B.Comp. Dissertation

Feature Subset Selection using Reinforcement Learning

By

Han Liang Wee, Eric

Department of Computer Science

School of Computing

National University of Singapore

AY 2016/2017

B.Comp. Dissertation

Feature Subset Selection using Reinforcement Learning

By

Han Liang Wee, Eric

Department of Computer Science

School of Computing

National University of Singapore

AY 2016/2017

Project No: H041790
Advisor: Assoc. Prof. Martin Henz, Dr. Liu Yong(A*STAR/IHPC)

Deliverables:
Report: 1 Volume
Program: 1 Git Repository

Abstract

Feature subset selection(FSS) is a significant problem in a litany of fields across Machine Learning(ML). FSS is an integral preprocessing step in the model building process. ML algorithms' accuracy often relies heavily on the quality of the data produced by the preprocessing step. FSS is the process of identifying the optimal number of features before learning. Moreover, FSS is a combinatorial optimization problem, searching exhaustively across all the subsets is an intractable problem. After studying the literature, we cannot find a method that addresses feature subset optimality and striking a right balance between prediction performance and feature selection speed. Hence, we proposed three wrapper based frameworks based on reinforcement learning to address this goal. Framework 1 and 2 are based on Deep Q-Learning and Neural Network Architecture respectively. We validated all three solutions on a preliminary set of data and found that Framework 3 performs well. Framework 3 uses a novel representation of the problem that was used with a modified version of the traditional Q-Agent, called Fast Q-Agent. We were experiencing some problem with the code and we have invalidated the Final results of the experiment.

Subject Descriptors:

10010294 Neural networks
10010321 Machine learning algorithms
10010329 Q-learning
10010336 Feature selection
10010178 Artificial intelligence

Keywords:

Implementation, Theory & Algorithms, Artificial Intelligence, Data Mining, Machine Learning

Implementation Software and Hardware:

Fedora 25 GNU/Linux 4.9.13-201.fc25.x86_64
Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz
Python 3.5.3

Fedora 25 GNU/Linux 4.9.13-201.fc25.x86_64
2 x Intel(R) Xeon(R) CPU X5680 @ 3.33GHz
Python 3.5.3

Acknowledgement

The author would like to thank Agency for Science Technology and Research (A*STAR), specifically for providing this opportunity to work on this project. Specifically, we like to thank Dr. Liu Yong of Institute of High Performance Computing (IHPC) for the guidance provided and for mentoring this project. We also like to extend thanks to Prof. Martin Henz of NUS and Dr. Rick Goh of IHPC for their overseeing this project.

We also like to thank the artificial intelligence group in IHPC and Prof. Lee Wee Sun for their advice. Lastly, we thank everyone who has contributed to this project but were not mentioned.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

1.1 Machine Learning

Machine Learning(ML) is a sub-field of Artificial Intelligence that allows computer systems to learn(Burdett, Burkhardt, Cumming, Hunter, Hurvid, Jaworski, Ng, Scheer, Southall, & Vella, 2008). There are many different approaches proposed in the literature to achieve learning systems; they are not limited to reinforcement learning, artificial neural networks, statistical learning methods, clustering and many others(Russell, Norvig, & Intelligence, 1995). We are currently living in an era of big data, where the rise of com-

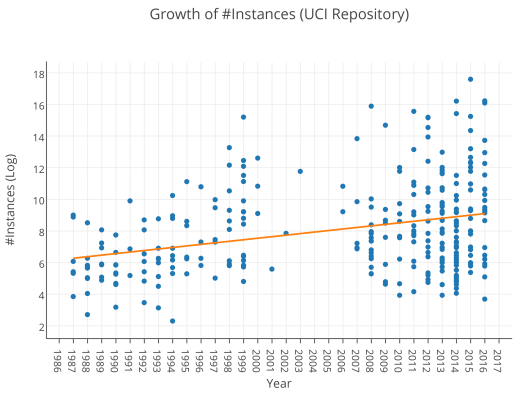

Figure 1.1: Instances Growth

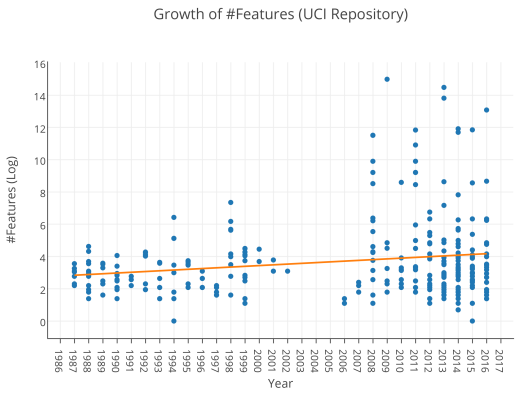

Figure 1.2: Features Growth

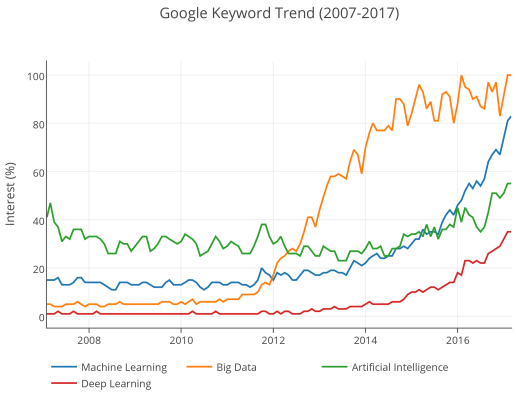

Figure 1.3: Google Trends

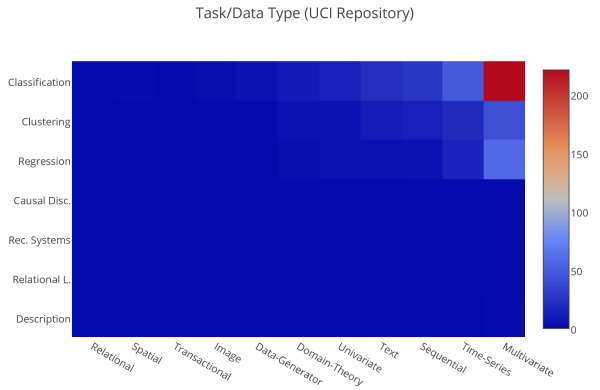

Figure 1.4: Heatmap of Task and Data Type

putational power has allowed entities of varying industries to collect high dimensional datasets, with an astronomical number of points. Figure 1.1 and 1.2 illustrate this trend by plotting the growth of both a number of instances and features in the popular UCI machine learning repository respectively(Lichman, 2013). With this advent of Big Data, entities can no longer afford to hire analysts or consultants to manually sieve through the data, thereby producing meaningful insigts(Lewis, Zamith, & Hermida, 2013). Increasingly, they recognize the need to automate analytics, adopting fully automated or hybrid approach. Hence, the demand for machine learning tools has risen, causing a list of commercial and open-source entities to grow. Figure 1.5 plots the interest of popular ML related keywords from 2007 to 2017 in the Google search engine(Google, 2017). From the graph, we can observe an increasing interest in the topic of machine learning. Moreover, many popular learning systems/tools have risen amongst novices, such as RapidMiner, DataRobot, BigML, Google Cloud Prediction Api and AutoML(Blog, Jain, Kaushik, Gupta, & Shaikh, 2017).

In Machine Learning(ML), learning algorithms learn patterns in data, to perform(typically) regression or classification tasks. The determination of tasks will depend on the nature of the problem and the dataset. Figure 1.4 is a heat map which plots tasks versus the datatype of all the datasets in UCI ML repository. It is clear that overwhelmingly, the

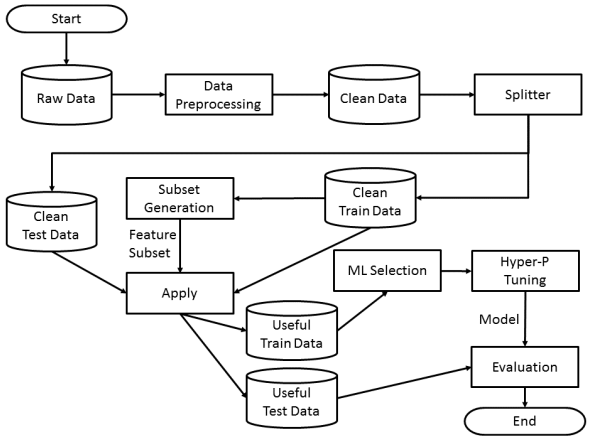

Figure 1.5: Process for model building

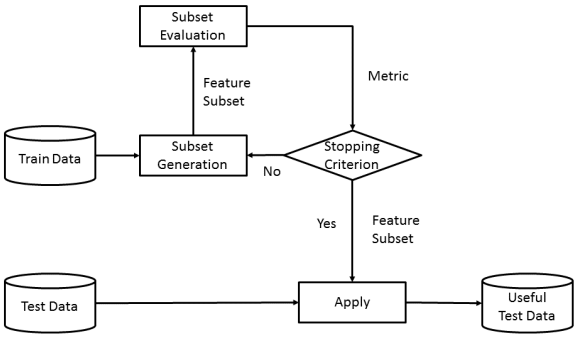

Figure 1.6: Process for Feature Selection

default task is classification. The goal of ML is to build a model, to perform prediction. To obtain the model, data scientists would need to create the model. In literature and industry, the steps required to build a model is fairly general, a generalized workflow of model building is presented in Figure 1.5.

Before the data can be used in any ML algorithm, it should be preprocessed, which includes Feature Engineering(FE). FE is the process where features are reinforced or abridged in a dataset. FE is a crucial step in the model building process, especially when applying ML on high dimensional data, due to the problem is known as the curse of dimentionality(John Lu, 2010). High dimensionality is where data in high dimensional space tend to be sparse and contains more opportunities to be affected by noisy, irrelevant redundant or noisy data, adversely affecting ML algorithms. Moreover, with a high dimensional dataset, the ML model tend to overfit, which is not desired. Hence, the efficacy of the ML model depends heavily on the representation and quality of the training data(Hall & Smith, 1998Kumar & Minz, 2014).

In literature, FE encompass both feature extraction and feature selection(Li, Cheng, Wang, Morstatter, Trevino, Tang, & Liu, 2016). Feature Extraction aims to derive features from the original set of features to improve the model. Feature Selection is the process of removing inadequate, extraneous and irrelevant features from the data(Hall &

Smith, 1998). In our discussion, we will not study Feature Extraction but study only Feature Selection exclusively.

1.2 Feature Selection

Feature Selection(FS) is also known as variable selection, attribute selection or variable subset selection in ML and Statistic communities(Li et al., 2016). In literature, it is understood that FS can potentially speed up both the learning and prediction processes for an ML algorithm(White, 2016). Moreover, FS can also simplify and enhance generalization of the ML model, allowing easier interpretation by scientists(James, Hastie, Witten, & Tibshirani, 2013Bermingham, Pong-Wong, Spiliopoulou, Hayward, Rudan, Campbell, Wright, Wilson, Agakov, Navarro, & Haley, 2015). In summary, we chose to study FS as it is not only an important technique but it is also a frequently used and indispensable part of the ML process(Kalousis, Prados, & Hilario, 2007).

Before automatic means of feature selection, an expert in the field would manually choose features using the domain-specific knowledge that he/she possesses. However, this process is highly subjective, slow and costly. As the trend continues, such manual methods become more and more intractable(Zhang & Hu, 2005) as both the size of data and the dimensional space of the data has dramatically increased. Hence, there is a need to perform FSS in automatically, without human intervention or support.

Through literature, the general procedure for feature selection has four key steps: Subset generation, Evaluation of Subset, Stopping Criteria, Result Validation(Kumar & Minz, 2014). The process is visualized in Figure 1.6, shown below.

Chapter 2

Literature Review: State of the Art

In this Chapter, we will discuss the state of the art Feature Selection Algorithms. We would first discuss how FS is categorized and sorted in literature.

Firstly, we would like to distinguish Feature Ranking from Feature Subset Selection. Feature Ranking(FR) is the process where features are ranked, but an explicit subset is not chosen. Feature Subset Selection(FSS) is the process where an explicit feature subset is chosen. In the literature, Feature Selection is sometimes loosely used interchangeably with Feature Subset Selection. In our discussion, when FS is mentioned, it will mean both FR and FSS. Hence, FS algorithms can be classified into two categories by the result they return, FR and FSS.

Additionally, FS can be distinguished by the labels on which it operates; they are classified as Supervised, Unsupervised and Semi-Supervised(Li et al., 2016). Supervised FS algorithms are typically used for classification and/or regression problems. It utilizes the class labels along with the data points to perform FS. Unsupervised FS algorithms are typically designed with clustering problems in mind. It does not utilize the ground truth to perform FS. Lastly, we have Semi-supervised which operates in the condition when the dataset contains both data points with and without their corresponding ground truth.

Lastly, FS algorithms can also be distinguished by the search strategy they employ, namely: Filter, Wrapper, and Embedded(Kumar & Minz, 2014Hall & Smith, 1998Guyon & Elisseeff, 2003Guyon, Gunn, Ben-Hur, & Dror, 2004).

2.1 Filter Strategy

Filter based methods, as the name suggests filters redundant features from the data. Algorithms in this category perform FSS independently from the ML algorithm, as illustrated in figure 2.1. Instead, filter based methods use a metric to be a surrogate to ML algorithm. Commonly used metrics include similarity, statistical, information theoric measures. Usually, algorithms in this category ignore the fact that the selection of features may depend on the learning algorithm(Janecek, Gansterer, Demel, & Ecker, 2008) and typically ignore each features' dependence on each other. There is a host of information theoric methods, which started with MIM(Lewis, 1992a) and was concluded with a generic framework called CIFE(Lin & Tang, 2006a). In the similarity-based methods, reliefF(Robnik-Šikonja & Kononenko, 2003a) is a very popular method and is widely used, compared and tweaked in literature. We also have CFS which is based on statistical correlation(Hall & Smith, 1999a). The biggest advantage of algorithms in this category is that it finds a reasonably good feature subset quickly. Interestingly some methods use reinforcement learning, most notably the model proposed by Romaric(Gaudel & Sebag, 2010), which spawned several papers which proposed improvements to his work(Fard, Hamzeh, & Hashemi, 2012aFard, Hamzeh, & Hashemi, 2012bFard, Hamzeh, & Hashemi, 2013).

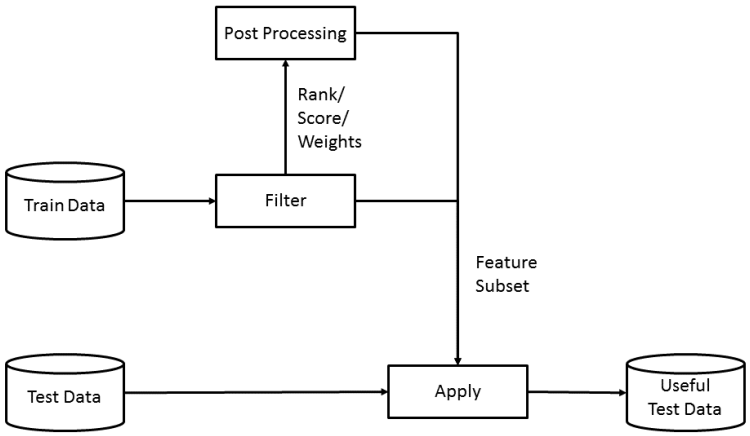

Figure 2.1: Typical process for filter-based methods

2.2 Wrapper Strategy

Wrapper-based methods, 'wrap' around a specific ML algorithm, searching for the optimal feature subset adapted for the model. Because it runs the ML algorithm as part of its process and relies on the ML algorithm's report on accuracy, such methods are typically slow. However, there are merits in using wrapper based approaches as it considers the dependence of feature subset on the machine learning algorithm(Janecek et al., 2008). Traditional methods include forward and backward selection(Guyon & Elisseeff, 2003). Forward selection, for instance, iterates from an empty set and incrementally chooses the best feature to be added.

2.3 Embedded Strategy

Embedded methods are ML algorithms that have FSS as part of their training process. Methods that fall into this category are CART(Lewis, 2000) decision trees and even ML algorithms that use kernels that have parameters to decide on the feature subset. An

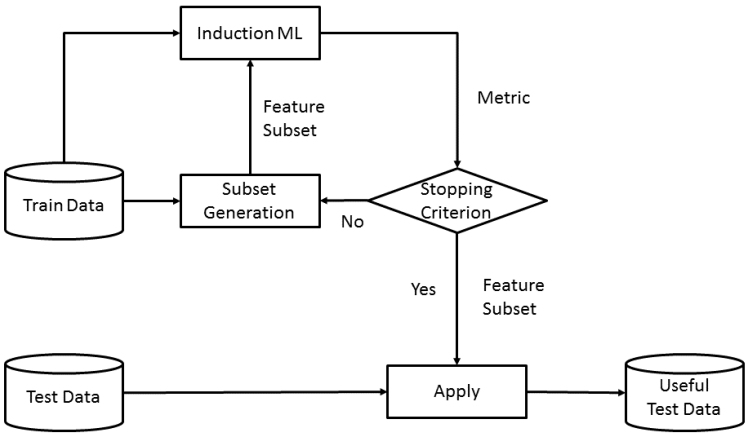

Figure 2.2: Typical process for wrapper-based methods

example of such a kernel is Radial Basis Function-Automatic Relevance Detection(RBF-ARD) in the context of Gaussian Processes (Cawley & Talbot, 2010). In our paper, we will not put our focus on embedded methods as they are unique to the machine learning algorithm.

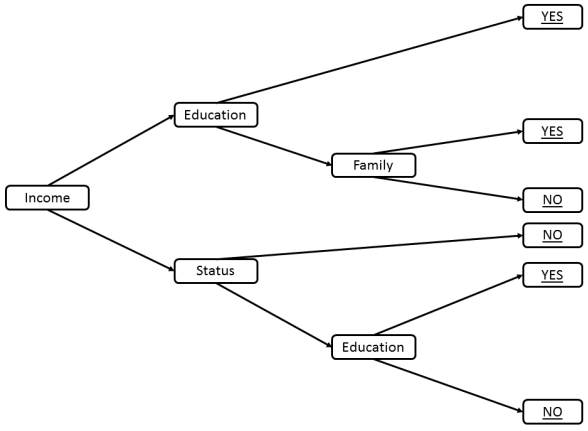

Figure 2.3: A Decision Tree

Chapter 3

Problem

In our discussion of the problem of Feature Selection, we will use the following conventions. A dataset is denoted D with its set of corresponding features to be $F = \{f_0, \ldots, f_N\}$. Hence, an instance $x \in D$ is such that x_i is a instance of $f_i, x_i \in f_i$. Any Machine Learning algorithm paired with some evaluator, called induction algorithm in our discussion for disambiguating, that operates on the dataset is denoted as $A : S \times D \to \mathbb{R}$, where $S \subseteq F$. The output of A is some metric in $\mathbb{R}$ where $A(S_a, D) > A(S_b, D) \iff S_a \succ S_b$. We also note that the Machine Learning algorithm used will limit the hypothesis space that we are searching. We also let n to denote the number of instances in D and m to denote the number of features in D.

3.1 Feature Ranking Problem

Feature Ranking problem is the problem of scoring all F features of a dataset D. In our discussion, we consider a ascending scoring metric, where $f_a > f_b$ means that the feature f_a is 'better' than its brethren f_b. We do not attempt to justify the definition of 'better' as it is dependent on the Feature Ranking Algorithm. We observe that the Feature Ranking problem can be reduced to Feature Subset Selection problem. Intuitively, a ranking of a

given features F can be reduced to a subset of features by greedily considering all possible subset of features in the order that is given.

3.1.1 Reduction

Given two subsets A and B, a set of functions F from the 2 solution spaces, $A \leq_F B$ if

$$\exists f \in F : \forall x \in A \iff f(x) \in B$$

We now give a function f to reduce a Feature Ranking problem to Feature Subset Selection problem.

Algorithm 1 Reducing Feature Ranking to Feature Subset

Require: x - m Feature Scores(ascending)

Require: m - Number of Features

Require: A - Induction Algorithm

Require: D - Training Dataset

$a_{\text{best}} \Leftarrow 0$

$S_{\text{best}} \Leftarrow \emptyset$

for S_i - Feature Subset with i highest scores in x, $\forall i \in [1, m]$ **do**

$a \Leftarrow A(S_i, D)$

if $a > a_{\text{best}}$ **then**

$a_{\text{best}} \Leftarrow a$

$S_{\text{best}} \Leftarrow S_i$

end if

end for

return S_{best}

We also note that the reduction from Feature Ranking problem to Feature Subset Selection problem is deterministic and runs in time of $\mathcal{O}(m \times f(m, n))$ where $\mathcal{O}(f(m, n))$ is

the runtime of the algorithm A. However, this is not interesting as it does not give us any guarantee over the solution, only a means to reconcile these two problems. Henceforth, we consider Feature Ranking to be an equivalent problem to Feature Subset Selection.

3.2 Feature Subset Selection Problem

Feature Selection problem is the problem of finding a feature subset $S \subset F$ of a dataset D which is the 'best'. Similarly, we do not attempt to define 'best'. Additionally, Feature Subset Selection can be visualized as undirected graph search problem. We represent the problem as a undirected graph G:

- Vertex V represent a feature subset S which is equivalent to a bit vector of size m with each bit at i representing f_i. The encoding is as follows: 1 means that the feature $f_i \subset S$ is selected, 0 means $f_i \not\subset S$ is not selected.
- Edge E connects two vertices if their bit vectors, V_a, V_b satisfy the condition $h(V_a, V_b) = 1$. h is the function computing the hamming distance between the 2-bit vectors. Equivalently, the feature subsets S_a, S_b can be translated to each other by adding or deleting one feature.

Hence, we want to find a feature subset in the graph G that is the 'best'. We will discuss the intractability of the problem in later chapters.

3.2.1 Definitions

In this section, we attempt to define the term 'best' that we have introduced earlier. There are many definitions of 'best' in literature, and some of the definitions were discussed earlier. We will only focus on the very general definition of relevance, found in the literature. In our definition and discussion, we assume that features and labels are

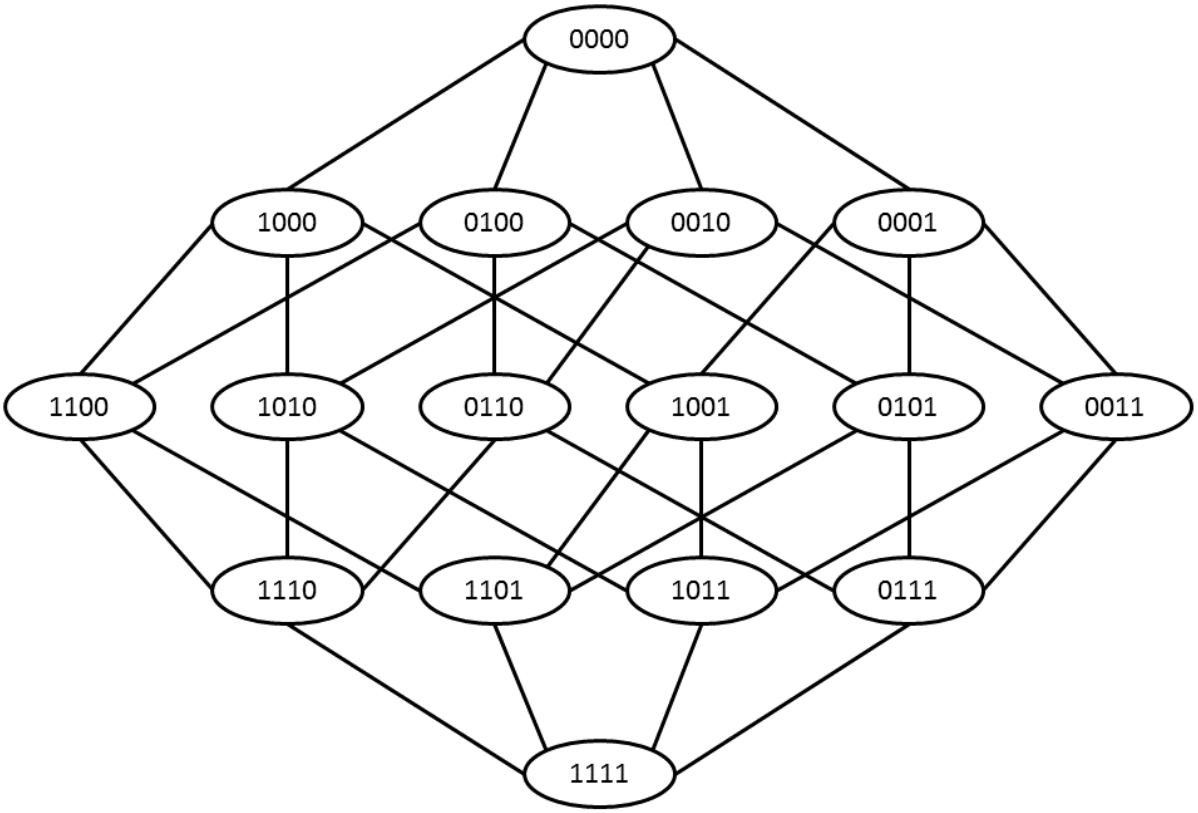

Figure 3.1: Graph representing a Feature Subset Selection Problem

booleans and have no noise. We note that our definitions and conclusions should extend to the cases without the assumptions. The definitions and examples are adapted from (Kohavi & John, 1997).

Definition 1 (Feature Relevance(Almuallim & Dietterich, 1994)) *A feature f_i is relevant to the concept C if f_i appears in every boolean formula that represents C and irrelevant otherwise.*

Definition 2 (Feature set Relevance) *Consquently, a feature subset S is relevant to a concept C if $\forall f \in S$ are all relevant.*

Definition 3 (Feature set Optimality) *A feature subset S_{opt} is optimal if and only if*

$$S_{opt} = \min\{s_{opt} : \text{argmax}_{s \in \mathcal{P}(F)} A(s, D)\}$$

where min *is defined on the number of features in the feature subset. Similarly, we do not commit to a particular evaluator in A.*

With those definitions, we now investigate the relationship of Relevance and Optimality in feature subsets.

Theorem 1 (Relevance does not imply Optimality) *Let* $\forall f_i \in F : f_i \in \mathbb{B}$ *for which has uniform distribution. We consider a dataset with* $m \geq 3$ *features. We also assume that the target concept is* $f(x_1, \ldots, x_m) = x_1 \vee (\bigwedge_{i=2}^{m} x_i)$*. Hence,* $\forall f_i$ *are relevant to the concept. If the hypothesis space is the space of monomials,* $\bigwedge x_i$*, then the only optimal feature would be* f_1*. Adding any other feature will decrease the accuracy. Hence, Relevance does not imply Optimality.*

3.2.2 Conclusion

Hence, any feature selection algorithm that does not consider the induction algorithm cannot guarantee optimality. Additionally, Optimality does not imply Relevance(Kohavi & John, 1997). Even though feature selection discards irrelevant features, it can result in a decrease in performance. Additionally, Isabelle Guyon has done an empirical study on multivariate and univariate distributions, and concluded that methods that score features independently of other features are at a loss to determine the optimal feature subset(Guyon & Elisseeff, 2003).

Henceforth, we conclude that optimal feature subset selection is a difficult problem. In literature, there are a lot of work done that tackle relevance but not much in algorithms that tackle optimality. Traditional graph search methods have been applied to feature subset selection, but due to the growing instance and feature size, it quickly becomes intractable. Feature selection methods that tackle relevance are popular because they are fast and provide a good performance compared to feature selection methods that tackle optimality. We want a method that addresses feature subset optimality and strikes a

balance between prediction performance and feature selection speed. We also note that the algorithm must not be intractable over large datasets.

3.3 Optimal Feature Subset Selection Problem

Finally, we define the Optimal Feature Subset Selection Problem that we want to solve in our paper. We note that in the earlier chapter we have seen that majority of the problems are classification problems. Additionally, since our optimality definition is based on the induction function A, naturally we consider the wrapper strategy. Wrapper strategy requires the induction algorithm, and it uses the labels in the dataset. Henceforth, our algorithm is a Supervised algorithm that employs the wrapper strategy.

As a result of the choice to tackle classification problems, we will use the accuracy metric as our definition of 'best'. Accuracy is well defined in literature and is as follows:

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} 1(y_i = \hat{y}_i)$$

We now define the problem that we are interested in. Given a dataset D with its set of features $F = \{f_0, \ldots, f_N\}$. Given an induction algorithm $A(S, D), S \subseteq F$ that produces a accuracy. Therefore, the objective is to find the optimal subset S_{opt} such that

$$S_{\text{opt}} = \min\{s_{\text{opt}} : \text{argmax}_{s \in \mathcal{P}(F)} A(s, D)\}$$

where min is defined on the number of features in the feature subset.

3.4 Search Space

3.4.1 NP-Hard

We also observe that the search space for the problem is $\mathcal{P}(F) = 2^m$ which is exponential in the size of the number of features m. This search problem can be said to be intractable

as it would require searching exhaustively, having a complexity of $\mathcal{O}(2^m \times f(m,n))$ where $\mathcal{O}(f(m,n))$ is the runtime of the algorithm A. Given a small dataset like Spambase(Lichman, 2013) with 57 features, assuming that the induction function $A(S,D)$ computes the score in a short amount of time(bounded), in 1 nanosecond, a deterministic machine would take 9.13 years to find the optimal solution by searching exhaustively. The largest dataset on UCI's machine learning repository is URL Reputation, which has an astounding 3231961 features. Hence, the search space for this dataset would be approximately $10^{10^{5.988075883858190}}$, taking about 5×10^{972897} millennias to compute. Clearly, the problem is NP-hard and had been shown in literature(Amaldi & Kann, 1998). We clearly see that using brute-force is infeasible and foolish. In this section, we will discuss similar problems that also has an intractable search space.

3.4.2 Reinforcement Learning

Reinforcement Learning has been recently used to solve large and intractable problems. Most notably, Google Deepmind's recent success in the design of AlphaGo, which successfully navigated a search space of 250^{150} possible states and won a human world champion in the chess game Go(Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, & others, 2016). We opine that FSS has a large state search space, similar to the chess game Go. Encouraged by their success in a similar problem, our objective is to investigate if reinforcement learning techniques can be applied to FSS to find the optimal feature subset.

3.4.3 Scope

Although the search space is intractable, we also note that there are some search space sizes for which the problem can be easily solved using brute force. We use a more realistic calculation for the time taken for the induction algorithm A to compute accuracy, 1

second. We want to find the n for which brute force can complete in 1 day. A quick calculation $2^n \times 1 = 3600 \implies n = 16.3$. Henceforth, we will not consider datasets of size smaller than 20.

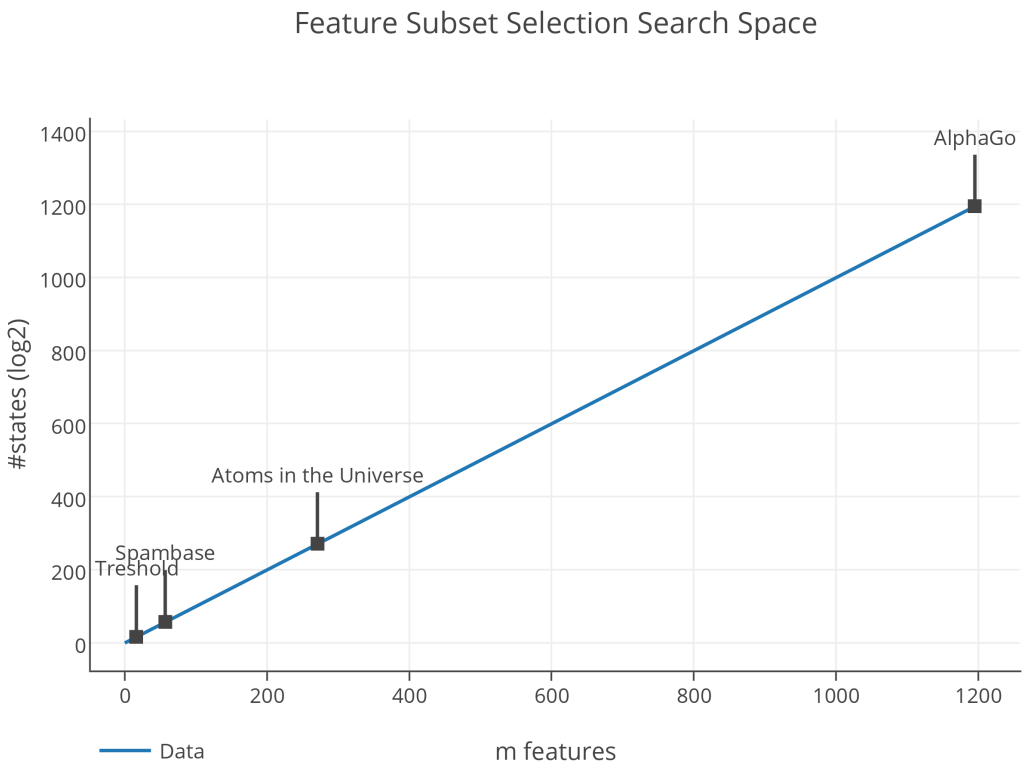

Figure 3.2: Order of Complexity for FSS Problem

Chapter 4

Proposal

4.1 Reinforcement Learning

Reinforcement learning(Sutton & Barto, 1998) is an area of ML where agents choose actions in a Markovian environment to maximize cumulative reward. For any reinforcement agent, we need to define the following:

- **States:** environment in which the agent will act.
- **Actions:** possible actions that the agent can take.
- **Transition function:** rules for the agent to transit to another state after taking an action.
- **Reward function:** rules defining current reward or penalty received by the agent after taking an action.
- **Observations:** rules defining what the agent can observe.

In our application, we have chosen a specific model-free reinforcement learning agent: Q-Learning. Since Q-Learning(Watkins & Dayan, 1992) is model-free, we do not need to define any utility function, Q-Learning learns the action-value function that gives the

optimal policy Π_{opt}. It is widely accepted in literature that the Q-Learning bellman update from state $s_t \xrightarrow{a_t} s_{t+1}$ is given in equation 4.1. Given states S, set of actions A, learning rate α, discount factor γ and reward function $r : S \times A \times S \to \mathbb{R}$:

$$\begin{aligned} Q(s_t, a_t) = Q(s_t, a_t) \\ + \alpha(r(s_t, a_t, s_{t+1}) \\ + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \end{aligned} \tag{4.1}$$

In our study we will consider the epsilon-greedy strategy to choose actions from the Q function. The strategy is explained below in equation 4.2, selecting an action a_t in state s_t, where the rational numbers $r, \epsilon \in [0, 1]$. As the Q-Learning agent runs, the ϵ is decayed exponentially, allowing the agent to be 'more greedy' as time passes. By choosing the epsilon-greedy strategy, we allow Q-Learning to converge to the optimal policy(Watkins & Dayan, 1992). Moreover, it strikes a balance between explore vs. exploit(EvE)(Tokic & Palm, 2011).

$$\epsilon\text{Greedy}(s_t, r) = \begin{cases} \text{random}_{a \in A} & r < \epsilon \\ \text{argmax}_a Q(s_t, a) & r \geq \epsilon \end{cases} \tag{4.2}$$

4.2 Framework 1

Our proposed framework is as follows:

- **States:** each state represents a subset of features.
- **Actions:** actions to add/remove features.
- **Transition function:** depending on the action taken, the appropriate feature is added/removed.
- **Reward function:** see discussion and equation 4.3.

- **Observations:** each state.

Firstly, we consider the FSS problem of finding a feature subset $S \subseteq F$ where the $|F| = N$. We define that at each action, the agent makes some choice on a particular feature f_i, to accept(added to S) or reject(removed from S) the feature. With that definition, it is trivial to see that N actions are sufficient for the agent to reach S_{opt} from any S. Hence, the agent will reach the terminal state after it performs N actions.

With those assumptions, it is possible to define a reward function such that the optimal policy Π_{opt} is a policy such that $s \xrightarrow[N]{\Pi_{\text{opt}}} s_{\text{opt}}$ finds the optimal subset after N actions. We also let the worse-case non-optimal policy Π_{nopt} such that $s \xrightarrow[N]{\Pi_{\text{nopt}}} s_{\text{nopt}}$. In our argument, we assume that $0 \leq r(s_t, a_t, s_{t+1})$ and the machine epsilon is given to be some small constant $u \in \mathbb{R}$. Let P to denote the set of possible sequences of state and actions taken in one episode and $R(p)$ to be the cumulative rewards collected by an agent on path p. Let P_{opt} and P_{nopt} to be the set of paths that contains S_{opt} and S_{nopt} as the terminal state respectively. We define p and p' as follows.

$$p = \text{argmin}_{p \in P_{\text{opt}}} R(p)$$

$$p' = \text{argmax}_{p \in P_{\text{nopt}}} R(p)$$

Q-Learning seeks to find the policy $\Pi_{\text{opt}} : \forall_s \pi_{\text{opt}}(s) = \text{argmax}_a Q(s, a)$. Hence, for Π_{opt} to always find the optimal S_{opt}, it is necessary that $R(p) > R(p')$. We want to define a reward function to satisfy that condition. Let $z(s_i)$ to be some arbitary scoring function such that $s \prec s' \implies 0 \leq z(s) < z(s') \leq 1$. Hence, we find the function $f(z(s))$.

$$r(s_t, a_t, s_{t+1}) = \begin{cases} f(z(s_{t+1})) & t+1 = N \\ z(s_{t+1}) & t+1 < N \end{cases}$$

For brevity, we represent scores at t with ρ_t and the rewards $z(s_{\text{opt}})$ and $z(s_{\text{nopt}})$ as ρ and ρ' respectively. We also assume that ρ' that can be distinguished from ρ, $\rho' + u = \rho$. We note that for any arbitary ρ_t, $0 \leq \rho_t \leq \rho$. Since u is small, $\frac{\delta f}{u} \approx f'$.

$$
\begin{aligned}
& R(p) > R(p') \\
& \implies \rho_1 + \ldots + \rho_{N-1} + f(\rho) > \rho'_1 + \ldots + \rho'_{N-1} + f(\rho') \\
& \implies 0 + f(\rho) > \rho + \ldots + \rho + f(\rho') \\
& \implies f(\rho) > (N-1)\rho + f(\rho') \\
& \implies f(\rho) - f(\rho') > (N-1)\rho \\
& \implies f(\rho) - f(\rho') > u(N-1) + (N-1)\rho' \\
& \implies \frac{f(\rho) - f(\rho')}{u} > N - 1 + \frac{N-1}{u}\rho' \\
& \implies f'(\rho') > N - 1 + \frac{N-1}{u}\rho'
\end{aligned}
$$

We rewrite the variable $\rho' \longrightarrow z$, and perform integration over the expression.

$$
\begin{aligned}
& f'(z) > N - 1 + \frac{N-1}{u}z \\
& \implies f(z) > \int N - 1 + \frac{N-1}{u}z \, \mathrm{d}z \\
& \implies f(z) > (N-1)z + \frac{N-1}{2u}z^2 + K
\end{aligned}
$$

Since we are using R to compare between 2 different paths, the exact value of f does not matter. Hence, we can just assign $K = 0$ for simplicity of computation. Henceforth, the reward function is given in equation 4.3. We also observe that our definition of r is both bounded and deterministic, not affecting any optimal convergence properties of Q-Learning(Watkins & Dayan, 1992Melo, 2001) and its derivatives. In order to simplify the following equation, z is $z(s_{t+1})$.

$$
r(s_t, a_t, s_{t+1}) = \begin{cases} (N-1)z + \frac{N-1}{2u}z^2 & t+1 = N \\ z & t+1 < N \end{cases} \tag{4.3}
$$

4.2.1 Model 1

The intuitive idea of this model is that the agent will choose any feature to add and to remove at each step. We consider a model where the states are directly representing the feature subset. Hence, there are 2^N states in the model. At each state, the agent can choose from $2N$ actions: $\{a_0, \ldots, a_{N-1}, \ldots, a_{2N-1}\}$.

- **States:** each state is a particular feature subset, modelled as a binary vector with size N. For example, binary vector $(0, 1, 1, 0, 1)$ represents the feature subset $\{f_1, f_2, f_4\}$ in a FSS problem with $N = 5$.
- **Actions:** $a_i, i < N$ adds the feature f_i to the feature subset, where $a_i, N \leq i < 2N$ removes the feature f_i from the feature subset. If the feature is already removed/existed, then the action does nothing to the feature subset(ie. no transition).
- **Transition function:** following the description of each action, it is the appropriate adding/removing of features specified through the action.
- **Reward function:** as specified in the framework

Figure 6.3 shows a visual representation of the model proposed for two features. Note that the directed cycles drawn in the diagram represent actions that perform no transition as the feature already existed or removed. For clarity, only the actions from the empty state are labeled in the figure. Additionally, we let the agent start from a random state and will perform N actions before reaching the terminal state. We also note that the state graph contains cycles.

4.2.2 Agent

As explained in the chapters before, this problem is a challenging problem. As observed from the exponential size of the number of states, it is not possible to use the vanilla

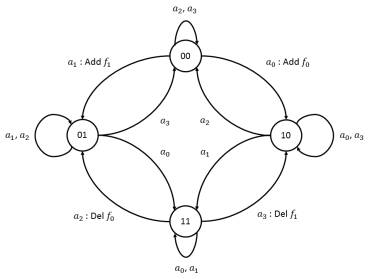

Figure 4.1: Graphical representation of model 1

version of Q-Learning that uses tables to store data. If we have chosen to store the Q-Function in tables, the table would need a size of $|S| * |A|$. In the models 1 and 2, the size required would be $2^{N+1} \times N$ and 2^{N+2} respectively. Additionally, for Q-Learning with epsilon strategy to converge to the optimum policy requires each state and action to be visited infinitely many times. Given the exponential size of the states on the size of features N, we need to find methods to approximate the Q-Table.

Deep Q-Network

Deep Q-Network(DQN) by is a variant of Q-Learning that uses a deep neural network to approximate the Q-Table(Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, & Riedmiller, 2013) proposed by Google Deepmind. A neural network is used as a non-linear function approximator to estimate the Q-Table. Instead of updating the Q-Table using the bellman update given in equation 4.1, the neural network is trained by minimizing a sequence of loss functions that changes at each iteration. Deepmind's researchers reported success in using DQN to achieve gameplay at a level that was comparable to that of a professional human games tester across the set of 49 Atari games(Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, & others, 2015). The most remarkable were that the models used by DQN is model free and are all using a general framework without game specific modifications. Most impressively, DQN

was eventually used with modification to beat Lee Sedol, a world champion in the game Go(Lee, Wang, Yen, Wei, Wu, Chou, Chou, Wang, Yang, & others, 2016). Although DQN does not guarantee that the network converges to a global optimum, we have seen that it works quite well in practice.

In deepmind's experiments, the states are images. Hence, each state is a 3-dimensional representation of the image(width, height, and color). Naturally, the choice of deep neural network is Convolutional Neural Network(CNN). We opine that CNNs may not be a good fit to our problem as each state is not an image. Henceforth, we consider the following neural networks:

- **Convolutional Neural Network(CNN)** is a type of feed-forward artificial neural network(ANN). They are biologically inspired and are tailored to recognize images.
- **Fully Connected Deep Neural Network(DNN)** is an ANN where each neuron is connected to every neuron in the previous layer. Moreover, each connection has its own weight.
- **Recurrent Neural Network(RNN):** is a kind of ANN where connections between neurons form a directed cycle. The directed cycle allows it to exhibit dynamic temporal behavior. The benefit of RNNs is that they can process arbitrary sequences of inputs. We also note that RNNs are good in approximating functions(Hammer, 2000).

4.3 Framework 2

The inspriation for this framework came from the paper titled 'Neural Architecyire Search with Reinforcement Learning'. In the paper, Barrat Zoph of Google Brain presented a Neural Architecture Search, which is a gradient based approach for finding good archiecture(Zoph & Le, 2016). His work is based on the insight that both the structue and

connectivity of a neural network can be specified by a variable length string. Hence, he exploited this insight by using a recurrent neural network to generate a string and this string is used to train and evaluate a child architecture. Figure 4.2(Taken from the original paper) illustrates this.

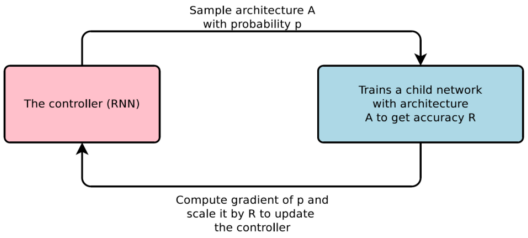

Figure 4.2: Neural Architecture Search

In his work, the problem that he is trying to solve is hyper parameter search. The architecture aims to find the hyper parameters that optimize the child architecture. Figure illustrates hyper parameter search using this architecture. In the diagram, the child network has parameters: Filter Height, Filter Width, Stride Height, Stride Width, Number of Filters. Previous prediction is fed into the next time step as input. It may look a little strange, but we also note that in our discussion in framework 1 that Recurrent Neural Network(RNN) are sensitive to sequences. Figure 4.3(Taken from the original paper) illurates hyper parameter search mentioned in his work.

The problem of hyper parameter search is slightly harder problem as hyper parameters are typically continuous. In his work, he discretized the hyper parameters. Henceforth, we can easily use the neural network architecture on our problem. We can modify the child network to be just induction algorithm A. Moreover, our search space is already discreteized. We will also define the reward similarly to what was mentioned in the paper. We summarize our formulation below:

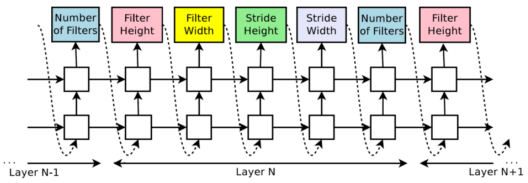

Figure 4.3: Searching for Hyperparameters

- **States:** each feature is represented as one step in an episode, hence $\mathbb{B}^m$
- **Actions:** performed by the RNN controller
- **Transition function:** after selecting the ith feature, go on to the $i+1$th feature.
- **Reward function:** cube of the accuracy, $r(s_t, a_t, s_{t+1}) = z(s_t)^3$

4.4 Framework 3

In this model, we think of FSS as a sequential game. We always start with the empty; which means nothing is chosen. At each step i, we decide on feature f_i: to select or not to select. After m steps, the agent would have decided on each of the features in F. As a result, we can represent the game as a binary tree of depth m. Hence, the number of states in this model is 2^{m+1}. At each state, the agent chooses between 2 actions; to select the particular feature or not. The framework is represented formally below:

- **States:** each state represents both the features selected and not selected. We model this as a ternary vector(of size N) with the 3 digits: $-1, 0, +1$. The number of digits that are either -1 or $+1$ represents the t-th timestep. The encoding is as follows: -1 and $+1$ corresponds to feature not selected and selected respectively, 0 represents

features for which the agent yet to decide. For example, $(-1, +1, +1, 0, 0)$ represents the selection in 3rd timestep $\{-f_0, +f_1, +f_2\}$ where the agent has yet to decide on f_3 and f_4. Additionally, each ternary vector also represents a feature subset by assuming that the features that have not been decided on are not selected. Hence, the vector $(-1, +1, +1, 0, 0)$ represents the feature subset $\{f_1, f_2\}$. This allows us to apply the same reward function as defined in the aforementioned framework.

- **Actions:** a_0 and a_1 selects and does not select the t-th feature respectively.
- **Transition function:** following the each action, a_0,a_1 causes the t-th digit to be changed from 0 to $+1$ and -1 respectively.
- **Reward function:** reward is just $A(S, D)$ at the leaf, there is no reward otherwise.
- **Observations:** each state.

Figure 4.4 explains model 2 in a graphical manner, displaying a FSS problem with only two features. In the figure, only the actions from leftmost states are labeled. We also note that the state graph is acyclic. We observe that the probelm is deteminstic and it always terminates. Hence, we may be able to exploit such properties in the model. This framework arose, after running experiments with framework 1, we have observed that each experiment takes an awful amount of time. This is due to the repeated evaulation of the $A(S, D)$ at every step. In equation 4.3, we observe at every update the agent is required to evaluate z, which is a wrapper over $A(S, D)$. The computation of z is not cheap and takes a consiable amount of time. We remember that one of our goals specified in the eariler chapters is that the feature selection algorithm must be able to find the feature subset in reasonable amount of time. In this model, it is natural for the agent to evaluate the goodness of the state at the end as the features are not fully selected yet at every other level, but only at the leaves. Along with our goal to complete the feature

subset selection in a reasonable amount of time, we modify the reward function from framework 1.

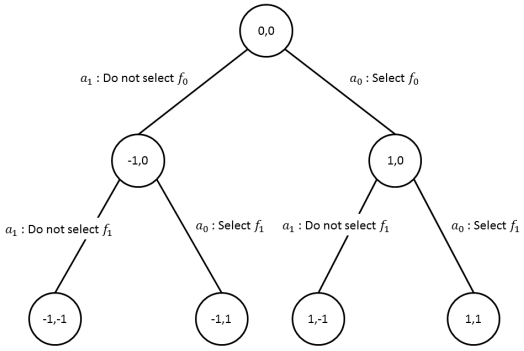

Figure 4.4: Graphical representation of framework 3

Similarly, we can model the states as ternary vectors. We leave the discussion of the modelling and the Q-Table in implementation.

4.4.1 Fast Q-Agent

In this section, we will discuss a few modifications that we have done to the traditional Q-Learning agent. First and foremost, we set the learning rate $\alpha = 1$ and discount factor $\gamma = 1$. We understand that this is a fully determistic environment, performing the same excact feature selection steps will result in the same feature subset. Henceforth, by setting the $\alpha = 1$, it is optimal. We also observe that modifying the discount factor will change the way how the agent views long term reward. Since, the agent will always perform only m steps, the agent will always terminate at the leaves. Hence, we would have no problems setting the discount factor $\gamma = 1$.

Inaddition to the above modifications, we would also modify how the Q-Learning agent performs the update. Traditionally, after every action $s_t \xrightarrow{a_t} s_{t+1}$ is performed, the Q-Table is updated. We would modify that update to be only performed on the leaf nodes. In Fast Q-Agent, the agent history H will be recorded as the agent progresses

from one level to the next. At the terminal nodes, the agent attempts to perform the updates for the entire history. We perform the update from the terminal nodes, all the way up to the root node.

Similar to framework 1, we are using the epsilon-greedy policy as discussed above in 4.2. We use a typical epsilon decay equation, however we modify it such that it is of arbitary base b instead of the typical e. E_0 and E_N represents the starting epsilon and the ending epsilon respectively. N is the number of episodes for which the agent will run. We note that b, E_0, E_N, N are hyperpatameters to our agent.

$$E(n) = E_0 \times b^{\frac{\log_b E_N - \log_b E_0}{N}} \times n \tag{4.4}$$

We note that in our Fast Q-Agent, we will be using datastructues that are similar to Q-Table. These data structures are deteminstic and will function like tables. We will discuss the data stuctures in more detail in the chapter later in implementation. In our discussion, we will refer the data sturctue as Q-Table.

We note that the a traditional Q-Agent converges to optimal when used with an epsilon greedy policy(Watkins & Dayan, 1992). We attempt to prove that our variant of Q-Agent converges to optimal as well, that our modifications does not affect convergence and also optimality of Q-Learning with epsilon greedy policy. We will only give arguments as to why is this so, but not a formal proof.

Converges to Optimal

We now prove that the Fast Q-Agent will converge to the optimal solution. We understand that a traditional Q-Agent with epsilon greedy policy converges to the optimal solution, we will bootstrap our argument onto a traditional Q-Agent. Let a Fast Q-Agent's Q-Table to be exactly the same state ad a traditional Q-Agent. Let H be a history of states and actions that the Fast Q-Agent has performed on the environment. For brevity, we

consider the environment to be deterministic and always terminates on every episode. The Fast Q-Agent will update its Q-Table with the history H, updating its Q-Table from the last state to the first state in H. We Let H' be a history that is in a traditional Q-Agent, this H' is composed of m runs of the H on the traditional Q-Agent. Since the length of H is composed of m actions, the history H' essentially propagates the final reward collected at the terminal state throughout its states. Since there are m states from the first state to the last state, the final reward value will take m steps to propagate. Henceforth, the traditional Q-Agent will have the same state as the fast Q-Agent after applying the update of H' and H respectively. Since we are talking about convergence at the infinite sequences, the extra $m-1$ episodes that the traditional Q-Agent does not matter. Hence, for every history of the Fast Q-Agent, we can always find a H' that will perform an equlivant update to the Q-Table. Since a traditional Q-Agent is convergent, all possible finite possibilities of histories are convergent. Hence, we can always find a convergent H'_{opt} in traditional Q-Agent that is equlivant to H_{opt} in Fast Q-Agent. Since H'_{opt} converges to optimal, H_{opt} must also converge to optimal.

4.4.2 Q-Table

We understand in our argument in Chapter 3, the number of states is very large. Henceforth, we need to be very careful how we represent the states in the Q-Table. Moreover, we recognize that we need to be diligent and careful in designing and implementing the Q-Table such that the Lookup and Update of the states and their respective Q-Value is fast. If the Q-Table is poorly implemented, the Q-Agent would perform miserably. In our algorithm, we will refer Q-Table as Q-Tree. The reason for this is because the datastructre is shaped like a tree instead of a table due to the nature of our problem, refer to Figure A.1 for a graphical representation. In this section, we will list down the 3 attempts at designing the Q-Tree. We will also discuss modifications to the state representation.

We also note that in our discussions, we do not consider the complexity of the distance calculations as the complexity is the same across all data structues.

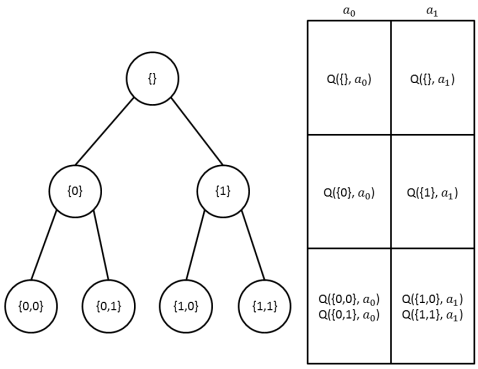

Figure 4.5: Q-Tree for $m = 2$

Attempt 1 - KD-Tree

KD-Tree is a generalized binary tree in k-dimentions(Bentley, 1975). KD-Trees are designed to address the inefficency of the brute-force in machine learning. The tree aims to minimize the number of distance calculations by encoding aggregate distance information. The intuition is given 3 n-dimentional points A, B, C, if we know that A and C are very far apart and if B is very close to A. Hence, we can automatically conclude that the A and B are distant, saving a distance computation. The complexity of KD-Tree for insertion, querying and deletion are the same, $\mathcal{O}(\log n)$.

We also note that in literature, KD-Trees are not suitable for problems with high-dimentional spaces. KD-Tree is most effective when the number of points n is large, more specifically when $n >> 2^k$, as it would be digressing to exhaustive search(Indyk, 2004).

Attempt 2 - Arrays

Arrays is a data sturctue storing an ordered collection of elements. In our discusssion, we will reaonably assume arrays are implemneted on a determinstic machine with con-

tiguous memory block. We note that we will benefit from cacheing effect. Intuntively, at Henceforth, the complexity for search is $\mathcal{O}(n)$, insertion and deletion would be $\mathcal{O}(1)$, random access woule be $\mathcal{O}(1)$. This is because we would need to perform a linear search over the entire array to find the closest match to any given state. Before we continue, we make note of an important insight into Q-Learning. As the algorithm progresses, it converges to optimal. Typically, the reward value increases as time passes and the algorithm will select the same state more frequently. With this insight, we can sort the underlying array. Experimentally and intuitively, it gives us a better performance due to the higher proability of matching a given state. However, we note that the worse-case complexity is the same as the aforementioned complextiy. An example is given in Figure 4.6.

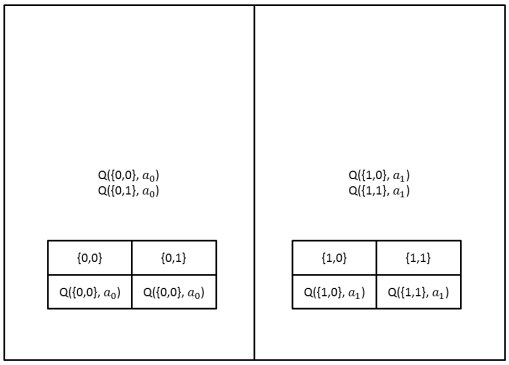

Figure 4.6: Arrays representing Q-Tree

We also note that we modify the representation of the state, deviating from the representation mentioned eariler. We now give a representation that is equlivant to the one that we had mentioned in the framework. We make two adjustments to the state representation. Firstly, the new state representation will be represented as bit vectors which can be efficiently implemented on any mordern machine. Secondly, we will represent the features not yet considered by the size of the bit vector. Hence, a state originally represented as $(1, 0, -1, -1)$ is represented as a bit vector 10. Similarly, $(1, 0, 0, 0)$ is represented as 1000. This allows us to use fast bit operations. In addition, we can use

Hamming-Disance to calculate the distance of the bit vectors which can be efficiently implemented using bit operations. This would give us a major advantage compared with KD-Tree, even though the complexity for KD-Trees are better.

We will also further assume that the size of the array at each level is fixed. We make note that the size at each level is bounded by $\min(2^{m_i}, n)$, where m_i is the number of features that have been considered at level i. From our discussion above, it should be clear that the binary vector processed at the i-th level must be of size i. Henceforth, the number of possible unique bit vectors would be 2^{m_i}. Additionally, the number of bit vectors cannot exceed the total number of episodes that the agent performs. Assuming that the agent finds 1 unique and new bit vector for level i, then it is n. Henceforth, we give an upper bound of the memory usage of the data structure in equation 4.5. We also make the reasonable assumption that $n > m$ as typical datasets the number of instances n is larger than the number of features m.

$$\mathcal{O}(\sum_{i=0}^{m} \min(2^{m_i}, n)) \implies \mathcal{O}(n \times m), \text{where } n > m \tag{4.5}$$

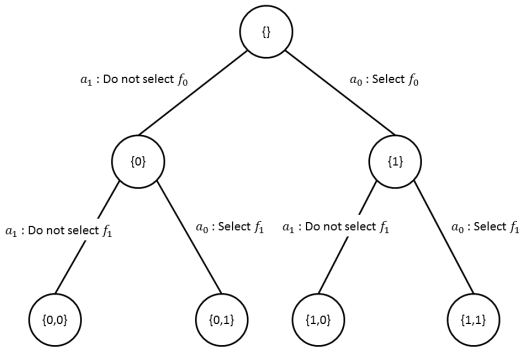

Figure 4.7: New Representation of Q-Tree's states

We now make another insight, deletion never happens in our array. This is because the operations that the Q-Tree performs only insert, update and search operations, updating of the Q-Value and searching for the state and its corresponding Q-Value. We also

note that we know the the difference $\delta = q_0 - q_1$ between the old Q-Value q_0 and the new q_1. Hence, we can update by simply performing a binary search of the value q_1 to the sub-array, either at the front or the back(determined by δ) relative to the old position. Thereafter, we shifts all the values between the old position and the new position. However, this does not change the wose case complexity of the update operation.

Attempt 3 - Doubly Linked List

We recognize that all the insights in attempt 2 can be applied to the doubly linked list. The difference between the array and the link list is that the link list would avoid the shifting of the elemnts. Shifting elements is an expensive operations as all the n elements will have to be rewritten to their adjacent cells. We also note that in this representaion, we lose the ability to perform binary search because we no longer have random access of $\mathcal{O}(1)$. However, the binary search operation is not the most expensive operation but the shifting of the elements. We attempt to rectify this by using doubly linked list. We can search in direction as determined by δ as mentioned in attempt 2 and just splice at the appropriate position. However, the worst-case complexity would be still the same, listed below. However, it would intutively reap a performance benefit as the number of operations peformed reduced and they are less expensive.

- Search: $\mathcal{O}(n)$
- Update: $\mathcal{O}(n)$
- Insert: $\mathcal{O}(n)$

Chapter 5

Testing Methodology

In this section we will describe how we tested our proposed frameworks with state of the art algorithms. We note that our discussion form this chapeter on will be organized by Prelimnary and also Framework 3. We need to make such a distinction as we have done prelimnary testing on all three frameworks. However, the results from framework 1 and 2 are not promising. Therefore, we will not further validate their results. We will focus on testing methodology and discuss the implementation details in the next chapter.

Before we go on on testing methodology, we summarize the scope of the project. We aim to find a fairly general method that tackles optimality that balances between accuracy and speed. In addition, we have seen in Chapter 1 that an overwhelming majority of the problems in machine learning is classification and we will focus on that. As a result, our induction algorithm will use accuraccy as an evaluation metric. Henceforth, we will need to define the competiting methods, datasets and the machine learning algorithms used in our testing. We will discuss that in this chapter.

5.1 Prelimnary

In this section, we detail the competeing methods, datasets and how the results are compared. Our purpose for prelimnary testing is to quickly iterate through hypothesis, results and analysis so that we can get quick feedback on the level of sucess of our methods. Hence, it is also understandable that the algorithms and algorithms compared are not comprehensive. We will also discuss the agents used for Framework 1 in this chapter, as they are not well defined earlier.

5.1.1 Competiting Methods

In our experiment, we want to compare models 1 and 2 with state of the art methods. The list of methods, including our method, are shown in table 5.1. We note that although Relieff amd CFS are Feature Ranking algorithms, we compared them against our algorithm by limiting the number of features selected to be the same as the subset that we have obtained. Please refer to the final experiment to have a complete understanding.

FSS Algorithm	Strategy	Label	Result
ReliefF	Filter(Sim)	Supervised	Ranking
CIFE	Filter(Info)	Supervised	Subset
CFS	Filter(Stat)	Supervised	Ranking
Framework 1	Wrapper	Supervised	Subset
Framework 2	Wrapper	Supervised	Subset

Table 5.1: Methods compared in Prelimary experiment

5.1.2 Datasets

In our experiment, we test our proposed methods using small datasets, some of which are widely used in FFS literature. The datasets used in prelimary testing are listed in Table 5.2. The dataset ID refers to the dataset's OpenML ID. We will discuss more on in implementation details. We also note that there is 1 dataset used in our prelimary testing that is not sourced from OpenML. IHPC is a dataset used in an internal project in IHPC's AI Group. The project is about predicting equipment failure using readings from sensors.

ID	Dataset	Field	Feature Type	#Features	#Instances	#Classes
44	spambase	Email Spam	Continuous	57	4601	2
1043	ada_agnostic	Marketing	Continuous	48	4562	2
-	IHPC	Failure Det.	Continuous	22	3985	2

Table 5.2: Datasets used in our experiment

In our prelimary testing, we used Stratified Shuffle-Split to optain the train and test sets. This garuntees that each folds preserve the ratio of the classes. For each dataset, we used 0.25 of the original data for training and the rest for testing.

5.1.3 Machine Learning Algorithm

We choose a machine learning algorithm: decision tree classifier to be used for testing purposes. We chose decision trees as it is relatively fast, used in wrapper methods(Kohavi & John, 1997) and is sensitive to noise(Atla, Tada, Sheng, & Singireddy, 2011). In addition, we have tested framework 3 with Support Vector Machines(Classification).

5.1.4 Evaluation

We also understand that we need to be able to properly evaluate any given subset of features. Hence, we will use K-fold Cross Validation for both training and testing. K-fold cross validation splits the orginal sample into k randomly partioned subsamples of equal sizes. One of the subsamples are used for testing while the rest are used for training. This process is repreated for all the k subsamples. We also note that 10-fold cross validation is commonly used in machine learning literature(McLachlan, Do, & Ambroise, 2005Kohavi & John, 1997).

In training, we use 5-fold cross validation. We will just use the mean accuracy as the return of the function z as mentioned earlier. For evaluation, we performed 10-fold cross validation. We relied only on the mean accuracy of the 10-fold cross validation to compare algorithms, following literature. We understand that this method of just comparing the mean is not ideal, however this is only prelimary testing, we can afford to be a little sloppy.

5.1.5 Environment/Agents - Framework 1

We will also need to define the specific agents that were used in framework 1. In our preliminary experiment, we tested our models using the following agents: DQN with DNN and DQN with RNN.

5.2 Final - Framework 3

Similarly, we detail the competeing methods, datasets and how the results are compared. The discussion here applies only to framework 3.

5.2.1 Competiting Methods

Instead of limiting feature ranking algorithms by the number of features that our algorithm select, we remember that we have given an algorithm to convert feature ranking problem to feature subset selection. Henceforth, all feature ranking algorithms will be thus converted to feature selection algorithms. We will perform our analysis on the feature subsets that arose from each of the feature selection algorithms.

We present the competiting methods found in literature. We have chosen the algorithms by their avaliablity of implementation and generality of the feature selection to several domains. We have avoided algorithms that tailor to a specific domain or tries to exploit certain structures of the data to perform feature subset selection.

Firstly we have Infomation theoric methods: Cife(Lin & Tang, 2006b), Cmim(Fleuret, 2004), Disr(Meyer, Schretter, & Bontempi, 2008), Fcbf(Yu & Liu, 2003), Icap(El Akadi, El Ouardighi, & Aboutajdine, 2008), Jmi(Yang & Moody, 1999), Mifs(Battiti, 1994), Mim(Lewis, 1992b) and Mrmr(Peng, Long, & Ding, 2005). These are filtering methods that uses infomation entropy to perform feature selection. Next, we have Similarity based methods: FisherScore(PEH, 2007), LapScore(He, Cai, & Niyogi, 2005), Relieff(Robnik-Šikonja & Kononenko, 2003b) and Spec(Zhao & Liu, 2007). Special mention goes to Relieff, as it is a very popular method amoungst data scientists due to its speed. Next, we have statistical based methods: Cfs(Hall & Smith, 1999b), FScore(Wright, 1965), GiniIndex(Gini, 1912), TScore(Davis & Sampson, 1986) and VarThreshold(Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, & others, 2011a). Finally, we have wrapper methods(Guyon & Elisseeff, 2003): ForwardSelection, BackwardSelection, RandomSearch and Simulated Annealing. We summarize the algorithms in Table 5.3, sorting them by the categories that we have mentioned earlier.

5.2.2 Datasets

All datatsets used for final testing come from UCI machine learning repository. We chose to use datasets from the repository as the datasets are widely used in feature selection and machine learning literature. Moreover, there are a litany of datasets from a wide veriaty of fields. Since our goal is to create a fairly general method, we will validate our method with datasets from many different fields.

In addition, we aim to test on datasets of varying sizes. As mentioned, we will test on datasets larger than 16 features as anything below the number can be 'easily' solved using brute force. The datasets that we will test on range from 30 to 1558 features. Additionally, the datasets are of varying instance sizes, some are large(34465) and some are tiny(1593). We also note that we have carefully chosen datasets that do not have problems such as the curse of dimentionality, with $m >> n$. The problem occurs when the number of instances are far lesser than the number of features. Such datasets require more in-depth analysis and in-order to avoid that, we will not choose such datasets.

We also note that we are using accuracy as our evaluator. In literature, there is such a problem called the accuracy paradox. Accuracy paradox occurs primarily when the dataset is imablanced. A balanced dataset is when the ratio of all the class instances are uniform. In such cases, a predictive model with lower accuracy can be better(predictive power) than one with higher accuraccy. It is understood that in such cases, matrics such as recall and precision should be used(Bruckhaus, 2007). In particular, we will not use the data set that is 'too' skewed. We use a general rule of thumb of 1/4, avoiding datasets that are skewed beyong 1/4. Additionally, we were carefully selected datasets that will not lead to problems in the K-fold cross validation. The problem is having insufficient instances in a class in a particular fold. Similarly, we used Stratified Shuffle-Split of ratio 0.25 to optain the train and test sets.

	Supervised		Unsupervised	
	Ranking	Subset	Ranking	Subset
Filter	FisherScore	Cife	Spec	
	Relieff	Cmim	LapScore	
	Cfs	Disr	VarThreshold	
	FScore	Fcbf		
	GiniIndex	Icap		
		Jmi		
		Mifs		
		Mim		
		Mrmr		
Wrapper		ForwardSelection		
		BackwardSelection		
		RandomSearch		
		SimulatedAnnealing		
		Framework 3		

Table 5.3: Feature Selection algorithms compared in the final experiment

OpenML ID	Dataset	Field	Feature Type	#Features	#Instances	#Classes
1475	first-order-theorem-proving	Theorem Proving	Continuous	51	6118	6
44	spambase	Email Spam	Continuous	57	4601	2
1486	nomao	Search Engine	Discrete,Continuous	118	34465	2
20	mfeat-pixel	Handwriting Reg.	Discrete	240	2000	10
1501	semeion	Handwriting Reg.	Continuous	256	1593	10
1485	madelon	Artificial	Continuous	500	2600	2
1468	cnae-9	Document Classification	Continuous	856	1080	9
1176	Internet-Advertisements	AdBlocker	Continuous	1558	3279	2

Table 5.4: Datasets used in Final experiment

5.2.3 Machine Learning Algorithm

We would test our method across 2 machine learning algorithms namely Naive-Bayes and also Decision Trees(CART). We chose the 2 aforementioned algorithms as they have fast runtimes for both training and testing, due to time constraints that we have for this project. We note that our method is employing the wrapper strategy and should work for any other machine learning algorithm. We make a note that we have run our method with other machine learning algorithms in our prelimary testing.

5.2.4 Setup

The Fast Q-Agent will run for 10000 episodes. It will run with default hyper-parameters, $E_0 = 1.0$, $E_N = 0.005$, $b = e$. We note that the author has made no effort to tune the hyper parameters. In addition, both random search and simulated annealing will run for 10000 episodes.

5.2.5 Evaluation

In this section, we will discuss the 6 metrics that will be used to compare the algorithms:

- Feature Subset Selection Runtime
- Test and Evaluation Runtime
- Number of features selected
- 5-fold accuracy mean and vairance for training
- 10-fold accuracy mean and vairance for testing
- T-test against Framework 3

K-fold mean and variance

Similar as in our prelimary testing, we will use k-fold Cross Validation with accuraccy as the metric. We will also use 5-fold and 10-fold respectively for training and testing. We will report both the mean and the standard deviation of the accuracies. We reiterate that the 10-fold cross validation scores reported are from the independent 'outer loop' which will be judged based on an unseen test dataset that is unknown to the feature subset algorithm. We note that some papers reported accuracies from the 'inner loop', in our case the results from the 5-fold cross validation, these results are optimiscially biased and inaccurate measure of the feature selection algorithm(Kohavi & John, 1997).

T-Test

Additionally, all the feature selection algorithms are competing against framework 3, we can perform t-tests of our algorithm against all the other algorithms to find out if the difference between the accuracies is significant. We make the assumption that a normal distribution is assumed. We report the p-values of the 2-tailed test, which will roughly translate to indicating the probability that one algorithm is better than the other. We will perform the test at 95% confidence level.

Feature Subset Selection Runtime

We make note of the wall clock time taken for the feature selection algorithm to run. We note that the feature selection algorithm typically run for an extended period of time, which the lenght is not short(in minutes/hours). We note that there are problems associated with measuring a short amount of time on the mordern multi-processors. We will discuss this problem in detail in implementation. Hence, to compare feature subset selection runtime, we would have no problems taking wall clock time.

Train and Evaluation Runtime

We will note the time taken for the machine learning algorithm to perform training and evaluation. We perform cross validation on the entire dataset and note the time taken. We also note that we run into the problem mentioned above, that measuring a short amount of time on mordern multi-processors can be tricky. We will discuss that in implementation.

Number of features selected

We will also make note of the number of features selected by the algorithm, so we can comment on the accuraccy achieved with the number of features selected.

Chapter 6

Results

6.1 Preliminary Results

Firstly, we ran all 3 frameworks on the Spambase dataset. We will present some of the results from the prelimary run here. We note that we have tried many other models under framework 1, we will simply list their results in the appendix.

6.1.1 Framework 1

The results, including the results for the baseline, are summarized in the table 6.1. Baseline represents the case for which all the features are chosen. We note that the metric used to compare the algorithms is different compared to the final and framework 3's prelimary results. Firstly, we summarize the baseline and other feature selection algorithms in the table below.

DNN

DQN with DNN performs miserably, with the loss function of the neural network increasing. Although the network converges, it fails to find a suitable subset, converging to a subset $S_{\text{DQN DNN}} = F$. We tried to tweak both the learning rate and the batch size of

Algorithm	# Features	Score
Baseline	57	0.0727
Relieff	28	0.0672
CIFE	41	0.00768
CFS	16	0.128

Table 6.1: Performance of competing algorithms

the DQN to no avail. We understand that DNNs are tough to train because of the depth and the interconnectedness of the neural nodes(Srivastava, Greff, & Schmidhuber, 2015). We conclude that DNN is not a good neural network to be used for our problem.

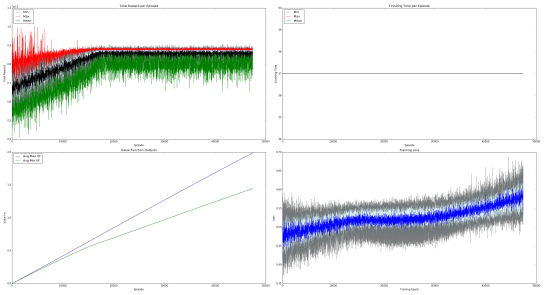

Figure 6.1: Results for DQN with DNN

RNN

DQN with RNN performed relatively better than DQN with DNN, converging to a subset where all the features are selected other than the first feature. We also observe that the training loss of the neural network reduces slowly, indicating that the neural network is learning. However, the performance of the subset is marginally better than baseline with a score of 0.0891. However, it fails to beat the score derived from the subset produced by CFS. We think that a customized neural network that is designed to take in a binary

vector may work much better than a DNN or RNN.

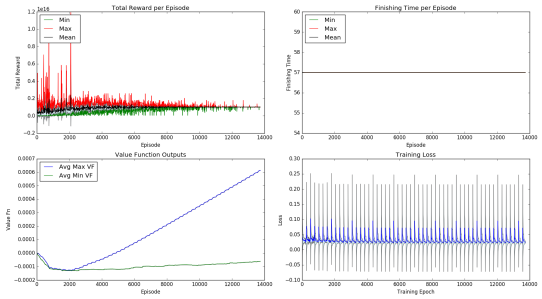

Figure 6.2: Results for DQN with RNN

KNN

We note that we have tried a similar modeling as in framework 3 in framework 1. We modeled exactly as what we have done in Framework 3, the only exception would be its reward function. We used the reward function specified in framework 1. Firstly, we observe that the rewards over time form an arc; it first increases then it decreases. One possible reason is that KNN struggles to keep up when the number of data points increases. For our experiment, we used $K = 5$ for all KNN. $K = 5$ may not be reasonable or sufficient. Additionally, the algorithm ran for only 5000 episodes which may be causing the problem as the agent did not explore the state space enough. It converged to a subset of size 31 that produces a miserable score of 0.0431. However, it is worth noting that while it was exploring, it found a subset of size 27 that scored 0.1474. We also note that Q-Learning with SVM had a similar performance to Q-Learning with KNN. We think that it can be the case of slow propagation of the Q-Value throughout the network that is causing the agent to not converge to the point that gives rise to the best score of 0.1474. This is the inspiration for us to design framework 2 and framework 3 to address some of the problems.

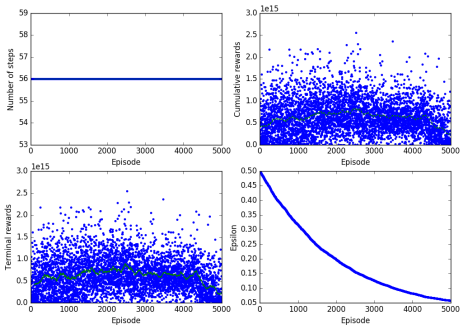

Figure 6.3: Results for Q-Learning with KNN

6.1.2 Framework 2

We ran the experiment for framework two only on the OpenMl-44 dataset. After running for 10000 episodes, the algorithm did not converge. The algorithm is still fluctuating a lot between accuracies even after the 9900-th episode. We illustrate this by showing the accuracies for two episodes: Episode 9986's testing accuracy is 0.822085889571. However, Episode 9988's testing accuracy is 0.747589833479. Another careful look at the data leads us to conclude that did not converge and it might be equivalent to performing random search. Please refer to the git repository *AutoDNN* for the logs of the results. We theorize that one of the possible reasons why Framework 2 fails to work is because the reward function provided to the controller is not good. We understand the reward functions are very important any reinforcement learning problem.

6.1.3 Framework 3

For Framework 3, we ran across three datasets. The git revision corresponding to the results is 'a19e01e16a31b6988068c1f194f1f53db6d9d672'. Firstly, we give the baseline and other feature selection algorithm's testing accuracy for the three datasets that we tested upon. We note that for wanglong, we used the machine learning algorithm SVC.

Firstly, we present the historical charts for the our method in Tables 6.4 and 6.5,

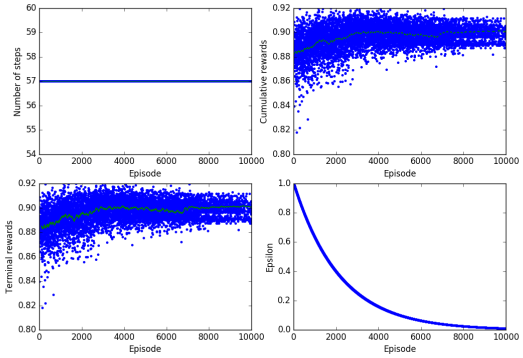

Figure 6.4: History for OpenMl-44

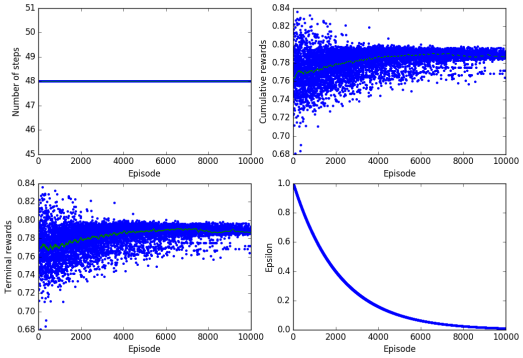

Figure 6.5: History for OpenMl-1043

showing the growth of rewards over the number of episodes. We will omit the historical chart for the dataset wanglong as the convergence pattern is similar.

From the datasets, we can see that our method works better than the rest of the competiting methods. The best competiting result are Relieff 0.897 which is comparable to our method's 0.880 for the OpenMl-44 dataset. We also observe that for the other data set OpenMl-1043, our method out performed the other methods, reaching an accuracy of 0.826. We also observed that in the last dataset, we performed better than the rest of the algorithms, losing only to Baseline. However, Baseline used 22 features, while our method used only 12 features but getting about the same performance: 0.910 vs 0.908. With this we will validate Framework 3 with more data in the final phase of the experiment.

Algorithm	**# Features**	**accuracy**
Baseline	57	0.893
Relieff	53	0.897
CIFE	39	0.854
CFS	13	0.842

Table 6.2: Performance on OpenMl-44

Algorithm	**# Features**	**accuracy**
Baseline	48	0.807
Relieff	20	0.784
CIFE	43	0.802
CFS	11	0.757

Table 6.3: Performance on OpenMl-1043

Algorithm	**# Features**	**accuracy**
Baseline	22	0.910
Relieff	7	0.895
CIFE	3	0.840
CFS	3	0.844

Table 6.4: Performance on WangLong

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)
0:39:49.343250	QLearner	3.7865	36	0.920 ± 0.012	0.880 ± 0.035

Table 6.5: OpenMl-44

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)
0:21:18.054767	QLearner	0.745256	19	0.836 ± 0.015	0.826 ± 0.025

Table 6.6: OpenMl-1043

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)
0:13:52.753311	QLearner	2.27872	12	0.907 ± 0.108	0.908 ± 0.074

Table 6.7: wangLong

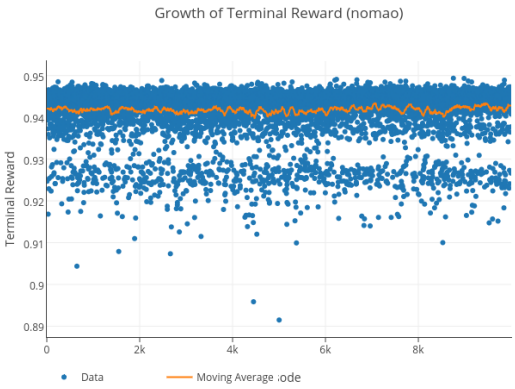

Figure 6.6: History for OpenMl-1486

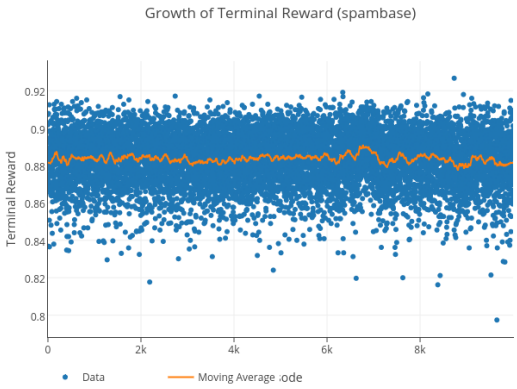

Figure 6.7: History for OpenMl-44

6.2 Final Results

Encouraged by our success with Framework 3, we attempt to test it over datasets of a larger variety. We show the full results in the Appendix. We are shocked at the result, as the algorithm did not converge. Figures 6.6 and 6.7 show that our method did not converge at all. Out of all the datasets, our method performs best in Openml-1486. Otherwise, our method can be shown to be significantly worse(by the T-test) when compared with the other methods. We are perplexed at why in our preliminary testing, our method converges and performs comparable or better than the competition, whereas in the final results it does not follow what we have concluded before. After much thinking and investigation, we have concluded that it is due to a bug in the code, introduced in between running the preliminary testing and the final testing. We were modifying the code that is part of the Q-Tree, we think that because of that bug it causes the Fast Q-Learner not to converge. We also note that in both preliminary and final datasets, we used OpenMl-44. For the dataset OpenMl-44, our method is shown to be convergent in preliminary testing, but shown to be not convergent in final testing. A visual comparison of the graphs in Figure 6.7 and 6.4 will tell the same story. Henceforth, we are confident that the results presented here are erroneous.

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
1:36:16.117269	ForwardSelection	3.7901	67	0.952 ± 0.000	0.937 ± 0.006	0.246521
10:01:43.145255	QLearner	4.93588	46	0.944 ± 0.002	0.933 ± 0.006	1
0:04:50.435381	GiniIndex	15.7306	117	0.947 ± 0.002	0.933 ± 0.009	0.874301
8:20:07.247130	BackwardSelection	10.8041	77	0.949 ± 0.002	0.933 ± 0.006	0.84342
0:04:38.058166	FScore	9.53049	91	0.947 ± 0.002	0.932 ± 0.007	0.564476
0:00:00.000144	Baseline	10.7084	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
1:34:18.259437	Disr	17.7005	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
0:17:36.542732	Jmi	11.1647	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
0:19:20.874951	Mim	10.9183	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
7:51:02.847676	SimulatedAnnealing	7.23281	60	0.944 ± 0.001	0.931 ± 0.007	0.396528
0:15:54.309767	Cmim	9.33369	95	0.944 ± 0.002	0.931 ± 0.009	0.467777
0:33:20.290407	Icap	14.8716	95	0.944 ± 0.002	0.931 ± 0.009	0.467777
0:34:29.493694	Relieff	8.82695	97	0.947 ± 0.003	0.931 ± 0.009	0.443816
0:10:03.007672	LapScore	15.036	99	0.946 ± 0.002	0.929 ± 0.008	0.257053
4:18:29.176265	RandomSearch	4.91086	61	0.949 ± 0.001	0.928 ± 0.008	0.126182
0:00:00.071364	VarThreshold	2.18691	34	0.937 ± 0.002	0.923 ± 0.008	0.00750944
0:00:48.514108	Fcbf	0.15799	4	0.926 ± 0.002	0.923 ± 0.007	0.00206919
0:28:09.306398	Cife	9.00142	71	0.933 ± 0.004	0.918 ± 0.008	0.000140857
0:01:44.384335	Mifs	0.422502	3	0.905 ± 0.003	0.889 ± 0.011	4.23588e-09
0:00:57.197468	Mrmr	0.407047	2	0.904 ± 0.003	0.889 ± 0.010	6.92833e-10
1:54:15.437697	Cfs	1.52938	11	0.885 ± 0.003	0.864 ± 0.007	1.36914e-14

Table 6.8: OpenMl - 1486

Chapter 7

Conclusion

Henceforth, with the erroneous results from the final testing, we will reject the results. Instead, we will perform our conclusions on our preliminary results.

From our preliminary results, we have seen that in framework 1 and 2, neural networks are very difficult to train. Even though we have tried many different values and variants of the model, we still cannot make the neural network converge. We conclude that we will require more work and study of neural networks, on how to train and utilize them. We opine that for framework 2, it is a very intuitive model and we would like to investigate more deeply how to configure the RNN such that it will work for feature selection, more importantly, we would like to re-look at the reword function used in this framework.

We have observed that Framework 3 was shown to work well in preliminary testing, but failed miserably in final testing. From the preliminary results, we conclude that it is still inconclusive if framework 3 achieves the goal that we have set out in the earlier chapter, to create a method to address feature subset optimality and striking a good balance between prediction performance and feature selection speed. To conclude this, we would need to find the bug and test it conclusively on the datasets mentioned in final testing.

7.0.1 Further Work

If the results from the final testing are admirable, we can further explore how to optimize framework 3. We have not spent much time to optimize framework 3. We can explore which order of the features would positively affect the accuracy results. Moreover, we can try to tweak the hyper-parameters of our method. I opine that by tweaking the hyper-parameters, it would result in some performance and speed improvement. Moreover, if it works, we can consider the Fast Q-Agent in other domains. One of the other domains that I am eager to test is the traveling salesman problem(TSP).

Moreover, I opine that we can further explore the literature and compare more feature selection algorithms. Leveraging on the framework that we have developed, we can study and test more feature selection algorithms. We would be able to publish a full literature review paper on feature selection. We also note that the last feature selection literature review was done in 2014. We can also explore open-sourcing the framework and writing a paper solely on the details of the framework.

References

Almuallim, H., & Dietterich, T. G. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, *69*(1-2), , 1994, 279–305.

Amaldi, E., & Kann, V. (1998). On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, *209*(1), , 1998, 237–260.

Atla, A., Tada, R., Sheng, V., & Singireddy, N. (2011). Sensitivity of different machine learning algorithms to noise. *Journal of Computing Sciences in Colleges*, *26*(5), , 2011, 96–103.

Battiti, R. (1994). Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, *5*(4), , 1994, 537–550.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), , 1975, 509–517.

Bermingham, M., Pong-Wong, R., Spiliopoulou, A., Hayward, C., Rudan, I., Campbell, H., Wright, A., Wilson, J., Agakov, F., Navarro, P., & Haley, C. (2015). Application of high-dimensional feature selection: evaluation for genomic prediction in man. *Scientific Reports*, *5*, , 5, 2015.

Blog, G., Jain, A., Kaushik, S., Gupta, A., & Shaikh, F. (2017). 19 data science tools for people who aren't so good at programming.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym.

Bruckhaus, T. (2007). The business impact of predictive analytics. *Knowledge discovery and data mining: Challenges and realities*, , , 2007, 114–138.

Burdett, A., Burkhardt, D., Cumming, A., Hunter, A., Hurvid, F., Jaworski, J., Ng, T., Scheer, M., Southall, J., & Vella, A. (2008). *Bcs glossary of computing and ict*. BCS, The Chartered Institute.

Cawley, G. C., & Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, *11*(Jul), , 2010, 2079–2107.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Davis, J. C., & Sampson, R. J. (1986). Statistics and data analysis in geology. , , 1986.

El Akadi, A., El Ouardighi, A., & Aboutajdine, D. (2008). A powerful feature selection approach based on mutual information. *International Journal of Computer Science and Network Security*, *8*(4), , 2008, 116.

Fard, S. M. H., Hamzeh, A., & Hashemi, S. (2012a). A game theoretic framework for feature selection. *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on* (pp. 845–850), IEEE, , 2012.

Fard, S. M. H., Hamzeh, A., & Hashemi, S. (2012b). Proposing a reinforcement learning based approach for feature selection. *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on* (pp. 380–385), IEEE, , 2012.

Fard, S. M. H., Hamzeh, A., & Hashemi, S. (2013). Using reinforcement learning to find an optimal set of features. *Computers & Mathematics with Applications*, *66*(10), , 2013, 1892–1904.

Fleuret, F. (2004). Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*, *5*(Nov), , 2004, 1531–1555.

Gaudel, R., & Sebag, M. (2010). Feature selection as a one-player game. *International Conference on Machine Learning* (pp. 359–366), , 2010.

Gini, C. (1912). Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi*, , , 1912.

Google (2017). Google trends - machine learning, big data, artificial intelligence, deep learning. https://trends.google.com.sg/trends/.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, *3*(Mar), , 2003, 1157–1182.

Guyon, I., Gunn, S., Ben-Hur, A., & Dror, G. (2004). Result analysis of the nips 2003 feature selection challenge. *Advances in neural information processing systems* (pp. 545–552), , 2004.

Hall, M., & Smith, L. (1998). Practical feature subset selection for machine learning. , , 1998, 181–191.

Hall, M., & Smith, L. (1999a). Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper. *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, Vol. 235 (p. 239), , 1999.

Hall, M. A., & Smith, L. A. (1999b). Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper. *FLAIRS conference*, Vol. 1999 (pp. 235–239), , 1999.

Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, *31*(1), , 2000, 107–123.

He, X., Cai, D., & Niyogi, P. (2005). Laplacian score for feature selection. *NIPS*, Vol. 186 (p. 189), , 2005.

Indyk, P. (2004). Nearest neighbors in high-dimensional spaces. , , 2004.

James, G., Hastie, T., Witten, D., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in r*. Springer Texts in Statistics. Springer London, Limited.

Janecek, A., Gansterer, W. N., Demel, M., & Ecker, G. (2008). On the relationship between feature selection and classification accuracy. *FSDM* (pp. 90–105), , 2008.

John Lu, Z. (2010). The elements of statistical learning: data mining, inference, and prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *173*(3), , 2010, 693–694.

Kalousis, A., Prados, J., & Hilario, M. (2007). Stability of feature selection algorithms: a study on high-dimensional spaces. *Knowledge and information systems*, *12*(1), , 2007, 95–116.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, *97*(1), , 1997, 273–324.

Kumar, V., & Minz, S. (2014). Feature selection: A literature review. *Smart CR*, *4*, , 2014, 211–229.

Lee, C.-S., Wang, M.-H., Yen, S.-J., Wei, T.-H., Wu, I., Chou, P.-C., Chou, C.-H., Wang, M.-W., Yang, T.-H., et al. (2016). Human vs. computer go: Review and prospect. *arXiv preprint arXiv:1606.02032*, , , 2016.

Lewis, D. D. (1992a). Feature selection and feature extraction for text categorization. *Proceedings of the workshop on Speech and Natural Language* (pp. 212–217), Association for Computational Linguistics, , 1992.

Lewis, D. D. (1992b). Feature selection and feature extraction for text categorization. *Proceedings of the workshop on Speech and Natural Language* (pp. 212–217), Association for Computational Linguistics, , 1992.

Lewis, R. J. (2000). An introduction to classification and regression tree (cart) analysis. *Annual meeting of the society for academic emergency medicine in San Francisco, California* (pp. 1–14), , 2000.

Lewis, S. C., Zamith, R., & Hermida, A. (2013). Content analysis in an era of big data: A hybrid approach to computational and manual methods. *Journal of Broadcasting & Electronic Media*, *57*(1), , 2013, 34–52.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2016). Feature selection: A data perspective. *arXiv preprint arXiv:1601.07996*, , , 2016.

Lichman, M. (2013). UCI machine learning repository.

Lin, D., & Tang, X. (2006a). Conditional infomax learning: an integrated framework for feature extraction and fusion. *European Conference on Computer Vision* (pp. 68–82), Springer, , 2006.

Lin, D., & Tang, X. (2006b). Conditional infomax learning: an integrated framework for feature extraction and fusion. *Computer Vision–ECCV 2006*, , , 2006, 68–82.

McLachlan, G., Do, K.-A., & Ambroise, C. (2005). *Analyzing microarray gene expression data*, Vol. 422. John Wiley & Sons.

Melo, F. S. (2001). Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, , , 2001.

Meyer, P. E., Schretter, C., & Bontempi, G. (2008). Information-theoretic feature selection in microarray data using variable complementarity. *IEEE Journal of Selected Topics in Signal Processing*, *2*(3), , 2008, 261–274.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, , , 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), , 2015, 529–533.

O'Shea, T. J., & Clancy, T. C. (2016). Deep reinforcement learning radio control and signal detection with kerlym, a gym rl agent.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011a). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(Oct), , 2011, 2825–2830.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011b). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, , 2011, 2825–2830.

PEH, D. (2007). Ro duda, pe hart, and dg stork, pattern classification, new york: John wiley & sons, 2001, pp. xx+ 654, isbn: 0-471-05669-3. *Journal of Classification*, *24*(2), , 2007, 305–307.

Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, *27*(8), , 2005, 1226–1238.

Robnik-Šikonja, M., & Kononenko, I. (2003a). Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, *53*(1-2), , 2003, 23–69.

Robnik-Šikonja, M., & Kononenko, I. (2003b). Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, *53*(1-2), , 2003, 23–69.

Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, *25*, , 1995, 27.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), , 2016, 484–489.

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training very deep networks. *Advances in neural information processing systems* (pp. 2377–2385), , 2015.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*, Vol. 1. MIT press Cambridge.

tanemaki (2016). basic rl.

tensorflow (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, *abs/1605.02688*, , May, 2016.

Tokic, M., & Palm, G. (2011). Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. *KI* (pp. 335–346), Springer, , 2011.

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, *15*(2), , 2013, 49–60.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), , 1992, 279–292.

White, R. W. (2016). *Interactions with search systems*. Cambridge University Press.

Wong, D. (2017). Autodnn. `https://gitlab.com/ihpc_ai/AutoDNN`.

Wright, S. (1965). The interpretation of population structure by f-statistics with special regard to systems of mating. *Evolution*, , , 1965, 395–420.

Yang, H. H., & Moody, J. E. (1999). Data visualization and feature selection: New algorithms for nongaussian data. *NIPS*, Vol. 12, , 1999.

Yu, L., & Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. *ICML*, Vol. 3 (pp. 856–863), , 2003.

Zhang, C., & Hu, H. (2005). Ant colony optimization combining with mutual information for feature selection in support vector machines. *Proceedings of the 18th Australian Joint Conference on Advances in Artificial Intelligence*, AI'05 (pp. 918–921), Berlin, Heidelberg, , 2005: Springer-Verlag.

Zhao, Z., & Liu, H. (2007). Spectral feature selection for supervised and unsupervised learning. *Proceedings of the 24th international conference on Machine learning* (pp. 1151–1157), ACM, , 2007.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, , , 2016.

Appendix A

Implementation and Setup

A.1 Machine

We ultilized 2 seperate machines for our work. Workstation is primarily used for quick testing and prelimary testing of the framework(s), Server is reserved for running the larger experiments. We also note that for our main experiment, it is entirely run on the Server. We summarize the details of its CPU, memory and OS in Table A.1. We note that Server is a multi-socket machine, hence we would run into the issue called Non-Uniform Memory Access(NUMA). We would need to consider that when executing the program. We have carefully ensured software on both machine matches to the author's best knowledge.

Name	Sockets	CPU	vCores	Memory	OS
Workstation	1	Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz	4	8 GiB	Fedora 25 4.9.13-201
Server	2	Intel(R) Xeon(R) CPU X5680 @ 3.33GHz	24	48 GiB	Fedora 25 4.9.13-201

Table A.1: Hardware used in experiment

A.2 Software

The programming languages used in this project are Python3 and bash. Table A.2 illustrates the languages, their versions and number of lines of code programmed.

Language	Version	Lines of Code
Python3	3.5.3	4012
Bash	4.3.43(1)-release	160

Table A.2: Summary of source codes

A.2.1 Python Dependencies

In this section, we will comment on the python dependencies and the work done. We will only list the immediate dependencies for which this project is based upon.We used OpenAI Gym's framework to define our environments(Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, & Zaremba, 2016). An environment defined using OpenAI Gym is generic and can allow a variety of different agents to run in the same environment. We will use their framework for Framework 1 and 3. Most of the implementations of feature selection algorithms listed is taken from the scikit-feature(Li et al., 2016). However, the library was designed for use with Python 2. Moreover, there were some problems with the code in the library. Hence, we have patched the library and ported to python 3. We used implementations for all the machine learning algorithms in the popular machine learning library, scikit-learn(Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, & Duchesnay, 2011b). Lastly, we relied on numpy and scipy for numerical, scientific and array processing.

For framework 1, we tested our models using the following agents: DQN with DNN, DQN with RNN, Q-Learning with KNN. We used the agent defined in Kerlym(O'Shea & Clancy, 2016), which is a third-party library(that uses the OpenAI framework) that implements Google Deepmind's DQN. The neural networks DNN and RNN are built using the deep learning library Keras(Chollet, 2015tensorflow, 2015Theano Development Team, 2016).

For framework 2, we used Daniel Wong's implementation of the Neural Architecture(Wong, 2017). He is a researcher in A*STAR/IHPC/AI, where I was attached for this FYP. We have forked his software repository and make the necessary modifications to make the Neural Architecture work for feature selection. We will discuss the work done in the next section.

A.2.2 Software repositories

For framework 3, we have written the enitre code from scratch. We modelled heavily after tanemaki's implementation of Q-Learning(tanemaki, 2016). We will discuss all the work done and list the software repositories that are of interest. Note that the repositories are set as private, please send an email to Eric Han(`eric_han@u.nus.edu`) to request for access.

scikit-feature

`https://bitbucket.org/eric_vader/scikit-feature` is a fork of the scikit-feature library. The library provides most of the feature selection implementations. We will list the specific feature selection algorithms that we have implemented in the section below. We have reviewed the code, fixed bugs and ported the library to Python 3. Many of the feature selection algorithms are donated by their original authors.

Python3 Dependency	Version	Description
gym	0.1.7	Reinforcement agents framework
numpy	1.12.0	Numerical/Array processing
scipy	0.18.1	Scitific computation
gmpy2	2.0.8	Numerical Computation(C)
simanneal	0.1.2	Simulated annealing framework
scikit-learn	0.18.1	Machine learning and data mining
tabulate	0.7.7	Pretty printer for tabular data
openml	0.3.0	Interface to data repository
plotly	2.0.6	Graphing library
KeRLym	0.0.2	Deep reinforcement learning agents
Keras	2.0.2	Deep Learning
tensorflow	0.9	Numerical Computation on GPU
Auto-DNN	-	Neural Architecture Search
scikit-feature(Patched)	1.0.1	Feature selection algorithms

Table A.3: Summary of source codes

AutoDNN

`https://gitlab.com/eric_vader/AutoDNN` is a fork of the Neural Network Architecture that Daniel had implemented. Framework 2 is implemented in this repository. We have reviewed his code, even fix a bug and implemented the child network using his framework. The simple network takes in a feature subset from the RNN controller and evaluates the feature subset.

feature-selection

`https://bitbucket.org/eric_vader/feature-selection` is the main repository that implements framework 1 and framework 3. For framework 1, we used the KeRLym framework to run our experiments: DQN with DNN, DQN with RNN. We created an environment that describes our model based on OpenAi's gym, and run it using KeRLym. For framework 3, we implemented our proposed solution from scratch. We have implemented the Fast-Q-Agent, the Q-Tree and all its necessary components.

We also note that we implemented 4 of the feature selection algorithms namely: Random Search, Forward Selection, Backward Selection and Simulated Annealing. Both Forward and Backward Selection implementations are taken from scikit-feature. However we have modified them to take in a generic machine learning algorithm instead of hard-coding the algorithm into the code. Simulated Annealing is implemented with the framework simanneal. We defined energy as the negative of the reward, as we want to maximize the reward while the Simulated Annealing agent wants to minimize energy.

A.2.3 Optimization of Q-Tree

In this section, we will discuss the implementation details of the Q-Tree that we have done. We also make note of the optimizations that we have performed. In this section, we have used the datasets in our prelimary set to test the Q-Tree. Figure A.1 shows the run times of the Fast Q Agent with default hyper parameters. Table A.4 lists the versions and their associated git revisions. We now describe breifly the optimizations done below:

- Version 0: No Optimization
- Version 1: Code refactoring and basic memory optimization(copying optimization, etc...)
- Version 2: Architectural changes, basic optimizations

- Version 3: Change of underlying data structure to numpy arrays
- Version 4: Upgrade numerical computation library to use gmpy2, performing computation of Hamming Distance in C code.
- Version 5: Optimize updates using functional style, without branching
- Version 6: Another architecture change in Qtree, to use functional style
- Version 7: Array sorted on Q-Value
- Version 8: Value and copy optimizations
- Version 9: Reuse state and optimized binary search
- Version 10: Change data structure to Linked List
- Version 11: Do not use Parallel computation in Machine Learning algorithm

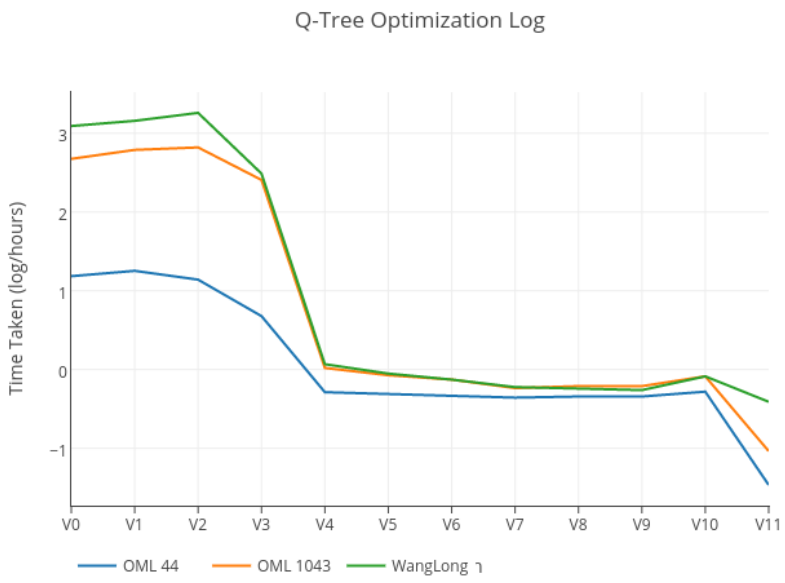

Figure A.1: Graph of the versions of Q-Tree and their runtimes

	Time Taken(hours)			
Ver	OML 44	OML 1043	WangLong	Git Rev
V0	3.27	14.5	22	2e0b57f93b3e45656afcd0df973f9ad3761b226c
V1	3.5	16.25	23.5	5209a1b55f711ef3b4212ba5c3363fab6b7204d1
V2	3.13	16.77	26	825abde4c20fb5a3851bcdb08b44993483925a63
V3	1.97	11.1	12.03	3d4fd09b90e57fc6b20445b477a2979aa8afa823
V4	0.75	1.02	1.07	7473fa3e6f6748454c4e9360ea57502c782fee04
V5	0.73	0.93	0.95	0fccafc459a2761f845eb6513b8940eb5186a362
V6	0.717	0.88	0.88	c1928562fb2b160f0bec0e6fe9b7395b657d254f
V7	0.7	0.79	0.8	ca6c6fc23e3c785a53cec7921270ec4aaf34ce69
V8	0.71	0.81	0.785	519931666019adc94c51b9ea49c7d1f862577408
V9	0.71	0.81	0.77	634084bf942b6d196c079a180c4b285f7345c312
V10	0.755	0.917	0.917	e91608b6d1e161ac2baf008dcf21b49a30f94bf9
V11	0.231	0.355	0.664	a19e01e16a31b6988068c1f194f1f53db6d9d672

Table A.4: Versions of Q-Tree and their runtimes

A.2.4 Software Engineering

The author has ensured that this project uses the best software engineering practices have gone into the project even though there is a short of time and this is a scientific project. We designed an architecture to run, compare and visualize the feature selection algorithms. The entire process is fully automatic, from the setup of the experiments to the visualization of the algorithms. Moreover, we have ensured that the critical parts of the code has been tested. We will detail more in the Software Engineering chapter.

A.3 Datasets

The datasets used in throughout all the comparisons are from the UCI machine Learning repository(Lichman, 2013). OpenML is a online data repository that serves many widely used datasets(Vanschoren, van Rijn, Bischl, & Torgo, 2013), including datasets hosted on UCI machine learning repository. In our project, the datasets are imported using OpenMl, as it allows us to import data with ease. We note that some of the datasets have seperated training and testing datasets in UCI machine learning repository. In our experiment, we will aggregate both train and test datasets and Stratified Shuffle Split the data ourselves to get our own train and test sets for consistency across all datasets.

Appendix B

Software Engineering

This chapter will give a brief discription of the program and its components for the feature-selection repository. Additionally, it will also provide a quickstart guide to using the code in the repository. The reason why we have engineered a proper software architecture to run the code is because we are increasingly frusturated at running and plotting the results manually. We also note that we have made every effort to

B.1 Software Architecture

We have ensured that the project has good architecture, the architecture diagram is shown below. There are 3 major and independent components, setup, experiment and visualization.

Setup's role is to define the experiment that we are running. For instance, defining the parameters for each of the feature selection algorithms, defining what feature selection algorithm should be run. It will also run all the experiments in parallel. We note that we will run the programs assuming the underlying architecture is NUMA enabled. In addition, setup is responsible for managing the runs of the experiments to the end. Since we are running multiple parallel experiments, we will need mechanisms to stop the process

easily instead of using command line tools. We have designed these mechanisms into the feature set of Setup.

Experiment's role is run the experiment. Experiment is designed to be flexible, we designed sub components to be 'swappable', which means that we can change the dataset's source easily. For instance, for the datsets component, we have 3 dataset sources implemented in code: OpenMl, Disk and SplittedDisk which loads from OpenML repository, file and multiple splitted files respectively. This enables us to be very flexible in adding or removing data sources. Similarly, the other sub componsents are designed with this flexibility. All sub-components in the experiment component are: datasets, experiment, evaluator, featsel and ml. Their purposes should be obvious from their names. We also note that the experiment will collect as much data as possible from each of the runs.

Lastly, we have visualization. Visualization's role is to take the data from experiment, process and visualize the data into tables and graphs. We are using Plotly and tabulate as our graphing and table generation engines respectively. It has 3

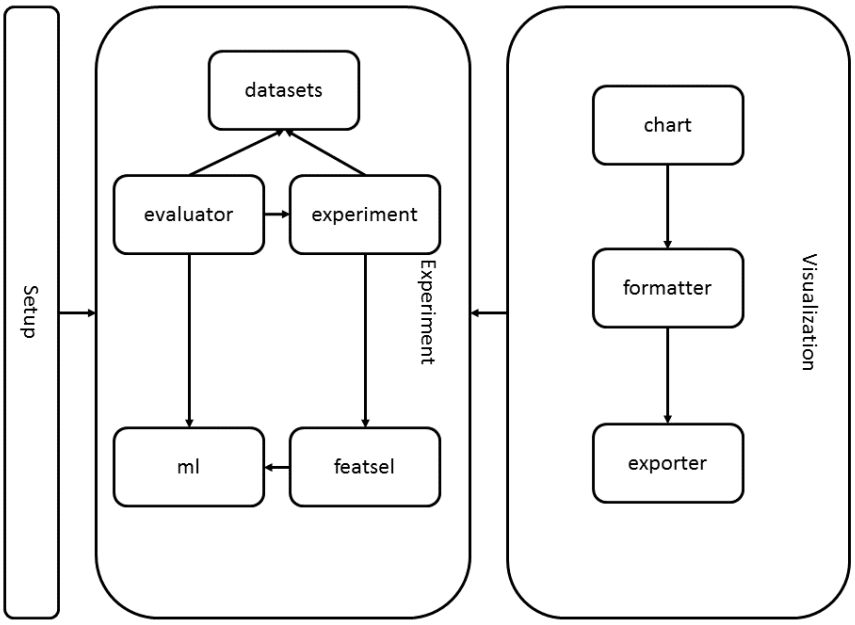

Figure B.1: Architecture of the project

B.2 Setup

When the experiment runs, setup track the process ids and some additional infomation about the process. The user may view the running process by running the command 'status.py' and the program will display the list of running processes. An screenshot of the output is given in Figure B.2

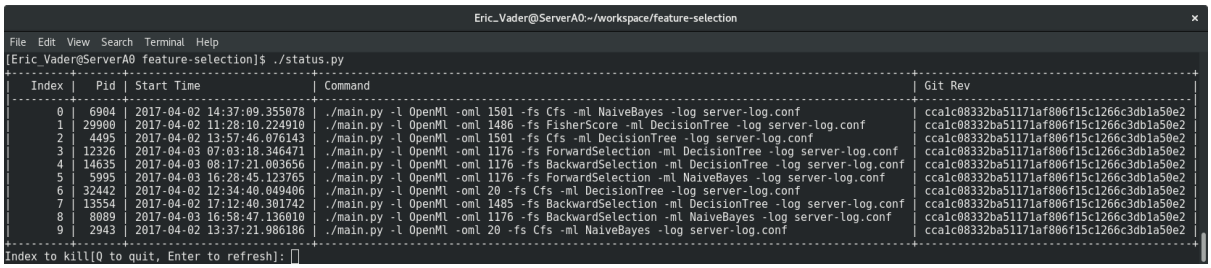

Figure B.2: Example output of status

Additionally, to run a experiment is simple. Simply run the script 'experiment.sh' with the openml id, an example is shown below. The Setup component will take care of running all the different feature selection algorithms with their respective machine learning algorithms in a multi-process fashion on a NUMA-enabled machine.

```
./experiment.sh 20
```

B.3 Experiment

Experiment is the main component and it will take in parameters to run a experiment. You may use the command 'main.py –help' to list all the possible parameters that the program can accept. An example screenshot of this is given in Figure B.4. As we can observe from the output, we can specify the feature selection algorithm to run, the machine learning algorithm and the dataset that will be used. All these sub-components are designed to be well abstracted from each other and easily extensible.

B.4 Visualization

Visualiztion is the component that handles the plotting of graphs and tables. We note that this component is totally independent from the other two components. In visualization, we can write code to analyze the data and display it in a graphical manner. You may use the command 'explore.py –help' to display the man page to list all visualizations that can be done. A screenshot of the help message is given in Figure B.3.

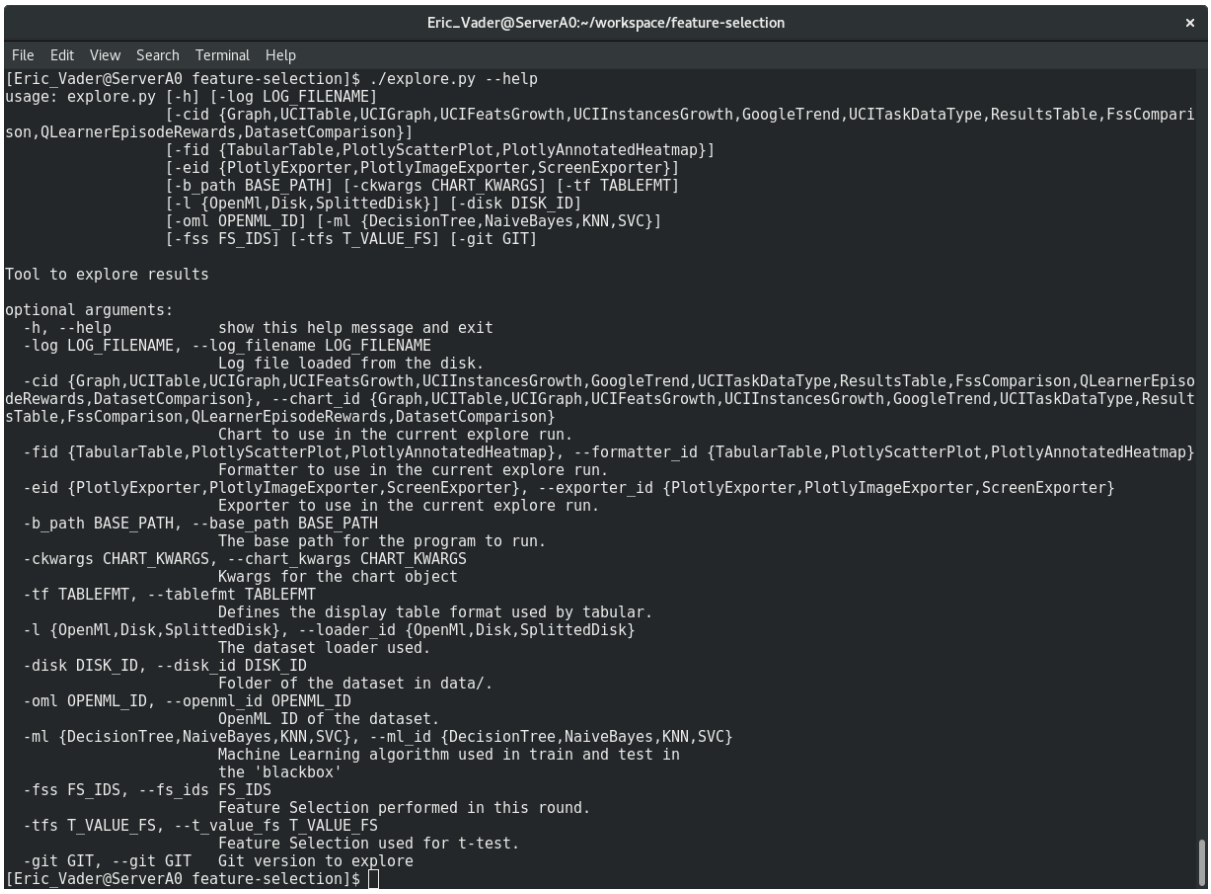

Figure B.3: Example output of explore's man page

Figure B.4: Example output of main's man page

Appendix C

Final Results

In this chapter we will list all the results from the final experiment. We will list both the Reward History for each of the Fast Q-Agent and also for the table comparing the Fast Q-Agent with all competing methods.

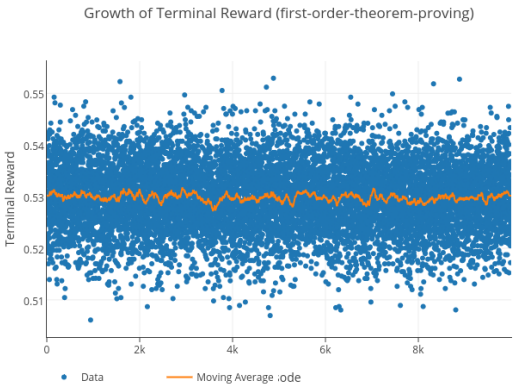

Figure C.1: History for OpenMl-1475

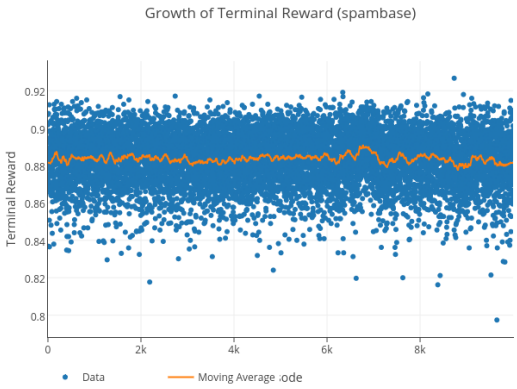

Figure C.2: History for OpenMl-44

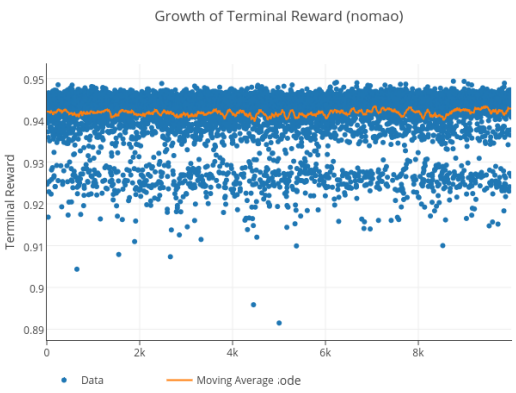

Figure C.3: History for OpenMl-1486

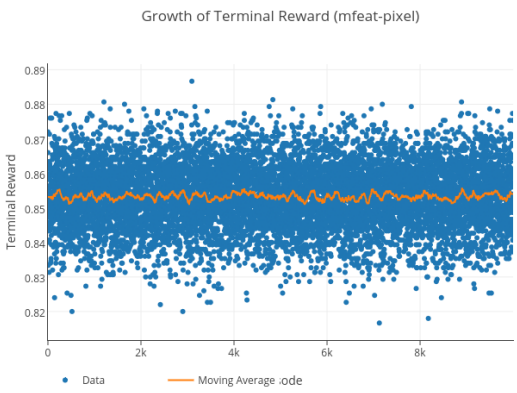

Figure C.4: History for OpenMl-20

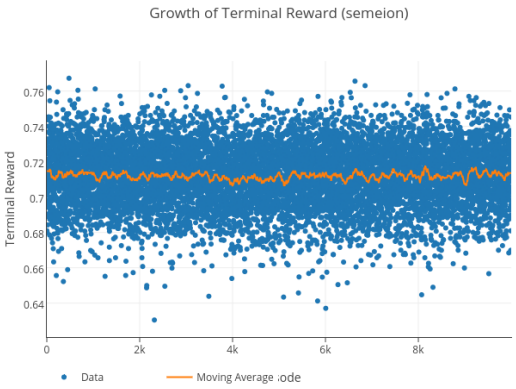

Figure C.5: History for OpenMl-1501

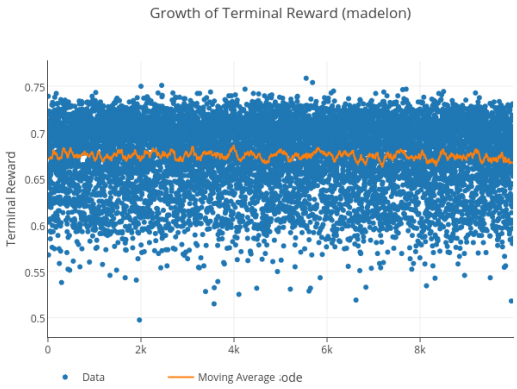

Figure C.6: History for OpenMl-1485

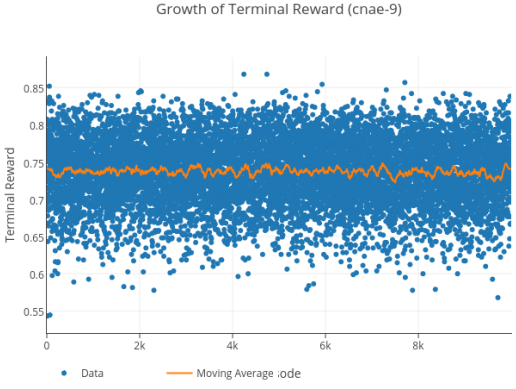

Figure C.7: History for OpenMl-1468

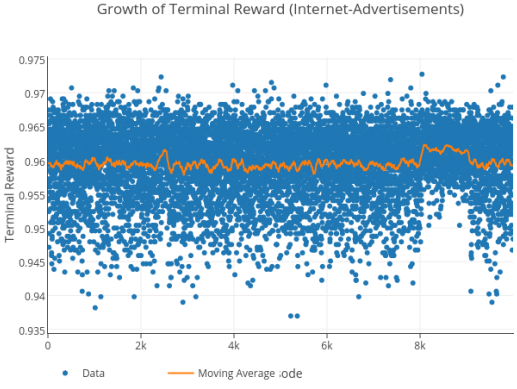

Figure C.8: History for OpenMl-1176

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:55:34.784336	RandomSearch	1.11415	30	0.553 ± 0.009	0.464 ± 0.032	0.00382751
0:00:00.000155	Baseline	1.94809	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:01:35.487266	Cmim	1.94352	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:06:42.144729	Disr	2.34561	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:02:47.768598	Icap	2.46861	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:01:34.727960	Jmi	1.95773	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:01:37.331538	Mim	1.94896	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:00:00.007013	VarThreshold	1.94189	51	0.527 ± 0.006	0.459 ± 0.033	0.0103016
0:00:25.278273	FisherScore	1.01678	24	0.547 ± 0.013	0.459 ± 0.026	0.00358087
0:00:20.206305	FScore	1.02652	24	0.547 ± 0.013	0.459 ± 0.026	0.00358087
0:00:51.599546	GiniIndex	2.42859	50	0.536 ± 0.008	0.458 ± 0.030	0.00816996
0:01:02.611036	LapScore	1.68358	40	0.541 ± 0.017	0.456 ± 0.021	0.00270193
0:04:40.266005	ForwardSelection	0.659277	21	0.557 ± 0.009	0.454 ± 0.032	0.0207543
0:01:11.012392	Relieff	1.15634	33	0.543 ± 0.013	0.454 ± 0.032	0.0239551
0:02:08.210622	Spec	1.37254	33	0.541 ± 0.012	0.453 ± 0.038	0.0472897
1:26:00.960628	SimulatedAnnealing	1.75874	29	0.534 ± 0.015	0.452 ± 0.037	0.0509341
0:15:19.552159	BackwardSelection	1.4787	28	0.552 ± 0.017	0.452 ± 0.032	0.0332113
0:06:51.067852	Cfs	0.282274	5	0.522 ± 0.010	0.432 ± 0.020	0.34623
2:22:35.913232	QLearner	0.322248	4	0.514 ± 0.020	0.424 ± 0.018	1
0:00:10.564024	Cife	0.222997	3	0.449 ± 0.009	0.406 ± 0.041	0.245997
0:00:06.655179	Mifs	0.125504	2	0.407 ± 0.008	0.351 ± 0.034	2.071e-05
0:00:07.177237	Mrmr	0.167625	2	0.407 ± 0.008	0.351 ± 0.034	2.071e-05
0:00:07.124033	Fcbf	0.115738	1	0.396 ± 0.009	0.342 ± 0.029	1.12203e-06

Table C.1: OpenMl - 1475

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:00:12.070542	FisherScore	0.424274	33	0.915 ± 0.007	0.897 ± 0.023	0.00186533
0:00:07.462365	FScore	0.424887	33	0.915 ± 0.007	0.897 ± 0.023	0.00186533
0:00:28.706980	Relieff	0.716664	53	0.910 ± 0.007	0.897 ± 0.018	0.00127612
0:00:00.000142	Baseline	0.755439	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:00:37.221192	Cmim	0.757258	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:04:06.670660	Disr	1.08531	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:01:17.438640	Icap	1.04812	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:00:36.838566	Jmi	0.74921	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:00:36.550050	Mim	0.750178	57	0.908 ± 0.006	0.893 ± 0.020	0.00290578
0:02:05.183228	ForwardSelection	0.294269	29	0.923 ± 0.008	0.892 ± 0.032	0.0115234
0:00:30.668332	LapScore	0.983525	55	0.910 ± 0.011	0.891 ± 0.027	0.0075007
0:01:31.001863	Spec	1.08994	56	0.910 ± 0.006	0.890 ± 0.024	0.00747671
0:00:00.003849	VarThreshold	0.539173	42	0.901 ± 0.013	0.889 ± 0.026	0.0103997
0:17:14.935999	RandomSearch	0.351473	32	0.922 ± 0.008	0.886 ± 0.018	0.00741214
0:00:15.402361	GiniIndex	0.901192	55	0.911 ± 0.009	0.885 ± 0.032	0.0257203
0:06:59.193245	BackwardSelection	0.271138	14	0.917 ± 0.009	0.874 ± 0.024	0.0756119
0:28:53.797283	SimulatedAnnealing	0.370481	28	0.882 ± 0.008	0.862 ± 0.031	0.375819
0:01:15.965164	Cife	0.655951	39	0.870 ± 0.008	0.854 ± 0.040	0.73058
1:15:56.242077	QLearner	0.273063	16	0.884 ± 0.008	0.848 ± 0.034	1
0:27:23.117506	Cfs	0.215215	13	0.848 ± 0.016	0.842 ± 0.036	0.723557
0:00:02.348845	Fcbf	0.0282051	2	0.755 ± 0.011	0.770 ± 0.035	0.000133581
0:00:06.181050	Mifs	0.10132	2	0.708 ± 0.007	0.713 ± 0.038	2.42617e-07
0:00:06.920748	Mrmr	0.105568	2	0.708 ± 0.007	0.713 ± 0.038	2.42617e-07

Table C.2: OpenMl - 44

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
1:36:16.117269	ForwardSelection	3.7901	67	0.952 ± 0.000	0.937 ± 0.006	0.246521
10:01:43.145255	QLearner	4.93588	46	0.944 ± 0.002	0.933 ± 0.006	1
0:04:50.435381	GiniIndex	15.7306	117	0.947 ± 0.002	0.933 ± 0.009	0.874301
8:20:07.247130	BackwardSelection	10.8041	77	0.949 ± 0.002	0.933 ± 0.006	0.84342
0:04:38.058166	FScore	9.53049	91	0.947 ± 0.002	0.932 ± 0.007	0.564476
0:00:00.000144	Baseline	10.7084	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
1:34:18.259437	Disr	17.7005	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
0:17:36.542732	Jmi	11.1647	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
0:19:20.874951	Mim	10.9183	118	0.946 ± 0.002	0.931 ± 0.011	0.615029
7:51:02.847676	SimulatedAnnealing	7.23281	60	0.944 ± 0.001	0.931 ± 0.007	0.396528
0:15:54.309767	Cmim	9.33369	95	0.944 ± 0.002	0.931 ± 0.009	0.467777
0:33:20.290407	Icap	14.8716	95	0.944 ± 0.002	0.931 ± 0.009	0.467777
0:34:29.493694	Relieff	8.82695	97	0.947 ± 0.003	0.931 ± 0.009	0.443816
0:10:03.007672	LapScore	15.036	99	0.946 ± 0.002	0.929 ± 0.008	0.257053
4:18:29.176265	RandomSearch	4.91086	61	0.949 ± 0.001	0.928 ± 0.008	0.126182
0:00:00.071364	VarThreshold	2.18691	34	0.937 ± 0.002	0.923 ± 0.008	0.00750944
0:00:48.514108	Fcbf	0.15799	4	0.926 ± 0.002	0.923 ± 0.007	0.00206919
0:28:09.306398	Cife	9.00142	71	0.933 ± 0.004	0.918 ± 0.008	0.000140857
0:01:44.384335	Mifs	0.422502	3	0.905 ± 0.003	0.889 ± 0.011	4.23588e-09
0:00:57.197468	Mrmr	0.407047	2	0.904 ± 0.003	0.889 ± 0.010	6.92833e-10
1:54:15.437697	Cfs	1.52938	11	0.885 ± 0.003	0.864 ± 0.007	1.36914e-14

Table C.3: OpenMl - 1486

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:01:19.829607	GiniIndex	1.15021	235	0.875 ± 0.012	0.864 ± 0.045	6.9689e-13
1:54:33.645320	BackwardSelection	0.263303	75	0.897 ± 0.010	0.858 ± 0.029	6.34525e-14
0:01:30.542317	FisherScore	0.638498	181	0.872 ± 0.020	0.850 ± 0.054	5.19541e-12
0:01:31.206136	Relieff	0.744162	229	0.867 ± 0.012	0.850 ± 0.056	7.46172e-12
0:00:32.470246	FScore	0.474519	181	0.872 ± 0.020	0.850 ± 0.054	5.19541e-12
0:00:00.000225	Baseline	0.541078	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:11:46.273304	Cife	0.981002	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:06:18.734724	Cmim	0.71556	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:26:15.942612	Disr	0.91066	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:11:41.825062	Icap	1.21013	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:06:11.674446	Jmi	0.736893	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:06:15.372993	Mim	0.717024	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:11:45.879622	Mrmr	1.00779	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:00:00.013499	VarThreshold	0.620376	240	0.852 ± 0.029	0.848 ± 0.046	1.47163e-12
0:02:54.542130	Spec	1.09956	209	0.863 ± 0.022	0.842 ± 0.042	1.08854e-12
0:16:44.323539	RandomSearch	0.224161	97	0.888 ± 0.014	0.842 ± 0.038	5.67568e-13
0:02:01.953281	LapScore	0.738668	185	0.875 ± 0.023	0.834 ± 0.043	1.71276e-12
0:30:54.064755	SimulatedAnnealing	0.350162	103	0.851 ± 0.015	0.818 ± 0.045	5.1503e-12
0:33:34.092850	ForwardSelection	0.114918	46	0.901 ± 0.015	0.806 ± 0.065	2.33986e-10
0:00:16.179755	Fcbf	0.0776708	28	0.861 ± 0.009	0.788 ± 0.044	1.85837e-11
0:00:35.078795	Mifs	0.0525891	14	0.845 ± 0.024	0.778 ± 0.083	1.21724e-08
4:26:35.210203	QLearner	0.0776837	10	0.463 ± 0.023	0.460 ± 0.051	1

Table C.4: OpenMl - 20

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:00:23.102075	Relieff	0.330859	221	0.760 ± 0.039	0.672 ± 0.053	1.99343e-12
1:37:30.097695	BackwardSelection	0.148785	57	0.744 ± 0.025	0.665 ± 0.075	5.03733e-11
0:00:17.734274	FisherScore	0.312971	213	0.754 ± 0.022	0.639 ± 0.067	4.54604e-11
0:00:17.171533	FScore	0.306738	213	0.754 ± 0.022	0.639 ± 0.067	4.54604e-11
0:00:12.596749	Fcbf	0.0611012	30	0.739 ± 0.036	0.636 ± 0.041	1.40086e-12
0:00:52.577964	Spec	0.580903	253	0.753 ± 0.029	0.624 ± 0.061	3.84793e-11
0:07:16.299544	Cife	0.571856	248	0.734 ± 0.032	0.621 ± 0.073	2.15609e-10
0:01:31.944594	Mifs	0.0909343	30	0.736 ± 0.026	0.621 ± 0.092	2.30388e-09
0:00:00.000146	Baseline	0.381868	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:02:55.279051	Cmim	0.379929	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:19:37.972723	Disr	0.585724	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:06:54.427827	Icap	0.593921	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:02:53.943105	Jmi	0.377431	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:02:55.190315	Mim	0.379683	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:07:24.278532	Mrmr	0.598911	256	0.738 ± 0.029	0.619 ± 0.064	7.55184e-11
0:00:44.663909	GiniIndex	0.589341	248	0.742 ± 0.026	0.619 ± 0.059	3.4546e-11
0:00:37.779056	LapScore	0.539122	233	0.769 ± 0.034	0.619 ± 0.081	6.75624e-10
0:11:31.291035	RandomSearch	0.193505	126	0.768 ± 0.024	0.616 ± 0.055	2.40933e-11
0:23:26.649914	SimulatedAnnealing	0.287252	119	0.710 ± 0.013	0.614 ± 0.045	6.13027e-12
0:26:08.704675	ForwardSelection	0.176301	117	0.788 ± 0.020	0.609 ± 0.048	1.12212e-11
0:00:00.007039	VarThreshold	0.370254	251	0.742 ± 0.017	0.604 ± 0.070	2.96059e-10
4:28:53.949300	QLearner	0.0298998	3	0.210 ± 0.027	0.223 ± 0.060	1

Table C.5: OpenMl - 1501

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:09:12.731607	Relieff	0.168278	10	0.808 ± 0.022	0.786 ± 0.041	2.48043e-08
1 day, 0:53:18.562516	ForwardSelection	0.175849	11	0.812 ± 0.013	0.772 ± 0.063	2.7824e-07
0:09:06.006739	FScore	0.245874	18	0.775 ± 0.014	0.732 ± 0.058	1.62882e-06
0:09:00.943109	FisherScore	0.208157	13	0.768 ± 0.010	0.730 ± 0.055	1.3507e-06
0:09:13.168024	LapScore	1.83497	138	0.735 ± 0.051	0.646 ± 0.078	0.000821486
0:18:34.934207	Spec	10.2162	417	0.724 ± 0.016	0.634 ± 0.048	0.000373792
5:37:00.896408	SimulatedAnnealing	5.91563	254	0.689 ± 0.020	0.619 ± 0.092	0.00702797
0:00:00.000163	Baseline	6.80957	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
1:47:17.988734	Cife	12.5701	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
0:42:35.088874	Cmim	6.7377	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
4:43:21.957977	Disr	10.6805	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
1:48:31.514008	Icap	11.4206	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
0:43:05.279648	Jmi	6.72532	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
0:44:12.845467	Mim	6.43507	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
0:00:00.024159	VarThreshold	6.4335	500	0.699 ± 0.015	0.604 ± 0.049	0.00299923
3:08:06.592670	RandomSearch	3.2198	226	0.751 ± 0.028	0.582 ± 0.087	0.039291
0:20:45.686222	GiniIndex	11.7389	498	0.708 ± 0.022	0.566 ± 0.053	0.0401261
0:00:37.022085	Mifs	0.0649647	2	0.532 ± 0.015	0.517 ± 0.061	0.487267
0:00:35.646724	Mrmr	0.0658895	2	0.532 ± 0.015	0.517 ± 0.061	0.487267
0:00:15.466253	Fcbf	0.0369353	1	0.509 ± 0.012	0.507 ± 0.069	0.679352
14:58:58.678453	QLearner	0.575607	17	0.498 ± 0.015	0.492 ± 0.085	1
0:52:54.012255	Cfs	0.0572776	2	0.519 ± 0.017	0.474 ± 0.054	0.587402

Table C.6: OpenMl - 1485

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:10:37.140378	Mifs	0.102765	87	0.878 ± 0.012	0.811 ± 0.084	1.80861e-15
5:28:53.356800	ForwardSelection	0.044595	59	0.915 ± 0.027	0.778 ± 0.050	4.35573e-19
0:01:07.560737	FisherScore	0.1151	195	0.881 ± 0.009	0.774 ± 0.061	1.70658e-17
0:12:44.246909	RandomSearch	0.262623	456	0.853 ± 0.018	0.767 ± 0.076	1.08163e-15
0:01:11.053690	Relieff	0.198902	361	0.873 ± 0.013	0.752 ± 0.062	4.59187e-17
0:00:00.000177	Baseline	0.632789	856	0.853 ± 0.011	0.748 ± 0.081	5.23455e-15
2:31:40.608685	Disr	0.98902	856	0.853 ± 0.011	0.748 ± 0.081	5.23455e-15
0:21:46.721482	Mim	0.6005	856	0.853 ± 0.011	0.748 ± 0.081	5.23455e-15
0:02:36.830234	Spec	0.668822	648	0.869 ± 0.018	0.741 ± 0.064	1.11494e-16
1 day, 2:15:26.624984	BackwardSelection	0.397202	371	0.880 ± 0.028	0.741 ± 0.070	5.55656e-16
0:02:40.563224	GiniIndex	1.01982	823	0.872 ± 0.020	0.733 ± 0.062	6.58014e-17
0:01:22.927546	LapScore	0.37198	589	0.875 ± 0.022	0.722 ± 0.058	3.15565e-17
0:00:20.768834	Fcbf	0.0415156	53	0.760 ± 0.016	0.722 ± 0.060	6.31614e-17
0:21:25.767443	Jmi	0.513683	749	0.860 ± 0.014	0.722 ± 0.038	1.67882e-20
0:55:24.749768	Mrmr	0.812156	749	0.860 ± 0.014	0.722 ± 0.038	1.67882e-20
0:01:07.343254	FScore	0.502196	759	0.873 ± 0.010	0.719 ± 0.090	7.08283e-14
0:16:37.204570	Cmim	0.236099	447	0.848 ± 0.018	0.715 ± 0.076	4.57983e-15
0:41:59.830839	Icap	0.43838	447	0.848 ± 0.018	0.715 ± 0.076	4.57983e-15
0:29:05.269804	Cife	0.260699	273	0.778 ± 0.026	0.689 ± 0.074	6.87768e-15
0:30:47.701238	SimulatedAnnealing	0.671319	437	0.665 ± 0.059	0.633 ± 0.109	2.70098e-11
0:00:00.015352	VarThreshold	0.0251744	14	0.610 ± 0.027	0.619 ± 0.098	7.25262e-12
9:24:19.005616	Cfs	0.0453169	23	0.181 ± 0.011	0.181 ± 0.026	1.90293e-07
17:55:58.230548	QLearner	0.0282976	3	0.117 ± 0.004	0.111 ± 0.000	1

Table C.7: OpenMl - 1468

Duration	Feature Selection	Test Time	#Features Selected	Train Accuracy(5-CV)	Test Accuracy(10-CV)	T-Test
0:08:54.693816	Mifs	0.0653412	10	0.966 ± 0.007	0.966 ± 0.021	1.94316e-11
0:01:37.315322	Fcbf	0.0696286	27	0.966 ± 0.005	0.962 ± 0.019	4.53509e-12
2:23:06.788162	LapScore	31.3766	1202	0.967 ± 0.007	0.956 ± 0.025	1.38871e-09
0:36:54.340197	FisherScore	0.614468	320	0.968 ± 0.007	0.956 ± 0.021	6.38796e-11
0:37:45.029520	FScore	0.627036	320	0.968 ± 0.007	0.956 ± 0.021	6.38796e-11
0:41:38.924559	Relieff	9.12981	1125	0.967 ± 0.008	0.956 ± 0.023	3.29754e-10
10:44:16.936349	Mrmr	26.3022	1366	0.960 ± 0.006	0.953 ± 0.029	1.33938e-08
2:23:07.322547	Spec	26.649	1089	0.967 ± 0.006	0.952 ± 0.028	1.15731e-08
0:00:00.000179	Baseline	14.4637	1558	0.961 ± 0.006	0.951 ± 0.024	1.58406e-09
1 day, 2:58:25.737611	Disr	24.7585	1558	0.961 ± 0.006	0.951 ± 0.024	1.58406e-09
3:37:40.694460	Jmi	14.5923	1558	0.961 ± 0.006	0.951 ± 0.024	1.58406e-09
3:41:42.032491	Mim	13.7794	1558	0.961 ± 0.006	0.951 ± 0.024	1.58406e-09
5:02:22.638143	RandomSearch	5.58867	771	0.972 ± 0.004	0.950 ± 0.028	1.60871e-08
3:02:06.300343	Cmim	7.77176	895	0.965 ± 0.008	0.950 ± 0.023	1.10755e-09
8:48:37.732928	Icap	19.5376	895	0.965 ± 0.008	0.950 ± 0.023	1.10755e-09
12:38:05.854345	SimulatedAnnealing	14.0515	761	0.959 ± 0.007	0.947 ± 0.023	1.23609e-09
4:29:39.091186	GiniIndex	39.6334	1551	0.968 ± 0.008	0.939 ± 0.026	4.91712e-08
0:00:00.087906	VarThreshold	0.0549575	11	0.938 ± 0.005	0.934 ± 0.023	2.23035e-08
6:52:43.746108	Cfs	0.0732549	11	0.942 ± 0.009	0.932 ± 0.030	1.05412e-06
5:28:32.235757	Cife	8.97778	507	0.960 ± 0.007	0.930 ± 0.028	5.98613e-07
1 day, 12:36:51.281438	QLearner	0.0476941	19	0.860 ± 0.001	0.860 ± 0.004	1

Table C.8: OpenMl - 1176