project. StatLog extended VBMS by considering a larger number of datasets' characteristics and produced a thorough imperial analysis of machine learning algorithms and models. It laid-out the foundations for the development of Meta-Learning systems by conducting a comprehensive study of 23 classification algorithms on 22 datasets taken from the UCI ML repository [50], showing their strengths, weaknesses, and applicability. The summary of the project results can be found in a book by Michie et al. [25].

Following the success of StatLog, another project was carried out by a group of researchers in Europe. The project was titled the *Meta Learning Assistant* (METAL) [47]. METAL extended StatLog by considering another class of supervised machine learning algorithms and targetted both classification and regression algorithms. It examined 53 UCI datasets and ten learning algorithms. One of the

outcomes of the project was a web-based Meta-Learning system for the automatic selection of learning algorithms in the context of machine-learning classification tasks, known as the *Data Mining Adviser* (DMA).

A recent work [51] shows how a combination of Meta-Learning with *Bayesian Optimization* can be used to build an automated machine learning system. The system can guide the user on selecting algorithms on different steps of the data mining process. AutoWEKA [52] is another recent example of Algorithm Selection for machine learning, which was an extension to the widely-used data mining tool WEKA [53].

3.2.1.2 Meta Learning beyond Machine Learning problems

To the best of our knowledge, Meta-Learning has not been previously applied to the task of Algorithm Selection for *Packet Classification* algorithms. However, other areas such as *Artificial Intelligence*, *Data Mining*, *Control Theory* and *Operation Research* have seen an extensive Meta-Learning related research due to the promising results and effectiveness that Meta-Learning methodologies provide in solving the *Algorithm Selection* problem.

In Operations Research, the *Travelling Salesman Problem* (TSP) is a very well-known combinatorial optimization problem. It is described as an NP-hard Problem, where it 's hard to find an optimal solution using exhaustive search methods. It is estimated that there exist approximately $1.22 * 10^{17}$ feasible solutions for a TSP of 20 cities using exhaustive sequential search. In its informal form a TSP problem is described as follows: Given a set of cities and the distance between each pair of them, find a traveling tour that starts in any one of the cities, visit each other

city once, and returns to the original city. A generalized solution for the general form of The TSP is difficult to find. However, there are lots of algorithms that can provide a near-optimal solution to a particular set of TSP instances. Meta-Heuristics (MHs) have been used to reduce the search for an optimal solution of specific TSP instances. MHs are guiding strategies that work to guide the search to the space of near optimal solution. In [54] Jorke Kanda et al, proposed a framework that uses Meta-Learning in selecting the proper MHs for any new instance of a TSP problem. The selection is based on Meta-Feature extraction from TSP instances and their appropriate MHS selected strategy. The model adopted MLP, K-ANN and decision tree. The result of this work concluded that meta-features based selection of MHs algorithm for new TSP instance works better than any other strategy to tackle a new unseen case of TSP.

The authors in [55] used concepts from *Reinforcement Learning* and *Dynamic Programming* to solve the Algorithm Selection problem for choosing between different sorting algorithms. Their system chooses between three sorting algorithms (Insertion, Quick, or Merge sort) for a given sequence based on its length (as a meta-feature). An extension of the work was carried out by [56] that included a larger set of algorithms (addition of Heap, and Shell sort). [56] utilized a measure of *"presortedness"* [57] as an additional meta-feature to the sorting problem. The work investigated the selection between the set of algorithms on a set of 43195 problem instances achieving over 90% accuracy on predicting the fastest sorting algorithm. The authors in [56] surprisingly didn't make any reference to "Meta-Learning".

Recent work in packet classification [58] shows a form of Algorithm Selection, where a memory consumption model is used. The model makes the decision on

whether to use a cut or a split algorithm for a particular ruleset based on the predicted memory footprint. However, the authors made it seem as if the model was built specifically for a set of algorithms (only the ones mentioned in their paper), showing no means of generalization to include other packet classification algorithms.

Knowing which algorithm is most appropriate a priori for a given dataset is not possible without manually implementing and evaluating a set of candidate algorithms. However, this is often not practical. Therefore, in this thesis, we present an efficient framework for automatically selecting the most appropriate algorithm for a given dataset. The work is presented as a generic solution to the problem with an application to the packet classification domain.

3.3 Summary

This chapter first introduced a taxonomy of packet classification algorithms and the features that each class of algorithms provides. It then introduced the problem of Algorithm Selection and its origin. This chapter also presented the research area of Meta-Learning as a solution to the Algorithm Selection problem. Examples of successful research projects were introduced with their main contributions to the literature in different domains. To the best of our knowledge, no real Meta-Learning system was ever introduced to address the Algorithm Selection problem in the area of packet classification. Furthermore, all of the previously developed systems and the choice of meta-features seem to target particular problems domains.

In this thesis, a framework for Algorithm Selection is developed keeping in mind

its generality of use. We demonstrate how it can be applied to the task of Algorithm Selection for packet classification as a new domain to this area of research. The framework methodology will be introduced in the following chapter.

Chapter 4

Methodology

This chapter details the proposed framework including its different operational phases. It discusses the training phase, including meta-features extraction, meta-learning, and algorithms ranking. It also discusses the testing phase and its effectiveness in recommending an algorithm when given a problem instance. The framework is explained and evaluated using packet classification algorithm selection as an example.

4.1 The Proposed Framework

The proposed framework consists of two operational stages: a training stage (illustrated in Figure 4.1) and a testing stage (illustrated in Figure 4.2). A detailed description of the framework is given in the following subsections.

4.1.1 Training Phase

The objective of this stage is to train a model that can be used to perform automatic algorithm selection. The four main components of the training framework stage are the (i) Benchmark generator, (ii) Descriptive feature extractor, (iii) Algorithm ranking module, and (iv) Machine-learning model. Each of these elements and their overall flow are explained next.

First, a combination of rulesets are generated and stored in a database. The training stage then applies all of the available packet classification algorithms on all of the considered rulesets, thus creating the performance metrics that are used to train the learning model. A feature extraction technique is also applied on each of the rulesets under consideration. The training data used to train the learning model includes both the data's descriptive (*Meta-features*), along with the candidate algorithms' performances. The framework is designed to be flexible, and can incorporate any type of machine-learning classification algorithm (e.g., Support Vector Machines (SVM), K-Neasest Neighbor (KNN), etc). A graphical illustration of the training phase is shown in Figure 4.1.

4.1.1.1 Ruleset Benchmarks

The performance of different classification algorithms is measured using standard benchmarks that are generated via an open source benchmark generator tool [32]. The tool comes equipped with seed files representing the different problem domains presented in Chapter 2, each with different data distributions and complexities. All of the seed files were considered for this work, with various (rule) size settings

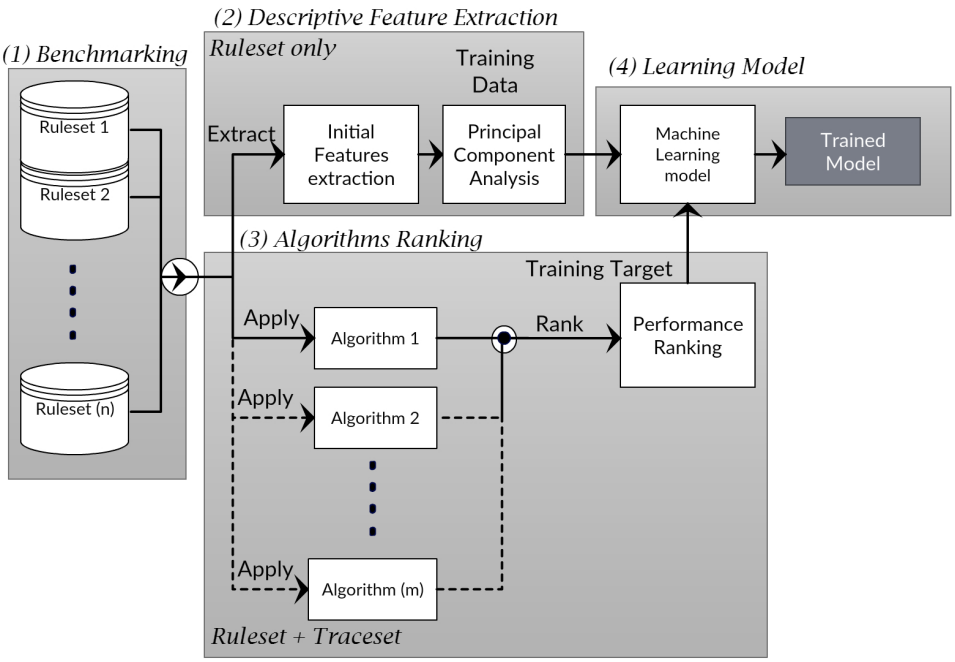

Figure 4.1: The Training Phase of The Proposed Framework

to ensure data variability. The tool generates test sets based on the generated rulesets. The test sets are known as *trace sets*. In practice, the trace sets are $10x$ larger than the original ruleset. When measuring the algorithm performance on a particular ruleset, the actual ruleset is only used in the preprocessing stage, while the corresponding trace set is the one considered for the classification (testing part). The packet classification algorithms' performance is measured as follows:

- **Memory requirement (storage)**: Measures the amount of memory needed to store a preprocessed ruleset during preprocessing.
- **Classification (search) speed**: Measures the time required to classify a trace in a trace set during classification. However, preprocessing time was not considered in this work.

4.1.1.2 Descriptive Feature Extraction

The meta-feature extraction technique used in this work is inspired by [28]. The generic nature of this technique allowed us to use it to extract general meta-features from rulesets (packet classification datasets) in the packet classification domain of problems without the need to handcraft packet classification specific meta-features. Furthermore, this technique has also shown its effectiveness when the task is to choose between Machine Learning classification algorithms [59]. This part of the framework only involves the original ruleset; that is, the trace set is not involved (not needed). The feature extraction process incorporates several steps, as explained below.

- **Step #1**

  Given a ruleset r (see Table 4.1), transform r from a categorical format to a binary format. This step is performed for easier feature extraction. A simplified ruleset is used for illustration purposes. The binarization process

Rule	Src Address (IP)	Dest Address (IP)	Flow (Action)
1	139.203.128.167/32	139.203.133.104/32	deny
2	139.203.128.167/32	139.203.133.104/32	deny
3	139.203.128.167/32	17.50.44.61/32	deny
4	0.0.0.0/0	0.0.0.0/0	permit

Table 4.1: An example two field ruleset

  involves identifying the unique values of the ruleset r. For each field (column), j in a rule k, the unique values of the particular field are treated as categories that act as the new fields for the binary ruleset representation. For the ruleset r in Table 4.1, the Source Address has two unique categories in four rules, while the Destination Address has three, as shown in Table 4.2.

Field	Unique Values	
Source Address	SA1	139.203.128.167/32
	SA2	0.0.0.0/0
Destination Address	DA1	139.203.133.104/32
	DA2	17.50.44.61/32
	DA3	0.0.0.0/0

Table 4.2: Ruleset unique value identification

The new category values are then used to reconstruct a new (binary) representation of the ruleset. Table 4.3 illustrates the new binary ruleset based on the unique values of Table 4.2. The unique values (unique categories) act as the ruleset's new fields where the values of each field for every rule would be either binary "1" or binary "0" otherwise.

	Source Address		**Destination Address**		
Rule	SA1	SA2	DA1	DA2	DA3
R1	1	0	1	0	0
R2	1	0	1	0	0
R3	1	0	0	1	0
R4	0	1	0	0	1

Table 4.3: A ruleset using binary representation

- **Step #2**

  The 1^{st} and 2^{nd} feature vectors are extracted. These vectors can be extracted simultaneously, as there is no dependency between them. For the 1^{st} FeatureVector, let $Cat_{(ijk)}$ be the i_{th} unique value (unique category) of a field j in a rule k. Equation 4.1 can then be used to compute the first feature vector of r.

$$Fv^{1st}(r) = \left(\frac{1}{n} \sum_{k=0}^{n} Cat_{ijk} \right), \ \forall(i,j) \tag{4.1}$$

where n is the number of rules in the ruleset r.

$$Fv^{1st}(r) = \left(\frac{3}{4}, \frac{1}{4}, \frac{2}{4}, \frac{1}{4}, \frac{1}{4}\right)$$

Note that the number of elements in $Fv^{1st}(r)$ is equal to the number of all the unique values in r across all fields (i.e., 2 unique values + 3 unique values = 5 elements). Unlike the 1^{st} FeatureVector, the 2^{nd} FeatureVector involves comparing the unique categories of two distinct fields - e.g., Source Address and Destination Address. If we let $Cat_{(ijk)}$ be the i_{th} unique value (unique category) of a field j in a rule k and $Cat_{i'j'k}$ be the i'_{th} unique value of a field j' in the same rule, where j and j' are different fields. Equation 4.2 can then be used to compute the second feature vector of the ruleset r.

$$Fv^{2nd}(r) = \left(\frac{1}{n}\sum_{k=0}^{n}\left(Cat_{ijk} \oplus\ Cat_{i'j'k}\right)\right),\ \forall(i,\ i', j \neq j') \tag{4.2}$$

$$Fv^{2nd}(r) = \left(\frac{1}{4}, \frac{2}{4}, \frac{4}{4}, \frac{3}{4}, \frac{2}{4}, \frac{0}{4}\right)$$

Note that the number of elements in $Fv^{2nd}(r)$ is equal to the product of the numbers of unique values in each field j in r (i.e., 2 unique values * 3 unique values = 6 elements).

- **Step #3**

  The two feature vectors are then concatenated, sorted, and summarized using the *Five Number Summary* statistics technique outlined in [60]. The summarized vector incorporates the minimum, maximum and seven statistical quantiles of each of the concatenated vectors. The use of *Five Number*

Summary ensures a reduction in the number of the feature vectors (typical rulesets have hundreds of rules, generating very large vectors). It also provides a unification of the size of the extracted vectors, regardless of the size or content of the original ruleset, which simplifies further rulesets comparison analysis. The final summarized vector is referred to as the ruleset's *Meta Features* vector.

Throughout our experimentation, we noticed that the model occasionally struggled to achieve a high prediction accuracy due to the large number of correlated features that were extracted from the previous process (as a result of information redundancy, and unnecessary correlation). Therefore a post-processing stage in the form of Principle Component Analysis (PCA) was included in the proposed framework and is explained next.

- **Step #4**

  The ***Principle Component Analysis*** technique [12] was used as preprocessing step to the learning model to ensure dimensionality reduction, and hence reduce the error generated from over-fitting the training data. PCA re-projects the data into a compact space using transformed features. It can ensure the availability of $\approx$ 99 % of the original variance in the relevant dimensions of the re-projected space, while keeping it fully representative.

  There are multiple ways to decide the number of PCA dimensions to retain. In this work, a variance cut-off value of 95% coupled with an exhaustive experimental accuracy analysis was used as the deciding factor. The 95% cut-off value indicates that the chosen PCA dimensions should keep 95% of

the variance in the original data (space). The PCA's chosen dimensions are coupled with each packet classification algorithm's ranking data and used to train the learning model.

4.1.1.3 Algorithm Ranking

In this stage, the framework generates the final training data (i.e., label) for the learning model. The labels represent the target *"most-suitable"* algorithm for each ruleset. To measure the performance of algorithms on rulesets, two metrics are used: memory consumption and classification speed. Each of the available algorithms is applied to every training ruleset, involving both the ruleset and the trace set. The performance of each run is recorded and stored in a database. Table 4.4 shows a sample example of the performances (not actual performances) of three algorithms (A1, A2, A3) on three rulesets. In the current implementation, ranking can be

	Algorithms' Performance					
	Classification Time *(ms)*			Memory *(bytes)*		
Ruleset	A1	A2	A3	A1	A2	A3
Ruleset 1	30	40	100	3000	2500	5000
Ruleset 2	20	70	14	4000	4700	3000
Ruleset 3	60	70	67	2000	2500	2700

Table 4.4: Algorithms Performance on Different Rulesets

performed based on one of the following metrics: (1) time, (2) memory, or (3) a weighted combination of both time and memory. Table 4.5 shows the ranking of algorithms based on time only. The ranking is then transformed into a binary format, where "1" is assigned to the best performance (first rank case) and "0" is given to all other cases. This transforms the problem into a classification problem

that can be solved using a machine learning classification technique such as an Artificial Neural Network. Table 4.6 shows similar results as those presented in Table 4.5 with ranking based on memory consumption. The same binarization of ranks is applied on the data, which transforms the ranking to selection.

	Algorithms Ranking			Target Label		
Rulesets	A1	A2	A3	A1	A2	A3
Ruleset 1	1	2	3	1	0	0
Ruleset 2	2	3	1	0	0	1
Ruleset 3	1	3	2	1	0	0

Table 4.5: Algorithms ranking (Classification time)

	Algorithms Ranking			Target Label		
Ruleset	A1	A2	A3	A1	A2	A3
Ruleset 1	2	1	3	0	1	0
Ruleset 2	2	3	1	0	0	1
Ruleset 3	1	2	3	1	0	0

Table 4.6: Algorithms ranking(Memory Consumption)

4.1.1.4 The Learning Model

The learning model part of our framework is a generic description of the type of system that would learn from task features and algorithms' rankings. When a user of the framework applies it to an algorithm selection task, a decision has to be made on what model to use. We decided to fix the learning model to be an Artificial Neural Network. This learning model fixing doesn't mean that other models such as Support Vector Machines (SVM) or Decision Trees (DT) cannot be used. However, ANNs are widely employed in many domains and are considered "state of the art" in many situations. ANNs are extendable in many different ways

as we will show in our experiments where they can model different tasks. At each iteration, the data is presented in an example-by-example basis to the network via the input layer. The data is then passed through the network layers in a forward run until it reaches the output layer.

The error is calculated at the output layer of the network outlining the difference between the output value and the true value. Using *back propagation* [4] (chain rule differentiation), the calculated error is then propagated across the network in a backward run accordingly updating the network's *weights* to produce a closer output to the actual value. This iterative offline process is known as training.

4.1.2 Testing Phase

At the end of the training phase, the final weights of the ANN are used as a testing model (function) by which the following online prediction is performed. Choosing a different learning model would require a different training and testing process. When a new $\text{ruleset}_{(i)}$ is being tested, its respective descriptive Meta-features are extracted using the same extraction process as the previous phase. These features are then reduced in dimensionality using PCA. The trained MLP then takes the features as inputs to identify (*predict*) the most suitable algorithm. This process neither requires a trace set nor any of the other training components of the previous training process. The testing phase (Figure 4.2) is fully online.

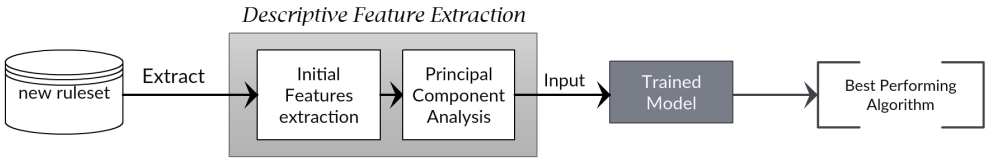

Figure 4.2: The framework's testing (recommendation) phase

4.2 Summary

In this chapter, the overall proposed framework, and the two stages comprising the framework, the learning stage, and the testing stage were described in detail. Descriptive Meta-feature extraction from incoming rulesets was explained as part of the learning phase using Meta-learning, and principle component analysis. We highlighted the standard benchmarking for the packet classification rulesets. We also described how the ANN weights are determined at the end of the learning phase and how these are used to determine the ranking of algorithms during the testing phase. In the next chapter, results based on the proposed framework will be discussed and analyzed.

Chapter 5

Results

In this chapter, results obtained based on the proposed framework introduced in Chapter 4 will be presented. First, the experimental setup and tools used will be introduced. This will be followed by preliminary investigations and results based only on simple attributes such as ruleset types and ruleset sizes. Evaluation of the proposed framework based on Meta-features will be presented and discussed.

5.1 Experimental Setup

The experiments were carried out using a Linux machine running on a Xeon 3.2GHz processor. The machine was used to generate the training data, i.e., record the packet classification algorithms' performance on the rulesets/trace sets. The original C/C++ source code was used to implement the algorithms, while the overall framework was implemented in Python (as a wrapper around the algorithms).

5.1.1 Benchmarks

The evaluation of the proposed framework was based on benchmarks generated using the ClassBench tool [32]. Twelve seed files were used to produce 108 rulesets with different sizes ranging from 1K rules to 5K rules (1-5K is a typical size range of rulesets). The seed files represent different data distributions for different types and domains of rulesets, i.e., ACL, FW, and IPC. The reader can refer to Chapter 2 for an explanation of these types of rulesets.

5.1.2 Algorithms

The packet classification algorithms that were initially used are GBSA [40] and PCIU [39] algorithms. The HybridCuts algorithm [38] was added as an additional case for memory consumption ranking. HybridCuts was not fully implemented (only preprocessing) in the work, due to missing parts of the original code. The algorithm itself is not explained in the background chapter. However, the reader can refer to [38] for an explanation of how the tree based algorithm works.

5.2 Preliminary Investigation

We conducted two separate experiments to evaluate the proposed algorithm selection framework. The first experiment was conducted to show that feature extraction of rulesets was necessary. It justifies the use of meta-features rather than relying on simple measures such as ruleset type or ruleset size. The second experiment is conducted to evaluate the proposed framework when all important features were extracted and fed to the framework along with performance metrics of the algorithms

involved.

For the first experiment, Table 5.1 confirms that there is no "single-best" algorithm based on only the ruleset type. On average, the two algorithms are almost equal on the number of times that they outperform each other. The same can also be said for the data shown in Table 5.2, especially for the rulesets in the range of $2K$ to $5K$. Clearly, there is no way to tell a priori how an algorithm will perform by only looking at the size of the ruleset, which indicates that a selection algorithm based only on the type or size of the ruleset as a selection criterion is not appropriate. It also justifies the use of the proposed meta-feature extraction technique to model the ruleset space by capturing as many of the characteristics that affect the packet classification algorithm's performance in this space, as possible.

Ruleset type	PCIU	GBSA	**Total rulesets**
Access Control List	24	21	45
Firewall	14	31	45
IP Chain	15	3	18
Total percentage	49%	51%	108

Table 5.1: Performance data "Speed" distribution (Ruleset ***type***)

	Number of rules								
Alg	1K	1.5K	2K	2.5K	3K	3.5K	4K	4.5K	5K
GBSA	12	11	7	7	6	4	4	4	3
PCIU	0	1	5	5	6	8	8	8	9

Table 5.2: Performance data "Speed" distribution (Ruleset ***size***)

An additional experiment was carried out, using the same feature data shown in Table 5.1 and Table 5.2, however, with a different performance metric: memory consumption. Additionally, HybridCuts (HC) was added to the pool of algorithms.

The addition of HybridCuts changed the class distribution from its original

semi-balanced state to an unbalanced state. Tables 5.3 and 5.4 show performance data distribution based on the new experimental parameters. It can be seen from Table 5.3 that this distribution (28% 49% 23%) is more challenging than the initial case (49% 51%). The additional complexity is due to the fact that the learning model can get biased by the majority class (PCIU in this case).

	Algorithms			
Ruleset type	HybridCuts	PCIU	GBSA	**Total rulesets**
Access Control List	16	15	14	45
Firewall	9	33	3	45
IP Chain	0	5	13	18
Total percentage	28%	49%	23%	108

Table 5.3: Performance data "Memory" distribution (Ruleset ***type***)

	Number of rules								
Alg	1K	1.5K	2K	2.5K	3K	3.5K	4K	4.5K	5K
GBSA	0	0	2	3	4	4	5	5	7
PCIU	9	9	8	7	5	5	4	4	2
HC	3	3	2	2	3	3	3	3	3

Table 5.4: Performance data "Memory" distribution (Ruleset ***size***)

5.3 Framework Evaluation

To evaluate the efficiency and robustness of the designed framework, an Artificial Neural Network (ANN) was implemented with five input units to match the chosen PCA dimensions and two output units to match the two algorithms (to select from). The ANN was trained on the extracted features along with algorithm performances. To evaluate the ANN, a *k-fold* cross validation technique [61] was used.

5.3.1 K-fold cross-validation

The data of $N = 108$ examples was divided into $k = 10$ folds. The model is then trained on $k - 1 = 9$ folds, and tested on the remaining 10^{th} fold. This training/testing process is repeated ten times until all parts are tested exactly once. This repetition ensures that each data example has been exactly once in both training and testing. The accuracy of a prediction model is a measure of how well the predictor can produce correct predictions. The accuracy is formally defined as the ratio of correctly classified instances to the number of classified instances, as shown by Equation 5.1 for binary classification (two classes only).

$$Accuracy = \frac{T_N + T_P}{T_N + T_P + F_P + F_N} \tag{5.1}$$

Note: T_P and T_N are the number of true positives and true negatives, respectively, while F_P and F_N are the number of false positives and false negatives, respectively.

5.3.2 Results using Classification Time

The model was trained to maximize its accuracy on predicting the defined output. Table 5.5 represents partial training data for algorithm selection based on classification speed. It shows data that was gathered from parts of the training process. The data is fed into the machine learning model. The first column represents the ruleset under consideration, the following five columns are the chosen PCA features, while the last two columns represent the most suitable algorithms (labeled "1"). The reader should note that the PCA features values under f1 to f5 are not to be

human interpretable. They were put for exemplification purposes only. Table 5.6

	Meta (Descriptive) Features w/PCA					Algorithms Labels	
Rulesets	f_1	f_2	f_3	f_4	f_5	PCIU	GBSA
Acl1 1K	4.84	1.45	-1.35	-0.24	0.95	1	0
FW4 1K	11.91	-2.13	-1.22	-3.06	0.83	1	0
FW1 5k	-2.79	-1.04	0.54	0.14	4.70	0	1
IPC2 2k	1.53	-0.40	-0.83	0.21	1.39	0	1

Table 5.5: Part of the training data for the ANN

shows the confusion matrix for the two-class classification problem based on classification time. We were able to achieve an average of 92.5% selection accuracy

		Prediction outcome		
		PCIU	**GBSA**	
Actual value	**PCIU**	52	3	51% of 108
	GBSA	5	48	49% of 108
		94.5% Hit rate	94.1% Hit rate	

Table 5.6: 2-Class Confusion Matrix

(100/108) on predicting the most suitable algorithm for the given unseen rulesets based on Equation 5.1. The training process involved using a momentum of 0.3, learning rate of 0.01, and 2000 epochs (number of training iterations). These ANN hyper-parameters were chosen empirically.

5.3.3 Results using Memory Footprint

To further illustrate the robustness of the framework a second experiment was carried out using the same feature data as shown in Table 5.7, but this time with memory consumption as the performance metric. Table 5.8 shows the confusion

	Meta Features w/PCA					Algorithms Labels		
Rulesets	f_1	f_2	f_3	f_4	f_5	GBSA	PCIU	HC
Acl1 1K	4.84	1.45	-1.35	-0.2	0.9	0	0	1
FW4 1K	11.91	-2.13	-1.22	-3.0	0.8	1	0	0
FW1 5k	-2.79	-1.04	0.54	0.1	4.7	0	1	0
IPC2 2k	1.53	-0.40	-0.83	0.2	1.3	0	1	0

Table 5.7: Training data for the ANN in the second experiment

matrix for the three-class problem based on memory footprint.

		Prediction outcome			
		GBSA	**PCIU**	**HybridCuts**	
Actual value	**GBSA**	28	1	1	28% of 108
	PCIU	3	47	3	49% of 108
	HybridCuts	2	3	20	23% of 108
		93.3% Hit rate	88.6% Hit rate	80% Hit rate	

Table 5.8: 3-Class Confusion Matrix

The flexibility of the framework was intended to be tested having a new performance metric. Using the same evaluation criteria, the ANN (with three output units) was re-trained using the new data. The newly trained model was then tested, where the accuracy remained in the same region as the previous experiment. Again, the accuracy is calculated as the sum of all correct classifications divided by the total number of classifications. The model achieved a selection accuracy of 88% on average (95/108), predicting the most suitable algorithm from the given three, on unseen rulesets.

5.4 Additional Work

The solution to the main problem as outlined in Section 2.4.2 of Chapter 2 consists of finding the mapping function $S(f(r))$. The function $S(f(r))$ defines the linkage between the feature space and the performance space. Finding $S(f(r))$ has shown to be useful when solving Algorithm Selection (*algorithm goodness*) based on algorithms performance criteria. The mapping function $S(f(r))$ can also be used to predict the algorithms performance itself as a real value. This prediction can be performed based on the same problem features that were extracted previously (*problem hardness*). We previously illustrated how the framework was used to tackle and solve the first task (algorithm selection). We will now use the same framework with minor model tweaks to be adapted to tackle the second outlined task (algorithm's performance prediction).

In this experiment, the objective is to predict the expected memory consumption of a certain algorithm given a ruleset instance. This prediction will result in a real-

valued number representing the amount of needed memory resources in *bytes*.

The second task is formulated as a regression format rather than the previous classification problem form. For this purpose, we used a regression-based ANN. The output layer of the same classification-based ANN that was used previously is now reconfigured having a single output linear unit instead of the 3-output softmax unit.

5.4.1 PCIU Memory Consumption Estimation

The regression ANN based model was trained on the same features data for the 108 instances (of 108 rulesets) shown in Table 5.5 and Table 5.7. However, the label is changed to being a single real-valued output, consisting of PCIU's memory consumption on rulesets. Table 5.9 shows part of the training data for the ANN in the new regression format. The ANN was trained for 1000 iterations (epochs) using a learning rate of 0.4 and a momentum of 0.2.

	Meta (Descriptive) Features w/PCA					Memory Consumption
Rulesets	f_1	f_2	f_3	f_4	f_5	PCIU (bytes)
Acl1 1K	4.84	1.45	-1.35	-0.24	0.95	92968
FW4 1K	11.91	-2.13	-1.22	-3.06	0.83	40264
FW1 5k	-2.79	-1.04	0.54	0.14	4.70	1247712
IPC2 2k	1.53	-0.40	-0.83	0.21	1.39	125296

Table 5.9: PCIU training data for the regression format

The model was evaluated using *Root Mean Square Error* measure (RMSE) [62] [63] (refer to Equation 5.2), where $\hat{y}_t$ in the equation is the predicted value for

memory consumption, while y_t is the true memory consumption, and n is the number of predictions.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_t - y_t)^2}{n}} \tag{5.2}$$

RMSE is widely used as a regression error measure. It is considered a scale dependent measure, where different data scales give different interpretations (different ranges) of the RMSE value. This scale dependency makes RMSE more suitable for comparing different models on the same data, but not very representative when observed for a single model or for different data with a different scale. To solve this issue, we normalized the target data by the $y_{t_{max}} - y_{t_{min}}$. This normalization step allows the RMSE measure to be more representative when observing a single model.

The trained ANN was able to achieve an average RMSE error of less than 0.03 on the normalized data. Table 5.10 shows a sample of the predicted values (after normalization) for PCIU's memory consumption on different rulesets. We unfortunately can't draw many conclusions from these results. However, we intended this experiment to count as a preliminary investigation for future development. The

Memory Consumption (normalized)		
Ruleset	True value	Prediction
FW4 4.5K	0.8801	0.8834
Acl2 2.5K	0.2179	0.2581
FW4 5k	0.8835	0.9121
IPC1 5k	0.5535	0.5597

Table 5.10: Normalized values for predicted memory consumption for PCIU

results in the table show how accurate the model's predictions can be versus the

true values.

5.4.2 GBSA Memory Consumption Estimation

Using the same evaluation criteria as above, the regression-based ANN model was restrained using the same feature data with changing only the label to match GBSA's memory consumption. We increased the number of iterations (epochs) to 1500 with a learning rate of 0.4 with the same previous momentum. The model achieved an average RMSE of 0.055 respectively. Based on the observed results, GBSA memory consumption seems to have a more complex relationship between the feature space and the performance space over PCIU. The mapping function $S(f(r))$ for both cases remains the same. However, the results are a little worse for GBSA. This could be solved by investigating the effect of using different models. Another approach would be using a different ANN architecture, which is more capable of handling the problem. A sample of the training data for GBSA's memory estimation model is shown in Table 5.11. A snapshot of what the model predicts is also shown in Table 5.12.

	Meta (Descriptive) Features w/PCA					Memory Consumption
Rulesets	f_1	f_2	f_3	f_4	f_5	GBSA (bytes)
Acl1 1K	4.84	1.45	-1.35	-0.24	0.95	160934
FW4 1K	11.91	-2.13	-1.22	-3.06	0.83	538330
FW1 5k	-2.79	-1.04	0.54	0.14	4.70	1229520
IPC2 2k	1.53	-0.40	-0.83	0.21	1.39	239078

Table 5.11: GBSA training data for the regression format

Memory Consumption (normalized)		
Ruleset	True value	Prediction
FW1 3K	0.5158	0.5594
FW4 5K	1.0	0.9177
FW5 1k	0.2496	0.2420
IPC2 1.5k	0.1298	0.1787

Table 5.12: Normalized values for predicted memory consumption for GBSA

The performance of the framework can be significantly contributed to the successful use of suitable descriptive features. The extracted features were highly representative, yet very general across domains. The 1^{st} feature vector was able to capture distributions of individual fields of the rulesets while keeping the rulesets' size information ($\frac{Number\ of\ occurrences\ of\ Cat_{ijk}}{Total\ Size}$). The 2^{nd} feature vector was able to capture fields correlation information (between fields) which represent the joint distribution information ($\frac{Number\ of\ common\ occurrences\ of\ Cat_{ijk}\ \&\ Cat_{i'j'k}}{Total\ Size}$). Those measures have a high correlation with algorithms performance. Therefore, their use yielded good algorithm selection results.

5.5 Summary

This chapter presented the main results of the framework when applied to the problem of algorithm selection for packet classification. We first conducted a preliminary investigation to validate the existence of an algorithm selection problem in the packet classification domain. We then showed that the problem cannot be easily solved by just knowing the size of a ruleset or the type (format) of it. We followed our investigation by extensive experiments validating the applicability of our framework on the problem. We showed that the framework is robust under dif-

ferent scenarios such as the change in the number of algorithms to select from, or the performance metrics governing the selection of packet classification algorithms. The framework showed a consistent algorithm selection accuracy of over 90% in all cases. Additionally, we showed how our framework could easily be adapted to tackle a different problem such as memory consumption estimation. We illustrated the framework's accuracy on predicting the performance of a single algorithm on an unseen ruleset based on previously seen performances on other rulesets.

Chapter 6

Conclusions and Future Directions

Many packet classification algorithms with variable performances and capabilities are available in the literature today. However, no single algorithm is guaranteed to outperform every other one in every case. In this thesis, we utilized Meta-Learning and Artificial Neural Networks (ANNs) to automate the process of algorithm selection. The developed framework was tested in different scenarios comprising combinations of different packet classification performance measures and different algorithms. Promising results were obtained when applied to solve the problem of algorithm selection for packet classification applied on a collection of 108 datasets. Using ANN as the learning model and 10-fold cross validation as the evaluation criteria, the framework was able to achieve an average accuracy of 92.5% (100/108) on predicting the most suitable algorithm that maximizes packet classification speed for an unseen dataset. The framework also showed similar results when the packet classification objective changed to minimizing memory footprint and the number of algorithms possibilities increased, achieving an average accuracy of 88% (95/108)

using the same learning model and the same evaluation criteria. Our research resulted in a novel framework for efficient, automatic packet classification algorithm selection. The framework can be used in wider industries as a tool for the automatic selection of algorithms based on the descriptive features that were extracted from the rulesets. While maintaining deployment flexibility in terms of performance metrics and the recommendation model, an empirical illustration of the models strength was carried out. The framework was able to illustrate accurate algorithm recommendations when tested in different scenarios using a machine learning approach to learning algorithms performance from previously observed cases.

6.1 Future Work

Several improvements could be introduced to further enhance the work achieved in this research. Enhancements could be made either improving the framework itself or in widening the experimental work to include other performance metrics.

6.1.1 Framework Enhancements

As a strong feature of robustness, the developed framework could accept different machine learning models, The use of different learning models such as the Support Vector Machines (SVM), or other variations of the Artificial Neural Network could be investigated to see the effect of these learning models on the accuracy of the algorithm selection.

6.1.2 Performance Metrics Extensions

Further investigations on the performance of packet classification algorithms could be achieved using different metrics other than the ones that were employed in this thesis. These metrics may involve adding single objectives such as ranking algorithms based on their scalability or update capability. Multi-objective metrics could also be used where algorithm ranking is then performed based on a weighted combination of two or more metrics.

6.1.3 Algorithm's Performance Estimation

From the additional work in the *Results* chapter, it can be seen that the framework is easily adaptable to tackle different types of problems, such as memory consumption estimation. This preliminary investigation leads to a more detailed analysis of how the framework can be suited to address such problems. Further improvements can be accomplished via using different ANN based architectures than the one used in our work. Using other models than the ANN can also be another direction of research.

Appendix A

Glossary

ML	: Machine Learning
MTL	: Meta Learning
ANN	: Artificial Neural Network
SVM	: Support Vector Machines
KNN	: K Nearest Neighbor
MLP	: Multi-Layer Perceptron
HC	: HybridCuts
PCIU	: Packet Classifcation using Incremental Updates
GBSA	: Group Based Search Algorithm
PCA	: Principle Component Analysis
CV	: Cross Validation
ACL	: Access Control List
IPC	: IP (Internet Protocol) Chains
FW	: Firewall

Bibliography

[1] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[2] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, 2006.

[3] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[5] M. T. Banday and T. R. Jan, "Effectiveness and limitations of statistical spam filters," *arXiv preprint arXiv:0910.2540*, 2009.

[6] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[7] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," 2007.

[8] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[10] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 936.

[11] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[12] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[13] R. Bellman, "A markovian decision process," DTIC Document, Tech. Rep., 1957.

[14] X. Guo and O. Hernández-Lerma, "Continuous-time markov decision processes," in *Continuous-Time Markov Decision Processes*. Springer, 2009, pp. 9–18.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[18] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*. Springer, 1995, pp. 195–201.

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[20] N. Kambhatla and T. K. Leen, "Dimension reduction by local principal component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1493–1516, 1997.

[21] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[23] D. Wolpert and W. Macready, "No free lunch theorems for search," Tech. Report SFI-TR-95-02-010, Santa Fe Ins., Tech. Rep., 1995.

[24] S. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[25] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine learning, neural and statistical classification," 1994.

[26] C. Castiello, G. Castellano, and A. M. Fanelli, "Meta-data: Characterization of input features for meta-learning," in *International Conference on Modeling Decisions for Artificial Intelligence*. Springer, 2005, pp. 457–468.

[27] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier, "Tell me who can learn you and i can tell you who you are: Landmarking various learning algorithms," in *Proceedings of the 17th international conference on machine learning*, 2000, pp. 743–750.

[28] N. Tatti, "Distances between data sets based on summary statistics," *The Journal of Machine Learning Research*, vol. 8, pp. 131–154, 2007.

[29] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Conference on Applications, technologies, architectures, and protocols for computer communication*. New York, NY, USA: ACM, 1999, pp. 147–160.

[30] D. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 238–275, 2005.

[31] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," Tech. Rep., 1998.

[32] D. Taylor and J. Turner, "Classbench: A packet classification benchmark," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 2068–2079.

[33] S. S. F. Baboescu and G. Varghese, "Packet Classification for Core Routers:

Is there an alternative to CAMs?" in *Joint Conference of the IEEE Computer and Communications*, vol. 1, April 2003, pp. 53–63.

[34] F. Yu, R. Katz, and T. Lakshman, "Efficient multimatch packet classification and lookup with tcam," *IEEE Micro*, vol. 25, no. 1, pp. 50–59, Jan. 2005.

[35] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *Micro, IEEE*, vol. 20, no. 1, pp. 34–41, 2000.

[36] G. V. S. Singh, F. Baboescu and J. Wang, "Packet classification using multidimensional cutting," in *Conference on Applications, architectures and protocols for computer communications*. New York: ACM, 2003, pp. 213–224.

[37] J. Rice, "The algorithm selection problem," 1975.

[38] W. Li and X. Li, "Hybridcuts: A scheme combining decomposition and cutting for packet classification," in *IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, 2013, pp. 41–48.

[39] O. Ahmed, S. Areibi, and D. Fayek, "Pciu: An efficient packet classification algorithm with an incremental update capability," in *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2010 International Symposium on*. IEEE, 2010, pp. 81–88.

[40] O. Ahmed and S. Areibi, "Gbsa: A group based search algorithm for packet classification," in *Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2011, pp. 1789–1794.

[41] S. S. V. Srinivasan and G. Varghese, "Packet classification using tuple space search," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. New York: ACM, 1999, pp. 135–146.

[42] S. S. P. Warkhede and G. Varghese, "Fast packet classification for two-dimensional conflict-free filters," in *INFOCOM 2001. Conference of the IEEE Computer and Communications Societies*, vol. 3, 2001, pp. 1434–1443. [Online]. Available: 10.1109/INFCOM.2001.916639

[43] H. L. J. Li and K. Sollins, "Scalable packet classification using bit vector aggregating and folding," vol. MIT-LCS-TM-637, April 2003.

[44] T. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *SIGCOMM Comput.Commun.Rev.*, vol. 28, no. 4, pp. 203–214, 1998.

[45] J. T. D. Taylor and F. Pt, "Scalable packet classification using distributed crossproducting of field labels." IEEE INFOCOM, 2005, pp. 269–280.

[46] R. King, C. Feng, and A. Sutherland, "Statlog: comparison of classification algorithms on large real-world problems," *Applied Artificial Intelligence an International Journal*, vol. 9, no. 3, pp. 289–333, 1995.

[47] P. Brazdil, C. Soares, and J. Da Costa, "Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[48] A. Kalousis and T. Theoharis, "Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection," *Intelligent Data Analysis*, vol. 3, no. 5, pp. 319–337, 1999.

[49] L. Rendell, R. Sheshu, and D. Tcheng, "Layered concept-learning and dynamically variable bias management." in *IJCAI*, 1987, pp. 308–314.

[50] C. Blake and C. J. Merz, "{UCI} repository of machine learning databases," 1998.

[51] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2944–2952.

[52] C. Thornton, F. Hutter, H. Hoos, and K. Leyton, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[53] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[54] J. Kanda, A. de Carvalho, E. Hruschka, C. Soares, and P. Brazdil, "Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features," *Neurocomputing*, 2016.

[55] M. G. Lagoudakis, M. L. Littman, and R. Parr, "Selecting the right algorithm,"

in *Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation, Cape Cod, MA*, 2001.

[56] H. Guo, "Algorithm selection for sorting and probabilistic inference: a machine learning-based approach," Ph.D. dissertation, Citeseer, 2003.

[57] V. Estivill-Castro and D. Wood, "A survey of adaptive sorting algorithms," *ACM Computing Surveys (CSUR)*, vol. 24, no. 4, pp. 441–476, 1992.

[58] P. He, G. Xie, K. Salamatian, and L. Mathy, "Meta-algorithms for software-based packet classification," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 2014, pp. 308–319.

[59] Q. Song, G. Wang, and C. Wang, "Automatic recommendation of classification algorithms based on data set characteristics," *Pattern recognition*, vol. 45, no. 7, pp. 2672–2689, 2012.

[60] D. Hoaglin, F. Mosteller, and J. Tukey, *Understanding robust and exploratory data analysis*. Wiley New York, 1983, vol. 3.

[61] R. Kohavi, "A study of cross-validation/bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2, 1995, pp. 1137–1145.

[62] J. S. Armstrong and F. Collopy, "Error measures for generalizing about forecasting methods: Empirical comparisons," *International journal of forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[63] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast

accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.