CS 691A : Machine Learning
Project
Online Learning of Decision Trees

G. Anastasovski, N. Salmani, V. Papakroni, S. Boothapati

December 14, 2012

Abstract

Contemporary hardware and software breakthroughs have enabled vast collections of data sets in numerous scientific fields such as genetics, astronomy, weather, web traffic etc. Most often these data sets contain repeating sequences that we can use to build a statistical model and extract previously unknown patterns such as groups, dependencies and anomalies. Collecting these data sets is not a trivial task; however, processing them is even harder. Most of the algorithms that have been used up to now build a statistical model using the whole data set (batch algorithms). These algorithms have their pros and cons but they have proven to be reliable for most of the time. In addition, they often fail, due to memory constraints, when the dataset never stops growing and when the training examples arrive at a very fast pace. Contemporary data mining algorithms can deal with massive datasets and also build models as instances arrive i.e. the model is created online. In this report, we will present and analyze three supervised decision tree algorithm that is able to learn online in an incremental fashion. Three datasets used to analyze these algorithms are KDD,Flight data, Waveform generator data. These datasets cannot be processed by a batch machine learning algorithm in an offline fashion.

1 Introduction

Just as the age of the Industrial Revolution had a profound effect on the lives of our forefathers, todays so-called Digital Evolution has become ubiquitous in ours. Web, computer, and mobile applications are everywhere and without these applications our lives are practically unimaginable. The emerging of such applications lead to massive information sharing and enormous data sets. Currently Amazon.com offers more than 2.2 million book titles [2], tens of millions of different queries are sent to Yahoo! Search every day [11], the Electricity Market dataset that was collected from the Australian New South Wels Electricity Market contains around 45,000 examples [7]. The accumulated data may contain some valuable information for the organization storing it [10]. Data mining this information is quite a challenge due to several reasons, as we shall see next.

Historically, the field of data mining used to extract data and build statistical models from small amounts of examples. Algorithms were fast, efficient, built a low error model, and all data used for training the model could fit in the operating memory. Such algorithms, where the entire set of data could fit in memory and be used for training the model, are called batch or offline algorithms. However, processing large amounts of data that cannot fit into the operating memory is needed. Stream data must generally be processed in an online manner in order to guarantee that results are up to date and that queries can be answered with small time delay. One possible solution is to store data on secondary storage devices, but this makes its access particularly expensive and computation can take long time. Often, to overcome these problems data is divided into several smaller subsets which are used for training and validating the created model. This approach has proven to be very effective in the past; however, this approach easily fails if we consider data streams coming at very fast pace such as 10,000 or more per second. A data stream is a sequence of items that can be seen only once, and in the same order it is generated. Examples of such data streams include air traffic, network event logs, credit card transaction, telephone call records, and surveillance video streams. To make things worse some data sources are up and generating instances all the time providing continuous flow of data, hence, a batch algorithm will never fit the entire set in memory nor compute it in a reasonable amount of time.

The principal task is to learn a concept incrementally by processing labeled examples one at a time [16]. Concept refers to the output function, which the model is trying to predict. In the literature, algorithms that process one instance at a time and update their model as each instance are referred to as Incremental or On-line learning algorithms [4, 6]. Since these algorithms do not fit all the examples in the working memory they can easily work with large volumes of data that are open ended. Furthermore, the speed that is needed to classify each example and update the existing model varies from algorithm to algorithm and in most cases is faster than batch learning algorithms.

Before we start elaborating the state of the art in incremental learning there is one more thing that needs to be considered and that is the so called concept drift; changes that occur in the underlying concept over time are called concept drift. Concept drift occurs when the process that is generating the data changes over time and this leads to a high error rate in the model that is trying to classify the data. Most of the offline and some of the online learning algorithms suppose that the data generation process comes from a stationary distribution i.e. the data generation process does not change over time. Unfortunately, this is not the case, data is seldom stationary. Often the time needed to collect the data is long. Furthermore, events exist that can rapidly change the underlying distribution (political, weather events).

One on the most cited papers in the data mining and machine learning research society is a paper by Domingos and Hulten in [3]. The authors propose an incremental learner called VFDT (Very Fast Decision Tree Learner) which is built upon Hoeffding trees. A Hoeffding tree represents a decision tree where each node contains a test on an attribute, each branch is a corresponding outcome to a test in a node, and each leaf is a decision or classification. When deciding the best test attribute for a node in batch learning algorithms like ID3 and CART, usually a heuristic function is used and all examples are in memory. VFDT too, relies on a heuristic function when choosing a test attribute in a node, but VFDT does not use all the examples in memory. The tree is built incrementally; the first examples to come from the data stream are used to decide a test attribute at the root and the others are passed down the tree to decide test attributes at the internal nodes. A problem arises, how many instances are needed to decide whether to split a node. The solution is trivial; the authors use a Hoeffding bound which is a statistical measure that determines the number of examples needed for a split with some given probability. The Hoeffding bound has the very attractive property that it is independent of the probability distribution generating the observations [3]. This algorithm suggests that in order to learn the tree each example needs to be inspected only once and that the tree is learned in constant time per example. Furthermore, the incremental nature of the algorithm does not affect precision quality of the tree, yet it improves it. Given enough examples, this algorithm produces more accurate and less overfitting trees than conventional decision trees. The authors prove that VFDT would have the same results as a particular batch decision tree, if that batch decision tree could be run on the entire data stream.

If all the above is considered VFDT represents a very efficient incremental learning algorithm which solves the problem of open ended data sets and very fast arrival of instances. However, VFDT supposes that the examples are generated by a stationary stochastic process which means that the generation of data does not change over time. Having that said, this algorithm will perform poorly in real world data sets, which require one to cope with concept drift. Therefore, a successor of VFDT, CVFDT (Concept-adapting Very Fast Decision Tree Learner) was proposed [9]. CVFDT keeps VFDTs speed and accuracy and extends it by being able to respond to changes in the data generation process i.e. it can respond to concept drift. It uses a fixed in size sliding window of training examples to build, modify, and keep up to date its Hoeffding tree. As new examples arrive they are added to the window and old ones are discarded. When it comes to dealing with concept drift, CVFDT grows an alternate subtree whenever an old one seems to be out of date, and replaces the old with a new subtree whenever the new subtree becomes more accurate [9]. If the underlying concept starts to change, CVFDT will start to grow an alternative subtree for the nodes where some of the splits no longer pass the Hoeffding bound. The nodes that no longer pass the Hoeffding bound represent root nodes of subtrees in the Hoeffding tree. Due to changes in the data generation, these nodes no longer pass the Hoeffding bound, and now, different attributes have higher information gain. CVFDT often enters in testing mode where it checks to see if alternate subtrees outperform subtrees rooted at the low performing node, in the original Hoeffding trees. If an alternative subtree proves to be more accurate than the rooted subtree in the Hoeffding tree, the rooted subtree is replaced by the alternative subtree. This way CVFDT always has a proper model that reflects the current concept generating the examples. CVFDT is advantageous to VFDT because it is able to quickly respond to concept drift. However, the down side of the CVFDT is that it discards subtrees that are out of date. Some of these subtrees might be useful in the future because there might be repeating patterns in the data. It is important to mention that in CVFDT a very small window will create a very accurate model when there is a big drift in the data, and a very large window is beneficial when the data generation process is stable. Therefore, it would be best if the size of the window would dynamically adjust itself to the rate of concept drift.

An online algorithm that dynamically adjusts its size of window, based on the current rate of concept drift, is OLIN (On-Line Information Network) [10]. OLIN uses an info-fuzzy network, or IFN, for classification of incoming examples where IFN represents a tree like structure, consisting of several layers. When building the tree a heuristic function called IN is used to determine which attribute should be used to split on which node. OLIN is divided into three modules: Learning module (creates and modifies an IFN tree by using the IN algorithm); Classification module (classifies currently arriving examples); and the Meta-Learning module (calculates the size of the next window and determines if the Learning model needs to be modified). Whenever a concept drift is occurring, OLIN reduces its window size, and whenever the concept seems to be stable, the size of the window is increased up to a predefined maximum size. We mentioned previously that the data streams considered by online learning algorithms can be open ended. In addition, depending on the nature of the problem studied, it is possible to have repeating concepts in the data stream. In other words there might be presence of long term concepts in the data stream.

Therefore the authors in [16] present a whole framework of online learning algorithms named FLORA, which tries to remember repeating concepts or patterns in the data. The first implementation of FLORA, called FLORA2 algorithm, attempts to dynamically adjust the size of the window during learning. FLORA3 is similar in approach to FLORA2, but also, improves FLORA2 by inspecting the current state of learning and deciding if a previously seen concept could represent the current concept. The idea is that when a context change seems to occur, the system should consult its store of old concept descriptions to see whether some old concept might better describe the examples currently in the window [16]. Concept re-installment is a very expensive task; therefore the successor of FLORA3, called FLORA4, adds stability to FLORA3 by trying to distinguish between random noise in the data stream and actual concept drift. As one can see there are many online learning algorithms, and in this paper we have only presented a handful. Other papers in the literature include online ensembles of classifiers [14], unsupervised online learning algorithms [8] and etc.

The proposed approach is described in Sec. 2.1. The datasets used for testing the VFDT implementation are explained in detail in Sec. 3. Performance of VFDT is compared with other existing methods. The experimenting process and other methods used to compare the performance of VFDT are described in Sec. 4 whereas Sec. 5 contains the results. Results are compared with Ensemble classifiers described in Sec. 2.2.

2 Classifiers

Two classifiers are implemented.

- VFDT
- Ensemble classifiers

2.1 VFDT

Given a training data represented by X and the class labels are represented by y, a model $y = f(X)$ has to be designed from a training data. Decision tree is one such model that generates a set of rules based on the information (information gain). Attribute that has highest information gain is considered to be the root node. This method is based on Hoeffding bound which states, “Consider a real values random variable r with values R. If n independent observations are available, Hoeffding bound states that, with probability 1-δ, the true mean is atleast $\overline{r} - \epsilon$ where $\overline{r}$ is the mean and ϵ is computed using

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

The Hoeffding tree algorithm is explained in detail in Sec. 2.1.1

2.1.1 Hoeffding tree algorithm

Procedure Hoeffding Tree (S, X, G, δ)

Inputs:	S	is a sequence of examples
	X	is a set of discrete attributes
	G(.)	is a split evaluation function
	δ	1-probability of choosing correct attribute
Output:	HT	is a decision tree

Table 1: Description f variables used in Procedure Hoeffding Tree.

```
Let $HT$ be a tree with a single leaf $l_1$ (the root).
For each class $y_k$
    For each value $x_{ij}$ of each attribute $X_i \in \mathbf{X}$
        Let $n_{jk}(l_1)$=0
For each example $(\mathbf{x}, y)$ in $S$
    Sort $(\mathbf{x}, y)$ into a leaf $l$ using $HT$
    For each $x_{ij}$ in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
        Increment $n_{jk}(l)$
    Label $l$ with the majority class among the examples seen so far at $l$
    If the examples seen so far at $l$ are not all of the same class, then
        Compute $\overline{G_l}(X_i)$ for each attribute $X_i \in \mathbf{X_l}$ using the counts $n_{ijk}(l)$.
        Let $X_a$ be the attribute with highest $\overline{G_l}$.
        Let $X_b$ be the attribute with second-highest $\overline{G_l}$.
        Compute $\epsilon$ using Equation 1.
        If ($\overline{G_l}(X_a) - \overline{G_l}(X_b) > \epsilon$ or $\overline{G_l}(X_a) - \overline{G_l}(X_b) < \epsilon < t$) and $n > n_min$ then
            Replace $l$ by an internal node that splits on $X_a$.
            For each branch of the split
                Add a new leaf
                For each class $y_k$ and each value $x_{ij}$ of each attribute $X_i \in \mathbf{X_m}$
                    Let $n_{ijk}(l_m)$=0
Return $HT$
```

The basic idea of the Hoeffding Tree algorithm is to collect the necessary statistics at each leaf of the tree to determine the subsequent splitting attributes. Each training instance passes through the current tree, until it ends to a leaf node. The algorithm then increments a set

of counters (n_{ijk}) according to the attribute values and the class value of the instance. The statistics of the leaf nodes consist of counters for each combination of attribute, attribute value and class value.

Utilizing these statistics, the algorithm computes an evaluation score for each attribute. Different evaluation functions G can be used to estimate the score of the attributes. In our implementation, we use the information gain function. The algorithm then estimates the Hoeffding bound ϵ defined in equation (1). If the difference ΔG is greater than the bound ϵ, then the current leaf node is split using the best attribute as condition attribute.

The value of the bound ϵ depends on three variables: the number of class values c (because $R = log(c)$), the confidence level 1-δ, and the number of training instances in the leaf nodes. With the increase of the number of training instances in a leaf node, the Hoeffding bound becomes eventually small enough for the difference ΔG to exceed the bound ϵ. In case ΔG could not exceed ϵ even after a large number of training instances, then the difference between the two best attributes could be considered insignificant. When the bound ϵ becomes smaller than a user specified τ threshold, then a tie is declared and the node is split based on the best attribute.

The estimate of the G scores and Hoeffding bound is done each time a training instance is presented. In order to save some computational time, the following condition: $n > n_{min}$ is added. The algorithm does not compute the G scores and Hoeffding bound until the minimum number of instances in the node is reached. n_{min} is a user specified parameter.

Whenever a node is split into several leaf nodes, the n_{ijk} are initialized to zero for each attribute, attribute value and class value. In order to save memory, the statistics in the splitting node are deleted.

2.1.2 Combining Hoefdding trees with Na$\ddot{i}$ve Bayes classifier

According to the pseudo-code, the class label of each leaf node is defined by the class with majority number of training instances in that node. When a test instance is presented, it passes the tree until it ends to a leaf node. The label of that leaf node is defined as the predicted class value.

Another approach for predicting the labels of the test instances is to build Na$\ddot{i}$ve Bayes classification model on each leaf node. The Na$\ddot{i}$ve Bayes classifier is based on the Bayes Theorem presented in equation 2.

The Na$\ddot{i}$ve Bayes classifier estimates the posterior probability for each class value based on the class prior probability and the class-conditional probability of each attribute. The class with the highest probability is defined as the predicted class value. Assuming independence of attributes, the probability of a given class value c is the following:

In order to define the class of the test instance, the Na$\ddot{i}$ve Bayes classifier needs to have the prior probability$P(c)$ and the class conditional probabilities $P(x_i|c)$. Whereas, $P(x)$ is the same for each class and does not contribute to the classification.

The n_{ijk} statistics used by the Hoeffding algorithm to build the tree can be later used to estimate the likelihood of each class value for a test instance. In this way, the algorithm does not need to do any further training to bueild a Na$\ddot{i}$ve Bayes classifier. The NB models is already defined by the counters kept at each leaf node for each combination of attribute, attribute value and class value.

2.2 Ensemble Classifiers

Unlike VFDT which uses one classifier, ensemble classifiers [15] use number of classifiers to determine the output. Each classifier is trained on a different training data. Classification learner C4.5 [12] is used, which is similar to ID3 in computing information entropy and information gain. C4.5 is capable of handling continuous and discrete attributes, handles training data with missing attributes and prunes trees after creation. In this method, we divide the data into several chunks. Decision trees are obtained from each of the first k chunks of training data. This method uses weighted classifier ensembles to mine streaming data with concept drifts. Assume a stream of 2-dimensional data is partitioned into sequential chunks based on their arrival time. Let S_i be the data that came in between time t_i and t_{i+1}. Each chunk of data will have a decision tree computed. The $k + 1^{th}$ chunk of data is tested based on the majority vote of all the five classifiers. Each classifier has an error rate on the test data. The classifier with higher error rate will be discarded replacing it with the decision tree computed using the test set S_{k+1}. For a test example y, a classifier outputs $f_c(y)$, the probability of y

being an instance of class c. A classifier ensemble pools the outputs of several classifiers before a decision is made. Majority voting is used to find the final output. To estimate the error, consider S_n consists of records in the form of (x, c), where c is the true label of the record. C_i classification error of (x, c) is $1 - {f_c}^i(x)$ where ${f_c}^i(x)$ is the probability given by C_i that x is an instance of class c. Mean square error of the classifier C_i is expressed by:

$$MSE_i = \frac{1}{|S_n|} \sum_{(x,c) \in s_n} (1 - {f_c}^i(x))^2 \tag{2}$$

The weights of the classifier C_i should be reversely proportional to MSE_i. A classifier predicts randomly will have mean square error:

$$MSE_r = \sum_c p(c)(1 - p(c))^2 \tag{3}$$

Weights w_i for a classifier C_i is given by:

$$w_i = MSE_r - MSE_i \tag{4}$$

2.2.1 Algorithm

```
**Input:** $S$ : a dataset of $ChunkSize$ from the incoming stream
        $K$ : the total number of classifiers
        $C$ : a set of $K$ previously trained classifiers
**Output**: $C$ : a set of $K$ classifiers with updated weights
train classifier $C'$ from $S$;
compute error rate of $C'$ via cross validation on $S$;
derive weight $w'$ for $C'$ using equations for weights update
**for** $each classifier C_i \in C$ **do**
    apply $C_i$ on $S$ to derive $MSE_i$;
    compute $w_i$ based on the equations;
$C \leftarrow K$ of the top weighted classifiers in $C \cup C'$;
return $C$;
```

The key feature here is that the ensemble machine learning algorithm is able to cope with datasets where concept drift is present. We saw previously that VFDT produces a single model that represents the entire data stream. This method suffers in prediction accuracy in the presence of concept drifts. This is because the streaming data are not generated by a stationary stochastic process, indeed, the future examples we need to classify may have a very different distribution from the historical data.

In order to make time-critical predictions, the model learned from the streaming data must be able to capture up-to-date trends and transient patterns in the stream. To do this, the ensemble classifier revises the model by incorporating new examples, and eliminating the effects of examples representing outdated concepts i.e., examples that represent outdated concepts are not used for training. The principal task here is to determine what training examples represent outdated concepts and thus remove them from the ensemble. It makes sense to think that the very first examples that come should be discarded as time progress since they will represent old concepts. However, repeating concepts exist in the data generation process and therefore the expiration of old data must rely on the datas distribution instead of only their arrival time.

Dataset	# Instances	# Instances used	#Attributes	#Target classes
KDD Cup 1999	5 million	2.5 million	42	25
Flight 2007	8 million	8 million	13	3
Flight 2008	5 million	5 million	13	3
Wave	1 million	1 million	22	3

Table 2: Summary of the datasets used

This means that instead of discarding data using the criteria based solely on their arrival time, decisions are made based on the datas class distribution. Historical data whose class distributions are similar to that of current data can reduce the variance of the current model and increase classification accuracy.

3 Data-sets

Threee data-sets are used for experiments.

- KDD Cup data 99
- Flight Data
- Waveform Data

The details about each dataset is summarized in Table 2.

3.1 KDD cup data-set 99

The data-set was used to build a predictive model that is capable of distinguishing between 'good' and 'bad' connections. The details of KDD data-set if found in [13].

The data-set contains approximately 4,900,000 instances each with 41 attributes. These instances are labeled as either normal or an attack. The target attribute contains 24 values from the training set, with an additional 14 types in the test set only. These attacks are divided into four types:

- Denial of service(DoS): The attacker makes computing or memory resource too busy or too full to handle legitimate requests.
- User to Root Attack (U2R): The attacker has access through a user account through sniffing passwords or dictionary attack.
- Remote to Local Attack (R2L): Attacker has unauthorized access from a remote machine.
- Probing Attack (Probe): Attacker gathers information to circumvent the surveillance.

The 24 target attributes can be mapped to these 4 types of attacks. Target attribute 'Normal' indicated a 'good' connection. The types of attacks and mapping is shown in Table 3.

The instances have 41 attributes. The detailed summary of the attributes is shown in Table 4.

Target Attributes	Group
back	DoS
buffer_overflow	U2R
ftp_write	R2L
guess_passwd	R2L
imap	R2L
ipsweep	Probe
land	DoS
loadmodule	U2R
multihop	R2L
neptune	DoS
nmap	Probe
perl	U2R
phf	R2L
pod	DoS
portsweep	Probe
rootkit	U2R
satan	Probe
smurf	DoS
spy	R2L
teardrop	DoS
warezclient	R2L
warexmaster	R2L

Table 3: Table indicates grouping target attributes.

Nr.	Attribute	Type	Description
1	duration	continuous	length of the connection in seconds
2	protocol_type	discrete	type of the protocol
3	service	discrete	network service on the destination
4	src_bytes	continuous	number of data bytes from source to destination
5	dst_bytes	continuous	number of data bytes from destination to source
6	flag	discrete	normal or error status of the connection
7	land	discrete	if connection is from/to the same host/port
8	wrong_fragment	continuous	number of wrong fragments
9	urgent	continuous	number of urgent packets
10	hot	continuous	number of hot indicators
11	num_ failed_logins	continuous	number of failed login attempts
12	logged_in	discrete	successful login
13	num_compromised	continuous	number of compromised conditions
14	root_shell	discrete	root shell obtained
15	su_attempted	discrete	'su root' command attempted
16	num_root	continuous	number of root accesses
17	num_file_creations	continuous	number of file creation operations
18	num_shells	continuous	number of shell prompts
19	num_access_files	continuous	number of operations on access control files
20	num_outbound_cmd	continuous	number of outbound commands
21	is_hot_login	discrete	login belongs to 'hot' list
22	is_guest_login	discrete	guest login
23	count	continuous	number of connections to same host in past 2 secs
24	serror_rate	continuous	% of connections that have "SYN" error
25	rerror_rate	continuous	% of connections than have "REJ" error
26	same_srv_rate	continuous	% of connections to the same service
27	diff_srv_rate	continuous	% of connections to different service
28	srv_count	continuous	number of connections to same service in the past 2 secs
29	srv_serror_rate	continuous	% of connections that have "SYN" errors
30	srv_rerror_rate	continuous	% of connections than have "REJ" error
31	srv_diff_host_rate	continuous	% of connections to different hosts
32	dst_host_count	continuous	
33	dst_host_srv_count	continuous	
34	dst_host_same_srv_rate	continuous	
35	dst_host_diff_srv_rate	continuous	
36	dst_host_same_src_port_rate	continuous	
37	dst_host_diff_src_port_rate	continuous	
38	dst_host_serror_rate	continuous	
39	dst_host_srv_serror_rate	continuous	
40	dst_host_rerror_rate	continuous	
41	dsr_host_srv_rerror_rate	continuous	

Table 4: Summary of training instances for KDD dataset

Nr.	Attribute	Type	Range/Values	Description
1	Year	continuous	1987-2008	Year of the flight
2	Month	discrete	1,2,3,...,12	Month of the flight
3	Day of Month	discrete	1,2,3,..,31	Day of month of the flight
4	Day of Week	discrete	1(Mon)-7(Sun)	Day of week of the flight
5	CRSDepartureTime	continuous	0000-2359	Actual local departure time(hhmm).
6	CRSArrTime	continuous	0000-2359	Actual local Arrival time(hhmm)
7	UniqueCarrier	discrete	List of carrier Ids	Unique Carrier code
8	FlightNum	continuous		Flight number
9	ActualElapsedTime	continuous		Elapsed time of the flight in minutes
10	Origin	discrete	List of Airport Ids	Origin airport code
11	Destination	discrete	List of Airport Ids	Origin airport code
12	Distance	continuous		Distance of flight in miles
13	Diverted	discrete	0(No)-1(Yes)	Flight Diverted
14	ArrivalDelay	discrete		Flight delayed

Table 5: Summary of training instances for Flight dataset

3.2 Flight dataset

The data sets used in this study contain details about departures and arrivals of commercial flights within US from 1987 to 2007. The data was taken from the Data Expo competition of 2009 and was later processed by Dr. Ikonomovska at Jozef Stefan Institute, Slovenia [1]. The data is grouped into different data sets according to the respective year of the flight. It is composed of 13 attributes related to different characteristics of the flights. In addition, the data sets have a target attribute called ArrivalDelay, which gives the delay time of a flight in seconds. The summary of attributes is shown in Table 5.

3.3 Waveform Generator Dataset

Waveform Generator dataset with one million records is being used in our experiments. This data set was donated to UCI repository by David Aha and it consists of three different waveform each of which is generated from combination of three base waves including 21 different attributes. The original version of this dataset has numeric attributes; however, in order to use this data for our experiments, we apply 10 bins discretization on entire data. The distribution of data is discretized as in [5].

3.4 Fitting the data in memory

There are two options in handling the stream of data to learn the decision trees. The first option is to load the entire dataset into memory and process the data in an online fashion. The second option is to sequentially read from a file, a fixed proportion of the data, until EOF is reached. The first approach is preferred in cases the data needs to be preprocessed before training. However, this approach violates one of the major reasons why online algorithms are developed, i.e. to deal with the constraint of limited memory. The second approach keeps the memory consumption low and the machine is able to store into memory larger trees.

In order to keep the memory consumption as low as possible, all attribute and class values are represented with specific integer ids. I.e. instances are stored into memory as arrays of integers. Hash tables are used to map between the attribute values and their corresponding ids. In this way, a string can be quickly transformed into array of integers and be further processed.

4 Experiments

4.1 Data preprocessing

Before running the experiments, the datasets were transformed in order to be processed by the implemented algorithms. The following subsections briefly describe the preprocessing procedures performed on the datasets.

4.1.1 Discretization of the class attributes

Both the algorithms implemented in this project work on classification problems, i.e. predicting class values from a fixed set of predefined classes. The target attribute in the original Flight datasets is continues. In this way, the target class values on both datasets need to be converted to discrete class values.

The target class of the Flight datasets was discretized using equal-frequency discretization. As the name suggests, this technique divides the list of instances sorted by class into a specific number of bins of equal size. [6] A discrete class is then assigned to each interval defined by the bins. The number of bins is arbitrarily defined by the user. However, the choice of the number of bins has a great impact on the future classification. In order to capture the underlying model the discrete values should approximately reflect the true distribution of the data.

Processing the Flight datasets has been difficult because of the large size of the datasets. Both datasets were initially combined into a single dataset. The data was loaded in MS SQL Server 2008 database and instances were sorted into by their class. A Transact-SQL script was then executed to discretize the data. The number of bins was equal to 3.

4.1.2 Discretization of other attributes

The implemented VFDT tree algorithm processes only data with discrete attribute values. All four datasets contains continuous numerical attributes. Consequently, these attributes had to be discretized before running the VFDT tree algorithm.

In case of the Flight dataset, equal frequency discretization is used for ActualElapsedTime and Distance attribute. The number of bins was respectively 9 and 5. The CRSDepartureTime and CRSArrTime were discretized converting the values to the corresponding hour (in 24 hour format). Again, the discretization of attributes was performed on the combined dataset loaded in MS SQL Server dataset.

In case of Wave dataset, all attributes were discretized using equal-frequency discretization with 10 bins. The discretization was performed using the Weka data mining tool.

For the KDD Cup data set, we have tried a supervised algorithm to discretize the numerical attributes. In case of supervised discretization, the class of the instances is taken into consideration when discretizing other attributes. The algorithm applied to discretize the KDD Cup data set is Fayyad-Irani discretizer. Initially, the instances are sorted by their attribute values. The algorithm divides the instances into two bins trying to maximize the information gain. The algorithm then iterates on each partition applying the same strategy until a stopping criteria is met. Again, the discretization was performed using the Weka tool[4].

4.1.3 Randomization of the datasets

The VFDT algorithm is assumed to run on datasets that are generated by stationary stochastic processes. Consequently, the algorithm cannot handle the cases when the target concepts changes as the stream of data come. For this reason, the sequence of the input data should be randomized.

The Wave dataset is randomly generated and contains no concept drift. The KDD Cup dataset used in the experiment was generated by applying a random stratified subsampling of the original dataset using the Weka tool.

For the Flight datasets a Python procedure is implemented to randomize the data (refer to Randomize method in the source code). The procedure initially loads all instances into memory and randomly shuffles them. Subsequently, the instances are rewritten to new files.

4.2 Machine and programming language

The VFDT and the Ensemble classifier were tested on different machines due to time and resource limits. This makes the comparison of the algorithms performance difficult to analyze.

The parameters of machine where VFDT was run are the following

- Processor: Intel (R) Core (TM) i7-2630 QM CPU @ 2.00 GHz
- Memory (RAM): 6.00 G
- System type: 64 bit
- Windows: Windows 7 Home Edition, Service Pack 1

The parameters of machine where Ensemble classifier was run are the following

- Processor: Intel (R) Core (TM) i7 CPU Q 720 @ 1.60 GHz
- Memory (RAM): 16.00 GB
- System type: 64 bit
- Operating System: Windows 7 Ultimate, Service Pack 1

VFDT was implemented using Python version 2.7. Ensemble classifier was implemented using MATLAB R2011a Student Version. MS SQL Server 2008 and Weka version 3.6 were also used for several preprocessing tasks described earlier.

4.3 The testing scheme

The scheme applied to test the algorithms is the following. The stream of data was processed in chunks of fixed size. In every iteration, both algorithms initially test their current models on the new chunk of data. Afterwards, they use the same chunk to modify their models.

The following measures and results are reported by VFDT on every iteration:

- The time in seconds and minutes for the test process.
- The test accuracy rate for simple VFDT and VFDT+NB.

- The current decision tree as a list of rules. For each rule, the algorithm reports the number of test instances for which the rule is applied and the respective accuracy rate for simple VFDT and VFDT+NB.
- The time in seconds and minutes for the train process.
- The memory size of the updated tree. The Python method getsizeof, is used to estimate the total number of bytes used to store all the nodes of the tree.
- The tree size of the updated tree in terms of number of internal nodes, leaf nodes and total nodes.

All this results are written in text files and can be found in the deliverable file named: VFDT_logs.zip.

The measures reported by the Ensemble Classifier on every iteration are the following:

- The test accuracy rate.
- The time in seconds for the train process.
- The total memory consumption of the program. The MatLab method memory was used to estimate the total memory in bytes.

The chunk size used for testing the Flight datasets is 100,000 instances. The VFDT algorithm was run on both versions of the flight datasets: sequential (by date) and randomized. The number of processed instances was 5,300,000 for each dataset. (The original datasets could not be preprocessed for more than 10,600,000.)

The chunk size used for testing the KDD Cup is 50,000. The dataset used to test the KDD Cup is half of the original dataset, i.e. 2,500,000 instances. (Weka could not discretize more than this number of instances due to memory constraints.)

The chunk size for the Wave dataset is 20,000. This dataset contains 1,000,000 records.

The following parameters were used when testing the VFDT algorithm:

- Confidence level: 95
- Minimum number of training instances in the leaf: n_{min} = 2000 (for Flight datasets) and n_{min} = 500 (for other datasets)
- Tie threshold: τ = 0.01

The model used in the Ensemble classifier is C 4.5. The number of modes is equal to 3

5 Results

Figure 1 to 4 show the accuracy results for the implemented algorithms on the four datasets. Figure 5 and 6 show the time needed respectively by VFDT and Ensemble classifier to train on every iteration. Figure 7 and 8 describe the memory consumption of VFDT respectively in terms of physical memory and total number of the tree nodes. Finally, Figure 9 shows the total memory consumption of program implementing the Ensemble classifier. For more details please refer to the raw data given in the Appendix.

Analyzing Figure 1 and 2, we notice that all the algorithms have approximately the same classification accuracy trend between the two Flight datasets. No one of the algorithms could reach an accuracy rate greater than 50%. This poor results may be due to the poor quality of the training data.

It is interesting to notice the unstable accuracy rate of VFDT when applied to the sequential datasets. As we have previously stated, the VFDT algorithms can be applied on data where there is no concept drift. However, this cannot be stated for the Flight data, because the target concept is highly depended on the time. Even Ensemble learning classifier is very unstable and could not produce better results than VFDT. On the other hand, VFDT applied on random data produces stable results throughout the iterations. VFDT combined with Na$\ddot{i}$ve Bayed produces the best results (almost 50% by the end of the execution).

Figure 3 shows that all the algorithms performed very well on the KDD Cup dataset. This dataset is a popular dataset used by many researchers. Consequently, we believe that the high accuracy rates are due to the very good quality of the data. Furthermore, the dataset was discretized using the supervised Fayyad-Irani algorithm. The algorithm discretizes the attributes in a way that increases the information gain, increasing in this way their predictive ability.

The Wave dataset produces the most interesting results. As we would expect the VFDT algorithm starts with very low accuracy values in the first iterations. With more training, the decision tree becomes more complex and could predict better the test instances. By a certain point of time the accuracy rate of VFDT gets stabilized. On the other hand, the Ensemble classifier performs in a stable way throughout the execution.

The VFDT combined with Nave Bayes classifier produces the best results on the Wave dataset, achieving an accuracy rate of 83% by the end of the execution. Furthermore, the algorithm performs pretty stable throughout the course of the program execution, outperforming all the other algorithms. These results show a positive impact of Na$\ddot{i}$ve Bayes when combined with a Decision tree. In case when the majority class of the leaf node determines the predicted class, the learner does not take into consideration the information that could be extracted by the instances of the other classes. Consequently, building Na$\ddot{i}$ve Bayes model on the leaf nodes could help the decision tree achieve better results.

Analyzing the training time between VFDT and Ensemble classifier (Figure 5- 6), we can conclude that VFDT takes more time to train especially for the Flight datasets. The VFDT is more expensive in terms of computational time, because it sequentially processes all the training instances. Furthermore, the training time of VFDT is not stable, although the chunk size of the training data is fixed throughout the execution. VFDT processes the node statistics only when certain number of training instances end up to the node. When new nodes are created, VFDT runs faster because it does not process the statistics.

Figure 7 and 8 show the trend of the tree size produced by VFDT in terms of memory and number of nodes. The VFDT algorithm does not include any stopping criteria on growing the tree. For this reason, as the learner is trained the size of the tree grows without bounds. This can be especially noted for the Flight datasets, where the size of the tree grows exponentially.

On the other hand, the memory consumption of the Ensemble classifier is more stable. This is due to the fact that the trees generated by C 4.5 do not change drastically in size as the program executes.

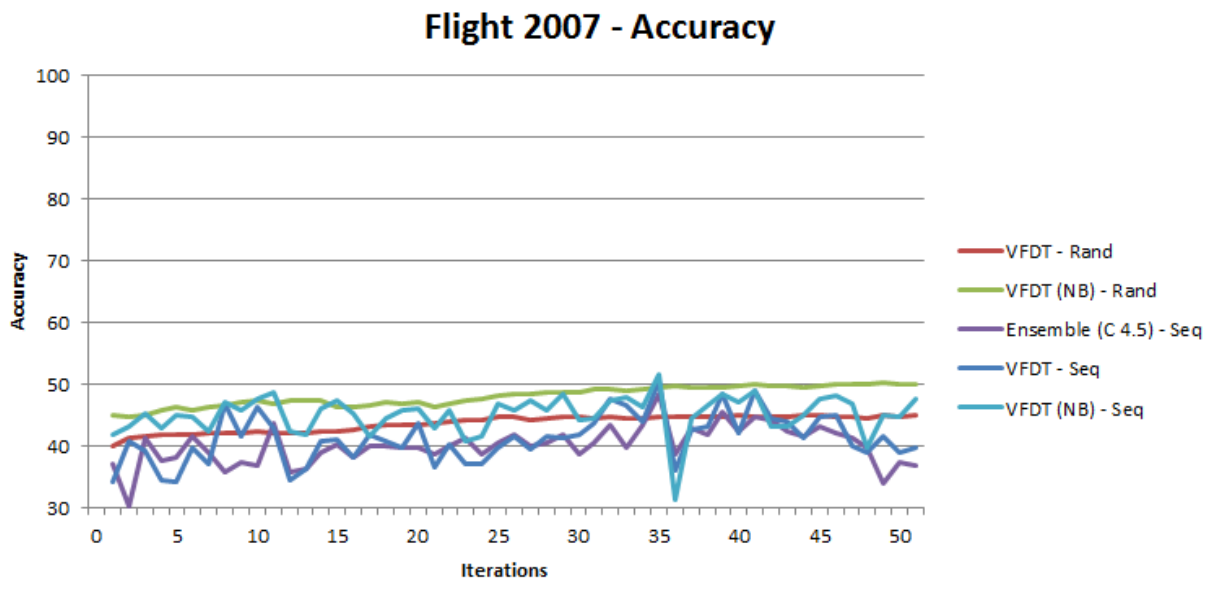

Figure 1: Flight 2007 dataset accuracy

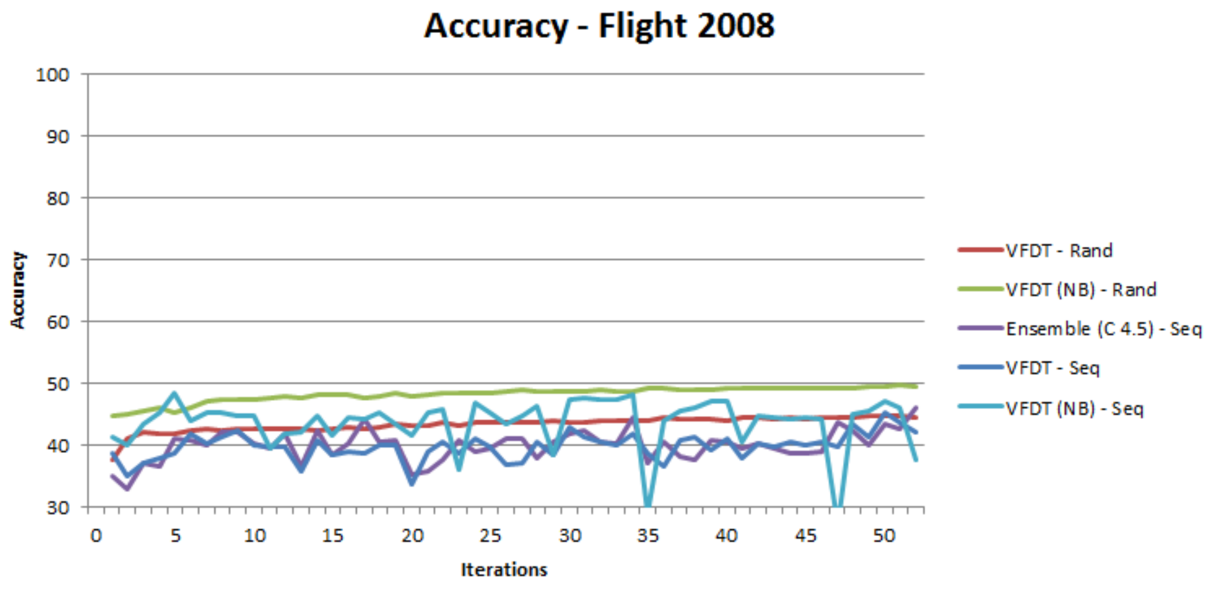

Figure 2: Flight 2008 dataset accuracy

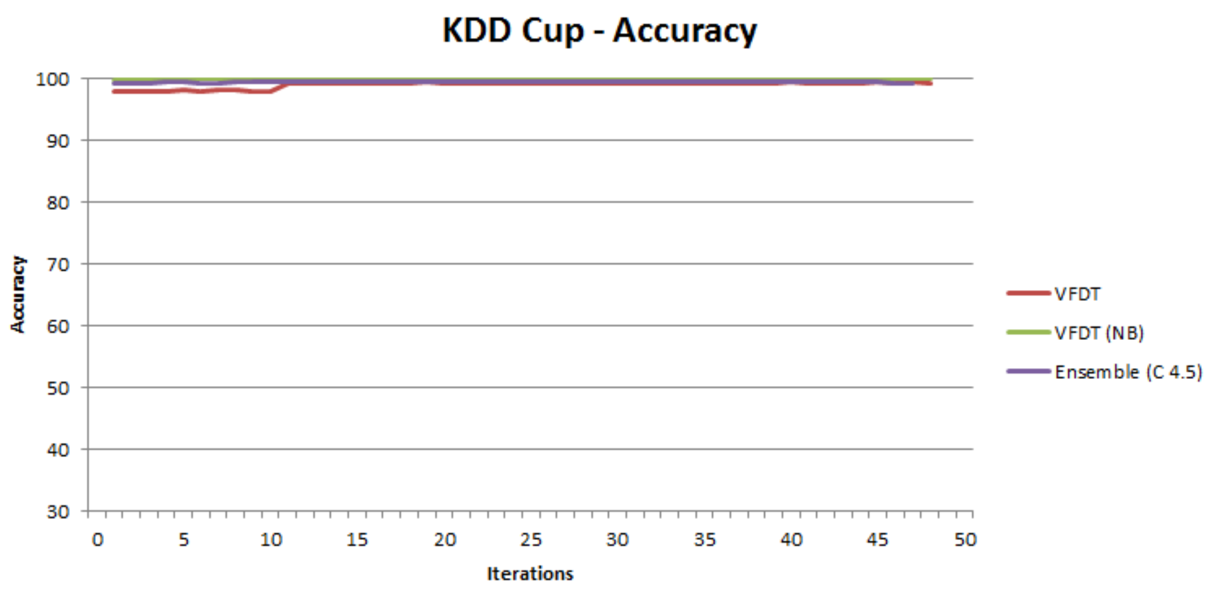

Figure 3: KDD Cup dataset accuracy

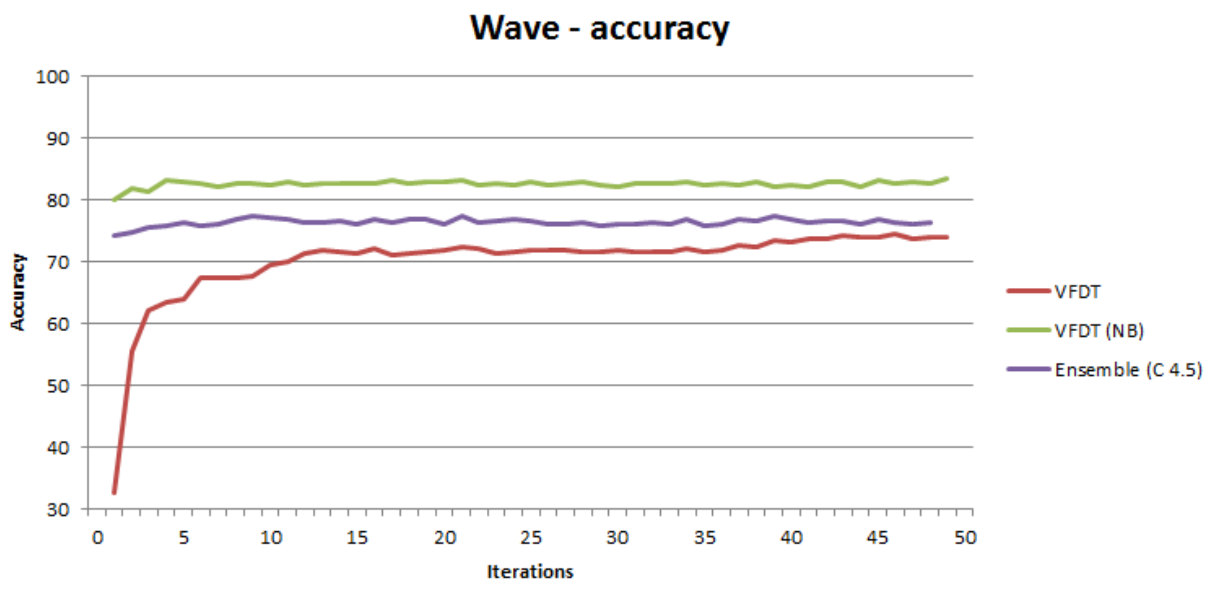

Figure 4: Wave dataset accuracy

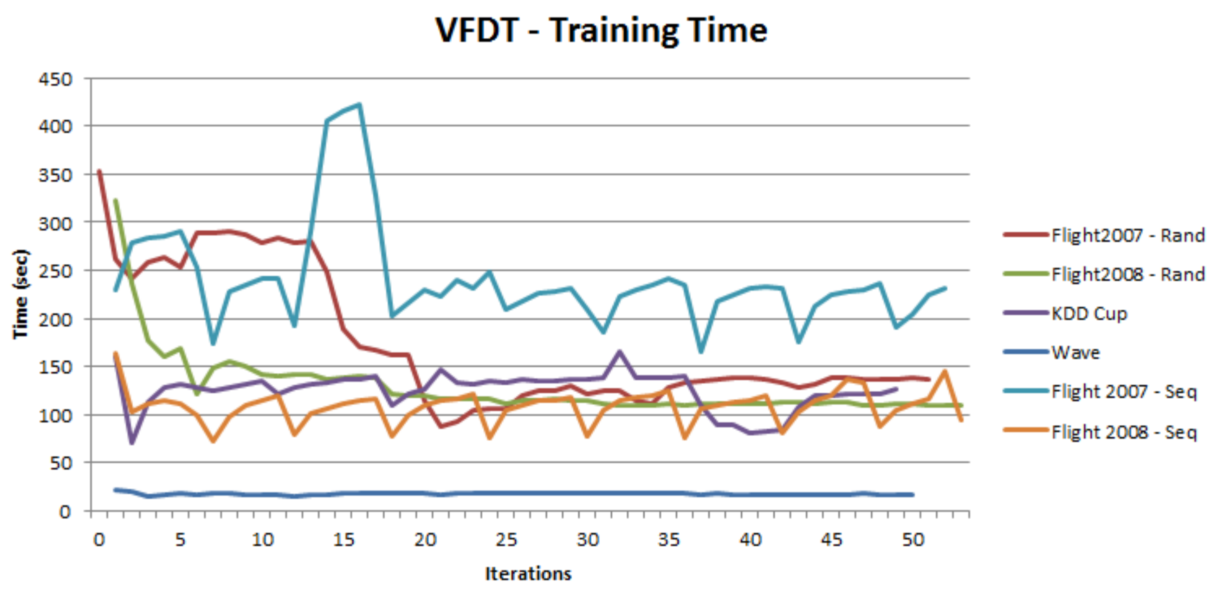

Figure 5: VFDT training time

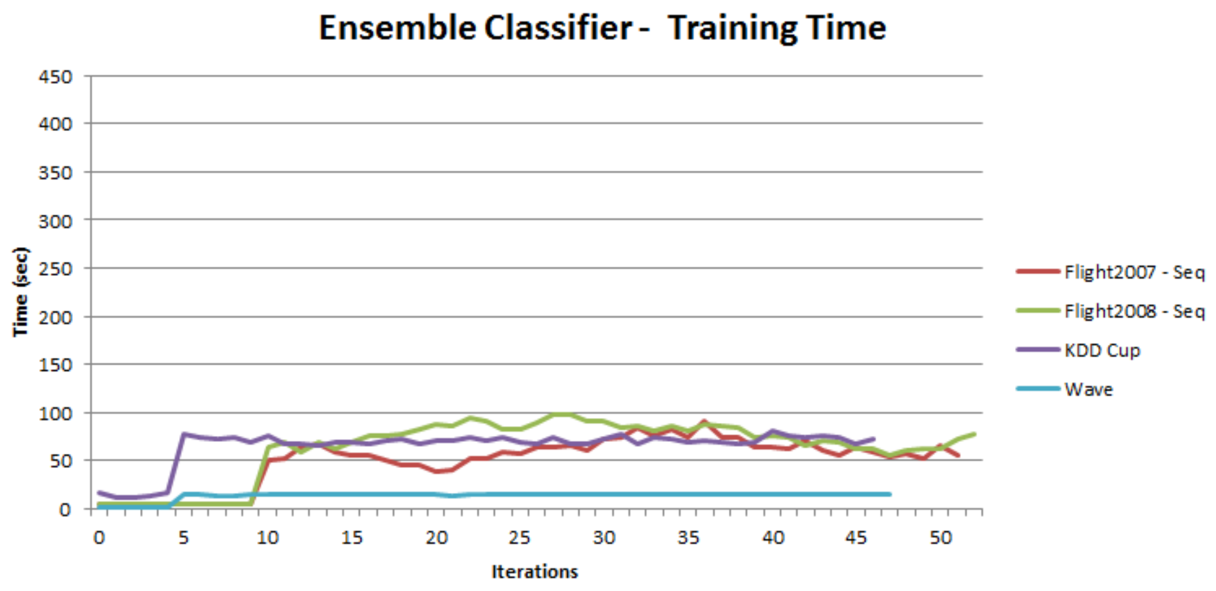

Figure 6: Ensemble training time

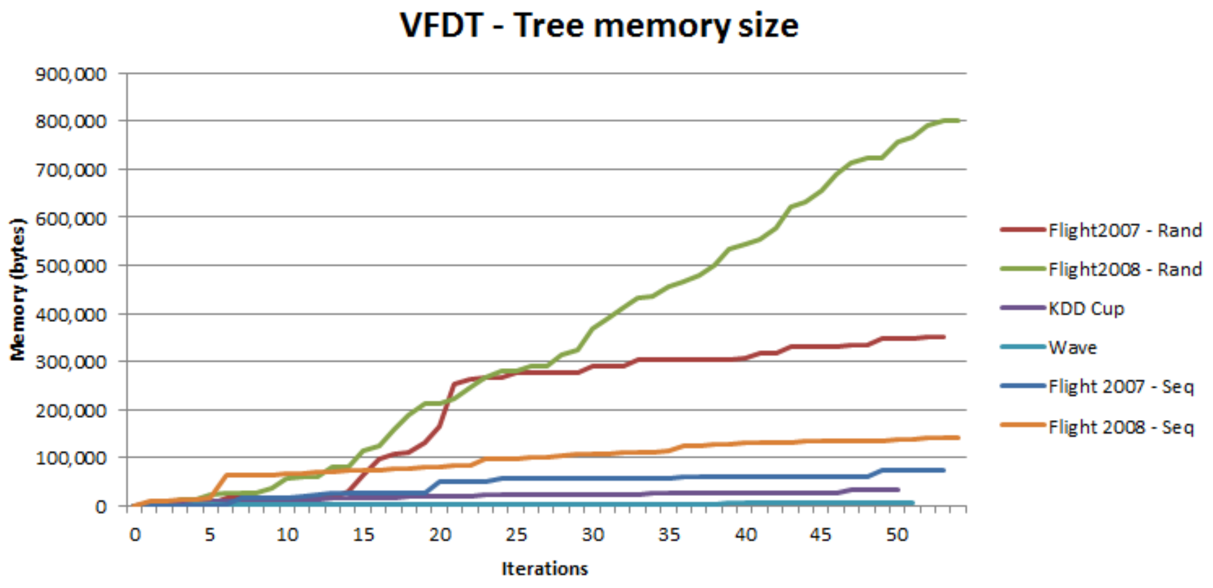

Figure 7: VFDT memory

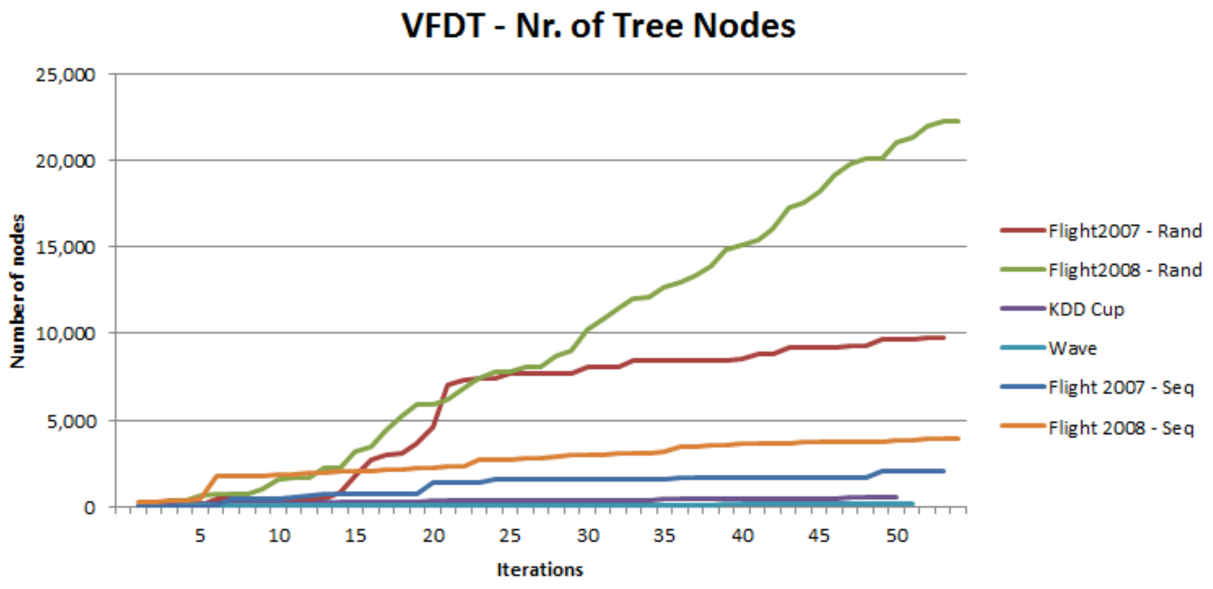

Figure 8: VFDT Number of tree nodes

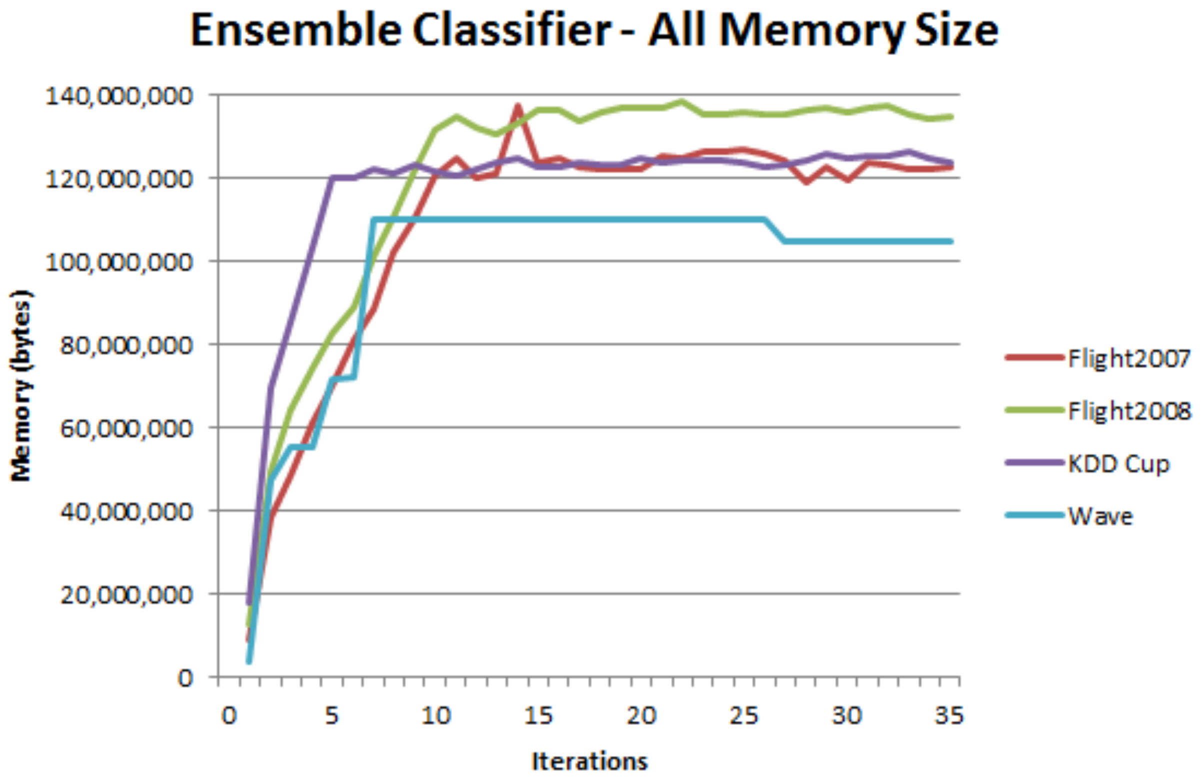

Figure 9: Ensemble classifiers memory

6 Conclusion

This project focused on online learning of decision trees. Online learning of decision trees was introduced as an answer to the problems that arise when batch algorithms are used. These problems include the size of the dataset (too big to fit in memory) and the speed of the arrival of the data (data streams).

We discussed and implemented two online learning decision tree algorithms. The first algorithm, called VFDT, built a tree incrementally i.e., learning on every instance as it comes. The attractive property of this algorithm was that at any point of time we had a tree ready for classification. Furthermore, this tree would theoretically produce the same results as a tree that would be learnt in an offline fashion on the data stream. The down side of this algorithm was that it was unable to deal with datasets that have concept drift.

As an answer to this issue we considered another algorithm, an ensemble of trees that tried to learn concepts of all the different distributions present in the data stream. This approach was fairly simple; it tried to keep the concepts that were consistent with the current chunk of testing examples by assigning weights to each tree in the ensemble. Trees that made a small error were given bigger weight and thus kept for further classification. The trees that made small error were the trees that were learn from training examples that had similar or same distribution of training examples.

As any other papers that compare machine learning algorithms, we used 4 different datasets. In two of them concept drift was present, one of them was a well-known benchmark dataset, another was a new dataset collected by a reached, and we also artificially created on dataset on our own.

References

[1] http://stat-computing.org/dataexpo/2009/.

[2] L.-d. Chen, M. L. Gillenson, and D. L. Sherrell. Consumer acceptance of virtual stores: a theoretical model and critical success factors for virtual stores. *ACM SIGMIS Database*, 35(2):831, 2004.

[3] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[4] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *ICML*, pages 194–202, 1995.

[5] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029, 1993.

[6] J. a. Gama and C. Pinto. Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 662–667, New York, NY, USA, 2006. ACM.

[7] J. a. B. Gomes, E. Menasalvas, and P. A. C. Sousa. Calds: context-aware learning from data streams. In *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques*, StreamKDD '10, pages 16–24, New York, NY, USA, 2010. ACM.

[8] C. Gupta and R. L. Grossman. GenIc: A Single-Pass Generalized Incremental Algorithm for Clustering. In *SIAM International Conference on Data Mining*, 2004.

[9] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[10] M. Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6:129–147, 2002.

[11] U. Manber, A. Patel, and J. Robison. Experience with personalization on Yahoo! Comm. 2000.

[12] J. R. Quinlan. Improved use of continuous attributes in c4.5. *J. Artif. Int. Res.*, 4(1):77–90, Mar. 1996.

[13] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press.

[14] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[15] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.

[16] G. Widmer and M. Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23:69–101, 1996.