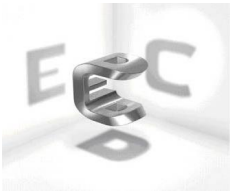
ECD Master Thesis Report



INCREMENTAL DECISION TREES

Bassem Khouzam

01/09/2009

Advisor: Dr. Vincent LEMAIRE

Location: Orange Labs

Abstract:

In this manuscript we propose standardized definitions for some ambiguously used terms in machine learning literature. Then we discuss generalities of incremental learning and study its major characteristics. Later we approach incremental decision trees for binary classification problems, present their state of art, and study experimentally the potential of using random sampling method in building an incremental tree algorithm based on an offline one.

Résumé :

Dans ce manuscript, nous proposens des définitions standardiséss pour quelques notions utilisées d'une manière ambiguë dans la literature d'apprentissage automatique. Puis, nous discutons les généralités d'apprentissage incrémentale et étudions ses majeurs charactéristiques. Les arbres de décision incrémentaux pour les problèmes de classification binaire sont après abordés, nous présentons l'état de l'art, et étudions empiriquement le potentiel de l'utilisation de la méthode l'echantillonage aléatoire en construisant un algorithm incrémentale d'arbre de décision basé sur un algorithm hors ligne.

" To Liana and Hadi ..."

Acknowledgments

I would like to thank all people who have helped and inspired me during my internship. I especially want to express my gratitude to my supervisor ***Dr. Vincent LEMAIRE*** for his inspiration and guidance during my internship at Orange Labs. He shared with me a lot of his expertise and research insight, his geniality was materialized in precious advices led me always in the right way. I am particularly grateful for his special courtesy that is second to none.

I was delighted to interact with ***M. Nicolas VOISINE*** by having him as a my supervisor for a short period of time, his unique vision in machine learning helped me a lot to push my experimentation ahead.

I am grateful to Professor ***José del Campo Avila and his research team members*** in the department of languages and sciences of computation, university of Malaga, Spain for his ultimate cooperation concerning his IADEM and IADEMc algorithms, and his priceless confidence and generosity concerning the unpublished source code of his Algorithms.

As this work is the end of my ECD Master degree, I would like to take the opportunity to thank all my professors and staff at the university Lyon 2. I am endlessly grateful to ***Dr. Julien VELCIN***, head of master 2 ECD and machine learning teacher, for his kind sympathy towards the difficulties I passed by. He gave me a head start in my new domain by clarifying all ambiguous concepts in KDD. His support, encouragement and trust accompanied me until today, making from him an unforgettable person in my life. Special thank go to ***Valerie GABRIELE***, secretary of master ECD, she helped me in all administrative works, and facilitated life to all ECD master students.

All my lab buddies at Orange Labs made it a convivial place to work. In particular, I would like to thank ***Aurélie LE CAM*** for her friendship and assistance in the past five months. Thanks go also to my literate friend ***Dr. Artem Sokolov*** who revised the language of this manuscript.

Finally, my deepest gratitude goes to ***my family*** for their unflagging love and support throughout my life. This love and care is now devoted to my beloved son who has lived among them during my absence.

BassemKhouzam : LBassem@gmail.com

Contents

List of Tables

List of Figures

1 Orange and France Telecom Group

One of the main telecommunication operators in the world, providing services to more than 170.000.000 customers over five continents. Number 3 in Europe for mobile services with almost 110.000,000 customers, European leader in broadband Internet (ADSL) and ADSL television. it has 17 research and development laboratories on 4 continents and 9 countries, with 5000 staff distributed between researchers, marketers, and engineers producing over than 400 inventions per year.

Orange Labs R&D is one of the most innovative operators on European market in 2004 and "A french revolution in innovation is unfolding" in 2006 according to Forrester. Best innovator France 2005 in the "innovation & technologies" category and "Best innovator" France 2007 in the category of "Organisational innovation and Marketing/R&D/IS network partnership" - according AT Kearney 2007.

R&D mission is an Orange two folds mission: in *research*, Orange's mission is to detect technological breakthroughs and acquire the know-how, produce high level intellectual property, explore new forms of technology, services and usage, and reduce the technical risk. In *development* orange seeks always to reduce time to market, industrialize products, services and network upgrades, design the services of the future while improving existing ones, and contribution to standardization.

Orange Labs subjects of research are defined on the basis of priority topics covering: network convergence and content access (voice, data, and Internet) irrespective of the Internet and terminal used, opening up to the ecosystem, having new partners and new communities, develop application to countries with high technical and economic constraints, optimisation of cost, and sustainable development and health care.

The team PROF[1] (PROFiling and datamining) is a research team specialized in information statistical manipulation and development of advanced data mining algorithms. The laters are intended to serve in many application like profiling, scoring, and recommendation. This team transmits his expertise to operational and marketing teams and help them in implementing their technical or strategic choices (methodologies, architectures, internal and external technologies, partnership..)

This internship falls in the context of continuous research efforts of Orange Labs R&D. Each year, they receive numerous internship students besides to PhD and postdoctoral scholarship ones. Students involve Orange research works when they comply to their studying subjects. Their efforts has pure research orientation, the whole priority is given to research paths that comply to their original subjects. This work is an example.

[1] This team was named TSI (Traitement Statistique de l'Information) until 07/01/2009.

Notations:

Here are the notations used in this manuscript:
S: Examples date's
M: A prediction model
X: Space of input variables of the model M
Y: Space of output variables of the model M
$f : X \rightarrow Y$: Target concept which the model tries to learn
$\tilde{f} : X \rightarrow Y$: Effective concept learnt by the model
h_i: Learning algorithm output hypothesis.
$I(S)$: Information entropy of the dataset S.
$ing(A)$: Information gain obtained by splitting the tree using attribute A
$ingr(A)$: Information gain ratio obtained by splitting the tree using attribute A
$gini$: Gini index for the dataset S.
$G()$: Heuristic measure used to choose test attributes.

2 Introduction

One of the most recent problems studied in the latest years is how to learn when the dataset is large or when new information can arrive at any time.

There is plenty of work trying to accommodate these situations, and many concepts were recently proposed in machine learning world, yet, no standard definitions exist for these concepts.

In this section, we are interested in finding standardized definitions to some confusing terms used in machine learning literature. Our definitions will consider classification problems rather than clustering ones, which are out of the scope of this study.

2.1 Supervised learning

In human and animal psychology, concept learning refers to the mental categories that help to classify objects, ideas, or events based on their features. It is defined in Wikipedia as " the search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories."

Development in machines and algorithms allowed machines to mimic humans and animals in their ability to discover categories or assign values to objects based on their attributes after learning some examples. This allowed extending the notion of concept learning to include a kind of machine learning, called supervised learning.

More specifically, supervised learning is the task of learning a function that is able to assign a label (classification) or a continuous value (regression) to some valid input, after having seen a set of training data, presented in pairs of input objects and output labels or values. The model will learn the rules that help to assign the appropriate labels or values to input objects (or examples) based on regarding their attributes (or variables). This is called **induction phase**, then it will be generalized and used to assign labels or values (or classes) to new input objects, this is the called **prediction phase**. Figure [1] shows the learning (induction) and working (prediction) phases of supervised learning.

Machine learning algorithms are not always supervised, other types of learning are unsupervised learning, semi-supervised learning, reinforcement learning, transduction learning, and learning to learn. However, unsupervised learning forms with supervised learning the two major domains that witnessed wide interest and got most efforts in machine learning world. Unsupervised Fernande's from supervised learning in that no previous information about the classes of examples is provided. On the Contrary, it focuses on grouping examples in clusters based on their natural similarity. Unsupervised learning is out of the scope of this study.

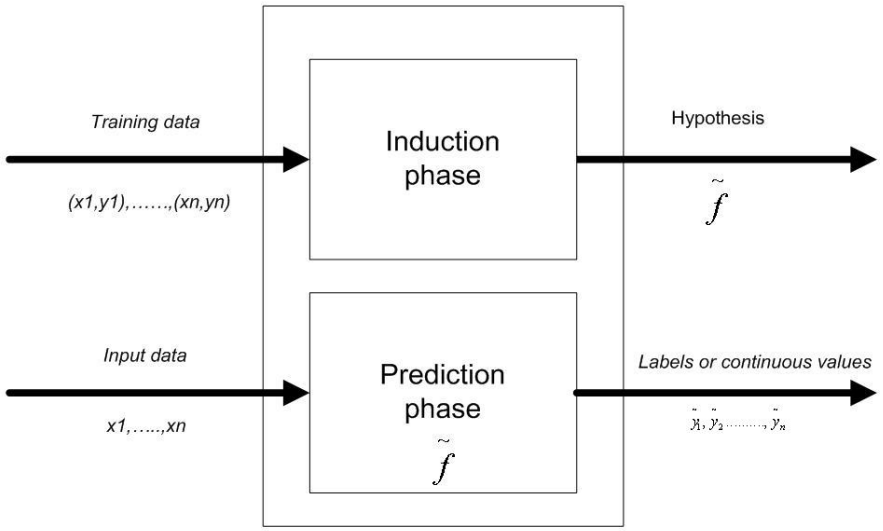


Figure 1: Supervised learning

2.2 Classification

As mentioned above, classification is a type of supervised learning where training data consist of examples labelled with finite number of discrete values that may be nominal, ordinal, or numeral values. Thus, in prediction phase, the function learnt by the classifier will output labels having the same nature and number of labels existing in training examples. Classification problem can be stated as follows:

Given a set of training data $(x_1, y_1), (x_2, y_2), .., (x_n, y_n)$ from $X \times Y$ having some implied unknown mapping $f : X \to Y$ between them, we are interested in finding the classifier $\tilde{f} : X \to Y$ that correctly assigns a true y to each x from X.

The rest of this section is organized as follows: a good start point will be to define offline learning, later we will talk about batch learning as a special type of offline learning. Sequential learning then will be approached and its intersection with offline learning will be clarified, it will be defined after listing some citations. We will continue by listing citations and define online learning and distinguish it from sequential learning. Incremental learning and its overlapping with online and sequential learning will be presented and defined. Anytime learning then will be presented and defined. Later we will remind with active learning as a special kind of supervised learning. We conclude this section with talking about data memorizing in classification models.

3 Types of supervised learning

3.1 Offline Learning

In offline learning, sample data that express the studied phenomenon are independently drawn from its population, according to the probabilistic distribution of these data, and then they are assigned the correct labels or classes. Later this data is presented to a learning system to build a classifier that verifies low real error and low generalization error, based on an underlying hypothesis. Then it will be validated and generalized. The second phase of the classifier's life cycle is the working phase; the classifier receives non-labelled examples and uses the learnt hypothesis to assign them the appropriate labels.

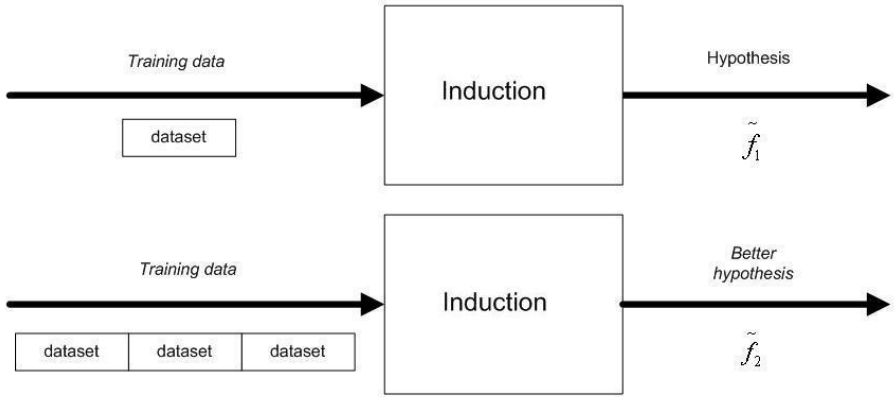


Figure 2: Batch learning from repeated dataset

3.2 Batch learning

Batch learning is the traditional offline learning mode. Labelled examples are first collected, then they are presented as one batch to the classification system to build the appropriate hypothesis. After the hypothesis has been learnt, the classification system can be put in the working phase.

Learning in batches of examples from the pre-collected dataset can be repeated on the same training dataset, aiming to improve the hypothesis quality before producing it by reducing the real error. According to the application, another possible choice is to stop the learnt classifier in the working phase and retrain the system again by batch of examples belonging to the same training dataset or to an updated training dataset. Training from a repeated dataset is accepted to improve hypothesis quality, because we admit that the dataset is a representative sample for its population. Batch learning from one dataset or from a repetition of a dataset is shown in figure [2].

3.3 Sequential learning

Over time, batch learning was proved to be hard to control and easy to diverge while learning in sequences of data was proved to be steady and to give the best results.

Sequential learning is a machine learning method that simulates the way we learn in real life, we accumulate received information to build knowledge, without knowing what will come on later.

"Pure sequential learning is important as a model of how learning occurs in real, biological brains. Humans and other animals are often observed to learn new things without forgetting old things." [42]

In offline learning, labelled examples are accumulated before model learning, and training occurs as a distinct step, this is why offline learning is sometimes called "one-shot learning". Differently, examples in sequential learning may not be available, but they become available over time. The model discards examples after processing them one by one or chunk by chunk. It is not needed to do a complete retraining on complete data.

"Sequential learning doesn't require retraining whenever a new data is received". [16]

Another possible necessity for sequential learning appear when gathered data are so huge to be fed to the learning algorithm at one time and it will be impractical to run the algorithm on a commercial machine because the processing time will be too long.

Sequential algorithms stop and return a hypothesis when example sequence ends; alternatively they could stop automatically when some condition is verified. For example in [19], if the distribution of examples is known in advance, the algorithm can be fed with the sufficient example size to obtain a good hypothesis, while in distribution independent learning, sample size can be determined by a concept class' mistake.

It is important to notice that the term "sequential learning" was used in literature to express different methods of learning, almost all of them adopt the principle of sequential flow of examples that are used to build a classifier's hypothesis, but we can find some works that use this term to denote other methods, for example to denote classifiers that label a sequence of incoming examples with one label.

[39] categorizes sequential learning into four main categories:

- Sequence prediction attempts to predict elements of a sequence on the basis of the preceding elements.
- Sequence generation attempts to generate elements of a sequence one by one in their natural order.
- Sequence recognition attempts to determine if a sequence is legitimate according to some criteria.
- Sequential decision making involves selecting a sequence of actions to accomplish a goal, to follow a trajectory, or to maximize or minimize a reinforcement (or cost) function that is normally the (discounted) sum of reinforcements (costs) during the course of actions.

Another categorization appears in [24]: "From these data, these systems extract information that may be used to detect objects (classification problem), or to estimate the parameters of a process (estimation problem), or to interact with its environment (sequential decision problem)."

[42], talks about learning of neural networks, tries to differentiate between multiple concepts in machine learning world, and correct the arbitrarily used terms. To reveal confusion about sequential learning, he suggests the term "pure sequential learning" instead of "sequential learning", to describe sequential example learning that don't need storage of any parameters other than network weights.

Many definitions were presented implicitly or explicitly to sequential learning, example in [10] : "If the complete dataset is represented as a set S of examples $S = (x_1, y_1)..(x_t, y_t)..(x_m, y_m)$. Sequential learning is the problem of learning, from such a dataset, a sequential classifier, i.e., a function f such that $f(x)$ produces a vector of class labels y".

Sequential learning is one step further than offline learning towards online learning. Although it can handle examples that are presented to the model gradually over time, it still doesn't produce a hypothesis unless training is ended, instead, it updates its hypothesis internally after processing each example, but it is never produced to be used. Thus, sequential training, like offline learning, has still two separate phases for learning and working. See figure [3].

Concept intersection between offline learning and sequential learning, allows us to conclude that offline sequential learning classifiers may be learnt via a repeated dataset in order to improve the hypothesis quality.

Another variation of non-pure sequential learning when working offline is to keep processed examples to be accessed later or to keep some examples only as will be discussed when talking about data memorization. This prevents the hypothesis from "drifting" with new examples information, and guarantees that the hypothesis takes in account the oldest examples in the same degree as newest ones, thus saving the system from what is called "catastrophic forgetting" or "catastrophic interference". Controlling this phenomenon is sometimes called "stability-plasticity dilemma", we will consider it when approaching incremental learning.

Our Definition: we define sequential learning as the problem of learning a classifier that receives labelled examples sequentially, one by one or chunk by chunk, and updates a hypothesis gradually and returns it when it stops learning, either when examples sequence finishes or when the system itself decides autonomously to stop learning.

Remarks:

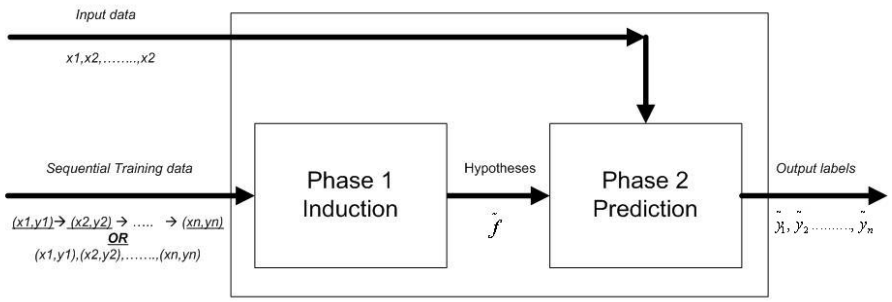


Figure 3: Sequential learning

- Sometimes, we find that sequential learning is used as a global notion that encapsulates all learning methods that process data sequentially (online learning and incremental learning), regardless of other considerations.
- Chunks of examples could be used to update the hypothesis by presenting its examples to the model one by one; this means that even if examples are received in chunks, they are not necessarily internally processed as chunks.

3.4 Online learning

Online learning holds its functionality in its name. It suggests that data is not collected before learning, instead they become available over time during the learning process. Online learning differs from sequential learning in many aspects, first of which is that there is no distinct phases for learning and operation.

"Online learning algorithms hold the promise of systems that adopt and learn gradually during operation without distinct phases of collection, labelling, training, and operation." [2]

While sequential learning algorithms produce a hypothesis after processing enough examples to enter later in operation mode, we find that online learning produces a hypothesis after processing each new example while being always able to label new examples.

"The on-line training changes all the weights within each backward propagation after every item from the training set." [41]

"Offline learning algorithms takes as input a set of training instances and output a hypothesis. In contrast Online Algorithms take as input a single labelled training instance as well as a hypothesis and output an updated hypothesis. Thus given a sequence of training instances an online algorithm will produce a sequence of hypothesis. Online algorithms are designed to reuse the previous hypothesis in various ways, allowing them to reduce update times and meet the constraint of online learning problems - the constraints are normally much tighter than for offline problems." [40]

Unlike pure sequential learning, online learning is authorized to store other parameters than model parameters. [42] gives some examples of these parameters.

"Pure sequential learning differs from on-line learning in that most on-line algorithms require storage of information in addition to the weights, such as the learning rate or approximations to the objective function or Hessian Matrix. Such additional storage is not allowed in pure sequential learning."

Like sequential learning, no access to previously processed examples is necessary. "In off-line learning, all the data are stored and can be accessed repeatedly. Batch learning is always off-line. In on-line learning, each case is discarded after it is processed and the weights are updated." [42].

Our definition: we define online learning as the problem of learning a classifier that receives labelled examples sequentially, one by one, and uses each of these examples with the previously learnt hypothesis

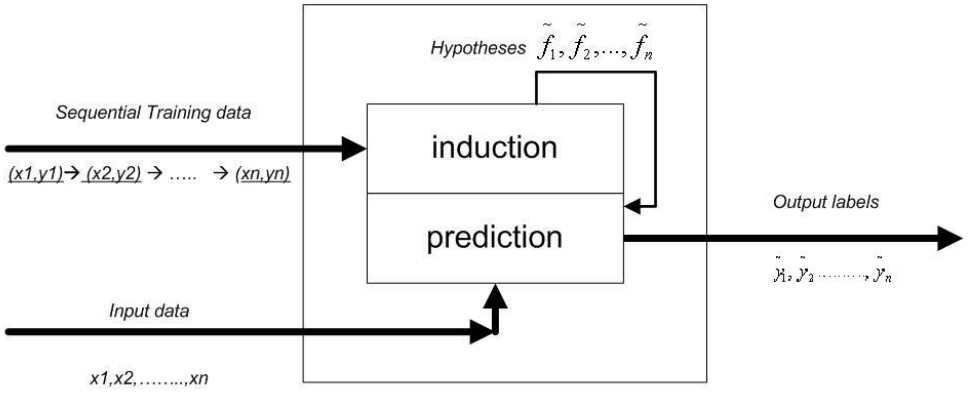


Figure 4: Online learning

to produce a new most-possible correct hypothesis regarding its changing environment, while respecting its constraints, for fully-working (learning and operation) in real time. see figure [4].

Remarks:

- Previous instances may be stored or not according to the used online algorithm.
- Online algorithm may be based in one learner or ensemble of learners.
- Online algorithms are designed to work in real time. They have to receive and process all coming examples. Thus, for high sample rates and complex algorithms, prediction accuracy is expected to decrease.
- Normally the most critical resources for an online algorithm are processing time and memory space.
- If the model's prediction of a new instance label is correct then hypothesis may be updated or not according to the used online algorithm.

3.5 Incremental learning

Regardless of our reservations, sequential and online learning methods are sometimes considered like incremental methods as in [26]: "Incremental learning methods (including online or sequential methods) have some features that make them very suitable to deal with huge amounts of data and to prevent excessively high memory requirements."

We think that incremental learning is considered sequential when using "sequential" as a global notion, as it receives examples sequentially, but it is different from sequential learning defined above in that incremental learning tends to produce a hypothesis after processing each new example while sequential learning updates hypothesis after processing each example but doesn't produce the hypothesis until the whole learning operation is completed. Hence, incremental learning is sequential by nature but differs from the specific definition of sequential learning.

[42] carefully cites the following concerning neural networks learning: "On-line training is always incremental" and "Incremental learning can be either online or offline".

This means that while online learning is always incremental, incremental learning may not necessarily be online. Consider the case when we have huge dataset to be learnt, model batch training is not feasible with current commercial computers as this requires impractical processing time and memory

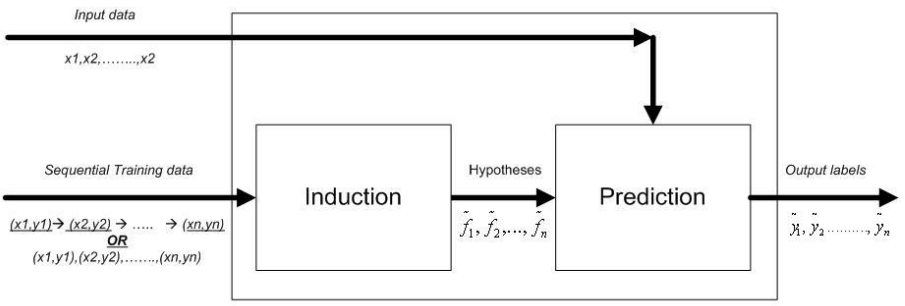


Figure 5: Incremental learning

requirements. However, it is preferred in this case to collect samples first and later feed the system sequentially with the previously collected examples. Here, we talk about *offline incremental learning.*

An experimental phenomenon shows that it is almost certain that incremental learning -in term of goal hypothesis production time- is better than offline learning. "It is believed that if the statistics for the E-step are incrementally collected and the parameters are frequently estimated, incremental learning converges more quickly because the information from the new data contributes to the parameter estimation more effectively than the batch algorithm does." [20]

When does incremental learning stop? [13] Gives the answer: "Incremental learning lasts almost indefinitely and the system can always produce an answer to a query using the best hypothesis learned so far." Clearly, this is not true when incremental learning is receiving data online for a specific period of time, or when it works offline with previously collected data that could be immense but limited in size.

A common dilemma is shared between incremental, online and sequential learning: "A learning algorithm is considered incremental if it can learn additional information from new data without having access to previously available data. This requires that the knowledge formerly acquired from previous data should be retained while new information is being learned, which raises the so-called stability-plasticity dilemma" [34]

The basic rules of incremental learning algorithms are listed in [33]:

1. It should be able to learn additional information from new data.
2. It should not require access to the original data, used to train the existing classifier.
3. It should preserve previously acquired knowledge (that is, it should not suffer from catastrophic forgetting).
4. It should be able to accommodate new classes that may be introduced with new data."

[13] defines incremental tasks and incremental algorithms as follows: " *Definition 1*: A learning task is incremental if the training examples used to solve it become available over time, usually one at a time. *Definition 2*: A learning algorithm is incremental if, for any given training sample $e_1, ..., e_n$, it produces a sequence of hypotheses $h_0, h_1, ..., h_n$ such that h_{n+1} depends only on h_i and the current example e_i."

Our definition: we define incremental learning as the problem of learning a classifier that receives labelled examples sequentially, one by one or chunk by chunk, and uses each of these examples with the previously learnt hypothesis to produce a new better hypothesis that encapsulates information held by the learnt examples so far. ***Or:*** online learning classifier that can be used offline. see figure [5].

Practical notes:

To help constructing an incremental learning system, [15] gives a good structural hint: "An incremental learning system should possess an information structure which can be updated incrementally. This structure is referred to as an Incremental Information Structure (IIS) by Fu [8]. An IIS should conserve useful information from examples as much as possible so that future updating can be constrained by this information."

Performance characteristics of incremental learning system are listed in [31]: "three most important assumptions characterizing an incremental learning system are: a) it must be able to use the learned knowledge at any step of the learning; b) the incorporation of experience into memory during learning should be computationally efficient (theory revision must be efficient in fitting new incoming observations); and, c) the learning process should not make unreasonable space demands, so that memory requirements increase as a tractable function of experience (memory requirements must not depend on the training size)."

To help design a logical workflow for incremental system operation in real time when it is learning online, [31] explains "a learning system should revise its learned knowledge when new observations are available, while still being able to provide at any stage of the learning process such a knowledge in order to carry out some task. This kind of learning is often named Incremental Learning."

Remarks:

- In [13] it is written:"Clearly, non-incremental learning algorithms can be used to solve incremental learning tasks. For example, the ID3 and back-propagation algorithms can learn by processing one example at a time. However, all previously encountered examples must remain available explicitly to enable learning - in ID3 to compute entropy and in back-propagation to compute mean sum-squared errors. In other words, learning incremental tasks with non-incremental algorithms requires complete re-training after each example."

- We have to insist on the idea that designing an incremental system to work offline, has different considerations from those which have to be taken in account when designing an incremental system to work online, as in online systems it is required to optimize hypothesis construction time to be shorter than incoming sample rate to get a system working in real time. This makes it difficult to always construct the best hypothesis. In incremental systems designed to work offline, there will be more flexibility in system design as such a constraint doesn't exist.

- The problem becomes more complex when considering sequentially learning system that accepts both labelled and unlabelled examples. This is one step farther for both semi-supervised learning and learning in sequence of examples domains. However, in this study we are interested only by supervised learning for classification problems.

3.6 Anytime learning

In [4] it is written: "The term anytime learning was introduced by [18] to denote a class of learning algorithms having the characteristics of anytime algorithms [12]: the algorithm can be suspended and resumed with negligible overhead, the algorithm can be terminated at any time and will return some answer, and the returned answers improve over time."

Hence, the key criterion of anytime algorithms is the possibility to be interrupted with negligible overhead. This is guaranteed especially when the environment could be described as "natural", i.e. all variations in the environment can be modelled by continuous functions, and no sudden variations in environment's phenomena could occur. Otherwise, in a non-natural environment in which events are severe, frequent and unexpected, if we suspend system learning for a while, it may loose its phenomena tracking and give decisions far from the real state of the environment. Here, no negligible overhead or suspending anytime algorithm is guaranteed.

Anytime learning resembles to online learning in that it learn and operates in real time, while it stays more general and more powerful.

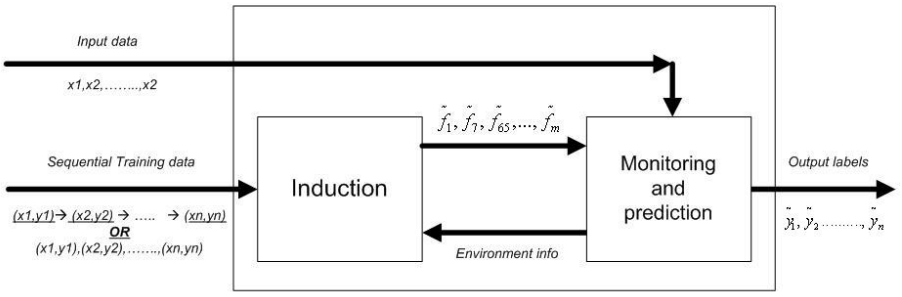


Figure 6: Anytime learning

Anytime learning systems differ from online learning that they are free from the narrow constraint of receiving environment samples one-by-one although they stay sequential in their behaviour, as they gather environment information gradually over time. Anytime systems may collect environment samples sequentially or in batches. Another difference is that online learning systems try to optimize the produced hypothesis regarding their environment conditions and the available resources, so it may not always produce the best hypothesis. At the inverse, anytime learning systems are more powerful systems due to their structure and complexity, they also contain efficient quality measurements to estimate the quality of controller under construction.

"What is special about anytime algorithms is the use of well-defined quality measures to monitor the progress in problem solving and allocate computational resources effectively." [47].

Our definition: Anytime learning is an ongoing process of learning, loading and execution of a classifier that receives the labelled examples collected by its host system, one-by-one or chunk by chunk, and integrates information held by them with the previously acquired knowledge to produce an improving hypothesis by time, for tracking its environment's changes in real time. see figure [6].

Appendix [.1] gives more details about the characteristics, structure and functionality of anytime learning systems.

3.7 Active learning

A special case of supervised learning arises when there are plenty of unlabelled examples because labeling is expensive, and few examples are only labeled, but it is possible to ask or query for the label of any example. An example of the expensive cost of labeling is to know if the enterprise's $15 * 10^6$ clients are satisfied with the services or not. This interactive kind of supervised learning is called active learning.

An important presentation on active learning can be found in [38], from which we quote: "The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer labeled training instances if it is allowed to choose the data from which is learns. An active learner may ask queries in the form of unlabeled instances to be labeled by an *oracle.*"

In active learning, the "learner" should be able to make decisions and to estimate the suitability of its decisions. The learner may not be always sure of its decisions, so it asks the help of an expert system, normally a human, the latest is called "the expert", and his role is to label the selected examples by the "active learning strategy". Learner-expert relation, resembles to student-teacher relation in real life, student can ask his teacher to correct him when he is not sure of his answer.

Active learning aims to select examples that help the best to develop the learning process of the concept/hypothesis, with a medium cost (low cost doesn't necessarily mean low examples number, except when example's labeling cost is equal). The number of examples required is much lower than the number

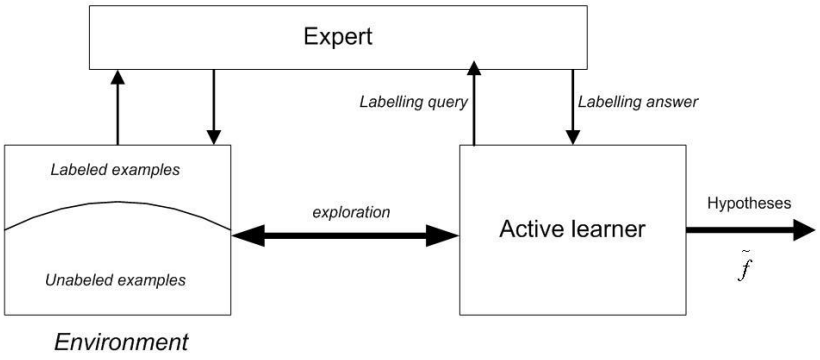


Figure 7: Active learning

required for normal supervised learning. As the reader may expect, there surely will be negative aspects for this miss of labeled examples, that is, the possibility of algorithm focusing on unimportant or even anomalous examples, thus obtaining a bad hypothesis.

Two scenarios are followed for examples sampling in active learning, selective sampling and adaptive sampling. The main difference between these two types of sampling is the nature of examples presented to the expert [5]. Selective sampling present examples to the expert that belongs to a fixed size set of examples, while adaptive sampling permits to an active learning strategy to generate examples by combining any set of variables values. This allows discovering all input space of examples; which is not true in the case of selective sampling. Sampling scenario is chosen according to the application nature.

Although active learning is not a vague term and there is wide agreement in literature on its meaning, we will propose our own definition.

Our Definition: we define active learning as the ability of a learning system to interact with its environment by choosing a subset of unlabeled examples, with the minimum labelling cost possible, which could help the best to develop the system's hypothesis. See figure [7].

3.8 Synthesis

table [1] and figure [**??**] show the synthesis of the major confusing concepts.

- Sequential learning:
  - Can be online (limited time) *, or offline
  - Returns one hypothesis
  - Not real time
  - One-by-one or chunks
  - Separate phases for learning and operation
- Online learning:
  - Online
  - Returns a series of hypotheses
  - Real time (limited accuracy) **
  - One-by-one only

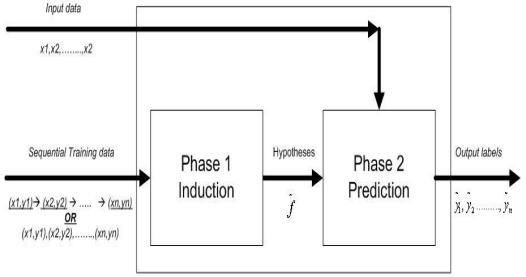
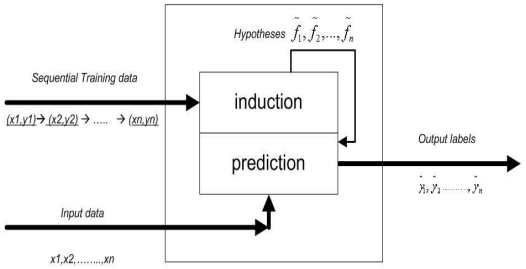
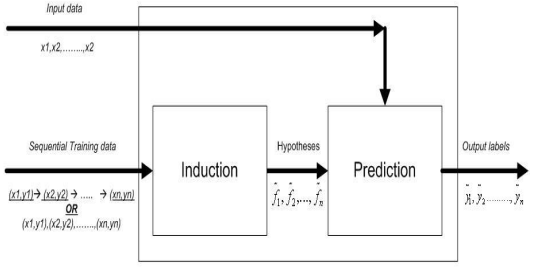
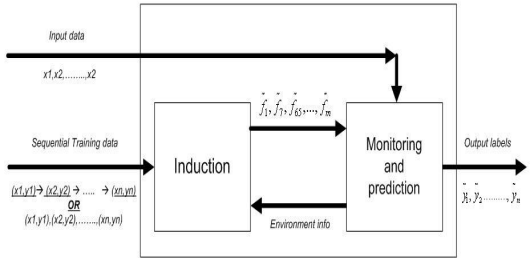
Learning method	Online/Offline	Series of Hypotheses	Real time	Handle Chunks	One phases learning/ Operation
Sequential	Yes*	No	No	Yes	No
Online	No	Yes	Yes**	No	Yes
Incremental	Yes	Yes	Yes***	Yes	Yes
Anytime	No	Yes	Yes	Yes	Yes

Table 1: Synthesis table 1

  - No separate phases for learning and operation

- Incremental learning:
  - Can be online, or offline
  - Returns a series of hypotheses
  - Real time (if online) ***
  - One-by-one or chunks
  - No separate phases for learning and operation
- Anytime learning:
  - Only online
  - Returns a series of hypotheses
  - Real time
  - One-by-one or chunks
  - No Separate phases for learning and operation

Table 2: Synthesis table 2

Definition	Figure	Definition	Figure
Sequential learning is the problem of learning a classifier that receives labelled examples sequentially, one by one or chunk by chunk, and updates a hypothesis gradually and returns it when it stops learning, either when examples sequence finishes or when the system itself decides autonomously to stop learning.		**Online learning** is the problem of learning a classifier that receives labelled examples sequentially, one by one, and uses each of these examples with the previously learnt hypothesis to produce a new most-possible correct hypothesis regarding its changing environment, while respecting its constraints, for fully-working (learning and operation) in real time.	
Incremental learning is the problem of learning a classifier that receives labelled examples sequentially, one by one or chunk by chunk, and uses each of these examples with the previously learnt hypothesis to produce a new better hypothesis that encapsulates information held by the learnt examples so far.		**Anytime learning** is the ongoing process of learning, loading and execution of a classifier that receives the labelled examples collected by its host system, one-by-one or chunk by chunk, and integrates information held by them with the previously acquired knowledge to produce a better hypothesis, for tracking its environment's changes in real time.	

3.9 A note about data memorizing

In offline learning, data is memorized to be accessed by the model within the learning phase. Data then could be discarded if no re-learning from the same dataset is planned. In online, incremental and anytime learning, examples are processed and discarded, so no data memorizing is needed, especially when some of these kinds of learning handle huge datasets or work continuously. However, this is not always the case, and there is an intermediate case where a model doesn't remember all data, nor discarding them all. This can be done by two ways:

- Memorizing representative examples that describe the example distribution in the best way, or memorizing examples that belongs to decision border regions.
- Memorizing statistics of data updated after processing each example, and discarding this example.

We distinguished three cases for memorization data in classification systems. They either memorize all data, some or statistics of data, or don't not memorize data at all.

Some sequentially working systems (online, incremental, anytime) are built with taking in account storing data or part or statistics of this data, when it is thought to produce better results, or when it is computationally possible to store this data with no side effects on system's performance. This will take these concepts far from our definitions of them, but at the origin, they ought not to store anything, except for model parameters.

4 Incremental Learning

4.1 Preamble

Incremental Learning was defined in the previous section. It was explained that its main aim is to learn a classifier that receives labelled examples in sequences and uses them with the previously learnt concept to produce a new better concept.

In the previous section we tried to draw sharp edges of the incremental learning frame boundaries, aiming to find some kind of isolation between incremental learning and other concepts. In fact, all what we did is showing our point of view in drawing these boundaries; we drew the limits we saw as the most suitable based on our reading to the related works in this domain. This may not necessarily conform to the point of view of someone else who tries to do the same thing. Nevertheless, in the literature of incremental learning there are many cases that were considered incremental by its authors, and considered non-incremental by others. Some of these cases don't completely conform to the strict definition we proposed for incremental learning, but they will conform if we make our definition less strict. Various algorithms were suggested as incremental learning algorithms, trying to solve different problems. Sometimes algorithms are considered incremental when they grow or prune classifier architecture, or when they select the most informative training samples. Some other times, they refer to the modification of classifier weights, by retraining with misclassified samples.

We will try to explain the extensions of our definition boundaries for incremental learning, in light of literature.

As mentioned in the previous section, there is a difference between an incremental learning task and an incremental learning algorithm. If the examples become available over time this means that the learning task is incremental but not necessarily the learning algorithm. For the learning algorithm to become incremental it must produce a new hypothesis for each new coming sample or set of samples (chunk) considered together in the learning process. [64]

In this meaning, incrementality of the algorithm has nothing to do with the way in which each hypothesis is built. If the time between coming samples were sufficient to do batch learning and produce a hypothesis, then the algorithm remains incremental. However, this is not often the case. Time between samples may not be sufficient to redo batch learning, and then, it is necessary to use another methodology to build the hypothesis.

This leads to the discussion of many other branch cases:

- What could be done if time between samples is regular?
- Even if this time is sufficient, is there the possibility to remember old samples or representatives of these samples?
- What facilities enable online or offline incremental learning?
- How incremental algorithms handle chunks of data?
- How should an algorithm handle unseen classes?
- Does the use of ensembles of classifiers instead of a single classifier give a better classifier?
- What methods are used in generalization?
- The importance of performance issues?
- Should incremental algorithm be affected by samples presenting order?
- Is it possible to approach anytime learning in incremental systems (i.e. is it possible to have two hypothesis, one is learning and the other is working)?

The above questions related to incremental learning help to extract some basic variables on each of which the incremental learning process depends. These basic variables are: time between samples, allocated data memory, sample flow life time, goal performance, and desired algorithm characteristics.

In literature, a variety of incremental learning algorithms are based each on a specific set of values of these variables or parameters. Also, future work in incremental learning described in this section will be often derived from assigning some other values for these variables. One can notice also that these variables are not all independent in between them, for example algorithm performance has more opportunity to be enhanced if time between samples were regular and long enough to improve the learnt classifier.

In this section we are interested in the state of art of incremental learning, rather than finding a generic model representing dependency of related parameters.

4.2 Time between samples

Let's start by discussing the problem related to time between incoming samples. Time sufficiency here is ambiguous, and generally one can't control the time separating between samples. It is possible to distinguish two cases of sample flow according to time between samples.

If time between samples is regular, it is possible to decide that batch learning represents the best strategy according to predefined measurements like quality of goal hypothesis or algorithm complexity. In this case, if the system provides enough memory to store all collected samples, one can test if this time is sufficient to do batch learning on all old samples. It may occur that when sample size becomes large enough over time, the batch learning strategy won't fit with the available time between samples. A possible alternative here to batch learning is to work on representatives of data as some data may be considered as better expressing the concept like SVs in SVMs. Another solution is to work on statistics of data instead of the whole data, and to update these statistics after each sample arrives.

If time between samples is not regular, one will find himself in face of a different problem. Normally, it is not possible here to adopt batch learning to all old data even if there is enough memory to store these examples. But if a statistical distribution of time between samples could be discovered, one may study the possibility of adopting batch learning in the irregular time between samples, even if it was obligatory some times to abort some examples. This especially could be done if samples hold little information, and then, some examples can be neglected without important overhead.

A special tricky case needs to be mentioned alone, it is when an algorithm is used someway to build a classifier and put it in work phase. This classifier may be used to classify any number of coming samples. It may be decided at some point of time that this classifier needs to be enhanced to fit some suspected concept drift or to accommodate new information held by the coming samples, and thus, the algorithm is resumed to incrementally update this classifier based on the coming examples. In this case, the algorithm is considered as incremental.

Incremental learning was in the origin thought of to reply for the need of batch learning problems, especially huge data size. Thus, the natural strategy for building hypotheses in incremental learning is the successive update of a pre-existent hypothesis each time a new sample arrives without the need of some or all old examples.

Concept drift: We say that there is a concept drift when the statistical properties of the class to be predicted by the model change over time in unpredicted way. By consequent, prediction accuracy will decrease over time. Concept drift is hard to avoid especially in predicting complex phenomena that are not governed by fixed laws of nature, such as socioeconomic processes, and some biological processes [111]. It is clear that online incremental learning can help to avoid bad accuracy when concept drift occurs, better than a one-time-learning method, because it takes in account new samples besides to old samples. A better strategy to track concept drift is to forget old data and take new data in account only. Proposed solutions to reduce the consequences of concept drift contain also enhancing the model by rebuilding it considering new variables thought to affect its prediction.

Stationarity: We say that a sample flow is stationary when the joint probability distribution for

samples class doesn't change if shifted in time or space. It is always the same probability distribution that samples are derived from. If exist, variance and mean values doesn't change over time or position for a variable with a stationary flow.

A flow is non-stationary when it doesn't verify the above condition, i.e. the sample class distribution changes over time. According to this change type in time, the environment can be classified as [113] abrupt or gradual, slow or fast, random or systematic, cyclical or not. In all cases of non-stationarity we say that the environment witnesses a concept drift.

4.3 Memory issues

Another issue related to incremental learning is data memorization. There are two types of memories concerning incremental learning, a memory for storing concept descriptions (or model parameters) and a memory for storing instances or samples, it is also called instance memory. We are interested here in the instance memory only, as the other memory exists with all types of learning algorithms to store parameters describing the resulting model.

In our strict definition it was not allowed to memorize examples. Avoiding the constraint of data memorization is preferred in all incremental learning efforts. This gives the target algorithm more dependency and makes it more flexible. However, sometime, one can't avoid the need of memorization of a minimum amount of data necessary to update the hypothesis. As we mentioned above, this minimum data may be statistics of data or representatives of data that best help to update the hypothesis. Even if batch learning is allowed in principle between two successive samples, it may be not possible to do complete batch learning (batch learning on complete old data) each time a new sample arrives. Here, if the incremental learning system was designed to work theoretically forever, memory lack will be evident at some point of time, even if classifier building time is guaranteed to be controlled. Incremental learning can be done in three ways according to the dedicated memory for examples, as named by [101], a fully instance memory where all previous examples are remembered like [105], partial instance memory where part of examples are remembered like AQ11-PM and GER-PM systems [101], and no instance memory where there is no need to remember examples like in STAGGER [102] and WINNOW [103] learners systems.

To solve the problem of memory lack over time in complete patch learning, the methodology proposed in AQ11 and GER [101] fit to partial instance memory, when a memory is used to store only a part of the coming samples. The proposed method depends on keeping examples and giving them more significance if they are repeated and if there were no concept drift detected. At the inverse, old examples are forgotten when a concept drift is detected. Repeated examples that occurred many times are given greater weights than other examples and are not directly dropped from the partial instance memory when a concept drift is detected; instead, their weights are reduced. In light of above definition of concept drift, it can be noted that this learning strategy suites both cases of incrementality and concept drift.

The selection of examples to be stored in the instance memory depends on another special algorithm that estimates the importance of examples. It is based on selecting and using a set of new coming examples in addition to examples from the boundaries of previous concept descriptions. It is proved that learners can decrease considerable memory requirements, concept complexity, and learning time with loosing very little predictive accuracy. The last thing to mention here is that batch learning on a number of examples equal to the partial memory size is used in this algorithm.

The reader has certainly noticed that the last described algorithm is able to manipulate a set of coming examples by involving them with the new batch learning operations. This is involving a coming chunk of data as is in the learning process, rather than presenting it internally one-by-one to the learner. Another example will be seen when talking about SVM methods. Consequently, this complies with our definition that incremental learning can handle new examples one-by-one or chunk by chunk.

It was advised to split the previous section into memorization and stationery, because CONCEPT DRIFT leads to stationary discussion.

4.4 Samples order

The effect of samples sequence order is studies in many works; [112] defines Order Sensitivity as follows: A learner L is order sensitive if there exists a training set T on which L exhibits an order effect. Presenting the same set of examples to the learning algorithm is preferred to give always the same final hypothesis, regardless of the order in which these examples are presented. While this is not guaranteed in many algorithms, we find that some studies paid special attention to resolve this problem, examples are [106], [107] from incremental decision tree algorithms and neural networks domains, they resolve the problem of example presenting order effect on the final concept.

4.5 Unseen classes

Another common challenge for incremental learning research is how to manipulate unseen classes. When a new sample arrives, it is almost required that the classifier assigns one of the previously seen classes in the previously learnt data to the new example. Nevertheless, some authors consider the criteria of accommodating new classes coming with new data as a necessary condition for the algorithm to be incremental. An example on considering this condition [107] is mentioned in the previous section, another example is ILbyCC [108], and ARTMAP [109].

However, in our definition to incremental learning algorithms we didn't require the feature of accommodating new classes as a principal one. We find that very rigid conditions will leave numerous algorithms unclassified, especially when it is required to find a good classification to all algorithm types in the whole supervised learning literature. Knowing this, we say that an incremental algorithm that accommodates new classes is a more powerful incremental algorithm, and it is much appreciated feature.

It is now clearer to what special cases lead incremental learning if it is online or offline. All the above discussion is applied to incremental learning when it is online, while there will be more tolerance in incremental learning conditions when it is offline. Memory requirements, sample flow type, time between samples can be all controlled much easier.

After receiving one sample or set of samples (batch), incremental algorithms aim to build the classifier that can be used in classification starting from the next arriving sample. This makes it a little hard to use a classifier based on ensemble of classifiers, because this normally requires longer time. however

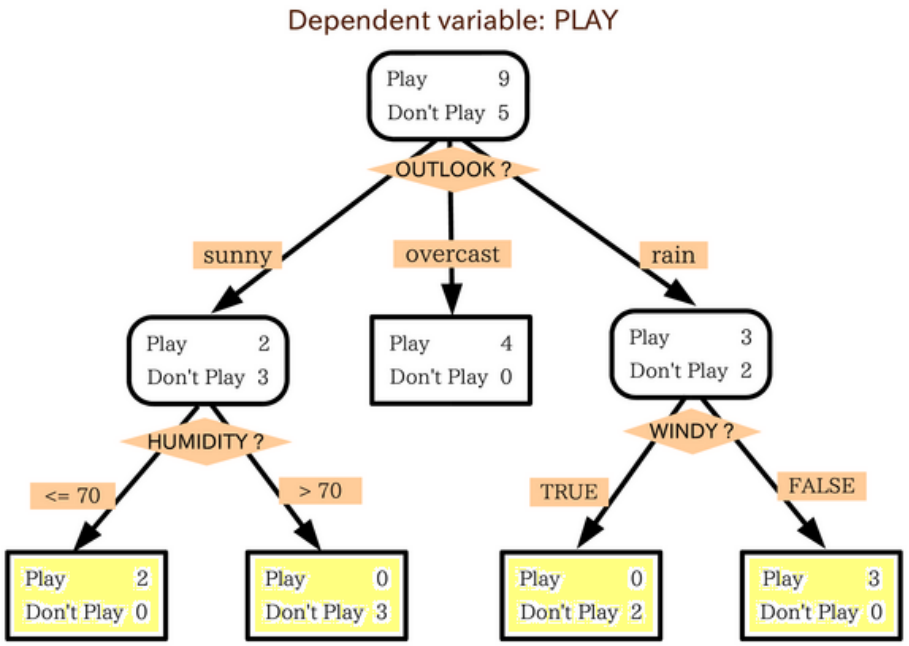


Figure 8: Decision tree example

5 Incremental Decision Trees

5.1 Introduction to decision trees

Decision trees are one of the most popular classification algorithms in current use in data mining and machine learning. They map observations about an item to inclusions about the item's target value. A decision tree assigns a class value or output (also called dependent variable) to an input pattern combined from categories or attributes (also called independent variables) by filtering the pattern down through the tests in the tree. The internal nodes of a decision tree represent tests and its leaves represent categories.

A famous example in learning decision trees is the playing tennis example. Observations about items contain outlook state which may take one of three values: sunny, overcast and rainy and humidity which is distinguished to be over or less than 70%, the third attribute is the windy weather which may be observed as true or false. 14 items are observed and the class value is known before for these observations, the learnt tree using ID3 tree algorithm is shown in figure [8].

A decision tree is learned by recursively replacing leaves by test nodes, starting at the root. The attribute to test at a node is chosen by comparing all the available attributes and choosing the best one according to some *heuristic measure*.

Tests in the tree may be uni-variate (based on one attribute) or multivariate based on several attributes. Each test results in two or more outcomes and this is known by segmentation. Attributes values may be numerical or categorical, class values may be two or more if it is categorical, or it may be an interval of continuous values.

Two major types of decision trees can be distinguished: *classification trees*, when the task is to label samples with some nominal or ordinal value, and *regression trees* when the test is to label samples by a continuous value. Regression trees are out of the scope of this study.

To resume, the learner induces models in the form of decision trees, where each node contains a test on an attribute, each branch from a node corresponds to a possible outcome of the test, and each leaf contains a class prediction.

Zighed [32] extended decision trees notion to induction graphs. Besides to nodes segmentation, he allows the fusion of two nodes in the tree, and thus, the tree loses its definition as a tree, because in trees

it is not allowed to join two branches, and therefore, to form a clique as called in graph theory.

When the decision tree is induced, rules are extracted from the tree in the form :

$$If(attributeI = value\ I_k)\ (and/or)\ (attribute\ J = value\ J_l)\ then\ (class = value\ C_h).$$

For example, in playing tennis example we obtain several rules, for example:

$$If(outlook = sunny)\ and\ (humidity > 70)\ then\ (play = no).$$

In prediction phase, the induced tree is used as a predictive model, the label y = DT(x) for an example x is obtained by passing the example down from the root to a leaf, testing the appropriate attribute at each node and following the branch corresponding to the attribute's value in the example.

The growing of Decision Tree (DT) from data is a very efficient technique for learning classifiers. The selection of an attribute used to split the dataset at each node of the DT is fundamental to properly classify objects; a good selection will improve the accuracy of the classification [3].

In the tree-induction phase the algorithm starts with the whole dataset at the root node. The dataset is partitioned according to a splitting criterion into subsets. This procedure is repeated recursively for each subset until each subset contains only members belonging to the same class or is succinctly small. In the tree-pruning phase the full grown tree is cut back to prevent over-fitting and to improve the accuracy of the tree.

During the induction phase the attribute selection measure (splitting criterion) is determined by choosing the attribute that will best separate the remaining samples of the nodes partition into individual classes. Once an attribute is associated with a node, it needs not be considered in the node's children.

The most time-consuming part of Decision Tree induction is obviously the choice of the best attribute selection measure. For each active node the subset of data fulfilling the conjunction of the splitting conditions of the node and its predecessors has to be constructed and for each remaining attribute the possible splits have to be evaluated.

Different attribute selection measures (*heuristic measure*)were proposed in the literature: for a dataset S containing n records the information entropy:

$$I(S) = -\sum_{i=1}^{K} P_i \cdot log_2 P_i$$

is defined as where P_i is the relative frequency of class i (there are K classes). For a split dividing S into m subsets: $S1(n1\ records), ..., Sm(nm\ records)$ in accordance with the attribute test X, the information entropy is:

$$I_X(S_1, ..., S_m) = -\sum_{i=1}^{m} \frac{n_i}{n} I(S_i)$$

The difference:

$$ing(X) = I(S) - I_X(S_1, ..., S_m)$$

measures the information that is gained by splitting S in accordance with the attribute test X. ID3 uses this criterion and selects the attribute test X that maximizes ing. Ing has one severe lack: a clear discrimination in favor of attribute tests with many outputs. For this reason C4.5 instead of ing, uses another attribute selection measure: information gain ratio ($ingr$):

$$ingr(X) = \frac{ing(x)}{splitinfo(X)}$$

Where:

$$splitinfo(X) = -\sum_{i=1}^{m} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

represents the potential information generated by splitting dataset S into m subsets S_i.
The Gini index ($gini$) for a dataset S is defined as

$$gini(S) = 1 - \sum_{i=1}^{K} P_i^2$$

and for a split:

$$gini(S_1, ..., S_m) = \sum_{i=1}^{m} \frac{n_i}{n} gini(S_i)$$

SLIQ and SPRINT algorithms, use gini index. Other attribute selection measure are in literature: quadratic entropy, Modified Gini index, symmetric Gini index, absolute weight of evidence, relief, chi2, relevance of an attribute, stochastic complexity, symmetric specificity gain ratio, and others.

5.2 State of art

5.2.1 One view

Plenty of efforts were done in the field of incremental decision trees. We will try here to pass over the most important ones. Incremental decision trees algorithms are often built based on a parent batch algorithm. We list here the most famous incremental tree algorithms:

CART family

- CART [6] 1984 is a non-incremental decision tree inducer for both classification and regression problems. developed in the mathematics and statistics communities.
- incremental CART (1989)[11]. Crawford modified CART to incorporate data incrementally.

ID3/C4.5 family

- ID3 (1986) [35] and C4.5 (1993) [36] are non incremental algorithms that were developed by Quinlan. ID3 was developed step by step over time, to incorporate incremental data, and enhance the specifications and performance each time, to end finally with the ripest algorithm, ITI (Incremental Tree Induction).
- ID3' (1986) [25] was suggested by Schlimmer and Fisher. It was a brute-force method to make ID3 incremental; after each new data instance is acquired, an entirely new tree is induced using ID3.
- ID4 (1986) [25] could incorporate data incrementally. However, certain concepts were unlearnable, because ID4 discards sub-trees when a new test is chosen for a node.
- ID5 (1988) [43] didn't discard sub-trees, but also did not guarantee that it would produce the same tree as ID3.
- ID5R (1989) [44] output the same tree as ID3 for a dataset regardless of the incremental training order. This was accomplished by recursively updating the tree's sub-nodes. It did not handle numeric variables, multi-class tasks, or missing values.
- ITI (1995) [45] is an efficient method for incrementally inducing decision trees. The same tree is produced for a dataset regardless of the data's presentation order, or whether the tree is induced incrementally or non incrementally (batch mode). It can accommodate numeric variables, multi-class tasks, and missing values.

STAGGER

- Schlimmer and Granger's STAGGER (1986) [25] was an early incremental learning system. It was developed to examine concepts that changed over time (concept drift).

VFDT

Very Fast Decision Trees learner reduces training time for large incremental datasets by sub-sampling the incoming data stream.

- VFDT (2000) [14], we will explain this algorithm later and use it in our experiments.
- CVFDT (2001) [23] can adapt to concept drift, by using a sliding window on incoming data. Old data outside the window is forgotten.
- VFDTc (2006) [17] extends VFDT for continuous data, concept drift, and application of Naive Bayes classifiers in the leaves.

OLIN and IFN

- OLIN (2002) [27], an online classification system of non-stationary data streams (with concept drift), using a dynamically adjusts sliding window on incoming data.
- IOLIN (2008) [9] - based on Info-Fuzzy Network (IFN) [28].
  describes a series of incremental algorithms that are based on an advanced decision-tree learning methodology called "Info-Fuzzy Network" (IFN) [28], which is capable to induce compact and accurate classification models.

IADEM and IADEMc

- IADEM (2006) [7], capable of keeping updated the estimation error of the tree that is being induced.
- IADEMc [8], adding the ability to deal with continuous data and applying Naïve Bayes at tree leaves.

AQ11

- AQ11 [37] is an Incremental learning algorithm using partial memory.

In the next three paragraphs we will compare the performance of two algorithms for categorical classification when the stream distribution is stationary (no concept drift). We will choose algorithms so that at least one should be incremental. To do so we have to choose the first algorithm from the best incremental ones and justify our choice. We will discuss the most famous and efficient algorithms: ITI (1997), VFDT (2000), and the new IADEM algorithm (2006).

5.2.2 VFDT Algorithm

Concept:
In order to find the best attribute to test a given node, it may be sufficient to consider only a small subset of the training examples that pass through that node. This number is calculated based on Hoeffding bound [21]. Based on it, it is proved that with a high probability, the attribute chosen using n examples (n as small as possible) is the same attribute that would be chosen using infinite examples. It

uses information gain or Gini index as the attribute evaluation measure.

VFDT algorithm is based on Hoeffding tree algorithm with addition of some enhancements and consideration and treatment to some special cases. The main idea behind Hoeffding tree is that in order to find the best attribute to test at a given node, it may be sufficient to consider only a small subset of the training examples that pass through that node. Thus, given a stream of examples, the first ones will be used to choose the root test; once the root attribute is chosen, the succeeding examples will be passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on recursively. Difficulty in this method lies in calculating how much examples are necessary at each node, it is solved using Hoeffding bound or additive Chernoff bound.

Consider a real-valued random variable r whose range is R (e.g., for a probability the range is one, and for an information gain the range is $log(c)$, where c is the number of classes). Suppose we have made n independent observations of this variable, and computed their mean $\overline{r}$. The Hoeffding bound states that, with probability $1-\delta$, the true mean of the variable is at least $\overline{r}-\epsilon$, where:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

The Hoeffding bound has the very attractive property that it is independent of the probability distribution generating the observations. The price of this generality is that the bound is more conservative than distribution-dependent ones.

Let $G(Xi)$ be the heuristic measure used to choose test attributes (e.g., the measure could be information gain as in C4.5, or the Gini index as in CART). The goal is to ensure that, with high probability, the attribute chosen using n examples (where n is as small as possible) is the same that would be chosen using infinite examples.

Assume G is to be maximized, and let Xa be the attribute with highest observed $\overline{G}$ after seeing n examples, and Xb be the second-best attribute. Let $\overline{\Delta G} = \overline{G}(Xa) - \overline{G}(Xb) \geq 0$ be the difference between their observed heuristic values. Then, given a desired δ, the Hoeffding bound guarantees that Xa is the correct choice with probability $1-\delta$ if n examples have been seen at this node and $\Delta\overline{G} > \epsilon^2$. In other words, if the observed $\Delta\overline{G} > \epsilon$ then the Hoeffding bound guarantees that the true $\Delta G \geq \Delta\overline{G} - \epsilon > 0$ with probability $1-\delta$, and therefore that Xa is indeed the best attribute with probability $1-\delta$. This is valid as long as the $\overline{G}$ value for a node can be viewed as an average of G values for the examples at that node, as is the case for the measures typically used. Thus a node needs to accumulate examples from the stream until ϵ becomes smaller than $\Delta\overline{G}$. (Notice that ϵ is a monotonically decreasing function of n.) At this point the node can be split using the current best attribute, and succeeding examples will be passed to the new leaves.

A key property of the Hoeffding tree algorithm is that it is possible to guarantee under realistic assumptions that the trees it produces are asymptotically arbitrarily close to the ones produced by a batch learner (i.e., a learner that uses all the examples to choose a test at each node).

Based on Hoeffding tree, VFDT is built with a number of refinements, When two attributes are very similar in G's value, a large number of examples should be collected to know which one is the best, VFDT chooses one of them to avoid waiting for large number of examples, when the difference in G's function is smaller than a user-defined value.

To avoid calculating G's value each time a new example arrives, a user-defined minimum number of examples is collected before doing the new test in the leaf. It is true that it is possible to collect more examples than needed before doing the test, but this reduces computation time of G value.

If VFDT maximum available memory is ever reached, it deactivates the least promising leaves in order to make room for new ones. The least promising leaves are detected by choosing leaves with minimum values of $p \cdot e$, where p is the probability that an arbitrary example will fall in the specified leaf, and e is the observed error rate at that leaf. Deactivated leaves can be activated later if the become more

promising than other active leaves. Other refinements are added to VFDT concerning early dropping of non promising attributes, and initialization by RAM-based learners to give the a algorithm a head-start. A rescan to previously seen examples can be done if example arriving rate is slow enough or if the dataset is small in size.

Test set and test plan:
A system like VFDT is only useful if it is able to learn more accurate trees than a conventional system, given similar computational resources. In particular, it should be able to use to advantage the examples that are beyond a conventional system's ability to process. The experimental study is comparing VFDT with C4.5 release 8 on a series of synthetic datasets. Two systems with the same amount of RAM, one for C4.5 and the other for VFDT. Concepts are tested (all with 100 attributes and two classes), they are generated by randomly generating decision trees.

Results:

- Better accuracy, and the decision trees don't fail at some point like C4.5 because of RAM lack.
- Less tree nodes
- Better accuracy in the presence of noise
- Better accuracy with concept size (leaves)

Characteristics:

- Store statistics of data
- Can scale to any allocated memory and handle the lack of memory by discarding the less promising nodes dedicated memory.
- Anytime(in the sense that a ready-to-use model is available at any time after the first few examples are seen, and its quality increases smoothly with time.)
- Needs to be parametrized
- Well documented
- Source code in C available
- Don't stop.

5.2.3 ITI algorithm

Concept:
Adopt binary tests on each node, and for non binary attributes it handle them as many binary tests. Use the principle of leaf nodes that can be converted to decision nodes and check the possibility of converting a leaf node to a decision one when incorporating a new example.

Test set and test plan:
The algorithms to be compared are three variants of ITI, three variants of DMTI (same article), and two variants of Quinlan's C4.5. All eight algorithms are applied to forty-six classification tasks in a perfectly balanced and perfectly crossed manner. Duncan's Multiple Range Test significance test is used to decide the best algorithm. The forty-six tasks differ in many ways, including number and type of attributes,

number of classes, noise, missing values, and default accuracy. The goal was to pick as many tasks as was practical. The chosen test sets was avoided to be large.

Results:
Defeated by VFDT after addition of a framework, the results of another article [206]. Domingos states in the latest article that ITI was very slow after 100.000 examples (thus some tests are not made beyond 100.000 examples). He states also that ITI results in higher error rate, and higher sensibility to the target concept noise than VFDT.

Characteristics:

- Online
- Needs to store all examples in RAM
- It handles numeric and categorical attributes.
- Handles missing values
- To the extent possible, independent from number of examples.
- Example order doesn't affect the resulting tree
- Handle inconsistent training examples
- Avoid fitting noise by avoiding over-fitting
- Has many modes: incremental, error correction, lazy, batch modes.
- Well documented
- Source code in C
- Needs to be parametrized
- Don't stop.

5.2.4 IADEM Algorithm

We needed help concerning the documentation of IADEM algorithm, we wrote to the authors and they provided us with the complementary information and software. There are problems that do not suffer any change, but generate huge amounts of information (even data streams). IADEM has been designed to deal with this kind of datasets those which concepts do not change over time.

Concept:
Using Chernoff and Hoeffding bounds to decide when to split a node. The algorithm stops when it reaches an error smaller than a prefixed superior limit of error. Expansions conditions are related to prefixed superior and inferior limits of error. Algorithm stop also when detected noise becomes 2 times the superior limit of error.

Test set and test plan:
LED dataset from UCI, and synthetic data randomly generated using decision trees. Compared with C4.5 and ITI and VFDT using their default configuration, i.e. no parameter tuning is tried. Tested until

10^6 examples. Used sampling with replacement from a dataset collected offline.

Results:
Less leaves, better accuracy, it takes much more time than other algorithms (e.g. sometimes 25 times than VFDT).

Characteristics

- Keep updated the estimated errors (could be displayed to user)
- Accurate and relatively small trees
- Deal with datasets of any size.
- Have to stop at some stage and give a result verifies its prefixed sup and inf error limits (even if the stream is not ended).
- Needs to be parametrized.

Discussion

VFDT will continue learning until every example in the dataset is evaluated. IADEM can stop processing examples in the stream when it has statistical evidence that it has learned the model with a maximum error. Thus, it may need less time (and examples) than VFDT. However, authors of IADEM state that their algorithm continues working if the sup(error) is fixed to 0, while expansions will always occur when the condition to expand are satisfied. If every possible expansion is made and the tree is completely expanded, the algorithm will stop as it can not do more expansions. IADEM authors didn't verify if their algorithm can track the same streams frequency compared with IADEM. Neither tested their algorithm on a real online stream.

ITI and VFDT source code exist in ANSI C language, and they can be tested on chosen datasets or real data streams.

When compared to ITI, VFDT is proved to be more efficient in term of classifier building time for the last few years. It is fast and gives a very good accuracy (if not the best), less sensibility to noise and less RAM requirements. It is a stream general purpose algorithm in the sense that it doesn't restrict any condition on the nature or the length of example stream. VFDT code is available for free and it is tested in two articles of its authors, besides, a well documented framework in ANSI C is presented so that addition and modification on the source code are easy.

We base on the above comparison between the most efficient incremental algorithms in literature, and we choose VFDT algorithm as a reference algorithm to evaluate our method that will propose in the following section.

6 Experimental Study

6.1 Objectives

The first objective of these experiments is to reproduce the experiments done in VFDT article [14] and to test a naïve idea for a baseline incremental system, based on classification trees with random sampling technique. In order to measure validity of this idea, we will compare with a reference algorithm which will be one of the most powerful algorithms in incremental decision trees.

Training time is an essential factor in the learning process, especially when dataset is too large, it is affected by how the algorithm accesses data. Some incremental decision tree algorithms rescan the database several times or need to store the whole seen dataset in memory.

Our second objective, is to be able to deal with large sample databases with only one pass on the dataset. Very few existing algorithms require only one pass over data. We will try to build an incremental method based on the batch learning decision tree C4.5, without retraining on the totality of seen samples.

Training on the whole seen dataset requires heavy memory and processing time consuming; besides, it is well known to get worse when the number of seen examples increases over time. We will compare the performance of our incremental method to the performance of VFDT incremental (anytime) algorithm. We will try also to find any possible improvement on the performance of our method by studying the effect of several factors on this method, all in comparing with VFDT performance, which is considered our reference to improve our incremental C4.5 method.

As our method requires storing some data in fixed size memories called reservoirs, and then it is considered as a partial memory method, we will benefit from our experiments to re-verify the truth of the following proposition: "Partial-memory learners tend to reduce predictive accuracy but notably decrease memory requirements and learning times." [29]

6.2 Data

6.2.1 Different ways to evaluate an algorithm

Various data types are used for algorithms learning evaluation. Datasets used in testing algorithms could be standard, real or synthetic:

- *Standard datasets* are previously prepared datasets with known statistical characteristics to test algorithms efficiency. The best common example is UCI repository data. It is commonly used as a standard platform for performance evaluation in machine learning field.
- The final purpose of any learning Algorithm is to apply it on *Real data*. However, the algorithm should be well studied and tested before applying it on real data. Real data are not always suitable to study algorithms characteristics because it is difficult to prepare them and to control their characteristics. sometimes, confidentiality issues may require not to use real data in the algorithm testing stage unless they are anonymized.
- Synthetic data replies to these requirements, they are artificially fabricated to test algorithms. the major benefit of using synthetic data is that it can be controlled and forced to hold any desired characteristics to serve exploring data effect on the model, and thus, synthetic data give a high degree of freedom when testing models.

A note about data anonymization:

There exist some possibilities to protect data before publishing it (anonymizing data), such as the masking method which implies adding noise to data or using some aggregate methods to deform data. An alternative is to use synthetic data by generating a set of random values adjusted in order to fulfill similar statistical characteristics of the original data. A hybrid dataset can be generated also by mixing masked data with synthetic data [30]. For example when a company organizes a challenge that helps to reveal the best algorithms related to its own data characteristics; in this case it is crucial for confidentiality reasons to hide private data that may be personal information about its customers but in the same time, to maintain the same data characteristics in the proposed dataset for the challenge.

In our experimental study we will use two types of data to evaluate our proposed method: synthetic and real data.

6.2.2 Synthetic data

We generated data stream with $350 * 10^6$ samples length. Each example has 100 binary attributes and 1 binary class variable. The generated data is balanced concerning the class values.

We benefited from Domingos framework to generate this data. Data generation procedure starts by creating a synthetic binary tree and then using it to label data which could be used to evaluate learning algorithms. The synthetic tree is generated starting from a single node as follows: A leaf is selected and is either split on one of the active attributes (each discrete attribute is used at most once on a path from the root to a leaf) or it is pruned. The probability of pruning is set by the *prunePercent* parameter but is 0 if the leaf is not below *firstPruneLevel* and is 1 if the leaf is below *maxPruneLevel*. If the attribute selected for a split is continuous a threshold is generated uniformly in the range [0,1] except that it is ensured that the chosen threshold is not redundant with an earlier split.

Once the tree's structure is created (all leaves have been pruned) a class label is randomly assigned to each leaf and then redundant subtrees (where every leaf has the same classification) are pruned. Data (training, testing, and pruning) is generated by creating an example and setting its attributes with uniform probability. The tree is used to label the data, and then the class and discrete labels are re-sampled with uniform probability (without replacement) as specified by the noise parameter and continuous attributes are changed by sampling from a Gaussian with mean of their current value and standard deviation of noise. Using the same *conceptSeed* (along with the same other parameter) results in the same concept being created. The same seed results in data being generated the same (so experiments are easily repeatable). Also, some additional statistics about the generated concept are outputs.

6.2.3 Orange data

Our data source for real data is Orange database constituted of customers in France. The purpose of the selected database is to evaluate the channel appetency to purchase a service using a specific channel (e-shop, phone, shop...).

Data selection: Attributes variables could be categorized as:
Demographic : name, age, income, addresses..
Behavioural and attitudinal: destination of a phone call, time of phone call, uploaded megabytes...
Bill: customer bill.
Services: what service does the client use, TV, Internet or phone.
Interests: sport, car, travel...

The class variable is the most commonly used channel of a customer.

Data Pre-treatment: Real data often require preparation before learning, a pre-treatment stage of Orange data was performed using Orange specific KHIOTREE software, and this included:

First: attributes discretization: We committed an automatic operation of grouping and discretization on attributes based on MODL approach. We forced the number of values to 2. This implies grouping values of each nominal attribute into 2 groups of and encoding them by 0 for the first group of values and 1 for the second. Choosing values of each group and encoding are automatically performed by KHIOTREE. For continuous attributes, we carried out a binary discretization, the split point for each attribute was chosen automatically by KHIOTREE.

Second: attributes selection: Orange data contains thousands of attributes and redundant attributes. The selection of 100 attributes is performed by applying Selective Naïve Bayes approach using KHIOTREE software, resulting in selection of the 100 most discriminating attributes.

Third: Treatment of damaged values: suppression of 3 examples of damaged class values (neither 0 nor 1), resulted while data preparation.

At the end of this process we obtained a file of 1,337,918 examples as a training set and a file of 41,321 examples as a test set. Each example is compound of 100 binary variables as attributes and one binary variable as the class.

6.2.4 Software

We used Domingos framework called VFML (Very Fast Machine Learning toolkit) [22], written in ANSI C. It was developed by the team of VFDT and CVFDT to test their algorithms. We modified in VFML code in some places to gain the maximum possible number of examples in one memory unit, the result was multiplying the memory unit capacity of storing examples by 4.

We wrote the code of random sampling [46] and integrated it within VFML after preparing it in the suitable way. The code of random sampling takes a sample file or a sample stream and returns files representing the image of random sampling reservoir after applying its algorithm. An image of the reservoir is taken after seeing each $10000 * (ScheduleParameter) * (ScheduleParameter)$ samples.

We used Weak's J48 algorithm to generate pruned C4.5 decision tree on resulted reservoirs. The latter are treated as the new training sets for C4.5 instead of the totality of examples seen so far. However, it was necessary to face and solve an operating system problem concerning Weka allocated memory, as the maximum memory allocated to java applications are limited to 64MB by default, this was not sufficient to deal with the huge amount of data in our generated reservoirs in addition to necessary Java runtime library. We extended this allowed memory to 1.5GB.

In our experiments, and like Domingos does in his VFDT article, we adopt memory usage as the base criterion of fair comparison between the two methods, while considering classification accuracy as the main evaluation criterion.

Generally, we can consider the memory used by a machine learning algorithm as the sum of memories used to store examples and model parameters. It can be noted as follows:

$$Algorithm\ memory = Memory\ to\ store\ examples + model\ parameters\ memory$$

On one hand, VFDT requires no example storage as it reads examples, updates tree statistics and then discard examples, so we can consider the first term equal to 0. On the other hand, C4.5 is a batch learning algorithm; it loads examples from disk into memory before the start of execution. Thus, we can consider only the memory used to load examples. This consideration is rightful as the model needs not to stay in memory (like in VFDT) until other examples come, examples are presented as one batch of data to the algorithm since its start. The model is output when model calculation is finished. An extra needed memory doesn't appear in the above equation, it is the memory taken by the application, this memory is implementation dependent memory, it is related to the used programming language and code optimization techniques. For example, used RAM for C4.5 depends on its implementation, an implementation of C4.5 in Java (Weka) requires much more memory than an implementation in ANSI C which can be optimized more efficiently. this application memory is not considered in our comparison for both C4.5 and VFDT.

By consequent, the fair comparison between C4.5 and VFDT is based on the following equality:

$$Memory\ used\ to\ store\ VFDT\ parameters = memory\ used\ to\ load\ C4.5\ examples$$

6.3 Protocol, results and discussion

In our experimental study we will compare VFDT to our new incremental C4.5 method with random sampling. It's close to what Domingos did in his VFDT article, and so, it is important to explain how Domingos tested VFDT versus C4.5 before starting to explain our experiments.

6.3.1 Introduction

What did Domingos do?

1. Making sure that both algorithms take the same amount of RAM (40MB).

2. Creating learning data stream:

   By creating 14 concepts with 100 binary attribute and 2 classes each. Then generating samples and using the 14 concepts to label data. Noise was added to data at the end.

   As described before, Domingos used decision trees to build concepts. At each level in each tree after the first three, a fraction f of the nodes was replaced by leaves; the rest became split on a random attribute that had not been used yet on a path from the root to the node being considered. When the decision tree reached a depth of 18, all the remaining growing nodes were replaced with leaves. Each leaf was randomly assigned a class.

   A stream of training examples was then generated by sampling uniformly from the instance space, and assigning classes according to the target tree. Various levels of class and attribute noise were added to the training examples, from 0 to 30% (A noise level of n% means that each class/attribute value has a probability of n% of being reassigned at random, with equal probability for all values, including the original one.) In each run, 50k separate examples were used for testing. C4.5 was run with all default settings. He ran his experiments on two Pentium 6/200 MHz, one Pentium II/400 MHz, and one Pentium III/500 MHz machine, all running Linux.

   We interpret Domingos's experiments plan in an algorithmic form as follows:

   Variables:
   $ExampeNo$: is the desired number of examples to test, it takes values in $[e+3, e+4, e+5, 2*e+5, ..]$.
   $ConceptNo$: is the number of concepts to test, it takes values in $[1, .., 14]$.
   $NoiseLevel$: is the ratio of added noise; it takes values in $[0, 5, 10, 15, 20, 25, 30]$.
   Algorithm:

   ```
   foreach ExampeNo do
       ;        // start or continue generating a sample stream X of length ExampleNo
       generate 50KS for test;
       foreach ConceptNo do
           assign label to samples;
           foreach NoiseLevel do
               add noise n% to stream;
               Train VFDT; Train C4.5;
               Store results;
           end
       end
   end
   Compare;
   Draw results;
   ```

3. Drawing results:

   - Accuracy of the learners averaged over all run. Accuracy = f1 (ExampleNo)
   - Average number of nodes in the trees induced by each of the learners NodesNo = f2 (ExampleNo)
   - How the algorithms respond to noise. Accuracy = f3 (n)

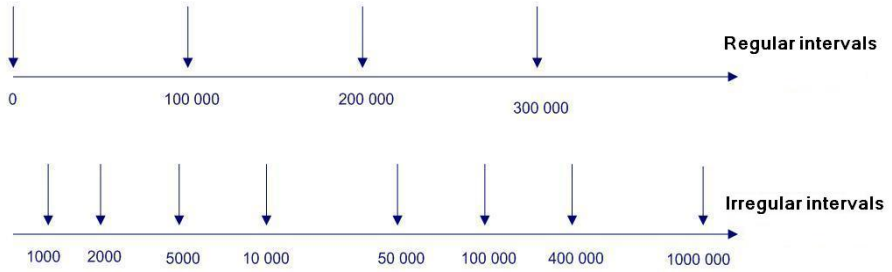


Figure 9: Test points selection in a stream

- How the algorithms compare on six concepts of varying size. Accuracy = f4 (LeavesNo)

Domingos also tested his VFDT on web data of the whole university campus, in order discover the most visited hosts, and thus, enhance web caching strategies.

What will we do?

In our experiments, we will try to follow the experiment logic of Domingos in his VFDT experimental study, but in some differences sometimes, besides, we will try to cover some special cases of data to discover whether VFDT is always better than C4.5 or not. Further, we will try also to do this comparison on some real data concerning Orange clients data. We will keep the same method for generating and labelling data as in VFML, i.e. maintain synthetic data specifications, 100 binary attributes, and 2 categorical classes. Data flow is stationary with normal distribution.

We will also mimic Domingos comparison between VFDT and C4.5 but with learning C4.5 over a sub dataset of the seen stream by using random sampling technique, and study possible enhancements in prediction accuracy when using Random sampling technique with C4.5.

C4.5 is a batch learning algorithm, thus, to compare its performance to that of an incremental one, the traditional strategy is to choose distinct points expressing the number of accumulated examples so far, and learn the algorithm on all previous examples. This means dividing the stream in regular or irregular sections according to the selected points. Figure [9] express a stream's possible test point selection.

However, a fair comparison between C4.5 and VFDT suppose that the memory requirements are the same. So, we have to fix a memory size and try to fit C4.5 training set into this memory.

If the stream is large in size or continuous, examples number will augment over time, and so will do the required memory to store them. Then, it will occur at some stage of C4.5 training process that the specified memory to store data will be exceeded.

Random sampling technique importance comes at this point, it randomly chooses a number of seen examples and accumulates then in a reservoir, this number is almost the same regardless of the number of seen examples. In this way, we will be able to overcome the obstacle of memory insufficiency while maintaining data distribution. We expect this gain to have a price which will be concreted in a negative aspect; it will be a probable loss of information that ties examples classes to their attributes values, and thus, a loss in the resulting tree accuracy. The result is an accuracy that is sure less than the accuracy of C4.5 learnt on all seen data so far.

We will use an algorithm of random sampling with a reservoir, called R algorithm. The reservoir is a memory space of size n dedicated to store the true random sample of the seen data. Every reservoir

algorithm begins by putting the first n sample in the reservoir. The rest of the samples are processed sequentially and they can't be selected for the reservoir until they are processed. A reservoir algorithm is an algorithm that maintain the invariant that after each sample is processed, a true random sample of size n can be extracted from the reservoir.

Algorithm R works as follows: When the $(t+1)st$ sample in the file or stream is being processed, for $t >= n$, the n candidates form a random sample of the first t samples. The $(t+1)st$ record has a $n/(t+1)$ chance of being in a random sample of size n of the first $t+1$ records, and so it is made a candidate with probability $n/(t+1)$. The candidate it replaces is chosen randomly from the n candidates. It is easy to see that the resulting set of n candidates forms a random sample of the first $t+1$ records.

Here is the pseudo code of the R algorithm:

Variables:
n : reservoir memory length.
Algorithm:
for $j = 0$ *to* $n-1$ **do**
  ; // Make the first n samples candidates for the sample
  $READ-NEXT-RECORD(C[j])$;
end
$t = n$; // t is the number of records processed so far
while *not eof* **do**
  ; // Process the rest of the records
  $t := t+1$;
  $A := TRUNC(t * RANDOM())$; // A is random in the range $0 <= A <= t$
  if $A < n$ **then**
    $READ-NEXT-RECORD(C[A])$; // Make the next record a candidate,
    ; // replacing one at random
  else
    $SKIP-RECORDS(l)$; // Skip over the next record
  end
end

Thus, in our experiments, instead of learning C4.5 on all previous seen examples that may result on huge training sets, we will train C4.5 on a true random sample of the data seen so far. We will use random sampling technique to make the training set best expressing the sample distribution with respect to the dedicated memory size for C4.5. We will also train VFDT on the same data and compare results as shown in figure [10].

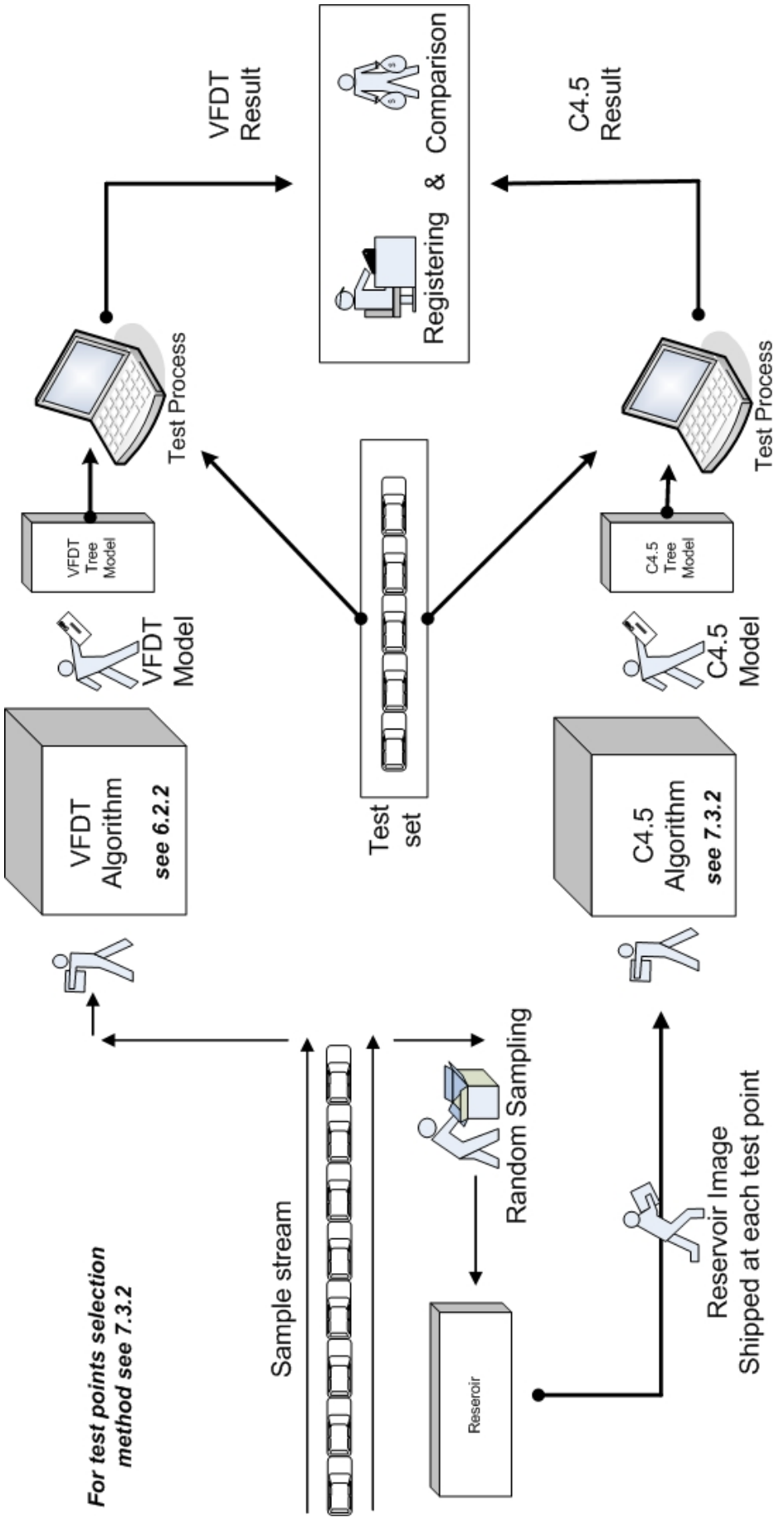


Figure 10: Experiments scheme

We will do the following experiments using our generated synthetic data:

- Testing parametrization effect to find a possible enhancement in C4.5 with random sampling performance and VFDT.
- Testing the Random sampling reservoir size effect on the performance of C4.5 with random sampling and comparing it to VFDT performance.
- Testing concept size effect on the performance of C4.5 with random sampling and VFDT.
- Discovering the effect of noise on C4.5 with random sampling performance and VFDT.
- Besides, we will test C4.5 with random sampling and VFDT on Orange specific real data.

In the following paragraphs, we will explain the details of our experiments, give the results and discuss them.

6.3.2 C4.5 parametrization effects on its performance and comparing it with VFDT

Here are the steps for this experience:

1. Use some samples seed (SamplesSeed 1=1234) to generate a stream of $10 * 10^6$ examples on the fly. Also generate a test file of 50000 samples.
2. Generate a concept using some seed (ConceptSeed 1=4).
3. Label generated examples using the generated concept while adding 10% noise to examples attributes and classes.
4. Generate a series of random sampling reservoirs (25800 sample, 5MB) using labelled data.
5. Find the best parameter set of C4.5 and present generated data to tuned C4.5 (C=3E-04, M=2).
6. Present reservoirs series to C4.5, test on the test file, and measure learning accuracy.
7. Generate the same sample stream by using the same seed (SamplesSeed 1=1234).
8. Generate another concept having the same size of the former using a different concept seed (ConceptSeed 2=111).
9. Label generated data using the generated concept.
10. Generate a series of 25800 samples random sampling reservoirs using labelled data.
11. Present reservoirs series to previously tuned C4.5 with the same parameters (C=3E-04, M=2).
12. Learn VFDT using the same labelled data and measure accuracy of incrementally learnt classifier at the same points of the stream as reservoir images are taken, while allowing 5MB of memory usage.
13. Compare.

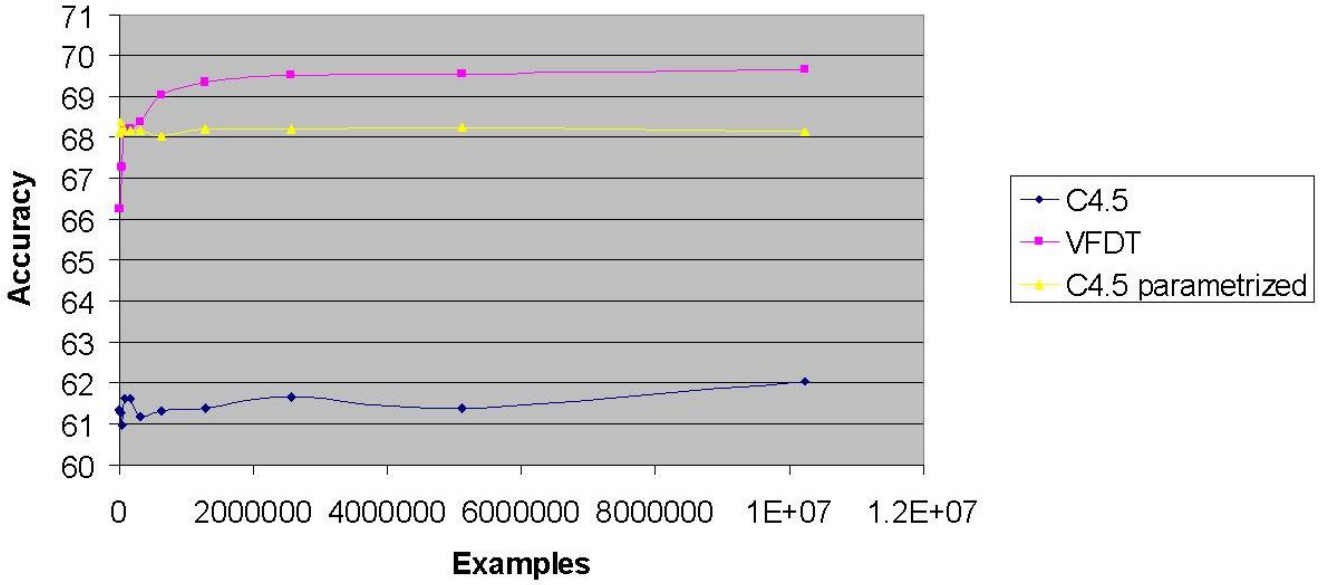


Figure 11: VFDT vs parametrized and non parametrized C4.5 - 1st concept (reservoir memory = 5MB, concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $10 * 10^6$ example, tested on 50000 examples)

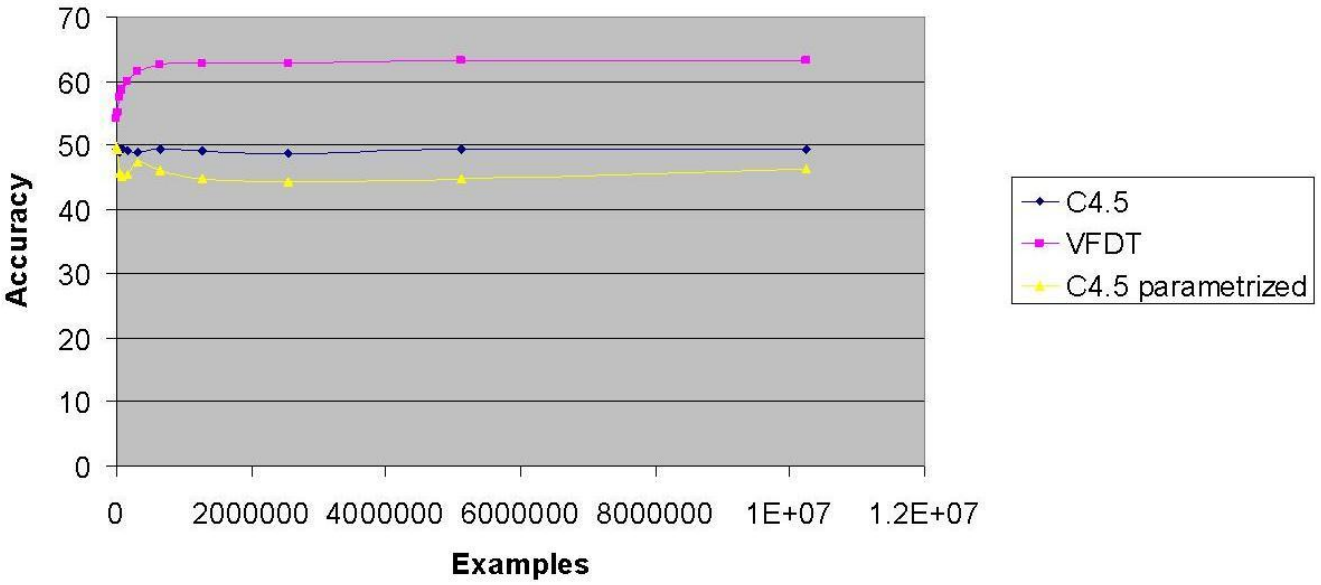


Figure 12: VFDT vs parametrized and non parametrized C4.5 - 2nd concept (reservoir memory = 5MB, concept seed=111, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $10 * 10^6$ examples, tested on 50000 examples)

The results of this experience are shown in figures [11] and [12].

In this first experiment, we notice that in both concepts used for labelling data, VFDT gave better results than C4.5, and the quality of results in term of accuracy increases always in the case of VFDT while C4.5 accuracy ripples while increasing sometimes. We notice also that both VFDT and C4.5 accuracies decreased when changing the concept used to label generated samples.

What is more important to notice is that C4.5 parametrization succeeded to highly improve its accuracy on the first concept. But when using the same parameters on another concept we got worse results than using C4.5 default parameters. We conclude that no parametrization of C4.5 should be used later as it is not possible to predict the underlying concept when incrementally learning real data and then choosing best parameters for C4.5. Thus, from now on, default parameters for C4.5 will be used.

As seen in the figures [11], [12] from the behaviour of VFDT, the accuracy enhances after seeing more examples, this is concurrent with the development of the tree size. As VFDT sees more examples it enhances the growing tree and consequently the resulting prediction accuracy. C4.5 behaviour is different, and it will be explained in the next experiment.

Note:

Default parameters values for C4.5 (J48) pre-set in Weka are: the confidence factor used for pruning is default to 0.25, and the minimum number of examples per leaf is default to 2.

6.3.3 Reservoir size effect on C4.5 and comparing with VFDT

Here are the steps for this experience:

1. Generate a stream of $350 * 10^6$ examples on the fly using some examples seed (without storing them in RAM - takes 11 hours for 40MB reservoir). Also, generate a test file of 50000 samples.
2. Generate a concept using some concept seed.
3. Label generated examples using the generated concept while adding 10% noise to examples attributes and classes.
4. Generate 4 series of random sampling reservoirs using labelled data, reservoirs sizes are: (1000 sample, 200KB), (5200 sample, 1MB), (25800 sample, 5MB), (207000 sample, 40MB).
5. Present the 4 series of reservoirs to C4.5 with default parameters and use the test file to measure accuracy.
6. Learn VFDT using the same labelled data and test at the same points of the stream as reservoir are taken using the test file, allowing 4 different memory usages (200KB, 1MB, 5MB, 40MB) (takes 6 hours for the reservoir 40MB).
7. Compare.

The results of this experience are shown in figures [13], [14], [15] and [16]. Those figures show that whatever was the size of reservoir, VFDT gives better results than C4.5. However, accuracy of both algorithms increases when increasing the allowed memory use and this increase is in the same amount for both algorithms, so that the difference in the resulted accuracy stays the same (around 8%).

We pay a special attention to the result of the experiment when allowed memory is 200KB. We notice that the tree gives a fixed value as accuracy, while when checking the growing tree of VFDT, we see that it varies from 4 nodes in size after seeing 10000 examples, to 95 after seeing 350,000,000 examples. We justify this by saying that VFDT was restricted to use a small memory of 200KB which was not sufficient

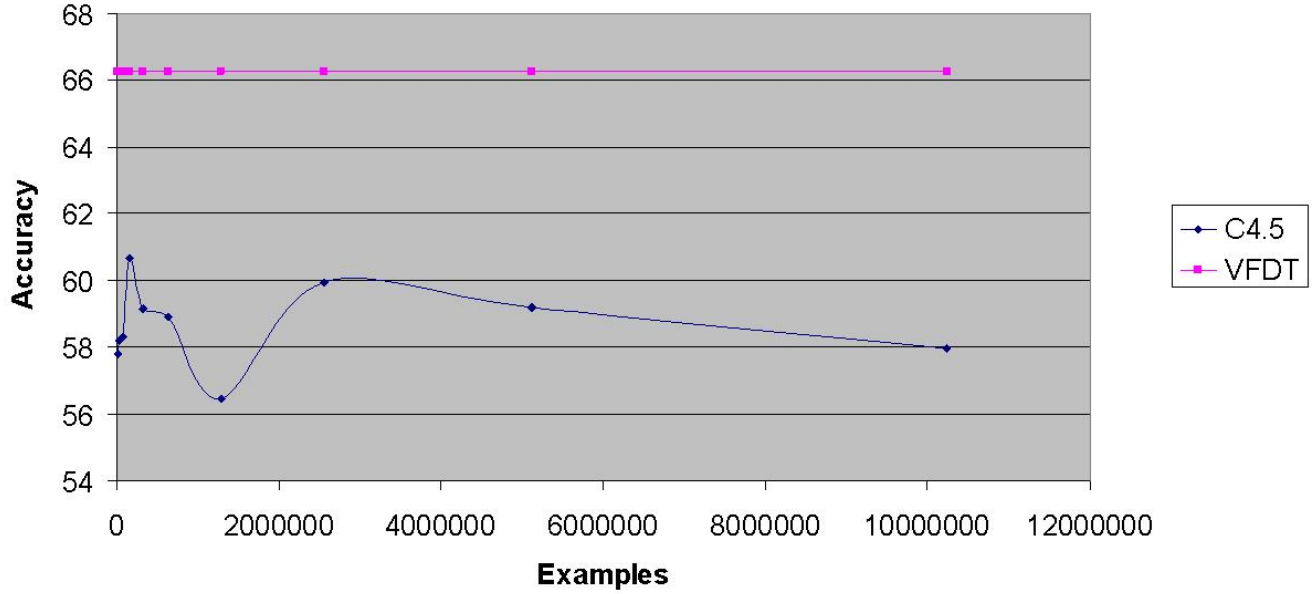


Figure 13: VFDT vs C4.5 - 200K Reservoir memory size (concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $350 * 10^6$ examples, tested on 50000 examples)

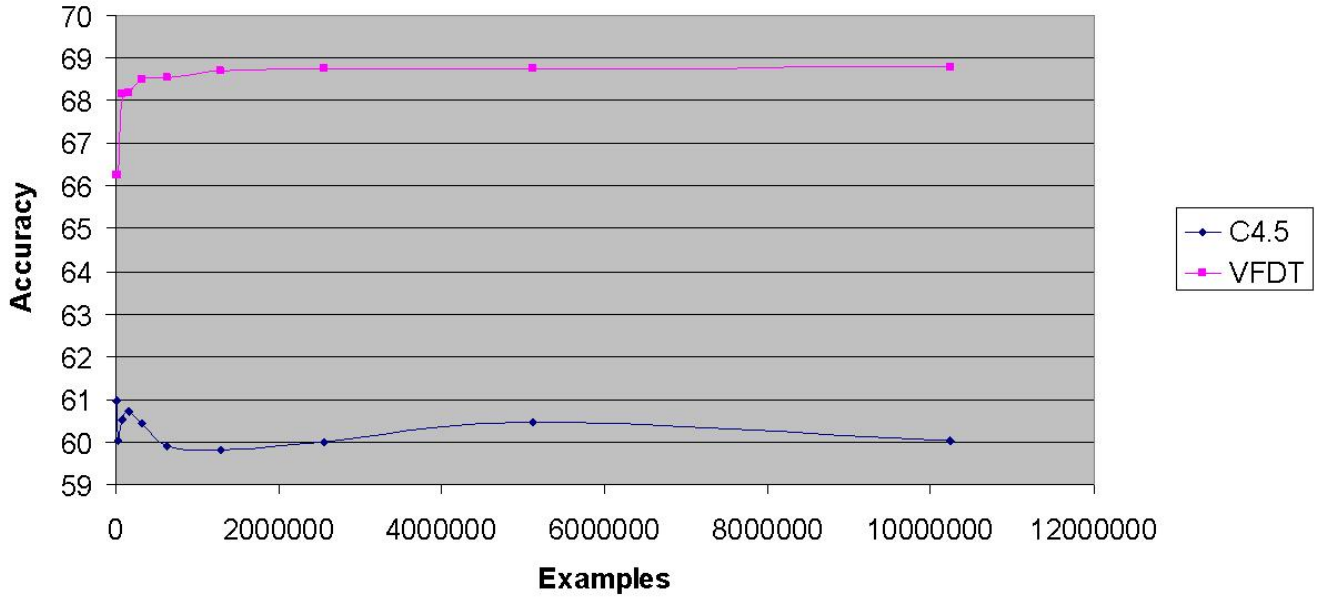


Figure 14: VFDT vs C4.5 - 1M Reservoir memory size (concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $350*10^6$ examples, tested on 50000 examples)

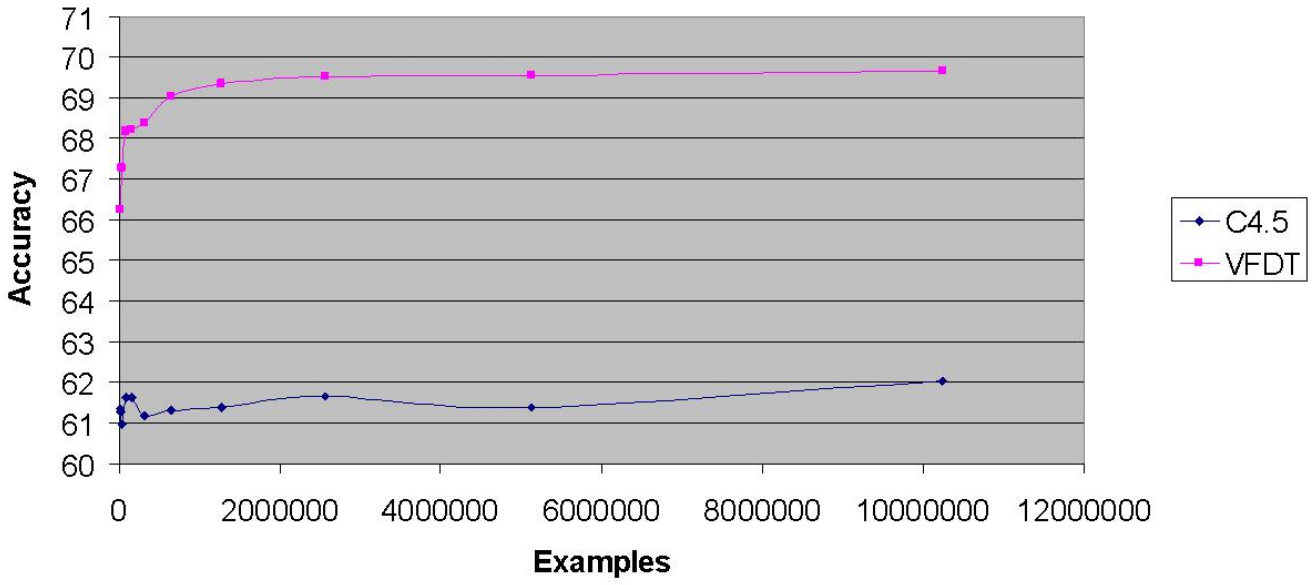


Figure 15: VFDT vs C4.5 - 5M Reservoir memory size (concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $350*10^6$ examples, tested on 50000 examples)

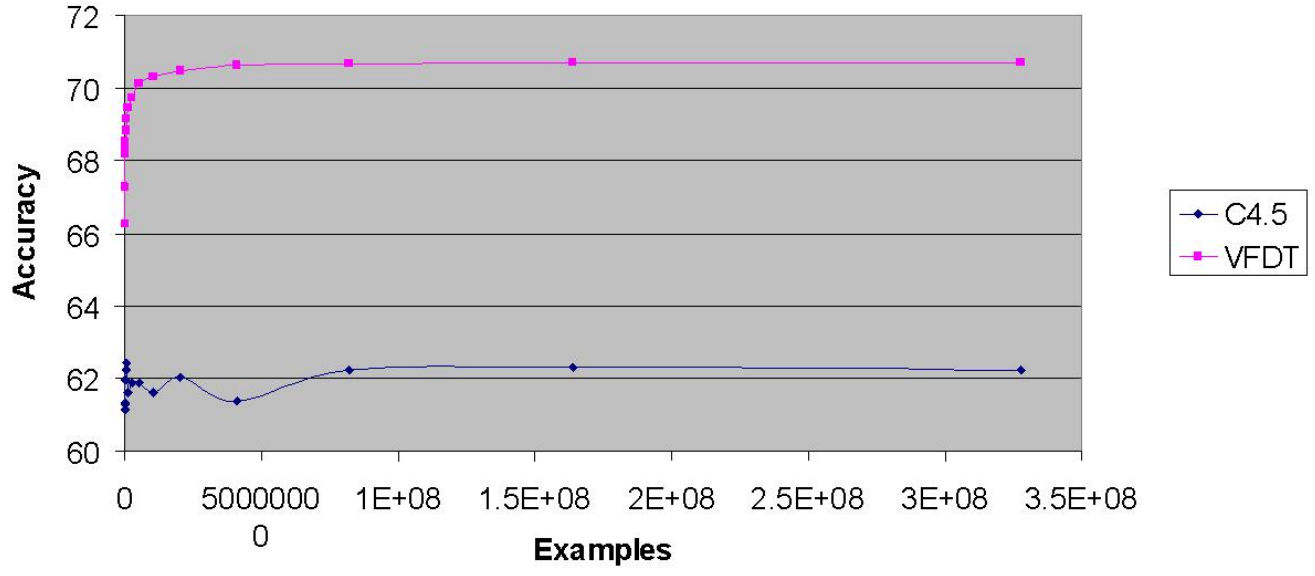


Figure 16: VFDT vs C4.5 - 40M Reservoir memory size (concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $350 * 10^6$ examples, tested on 50000 examples)

to store parameters for a large tree. And thus, the depth of the tree was not sufficient to cover enough discriminating attributes.

We return back to the explanation of C4.5 behaviour, which it is essentially related to random sampling method. Before the filling of the reservoir, C4.5 accuracy increases by example number, because it sees more information to build a better classifier. After this point, and generally speaking, when the number of seen examples increases, we notice ripples in its behaviours. If there is no special pattern in data, C4.5 accuracy ripples around some slightly bent axe. The behaviour is related to many factors, mainly to the complexity of the concept to learn and to reservoir size, because if the learnt concept is complicated, then the size of the reservoir may not be sufficient to store enough examples representing the concept. If the concept is simple, a good accuracy will be maintained over time. The bent of the axe around which the Accuracy curve ripples gives a good indication about the reservoir size fit to the concept. If this bent is positive, then this means that the underlying concept of data is complex and thus examples in the reservoir become more representative to the real concept. A zero bent of the ripple axe means that the reservoir size is enough to stock enough examples to represent the underlying concept. However, when the bent is zero, if we increase the size of the reservoir, this ripple axe should shift up only. No negative bent result.

We conclude from this experience that increasing allowed memory enhances the performance of both VFDT and our incremental C4.5 in the same degree. It doesn't favour C4.5 over VFDT.

Note:

The difference in generation time of $350 * 10^6$ samples is due to time needed to store examples in memory every time an image of the reservoir is generated. Besides, there exist no latency between example generation and presenting them to VFDT, while the example generator have to wait until each example is processed and its probability of being in the reservoir is calculated. At the inverse, VFDT does once statistical test only each 300 read examples.

6.3.4 Concept size effect on C4.5 and comparing with VFDT

Here are the steps for this experience:

1. Generate streams of $10 * 10^6$ examples on the fly using some examples seed.
2. Generate 2 concepts of different sizes and the same seed (ConceptSize 1= 20943 nodes, ConceptSize 2=101 nodes).
3. Label generated examples using the generated concepts while adding 10% noise to examples attributes and classes. Also, Create 2 training sets.
4. Generate 2 series of random sampling reservoirs using labelled data, reservoirs sizes are (25800 samples, 5MB).
5. Present the 2 series of reservoirs to C4.5 with default parameters, and measure accuracy using the test file corresponding to each series.
6. Learn VFDT 2 times using the same labelled data of both concepts and measure accuracy for the incrementally learnt classifier so far at the same points of the stream as reservoir images are taken using the corresponding test file, allowing 5MB of memory usages.
7. Compare.

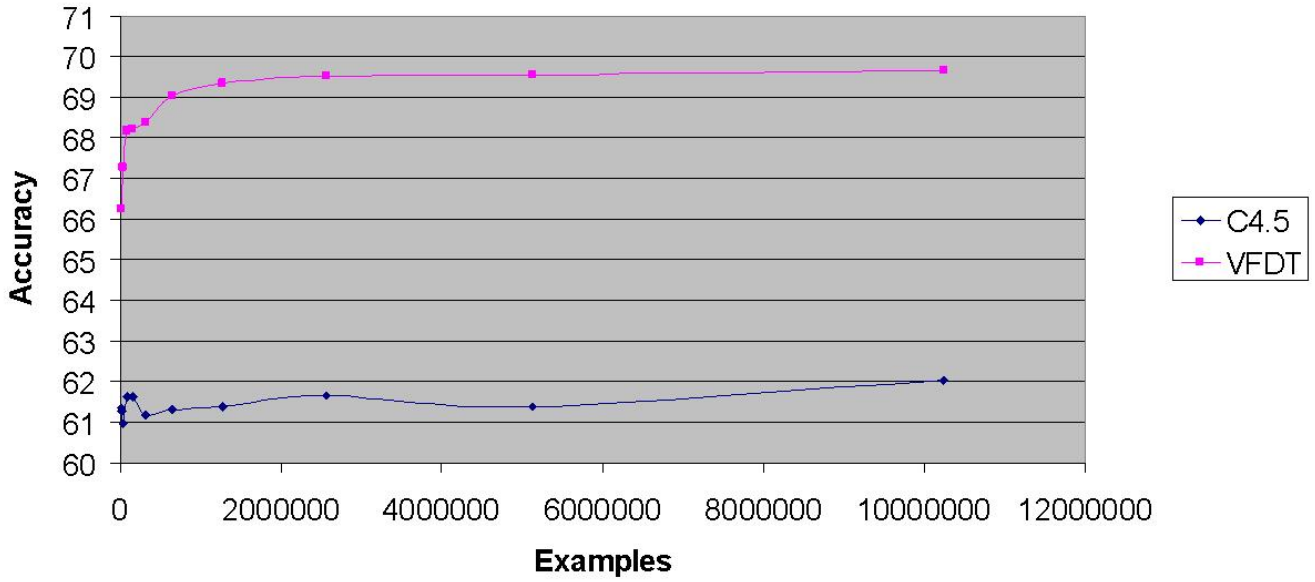


Figure 17: VFDT vs C4.5 - concept size 20943 nodes (reservoir memory = 5MB, concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 18, learnt on $10 * 10^6$ examples, tested on 50000 examples)

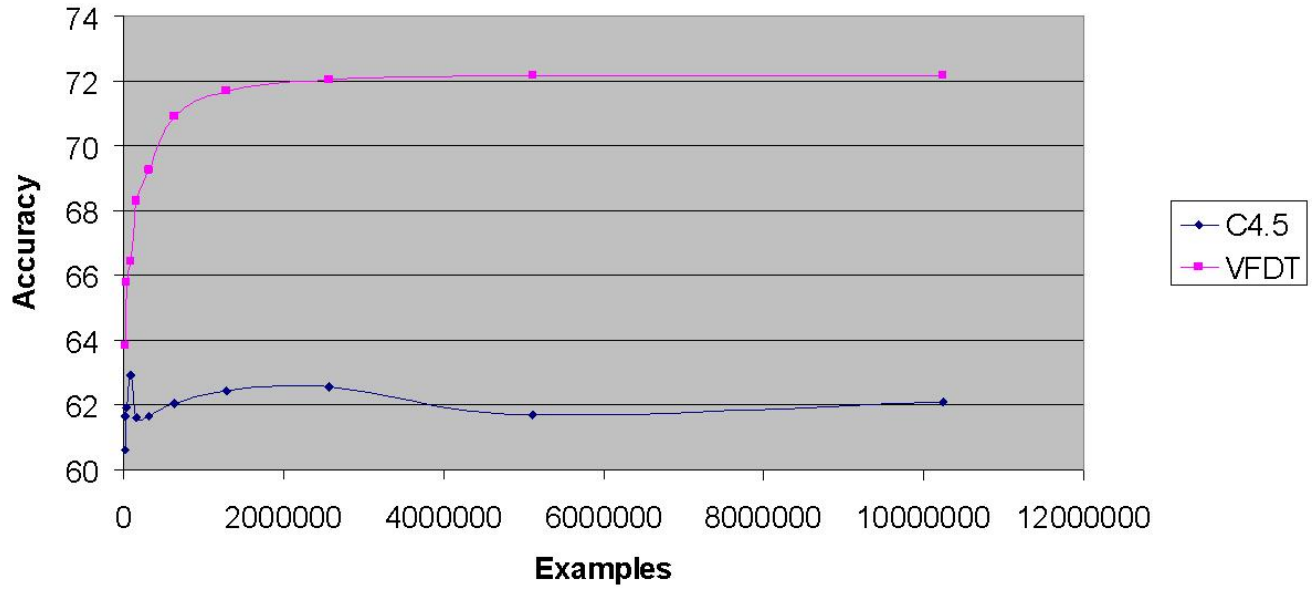


Figure 18: VFDT vs C4.5 - concept size 101 nodes (reservoir memory = 5MB, concept seed=4, samples seed=1234, noise level 10.00%, prune percent 25%, max prune level = 7, learnt on $10 * 10^6$ examples, tested on 50000 examples)

The results of this experience are shown in figures [17] and [18].

In the first concept, which is complicated, the accuracy of C4.5 ripples while increasing and the ripple axe bent is positive, this means that the reservoir becomes more representative to the data underlying concept. While, in the second concept, the ripple axe is horizontal, and thus no new information is learnt after seeing more examples.

No worthy remark can be listed, except for a slight improvement in accuracy for both algorithms in the simpler data underlying concept. We can say that concept size doesn't favour C4.5 over VFDT.

6.3.5 Noise effect on C4.5 and comparing with VFDT

Here are the steps for this experience:

1. Generate a stream of $5 * 10^6$ examples on the fly using some examples seed.
2. Generate a concept using some concept seed.
3. Label generated examples using the generated concepts while adding 7 different levels of noise each time (n= 0,5,10,15,20,25,30), by reassigning attributes and class values with a probably of n% and with the same probability of all attribute values or class values. Also, generate 7 test files of 50000 samples each for each noisy stream.
4. Generate 7 random sampling reservoirs at the point of $5 * 10^6$ examples seen using noisy labelled data; reservoirs sizes are (25800 samples, 5MB).
5. Present the 7 reservoirs to C4.5 with default parameters. And test using the test file of the same noise level.
6. Learn VFDT 7 times, one for each noise level using the 7 different labelled data streams at the same point of the stream ($5 * 10^6$ samples) allowing 5MB of memory usages.
7. Compare.

The results of this experience are shown in figure [19].

As expected, increasing noise effect affects the accuracy of both classifiers, while VFDT is still doing better; it is affected by noise at the same degree of C4.5. Also here we can say that noise doesn't favour C4.5 over VFDT.

6.3.6 Real data

Plan of experiment:

1. Preparation of Orange data as described above.
2. Generation of 8 random sampling reservoir images using data in training file, reservoirs sizes are (207000 samples, 40MB).
3. Present the 8 reservoir images to C4.5 with default parameters and do tests on the test file.
4. Learn VFDT allowing 40MB of memory usage, using the training file data and testing at the same points of the stream as reservoir are taken using the test file.
5. Compare.

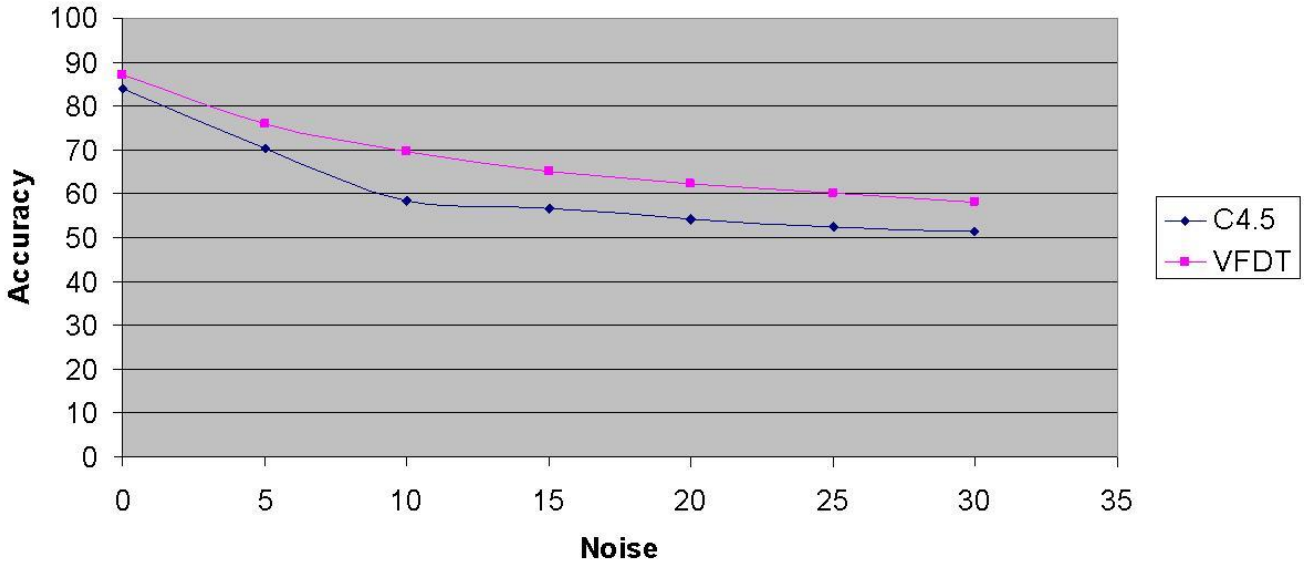


Figure 19: Noise effect (reservoir memory = 5MB, concept seed=4, samples seed=1234, prune percent 25%, max prune level = 18, learnt on $5 * 10^6$ examples, tested on 50000 examples)

Figure [20] shows VFDT and C4.5 performance until 640,000 samples only. Figure [21] shows the results in the form: $Accuracy = log_2(examples)$. Class 1 ratio in the training data file and the resulting random sampling reservoirs is shown in figure [22].

It can be seen that in the first three points, the ratio is identical between the training data file and the reservoirs, because the reservoirs are not filled yet, so random selection and replacement of values didn't start yet. Later, the ratio differs lightly because random guarantees the appearance of same ratios of classes as in the data file.

In our randomly selected training dataset we notice class appears 2 times only in the first 10000 samples, and 9 times in the first 20000 samples, and 54 times in the first 40,000 examples. While it the ratio of class 1 example increases later to 39.14 after seeing 640000 samples. This is a real world case, data has some kind of order and one should expect such a case in real world applications. This doesn't mean that there is any concept drift in the data, because appetency distribution with attributes had -surely- a single form at the specific time interval when data is collected. Appetency to the channel may change over long period of time, when some new marketing technologies are thrown in the market, or when some abnormal event affects some parameter of the whole operation environment, like unexpected crises or exceptional events, Noel period is a good example, people marketing behaviour is expected to change, so they would likely prefer to profit from vacations and offers comparison to buy directly from shops channel. We suppose that our data is collected over a period where there is no exceptional event or crisis.

C4.5 and VFDT gives identical results for the first three points (10000, 20000, 40000 samples). The justification for this phenomenon is that as sampled data so far are less than the size of the reservoir (207000 samples), and the reservoir will be partially filed by the same data as in training file. Before this point in the random sampling algorithm life, random sampling is not started yet.

Based on this, we notice that C4.5 and VFDT give exactly the same results. The reason is that there is very few examples labelled by 1 as class. And the built classifier by c4.5 is naïve. We expect this classifier to be a very simple rule using only zero attributes to predict the class. The classifier rules here take this simple form: $Classifier : Class = 0$. This justifies the identical small accuracy in the first 3

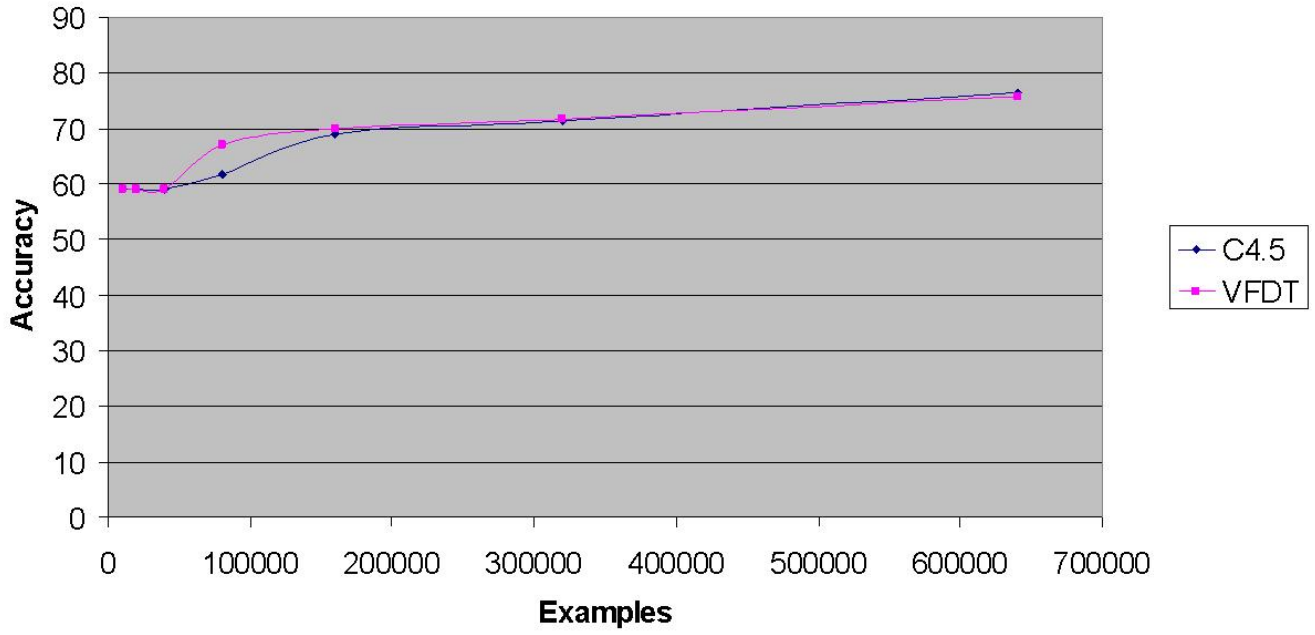


Figure 20: C4.5 and VFDT on Orange data (reservoir memory = 40MB, learnt on $640 * 10^3$ examples, tested on $50 * 10^3$ examples)

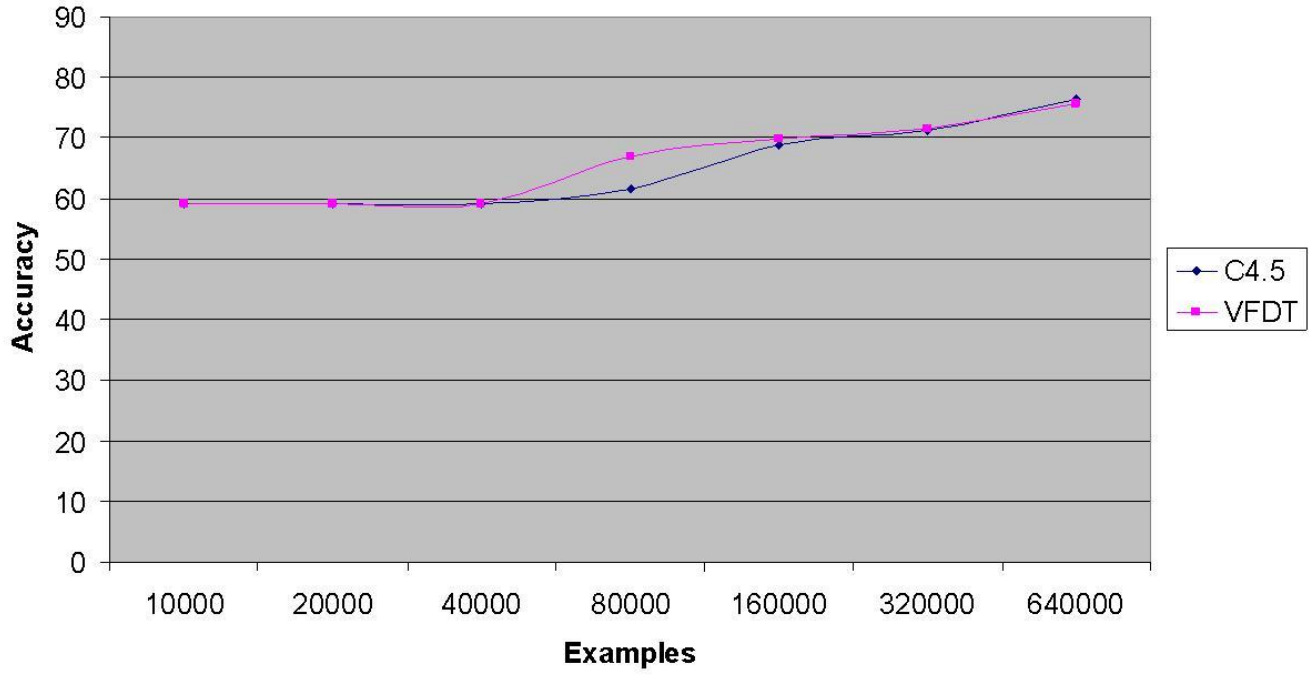


Figure 21: C4.5 and VFDT on Orange data - $Accuracy = log_2(examples)$ (reservoir memory = 40MB, learnt on $640 * 10^3$ examples, tested on $50 * 10^3$ examples)

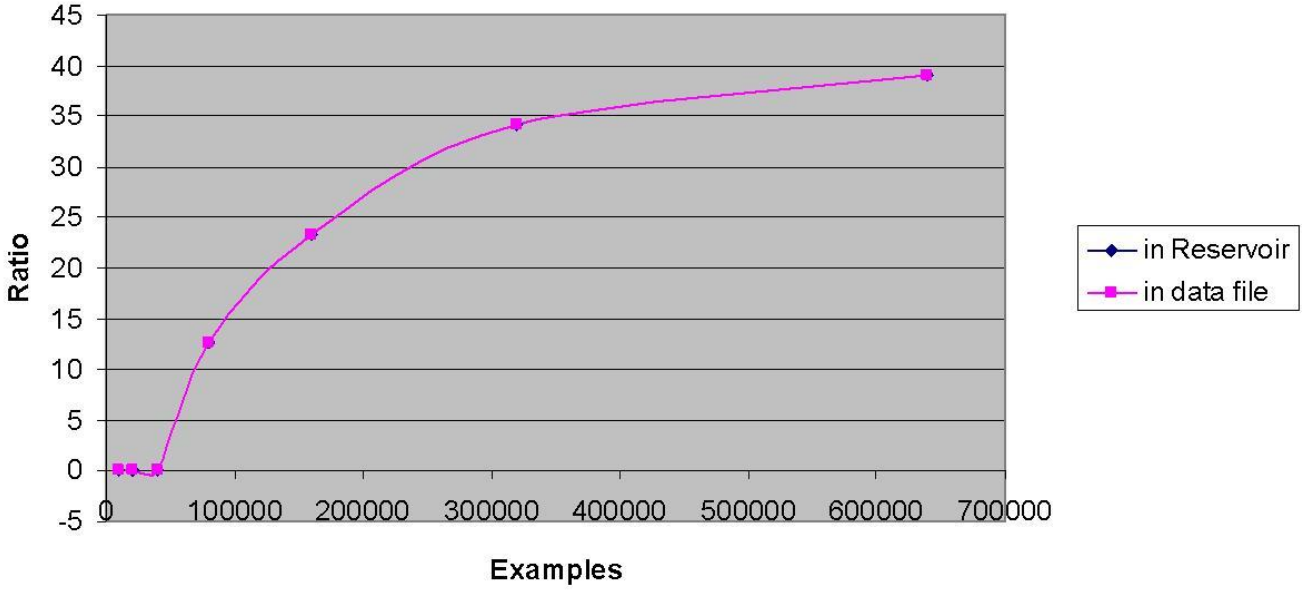


Figure 22: Class 1 ratio in training data file and random sampling reservoirs

Examples	10000	20000	40000	80000	160000	320000	640000
C4.5	1	1	1	1924	3685	6044	5612
VFDT	3	11	23	59	119	217	427

Table 3: Tree node number growth

points for C4.5.

For VFDT, it built a more developed tree than C4.5 as seen in Table [3] and used calculated statistics based on the seen data, but it didn't dare to assign any seen example to the class 1 because it has no sufficient statistical evidence on any attribute. It carefully builds it tree using strongly discriminate attributes only.

On the fourth point (80000 samples), C4.5 builds a larger tree than VFDT, but with less accuracy. Here we say that VFDT exploited its previously built statistics to give a good accuracy on this point. Until this point C4.5 gives bad results because it sees a non balanced training set as class 1 ratio is still low (12.7%), and this is a common problem in traditional decision trees, because the ratio of leaves with considerable error rate is high when the class training set is not balanced.

In the fifth point(160000 samples), C4.5 performance gets better, as the non balance of class distribution is reduced, and gives a good result almost identical to the result of VFDT. In the sixth point (320000 samples) C4.5 were slightly better while VFDT were slightly better on the seventh points (640000 samples).

In fact, there is a major difference between our results and those of Domingos. C4.5 performance in Domingos experiments were better than that of VFDT at the start of learning process, and it is improving over time until VFDT's performance exceeds C4.5 performance. In our expirements, we found that VFDT give better results than C4.5 at the start of the learning process, even before the filling of the reservoir and thus the launch of random sampling algorithm. Here it is important to mention that we couldn't get the seeds used in Domingos experiments, thus, we chose seeds randomly for our expirements.

However, the reader may have noticed that our experiment results on real data gave an interesting behaviour for C4.5 with random sampling, it is almost the same performance as VFDT and regardless of produced tree size, there exist no preference for VFDT on C4.5.

We couldn't capture the origin of the difference between our results and those of Domingos on synthetic data, but it is likely due to bad chance in concept and data stream seeds selection.

Note: For the benefit of those who desire to use VFML 1.3.2 framework, it is important to mention that we encountered bug of VFDT implementation in reading files larger than 100MB, so we couldn't test the point 1,028,000 of our real dataset.

7 Conclusion

In this internship thesis we briefly introduced supervised learning and classification problems. We defined our scope of interest to be binary classification decision trees when the data flow is stationary. We searched in machine learning literature to find clear definition for inconsistently used terms in previous works. We presented several point of views of scientists and tried to extract and propose standardized definitions for the underlying concepts behind these terms. At the end of our work we defined offline learning, batch learning, sequential learning, online learning, incremental learning, anytime learning and active learning.

Later, we narrowed our research subject to focus on incremental learning. We explained the extensions and permissions of our definition boundaries, extracted the basic variables on each of which the incremental learning process depends, this included time between samples, allocated data memory, sample flow life time, goal performance and desired algorithm characteristics. Then, we studied the effect of time between samples, memory issues, samples order and unseen classes on incremental learning performance.

We further narrowed our research subject to specifically study incremental decision trees, presented its state of art and focused on the most powerful algorithms in the domain. We compared their performance and selected VFDT to be a reference algorithm to which we compared later our proposed method in our experiments.

In our experimental study, we explained the sufficient theoretical base to advance in our experiments, described the source and characteristics of used synthetic and real data, besides to the used software in experimentation. We proposed the method of using C4.5 algorithm with random sampling technique, trying to approach the performance of VFDT while being able to overcome the obstacle of memory, which is normally issued in batch learning algorithms.

Using synthetic data, we studied the effect of C4.5 parametrization, random sampling reservoir size effect, concept size effect and noise effect on C4.5 with random sampling method, and compared its performance with VFDT performance, while applying both algorithms on the same data.

Finally, we compared the performance of both algorithms on Orange specific real data. We enhanced the difference between results on synthetic data and on real data. We succeeded to use C4.5 on very large database using a *small* amount of memory, and that was reflected in a fast training of the decision tree.

Many lessons are learnt from this study. Wider vision and better comprehension to machine learning problems are acquired when surfing large amount of literature. In our experimental study we were faced by several traditional data mining problems including memory lack, high processor quota and long processing time, we solved these problems and enhanced the performance to the extent possible.

References

[1] Principal Investigator Alan C. Schultz. Anytime learning - continuous and embedded learning. http://www.nrl.navy.mil/aic/as/AnytimeLearning.php.

[2] Hiyan Alshawi. *Online Multiclass Learning with K-way Limited Feedback and an Application to Atterance Classification*, volume 60 N/1/2/3. Springer, 2005.

[3] Laviniu Aurelian Badulescu. Attribute selection measure in decision tree growing. In *Proceedings SINTES 13 The International Symposium on System Theory, Automation, Robotics, Computers, Informatics, Electronics and Instrumentation 2(1)*, pages 1–6, 2007.

[4] Andrea Bonarini. Anytime learning and adaptation of structured fuzzy behaviors, 1997.

[5] Alexi Bondu. *Apprentissage Actif par Modèles Locaux*. Orange Labs, 2008. Phd thesis.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

[7] José del Campo-Ávila and Morales-Bueno Ramos-Jiménez. Incremental algorithm driven by error margins. In *9th Int. Conf. on Discovery Science*, 2006.

[8] José del Campo-Avila Campo-Ávila, Gonzalo Ramos-Jiménez, João Gama, and Rafael Morales-Bueno. Improving prediction accuracy of an incremental algorithm driven by error margins. In Ralf Klinkenberg João Gama, Jesus S. Aguilar-Ruiz, editor, *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams (IWKDDS 2006), held in ECML/PKDD 2006*, Berlin, Germany, September 2006.

[9] Lior Cohen, Gil Avrahami, Mark Last, and Abraham Kandel. Info-fuzzy algorithms for mining dynamic data streams. *Appl. Soft Comput.*, 8(4):1283–1294, 2008.

[10] William W. Cohen. Stacked sequential learning. In *IJCAI-2005*, 2005.

[11] S. L. Crawford. Extensions to the cart algorithm. *Int. J. Man-Mach. Stud.*, 31(2):197–217, 1989.

[12] Dean and Boddy. An analysis of time-dependent planning. In *Seventh National Conference on AI*, 1988.

[13] Christophe Giraud-Carrier Department and Christophe Giraud-carrier. A note on the utility of incremental learning. *AI Communications*, 13:215–223, 2000.

[14] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2000. ACM.

[15] LiMin Fu. *Incremental Knowledge Acquisition in Supervised Learning Networks*, volume 26. 1996.

[16] P. Saratchandran G.-B. Huang and N. Sundararajan. *An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks*, volume 34. 2004.

[17] Joao Gama, Ricardo Fernandes, and Ricardo Rocha. Decision trees for mining data streams. *Intell. Data Anal.*, 10(1):23–45, 2006.

[18] Grefenstette, Ramsey, and Connie Loggia Ramsey. An approach to anytime learning. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 189–195. Morgan Kaufmann, 1992.

[19] Greiner. http://www.cs.ualberta.ca/~greiner/RESEARCH/seq.html.

[20] JIAN SU CHIH-LUNG LIN HAIZHOU LI, JIN-SHEA KUO. *Mining Live Transliterations Using Incremental Learning Algorithms*, volume 21. World Scientific Publishing Company, 2008.

[21] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[22] Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003.

[23] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, New York, NY, USA, 2001. ACM.

[24] INRIA. Project-team presentation. http://ralyx.inrialpes.fr/2007/Fiches/sequel/sequel.html.

[25] D Fisher J C Schlimmer. A case study of incremental concept induction. *Artificial Intelligence.*

[26] Gonzalo Ramos-Jiménez José del Campo-Avila and Rafael Morales-Bueno. *Incremental Learning with Multiple Classifier Systems Using Correction Filters for Classification*, volume 4723/2007. Springer Berlin / Heidelberg, 2007.

[27] Mark Last. Online classification of nonstationary data streams. *Intell. Data Anal.*, 6(2):129–147, 2002.

[28] Last-M. Maimon, O. The info-fuzzy network (ifn) methodology. 2000.

[29] Marcus A. Maloof and Ryszard S. Michalski. Incremental learning with partial instance memory. *Artif. Intell.*, 154(1-2):95–126, 2004.

[30] Josep Maria Mateo-Sanz, Antoni Martínez-Balleste, and Josep Domingo-ferrer. Fast generation of accurate synthetic microdata. In *Privacy in Statistical Databases, vol.3050 of LNCS*, pages 298–306. Springer, 2004.

[31] Stefano Ferilli Nicola Di Mauro, Floriana Esposito and Teresa M.A. Basile. *Avoiding Order Effects in Incremental Learning*, volume ISBN978-3-540-29041-4. Springer-Verlag Berlin Heidelberg 2005, 2005.

[32] Patrice Bertrand Paula Brito, Guy Cucumel and Francisco de Carvalho. *Induction Graphs for Data Mining, from: Selected Contributions in Data Analysis and Classification.*

[33] Robi Polikar, R. Polikar, Lalita Udpa, Vasant Honavar, Satish Udpa, S. S. Udpa, and V. Honavar. Learn++: An incremental learning algorithm for multilayer perceptron networks. *IEEE Transactions on System, Man and Cybernetics (C), Special Issue on Knowledge Management*, 31:497–508, 2000.

[34] Robi Polikar, Lalita Udpa, Senior Member, Satish Udpa, and Vasant Honavar. An incremental learning algorithm with confidence estimation for automated identification of nde signals. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, 51:990–1001, 2004.

[35] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106.

[36] J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[37] R.L. Chilausky R.S. Michalski. Learning by being told and learning from examples. In *4 International Journal of Policy Analysis and Information Systems*, pages 125–161, 1980.

[38] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[39] Ron Sun and C. Lee Giles. Sequence learning: From recognition and prediction to sequential decision making. In *IEEE Intelligent Systems*, volume 16, pages 67–70, 2001.

[40] Pedro Domingos Daniel S.Weld Tessa Lau, Steven A. Wolfman. *Online Ensemble Learning: An Empirical Study*, volume 53 N/1/2. Springer, 2003.

[41] Frank M. Thiesing. On-line training, 1994. http://www.inf.uos.de/papers_html/pvmug94_frank/node4.html.

[42] Usenet. Usenet newsgroup monthly post. http://www.faqs.org/faqs/ai-faq/neural-nets/part2/preamble.htm.

[43] P. Utgoff. An incremental id3. *Fifth International Conference on Machine Learning.*

[44] Paul E. Utgoff. Incremental induction of decision trees. *Mach. Learn.*, 4(2):161–186, 1989.

[45] Paul E. Utgoff. Decision tree induction based on efficient tree restructuring. In *Machine Learning*, pages 5–44, 1995.

[46] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[47] S. Zilberstein. Anytime algorithms in intelligent systems. *American Association for Artificial Intelligence*, 17(3), 1996.

APPENDICES

.1 Anytime learning

Anytime learning is also called "continuous and embedded learning" [1] to denote a particular way of learning and execution, where the basic idea lies behind is to integrate two continuously running modules: learning and execution.

Anytime learning systems contain two internal modules for learning and execution, each of which is compound of several sub-modules. Thus, they require significant computational capabilities, and important design and realisation efforts proportional to their complexity.

In order to get the best controller over time, anytime systems are considered as Learning Classifier Systems and are usually implemented with a core of genetic algorithm that evolves a population of controllers (based on strategies or hypotheses) [4]. This means that each generated controller have to be tested against an in-system simulation model of the external environment, to be refused or accepted. This also augments the system's computational effort and implementation complexity.

Contrarily to offline incremental learning systems, anytime systems must be immersed in their working environment and are not built to work offline.

[47] Defines anytime algorithms as "algorithms whose quality of results improves gradually as computation time increases." Based on this latest citation, we can imagine that anytime learning could be seen as a bi-dimensional space, the first axe is time and the second axe is the decreasing quality of results, and our mission is to calibrate the position of the operational point $(time, -quality)$ on a system specific performance curve, that suites the best the special environment of the current application. For example, a "natural" environment whose events are slow could be assigned a functional point with high processing time and high results quality as we can guarantee that no sudden change will occur. At the inverse, a fast changing environment should be well studied in order to decide what is critical, tracking time or quality of results and calibrate the operational point based on this decision.

Practical notes:

[4] explains the common used architecture and functionality of anytime systems: "The anytime learning algorithm proposed by Grefenstette has two modules running in parallel: a learning system that learns behaviours running on a simulated environment, and an execution system that controls the embodied agent in the real environment. The execution system uses the best controller produced so far by the learning system, which continues to try to improve it, running in parallel. At the same time a monitor checks whether the model adopted for the simulation still matches the actual environment, and, eventually, updates the simulation model. Thus, when the environment changes, the simulation model is modified accordingly, and the learning system works on a simulated environment always reflecting the real one."

[1] propose the same architecture and modules interaction, figure [23] shows a structural block diagram of anytime systems:

[47] distinguishes between two types of anytime learning: "A useful distinction has been made between two types of anytime algorithm, namely, interruptible and contract algorithms. An interruptible algorithm can be interrupted at any time to produce results whose quality is described by its PP (performance profile). A contract algorithm offers a similar trade-off between computation time and quality of results, but the total allocation must be known in advance. If interrupted at any point before the termination of the contract time, it might not yield any useful results. Interruptible algorithms are in many cases more appropriate for the application, but they are also more complicated to construct."

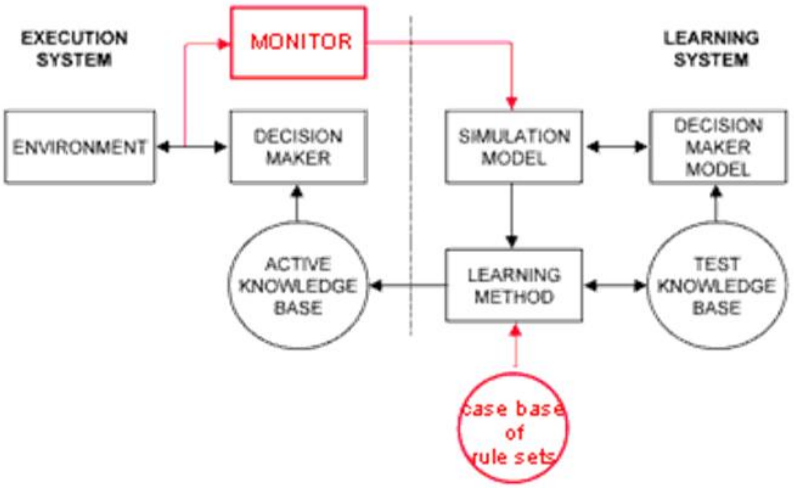


Figure 23: Block diagram of anytime systems structure [1]