

A SCALABLE ALGORITHM FOR CLASSIFICATION OF STREAMING DATA

A PROJECT REPORT

Submitted By

BEZZAM VARUN **312212104020**

KIRAN SUDHIR **312212104044**

MAYANKA PACHAIYAPPA 312212104054

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

SSN COLLEGE OF ENGINEERING

KALAVAKKAM 603110

ANNA UNIVERSITY :: CHENNAI - 600025

April 2016

ANNA UNIVERSITY : CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report titled “**A SCALABLE ALGORITHM FOR CLASSIFICATION OF STREAMING DATA**” is the *bonafide* work of “**BEZZAM VARUN (312212104020), KIRAN SUDHIR (312212104044), and MAYANKA PACHAIYAPPA (312212104054)**” who carried out the project work under my supervision.

Dr. CHITRA BABU

Head of the Department

Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Dr. T. T. MIRNALINEE

Supervisor

Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Place:

Date:

Submitted for the examination held on.....

Internal Examiner

External Examiner

ACKNOWLEDGEMENTS

We thank God, the almighty for giving us strength and knowledge to do this project.

We would like to thank and express a deep sense of gratitude to our guide **Dr. T. T. MIRNALINEE**, Professor, Department of Computer Science and Engineering, for her valuable advice and suggestions as well as her continued guidance, patience and support that helped us to shape and refine our work.

Our sincere thanks to **Dr. CHITRA BABU**, Professor and Head of the Department of Computer Science and Engineering, for her words of advice and encouragement and we would like to thank our project Coordinator **Dr. S. SHEERAZUDDIN**, Associate Professor, Department of Computer Science and Engineering for his valuable suggestions throughout the first phase of this project.

We express our deep respect to the founder **Dr. SHIV NADAR**, Chairman, SSN Institutions. We also express our appreciation to our **Dr. S. SALIVAHANAN**, Principal, for all the help he has rendered during this course of study.

We would like to extend our sincere thanks to all the teaching and non-teaching staffs of our department who have contributed directly and indirectly during the course of our project work. Finally, we would like to thank our parents and friends for their patience, cooperation and moral support throughout our lives.

Bezzam Varun

Kiran Sudhir

Mayanka Pachaiyappa

ABSTRACT

A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities. Stream mining is the capability of extracting useful information from these large datasets or streams of data. New mining techniques are required due to the large volume, velocity and variability of such data. In this project, a fast, scalable architecture based on Very Fast Decision Trees (VFDT) and a novel drift detection method is designed for processing stream data. Existing systems that integrate online learning models with drift detection methods suffer from various drawbacks such as the requirement to maintain a variable size sliding window over the data. Another disadvantage is that they assume that a large number of training examples can be stored in memory. Moreover, existing drift detection techniques such as Drift Detection Method (DDM) and Early Drift Detection Method (EDDM) do not perform well in real world applications. Finally, we experimentally compare the proposed system to existing systems using benchmark data, and also present the accuracy of proposed system with numerous data sizes to show scalability.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
1 Introduction	1
1.1 Data Stream Mining	2
1.2 Scalable Algorithms	2
1.3 Concept Drift	3
1.4 Problem Definition	4
1.5 Organisation of the Report	5
2 Literature Survey	6
3 Proposed System	9
3.1 Preprocessing	10
3.1.1 Simple Random Sampling	10
3.1.2 Stratified Random Sampling	11
3.1.3 Reservoir Sampling	12
3.2 Decision Tree Based Model	13
3.2.1 Classification and Regression Trees (CART)	14
3.2.2 Random Forest	15
3.2.3 Very Fast Decision Trees	16

3.3	Model Updation	18
3.3.1	DDM: Drift Detection Method	19
3.3.2	EDDM: Early Drift Detection Method	20
3.3.3	UDDM : Ubiquitous Drift Detection Method	22
3.3.4	Ensembling with UDDM	23
4	Experimental Results and Analysis	25
4.1	Experimental Setup	25
4.2	Implementation Results	26
4.2.1	Sampling analysis	26
4.2.2	Decision tree based analysis	27
4.2.3	Results of drift detection using EDDM	27
4.2.4	Results of drift detection using UDDM	28
4.2.5	Comparison of drift detection methods	29
4.2.6	Scalability of UDDM	29
4.2.7	Analysis of UDDM using different <i>allowed space</i> values .	30
4.2.8	Results of integration of VFDT with UDDM	31
4.3	Screenshots	31
5	Conclusion and Future Work	38

LIST OF TABLES

4.1	Comparison of Sampling strategies	27
4.2	Comparison of Classification strategies	27
4.3	Results of drift detection using EDDM	28
4.4	Results of drift detection using UDDM	28
4.5	Comparison of drift detection methods	29
4.6	Results of UDDM on different dataset sizes	29
4.7	Results of UDDM using different <i>allowed space</i> values	30
4.8	Results of integration of VFDT with UDDM	31

LIST OF FIGURES

3.1	Architectural Diagram of Proposed System	9
3.2	Example of Decision Tree	14
3.3	Hoeffding Tree Algorithm	17
4.1	Random Sampling with 100,000 records	31
4.2	Random Sampling with 300,000 records	32
4.3	Reservoir Sampling with 100,000 records	32
4.4	Reservoir Sampling with 300,000 records	33
4.5	Stratified Sampling with 100,000 records	33
4.6	Stratified Sampling with 300,000 records	34
4.7	Random Forest on SEA dataset	34
4.8	CART on SEA dataset	34
4.9	VFDT on SEA dataset	35
4.10	EDDM	35
4.11	UDDM	36
4.12	Ensembling with UDDM	36
4.13	VFDT integrated with UDDM	37

CHAPTER 1

Introduction

Over the years there has been an exponential increase in the amount of data being generated and stored. One of the key reasons for this is because they are increasingly gathered by cheap and numerous information-sensing mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; [4] as of 2012, every day 2.5 exabytes (2.51018) of data are created. An ever increasing number of organizations worldwide are now focusing on making use of their data to find and understand correlations. This in turn helps them stand out among their competitors and has led to better customer understanding. Apart from just spotting business trends, data mining also finds its use in gaining crucial information in preventing diseases, combating crimes and so on.

In order to retrieve useful information from the data one of the biggest challenges faced is handling the large amount of data available. Moreover, in most cases the data is available in the form of stream which has to be analysed in real time. Thus, one of the key research areas in computer science today is the development of efficient scalable algorithms which can handle the volume and velocity of data. Apart from this, the development of algorithms that can adapt to the changes in data have also become increasingly popular and relevant.

1.1 Data Stream Mining

Data Stream Mining is the process of extracting knowledge structures from continuous, rapid data records. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities. Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data. Data stream mining can be considered a subfield of data mining, machine learning, and knowledge discovery.

In many data stream mining applications, the goal is to predict the class or value of new instances in the data stream given some knowledge about the class membership or values of previous instances in the data stream. Machine learning techniques can be used to learn this prediction task from labeled examples in an automated fashion. Often, concepts from the field of incremental learning, a generalization of Incremental heuristic search are applied to cope with structural changes, on-line learning and real-time demands.

1.2 Scalable Algorithms

Due to the availability of large volumes of data that arrive at high velocity, the development of scalable algorithms that can handle a growing amount of data without adversely affecting accuracy is the need of the hour. Standard Machine Learning techniques do not perform well in the streaming data scenario because of their inability to develop a scalable model that accurately models streaming data. Also, standard Machine Learning algorithms such as Decision Trees and

Random Forests operate within the batch machine learning setting, in which data is collected into batches and a model is trained on the collected data in one pass. An interesting new development in the field of knowledge discovery in data is the development of a class of scalable algorithms known as online learning models. Online machine learning models are a class of machine learning techniques in which data becomes available in a sequential order and is used to update the best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once. Online learning is a common technique used in areas of machine learning where it is computationally infeasible to train over the entire dataset, such as in the streaming data setting. Online learners are also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data. These algorithms can accurately handle the volume, velocity and variability of streaming data.

1.3 Concept Drift

In many applications, especially operating within non-stationary environments, the distribution underlying the instances or the rules underlying their labeling may change over time, i.e. the goal of the prediction, the class to be predicted or the target value to be predicted, may change over time. This problem is referred to as concept drift. The term concept refers to the quantity to be predicted. More generally, it can also refer to other phenomena of interest besides the target concept, such as an input, but, in the context of concept drift, the term commonly refers to the target variable.

For example, the behavior of the customers in an online shop may change over time. For example, let's say the weekly merchandise sales are to be predicted and a predictive model has been developed that works satisfactorily. The model may use inputs such as the amount of money spent on advertising, promotions being run, and other metrics that may affect sales. The model is likely to become less and less accurate over time - this is concept drift. In the merchandise sales application, one reason for concept drift may be seasonality, which means that shopping behavior changes seasonally. Perhaps there will be higher sales in the winter holiday season than during the summer, for example.

1.4 Problem Definition

Since most real world applications today generate data streams on a daily basis, efficient processing of streaming data is the need of the hour. The most common issues that are faced while handling stream data are accuracy, time constraints and storage space restrictions. While the VFDT [1] algorithm provides efficient handling of streaming data, it does not account for concept drift. Concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions become less accurate as time passes. Moreover, concept drift based techniques such as those proposed by Gama et al. [5] do not clearly define how to accurately determine when the error rate increases. However, with an online learning model such as VFDT, even a very small number of training examples can be used to develop the incremental decision tree model, so the error rate can be easily determined. A novel algorithm (UDDM) is developed that integrates

the detection of concept drift within VFDT algorithm to handle the Velocity and Variability of the data. In order to handle the Volume of the data, we include a preprocessing step within the stream that samples the stream data and produces only a small representative stream of records.

1.5 Organisation of the Report

Chapter 2 talks about the existing work on classifier algorithms that use online learning methodologies and on the various drift detection techniques.

Chapter 3 gives a detailed explanation of the proposed system. The various algorithms studied and implemented for each module are explained.

Chapter 4 contains details about the datasets used and the type of system and computing environment used for implementation. Following this, the results of the implementation of all the specified algorithms are included along with screenshots.

Chapter 5 concludes the project with the outcome of our implementation and specifies the scope for further improvement.

CHAPTER 2

Literature Survey

When dealing with non-stationary data streams, the optimal solution is to have data mining systems that operate continuously, constantly processing the data received so that potentially valuable information is never lost. In order to achieve this goal, several methods for extracting patterns from non-stationary streams of data have been developed, all under the general title of online (incremental) learning methods.

The incremental learning methods take into account every new instance that arrives. These algorithms may be irrelevant when dealing with high-speed data streams. Widmer and Kubat [3] have described a series of purely incremental learning algorithms that flexibly react to concept drift and can take advantage of situations where context repeats itself. The series of algorithms is based on a framework called FLORA. FLORA maintains a dynamically adjustable window during the learning process and whenever a concept drift seems to occur (a drop in predictive accuracy) the window shrinks (forgets old instances), and when the concept seems to be stable the window is kept fixed. Otherwise, the window keeps growing until the concept seems to be stable. FLORA is a computationally expensive methodology, since it updates the classification model with every example added to or removed from the training window. Another category of KDD systems attempt to modify batch learning decision tree methods in order to handle data streams.

Previous work on scaling up decision tree learning produced systems such as SLIQ [10], SPRINT [7] and Rainforest [6]. These systems perform batch learning

of decision trees from large data sources in limited memory by performing multiple passes over the data and using external storage. Such operations are not suitable for high speed stream processing. Domingos and Hulten [1] have proposed the VFDT (Very Fast Decision Trees learner) system in order to overcome the memory limitations of the aforementioned systems. The VFDT system is based on a decision tree learning method, which builds the trees based on subsampling of a stationary data stream. While VFDT works well for online learning of streaming data without storing any examples, it does not handle concept drift.

To handle concept drift, Domingos and Hulten [2] proposed an improvement to the VFDT algorithm which is the CVFDT (Concept-adapting Very Fast Decision Tree learner). CVFDT applies the VFDT algorithm to a sliding window of a fixed size and builds the model in an incremental manner instead of building it from scratch whenever a new set of examples arrives. CVFDT increases the computational overload vs. VFDT by growing alternate sub-trees at its internal nodes. The model is modified when the alternate becomes more accurate than the original. While CVFDT is capable of handling abrupt concept drift, it does not accurately handle gradual concept drift. Moreover, it requires maintaining a sliding window over the incoming training examples and growing an alternate decision tree over the newer examples, that is, whenever the window slides. Therefore it requires the tree to be reconstructed a large number of time. This approach depends on the size of the sliding window and also assumes that the sliding window can always fit in memory.

Certain systems that detect concept drift such as the Drift Detection Method (DDM) [5] proposed by Gama et al. to solve the problem of concept drift are

incapable of handling gradual concept drift. Other systems such as On-Line Information Networks (OLIN) [8] based on info-fuzzy networks adapt automatically to concept drift by repeatedly constructing a new model from a sliding window of latest examples. Repeated construction of a new model is computationally expensive and adversely affects performance. Moreover, similar to CVFDT, such systems make assumptions regarding the size of the sliding window and the ability to store the window of training examples in memory.

In conclusion, out of all the above mentioned online learning methods, VFDT works best for streaming data and also eliminates the need to store any examples. However, it does not handle concept drift. While CVFDT builds on VFDT and handles abrupt concept drift, it does not accurately handle gradual concept drift. It also requires the tree to be reconstructed a large number of times (whenever the window slides), and also assumes that the window can always fit in memory. Thus, the VFDT method integrated with an accurate drift detection mechanism would show more promising results than the CVFDT method. This drift detection method should accurately detect both gradual and abrupt concept drift, while maintaining scalability and efficiency.

CHAPTER 3

Proposed System

A scalable algorithm to classify streaming data using a novel method that handles concept drift within the VFDT framework is proposed. The architecture of the system can be seen in Figure 3.1.

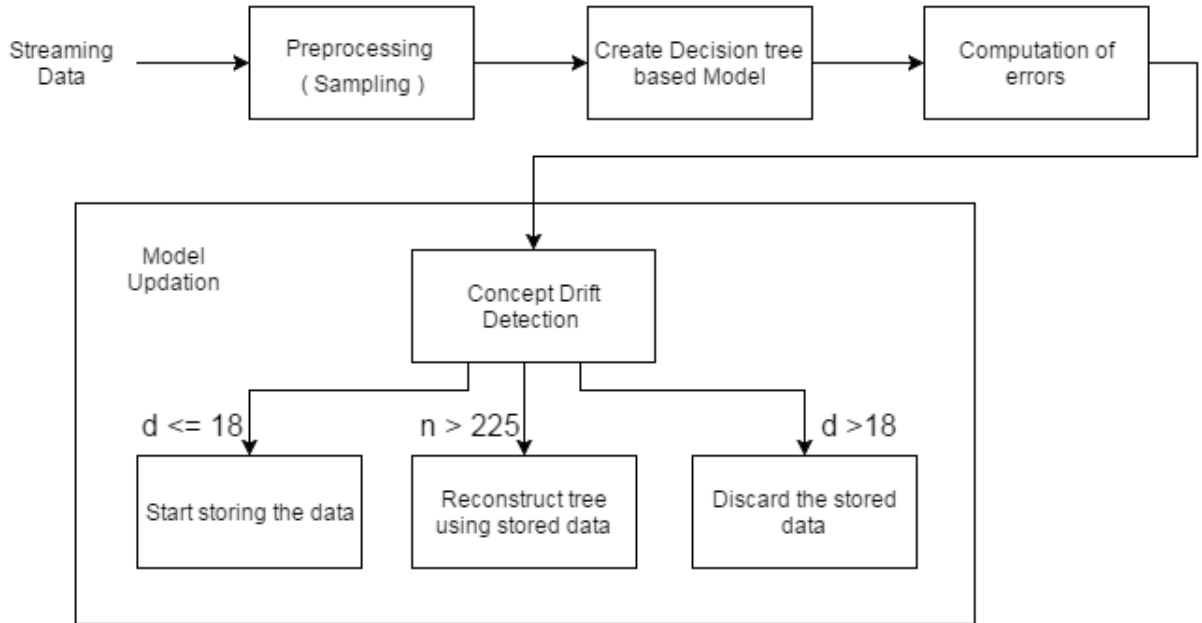


FIGURE 3.1: Architectural Diagram of Proposed System

Different algorithms that were implemented for each module of the architecture are listed below. For each module, we have chosen the most suitable algorithm in terms of scalability, accuracy and storage requirements, and also justified the reasons for choosing that specific algorithm over the others.

3.1 Preprocessing

When we decide to process streaming data, very often we are unable to look at all the individual examples due to the volume and velocity with which they arrive. Instead we choose a sample of training examples from the dataset, which reflects its structure and nature. We want our results to be reliable and dependable, and for those reasons our sample must represent the entire population. Thus, sampling is the process of selecting units from a dataset of interest so that by studying the sample we may fairly generalize our results back to the dataset from which they were chosen. In our architecture, sampling strategies are used to construct representative batches from training data.

Preprocessing in the streaming data scenario is an attractive solution to the problem of processing large volume of high-speed streaming data while still maintaining accuracy comparable to the accuracy obtained using the entire dataset. Including the preprocessing step in the system improves its ability to handle the volume and velocity of the streaming data.

We experiment with three different sampling strategies - simple random sampling, reservoir sampling and stratified sampling - to determine the optimal strategy for sampling stream data. These sampling strategies were selected because they are the most common techniques for accurately representing the entire dataset.

3.1.1 Simple Random Sampling

Simple random sampling is a type of probability sampling where there is an equal chance (probability) of selecting each unit from the population being studied when

creating the sample. Choosing a sample by chance eliminates bias.

This sampling was performed with two sampling sizes and the accuracy of the sampled dataset was found to be low, indicating that the sample does not represent the entire population.

3.1.2 Stratified Random Sampling

Stratified random sampling is a type of probability sampling technique. Unlike the simple random sample, sometimes we are interested in particular strata (meaning groups) within the population (e.g., males vs. females; houses vs. apartments, etc.). Stratification is the process of dividing members of the population into homogeneous subgroups before sampling. The strata should be mutually exclusive: every element in the population must be assigned to only one stratum. The strata should also be collectively exhaustive: no population element can be excluded. Then simple random sampling is applied within each stratum. This often improves the representativeness of the sample by reducing sampling error. With the stratified random sample, there is an equal chance (probability) of selecting each unit from within a particular stratum (group) of the population when creating the sample.

Stratified Random Sampling was also performed with two sampling sizes and the sampled dataset was found to be a close representation of the actual dataset.

3.1.3 Reservoir Sampling

Reservoir sampling is a family of randomized algorithms for randomly choosing a sample of k items from a list S containing n items, where n is very large or an unknown number. Reservoir sampling is particularly well suited to the sampling of streaming data because it ensures that every item has an equal probability of being present in the final sampled dataset.

Suppose we see a sequence of items, one at a time. We want to keep a single item in memory, and we want it to be selected at random from the sequence. If we know the total number of items n , then the solution is easy : select an index i between 1 and n with equal probability, and keep the i^{th} element. The problem is that we do not always know n in advance. A possible solution is the following :

- Keep the first item in memory.
- When the i^{th} item arrives (for $i > 1$) :
 - with probability $\frac{1}{i}$, keep the new item instead of the current item;
 - with probability $1 - \frac{1}{i}$, keep the current item and discard the new item.

So :

- when there is only one item, it is kept with probability 1;
- when there are 2 items, each of them is kept with probability $\frac{1}{2}$;
- when there are 3 items, the third item is kept with probability $\frac{1}{3}$, and each of the previous 2 items is also kept with probability $(\frac{1}{2})(1 - \frac{1}{3}) = (\frac{1}{2})(\frac{2}{3}) = \frac{1}{3}$;

- by induction, it is easy to prove that when there are n items, each item is kept with probability $\frac{1}{n}$.

Reservoir Sampling was also performed with two sampling sizes and the sampled dataset was found to be a very close representation of the actual dataset. Out of the three sampling algorithms, reservoir sampling was determined to be most suitable because apart from generating a very close representation of the actual dataset, it also works well for streaming data.

3.2 Decision Tree Based Model

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. The goal is to create a model that predicts the value of a target variable based on several input variables. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

For this module, three classifier methods have been implemented-CART, Random Forests and VFDT. The classification score was calculated for each of the classifier algorithms.

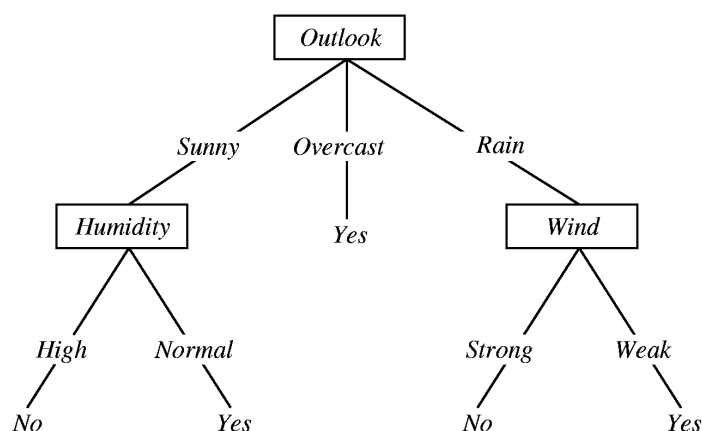


FIGURE 3.2: Example of Decision Tree

3.2.1 Classification and Regression Trees (CART)

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively.

Decision trees are formed by a collection of rules based on variables in the modeling data set :

- Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable.
- Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure)
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. (Alternatively, the data are split as much as possible and then the tree is later pruned.)

3.2.2 Random Forest

Random forest is an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random Forests grows many classification trees. To classify a new object from an input vector, it puts the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree votes for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows :

- If the number of cases in the training set is N , sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
- If there are M input variables, a number $m \ll M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
- Each tree is grown to the largest extent possible. There is no pruning.

The forest error rate depends on two things :

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.

- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

3.2.3 Very Fast Decision Trees

Hoeffding trees were introduced by Domingos and Hulten in the paper Mining High-Speed Data Streams [1]. They refer to their implementation as VFDT, an acronym for Very Fast Decision Tree learner. The key idea depends on the use of Hoeffding bounds. Hoeffding trees are being studied because they represent current state of-the-art for classifying high speed data streams. The algorithm fulfills the requirements necessary for coping with data streams while remaining efficient, an achievement that was rare prior to its introduction. The Hoeffding tree induction algorithm induces a decision tree from a data stream incrementally, briefly inspecting each example in the stream only once, without need for storing examples after they have been used to update the tree. The only information needed in memory is the tree itself. The measure used to split a node is the entropy or information gain, given by

$$entropy(p_1, p_2, \dots, p_n) = \sum_{i=1}^n -p_i \log_2 p_i$$

The Hoeffding bound states that with probability $1 - \delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

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

What makes Hoeffding bound attractive is its ability to give the same results regardless the probability distribution generating the observations. However, the number of observations needed to reach certain values of ϵ and δ are different across probability distributions.

For example, say the information gain difference between two attributes (A and B, where A's information gain is greater than B) is 0.3 and $\epsilon = 0.1$. This means, in the future, the minimum difference between A's info gain and B will be at least $0.3 - 0.1 = 0.2$.

In other words, With probability $1 - \delta$, one attribute is superior compared to others when observed difference of information gain is greater than ϵ .

Algorithm : Hoeffding tree induction algorithm.

```

1: Let  $HT$  be a tree with a single leaf (the root)
2: for all training examples do
3:   Sort example into leaf  $l$  using  $HT$ 
4:   Update sufficient statistics in  $l$ 
5:   Increment  $n_l$ , the number of examples seen at  $l$ 
6:   if  $n_l \bmod n_{min} = 0$  and examples seen at  $l$  not all of same class then
7:     Compute  $\bar{G}_l(X_i)$  for each attribute
8:     Let  $X_a$  be attribute with highest  $\bar{G}_l$ 
9:     Let  $X_b$  be attribute with second-highest  $\bar{G}_l$ 
10:    Compute Hoeffding bound  $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n_l}}$ 
11:    if  $X_a \neq X_b$  and  $(\bar{G}_l(X_a) - \bar{G}_l(X_b)) > \epsilon$  or  $\epsilon < \tau$  then
12:      Replace  $l$  with an internal node that splits on  $X_a$ 
13:      for all branches of the split do
14:        Add a new leaf with initialized sufficient statistics
15:      end for
16:    end if
17:  end if
18: end for

```

FIGURE 3.3: Hoeffding Tree Algorithm

Even though the Decision Tree and Random Forest classifiers have better accuracy, they require the whole dataset to be maintained in memory for training the tree,

whereas VFDT works specifically for data streams as it maintains only the tree and some statistics required, such as the number of training records that reach each node of the decision tree, in memory. Thus VFDT is selected as the decision tree model. It is an online learning model that incrementally uses each training example to update the model. The VFDT algorithm also makes a prediction on the target variable for each training example after training (This is the Train followed by Test approach to evaluate online learning models). This is compared against the given label of the training data to determine whether the example is misclassified, that is, an error or not.

3.3 Model Updation

In the streaming data scenario, a model must satisfy two conditions. First, it should take into account recent history when it makes its predictions. Additionally, it should be updatable, that is, the model needs to in some sense evolve as data streams through its infrastructure. However, even incremental algorithms that update the model for each training instance are unable to detect and handle concept drift, which is why a system which uses an online learning model that is periodically updated is necessary. Concept drift is selected as the factor that determines model updation because it occurs frequently in streaming data and adversely affects the accuracy of the trained model. If the algorithm does not detect concept drift, then the model becomes outdated and makes incorrect predictions. This could adversely affect organisations that heavily rely on such predictions to make crucial business decisions. In this module, the model is updated based on whether concept drift is detected or not.

The examples are considered to arrive one at a time. In online learning approach, the decision model must make a prediction when an example becomes available. Once the prediction has been made, the system can learn from the example (using the attributes and the class) and incorporate it to the learning model. Examples can be represented using pairs (\vec{x}, y) where \vec{x} is the vector of values for different attributes and y is the class label. Thus, i^{th} example will be represented by (\vec{x}, y) . When the current model makes a prediction (y'_i) , it can be correct ($y_i = y'_i$) or not ($y_i \neq y'_i$).

3.3.1 DDM: Drift Detection Method

DDM is an approach that pays attention to the number of errors produced by the learning model during prediction. The drift detection method (DDM) proposed by Gama et al[5]. uses a binomial distribution. That distribution gives the general form of the probability for the random variable that represents the number of errors in a sample of n examples. For each point i in the sequence that is being sampled, the error rate is the probability of missclassifying (p_i), with standard deviation given by $s_i = \sqrt{p_i(1 - p_i)/i}$. They assume that the error rate of the learning algorithm (p_i) will decrease while the number of examples increases if the distribution of the examples is stationary. A significant increase in the error of the algorithm, suggests that the class distribution is changing and, hence, the actual decision model is supposed to be inappropriate. Thus, they store the values of p_i and s_i when $p_i + s_i$ reaches its minimum value during the process (obtaining p_{min} and s_{min}). And it checks when the following conditions triggers:

- $p_i + s_i \geq p_{min} + 2s_{min}$ for the warning level. Beyond this level, the examples are stored in anticipation of a possible change of context.
- $p_i + s_i \geq p_{min} + 3s_{min}$ for the drift level. Beyond this level the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p_{min} and s_{min} are reset too.

This approach has a good behaviour detecting abrupt changes and gradual changes when the gradual change is not very slow, but it has difficulties when the change is slowly gradual. In that case, the examples will be stored for long time, the drift level can take too much time to trigger and the examples memory can be exceeded.

3.3.2 EDDM: Early Drift Detection Method

Early Drift Detection Method (EDDM), was developed to improve the detection in presence of gradual concept drift. At the same time, it keeps a good performance with abrupt concept drift. The basic idea is to consider the distance between two errors classification instead of considering only the number of errors. While the learning method is learning, it will improve the predictions and the distance between two errors will increase. The average distance between two errors (p'_i) and its standard deviation (s'_i) are calculated. What is store are the values of (p'_i) and (s'_i) when $p'_i + 2s'_i$ reaches its maximum value (obtaining p'_{max} and s'_{max}). Thus, the value of $p'_{max} + 2s'_{max}$ corresponds with the point where the distribution of distances between errors is maximum. This point is reached when the model that it is being induced best approximates the current concepts in the dataset.

The method defines two thresholds based on the value of $(p'_i + 2s'_i)/(p'_{max} + 2s'_{max})$, henceforth known as Gama's ratio (GR) :

- $GR < \alpha$ for the drift indication threshold. Beyond this threshold, the examples are stored in advance of a possible change of context.
- $GR < \beta$ for the drift confirmation threshold. Beyond this limit, the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p'_{max} and s'_{max} are reset too.

The method considers the thresholds and searches for a concept drift when a minimum of 30 errors have happened (note that a large amount of examples could appear between 30 classification errors). After 30 classification errors have occurred, the method uses the thresholds to detect when a concept drift happens. The number of classification errors was set to 30 to estimate the distribution of the distances between two consecutive errors and compare it with future distributions in order to find differences. Thus, $p'_{max} + 2s'_{max}$ represents the 95 percent of the distribution. For the experimental section, the values used for α and β were set to 0.95 and 0.90. These values have been determined after some experimentation.

If the similarity between the actual value of $p'_i + 2s'_i$ and the maximum value $(p'_{max} + 2s'_{max})$ increase over the warning threshold, the stored examples are removed and the method returns to normality.

3.3.3 UDDM : Ubiquitous Drift Detection Method

EDDM was developed to improve the detection in presence of gradual concept drift while also maintaining good performance with abrupt concept drift. However, EDDM wrongly detects concept drift if Gama's ratio drops below β for even a single example. Moreover, the accuracy of EDDM is unsatisfactory in the case of gradual concept drift. In this project, we developed a novel drift detection method (UDDM) that accurately detects both abrupt and gradual concept drift with better performance and accuracy than EDDM.

UDDM defines two parameters, the *allowed space* and *number of errors to be seen*. The *allowed space* parameter is defined to be the maximum allowed distance between two consecutive errors and the *number of errors to be seen* parameter is defined to be the number of errors required to have occurred for confirmation of concept drift. With respect to these parameters the three cases can be defined - drift indication, drift confirmation and no drift.

- **Drift Indication** : While the distance between errors d falls less than the *allowed space*, then it is assumed that there is a possibility that the concept has changed, so all training examples from here on are stored temporarily.
- **Drift Confirmation** : If the number of errors n exceeds the *number of errors to be seen*, then a new concept is declared and all the training examples stored are used to train a new VFDT, which replaces the old model. The value of n is reset to zero.

- **No Drift** : If the distance d between two consecutive errors exceeds the *allowed space* parameter, the same old VFDT model is maintained and the stored training examples are discarded. The value of n is reset to zero.

The above approach has not been implemented so far. Moreover, it is a scalable, incremental learning model that automatically adjusts to concept drift based on the structure of the data without making any assumptions on window sizes and other variables. At any point of time, it only stores a single decision tree model (VFDT) and the two online learning measures d and n in memory. Scalability is shown as the accuracy of the algorithm is maintained when the volume of the dataset is increased.

3.3.4 Ensembling with UDDM

In our proposed algorithm, every time concept drift is confirmed, we reconstruct a new tree using the stored training examples and discard the existing tree. We have also tried out a modified version of this algorithm where an ensemble of trees is used.

As in the original algorithm, when a change in concept is confirmed by UDDM, a new decision tree is constructed using the stored training examples. This tree is then added to a collection or ensemble of trees. Thus, the old trees aren't discarded and each new tree is simply added to the ensemble of trees. Every incoming example to be classified is put down each tree in the ensemble. Each tree outputs a classification, i.e. it votes for a particular class. The algorithm finally outputs the class having the most number of votes, i.e. the mode of all the classifications.

This method was found to perform similar to the original method in terms of accuracy. However, this method requires the ensemble of trees to be maintained in memory. Moreover, to make a single classification, each test example has to be provided as input to all the trees in the ensemble, making it both computationally expensive and time consuming. Due to these reasons, we have decided against using an ensemble of trees.

CHAPTER 4

Experimental Results and Analysis

4.1 Experimental Setup

The experiments were conducted on a machine with Intel PIII 3.2-GHZ CPU, 4-GB RAM, and running Windows 8.1. The implementation was carried out in Python 2.7. Three Python libraries were used, namely numpy for multi-dimensional array handling, scikit-learn for various Machine Learning methods such as Decision Trees and Random Forests, and pandas for data manipulation. Additionally, the MOA Java library was used to simulate the high-speed streaming data environment.

Two different datasets were used, namely the PokerHand dataset for the comparison of sampling strategies and classifier methods, and the SEA dataset for the comparison of drift detection methods. The PokerHand consists of 1,000,000 training examples and 30,000 testing examples. Each example belongs to one of the ten classes, numerical value ranging 0 to 9, representing each type of hand in the game of Poker. Every example is defined by eleven numeric attributes, out of which five denote the suit of the five cards, five denote the rank of the five cards and last denotes the class label, which is the "Poker Hand". The suit of the card is a numeric value ranging 1 to 4, where each number represents Hearts, Spades, Diamonds, Clubs respectively, and the rank of the card is also a numeric value, but ranges from 1 to 13 representing (Ace, 2, 3, ... , Queen, King) respectively. The data has no noise.

For drift detection, the data set used is a synthesised data set, SEA, which contains concept drift. It contains 60,000 examples with four concepts. The concept drift occurs after every 15,000 examples. Each example can belong to one of two classes, either class 0 or class 1. Every example is defined by three numeric attributes whose values can range from 0 to 10, out of which only two are relevant. If the sum of the two relevant features is greater than a threshold, then class is 0. The threshold value varies for each concept. Threshold values for each concept are 8,9,7, and 9.5. The data has about ten percent noise.

4.2 Implementation Results

4.2.1 Sampling analysis

The execution time for the optimised implementation of the CART algorithm reduces significantly on performing sampling. Out of the three sampling techniques used, reservoir sampling produced the best results for a sample size of 100,000 records while simple random sampling produced the best results for a larger sample size of 300,000 records. Reservoir sampling was concluded to be the best sampling technique for streaming data because it produced the highest accuracy for 100,000 records and also improved performance by taking the least time to execute.

Sampling Strategy	Size of the dataset	Accuracy	Time to execute
No Sampling	1,000,000	66%	10.96 seconds
Random Sampling	100,000 300,000	38% 60%	0.72 seconds 2.24 seconds
Reservoir Sampling	100,000 300,000	56% 59%	0.6 seconds 2.95 seconds
Stratified Sampling	100,000 300,000	56% 59%	0.74 seconds 2.80 seconds

TABLE 4.1: Comparison of Sampling strategies

4.2.2 Decision tree based analysis

Out of the three decision tree classifiers implemented, RandomForest has the highest accuracy at 71 percent. However, it cannot be extended easily to handle streaming data as it requires storing large amounts of data to construct the tree, in certain cases even the whole dataset. Out of VFDT and the optimised CART, VFDT achieves almost similar accuracy to the CART algorithm. However, VFDT is preferred because it is better suited for streaming data. CART has the same drawback as RandomForest.

Classifier Used	Accuracy
CART	66.0%
Random Forest	71.0%
VFDT	65.2%

TABLE 4.2: Comparison of Classification strategies

4.2.3 Results of drift detection using EDDM

Though the dataset has only four concepts, that is, it has only three concept drifts, EDDM incorrectly detects multiple concept drifts. This is because EDDM considers even noise as a change in concept and reconstructs the tree using just

that single noisy training example. Table 4.3 shows how multiple concept drifts are detected between data records 15,000 and 30,000 even though there is no actual change in concept in that range.

Actual Drift Index	Drift Detection Index	Drift Confirmation Index
15000	18226	25360
15000	25499	25499
15000	25577	25577
15000	25630	25630
15000	25872	25872
30000	31023	31023
30000	31089	31089

TABLE 4.3: Results of drift detection using EDDM

4.2.4 Results of drift detection using UDDM

UDDM detects concept drift much more accurately in comparison to EDDM. The index when drift is detected is close to the actual location of concept drift and the drift is confirmed soon after. This method is a lot less vulnerable to noise, and thus does not provide incorrect results. This shows the proposed system is promising.

Table 4.4 shows the results obtained on implementation.

Actual Drift Index	Drift Detection Index	Drift Confirmation Index
15000	15475	16375
30000	30390	31100
45000	45020	45673

TABLE 4.4: Results of drift detection using UDDM

4.2.5 Comparison of drift detection methods

EDDM was found to have very poor accuracy in comparison to our Drift Detection method. The same drift detection method was also used to generate an ensemble of trees which could be used for classification. The ensemble method was found to have very close accuracy to the drift detection method. The drift detection method was concluded to have the best accuracy of 81.2 percent.

Drift Detection Method	Classification Accuracy
EDDM	54.4%
UDDM	81.2%
Ensembling using UDDM	80.5%

TABLE 4.5: Comparison of drift detection methods

4.2.6 Scalability of UDDM

The efficiency of a method should not be affected by the volume of the dataset. The accuracy of the proposed method was checked for varying sizes of the dataset in order to check scalability. The algorithm was able to maintain its classification accuracy with the increase in the size of data. Table 4.6 shows the result when the system was implemented with datasets of sizes 60,000, 120,000 and 240,000.

Number of instances	Classification Accuracy
60000	80.57%
120000	80.09%
240000	80.02%

TABLE 4.6: Results of UDDM on different dataset sizes

4.2.7 Analysis of UDDM using different *allowed space* values

The allowed space value was determined experimentally as 18 by the method of trial and error for this dataset. This value may vary depending on the dataset.

Allowed Space	Actual Drift Index	Drift Detect Index	Drift Confirm Index
16	15000	16122	16964
	30000	31139	31885
	45000	44923	45725
		45850	46770
17	15000	16122	16992
	30000	30006	30770
	45000	44893	45670
18	15000	15475	16375
	30000	30390	31100
	45000	45001	45673
19	15000	14601	15573
		15573	16565
		17608	18711
	30000	30553	31295
	45000	44999	45785
20	15000	14596	15601
		15601	16598
		22063	23205
		23502	24524
	30000	30012	30716
		37783	38929
	45000	45095	45843
		45843	46866
		49307	50528
		56300	57462

TABLE 4.7: Results of UDDM using different *allowed space* values

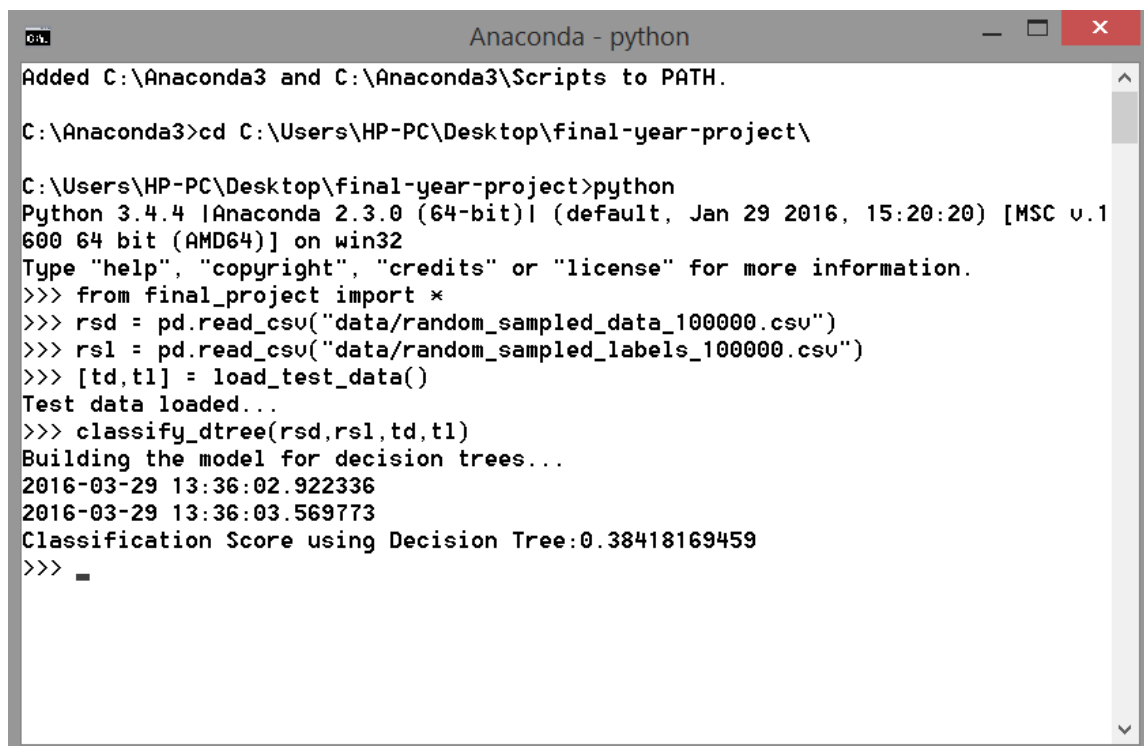
4.2.8 Results of integration of VFDT with UDDM

The VFDT method alone showed a satisfactory accuracy level at 84 percent. On integrating our drift detection method with VFDT the accuracy increased to 87 percent. This has been depicted in Table 4.8.

Algorithm	Accuracy	Time
VFDT without drift detection method	84.69%	2.82 seconds
VFDT with UDDM	86.71%	4.40 seconds

TABLE 4.8: Results of integration of VFDT with UDDM

4.3 Screenshots



```

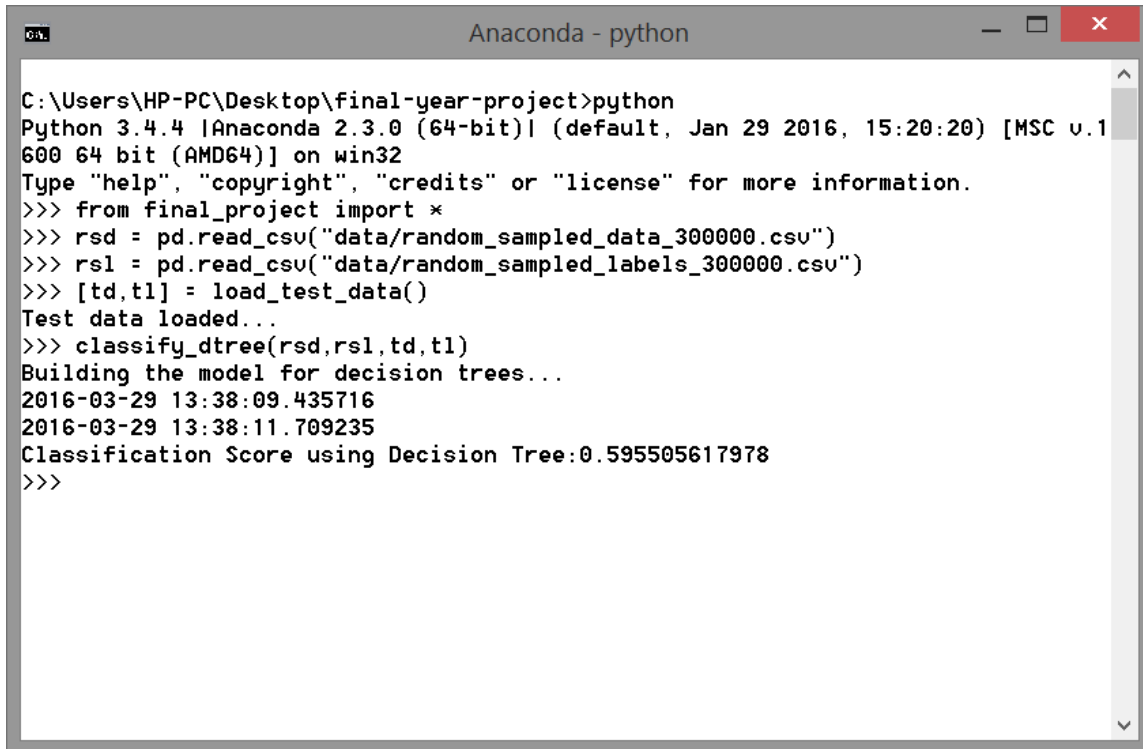
Anaconda - python
Added C:\Anaconda3 and C:\Anaconda3\Scripts to PATH.

C:\Anaconda3>cd C:\Users\HP-PC\Desktop\final-year-project\

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> rsd = pd.read_csv("data/random_sampled_data_100000.csv")
>>> rsl = pd.read_csv("data/random_sampled_labels_100000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(rsd,rsl,td,t1)
Building the model for decision trees...
2016-03-29 13:36:02.922336
2016-03-29 13:36:03.569773
Classification Score using Decision Tree:0.38418169459
>>> _

```

FIGURE 4.1: Random Sampling with 100,000 records



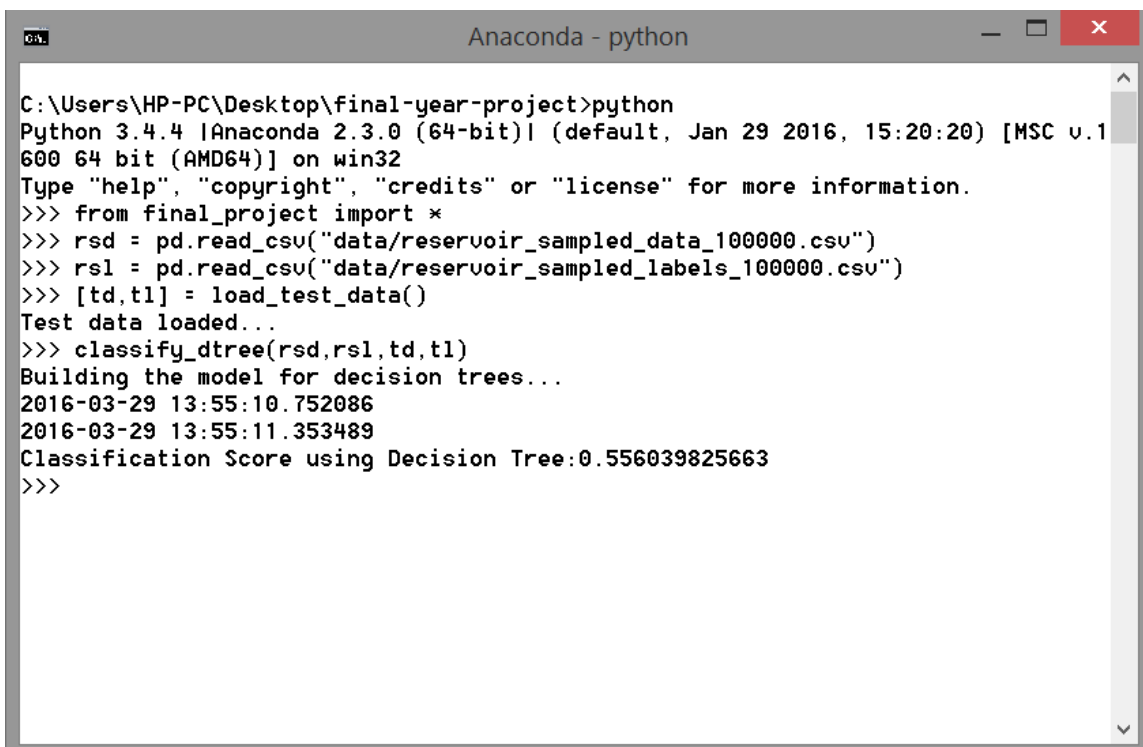
```

Anaconda - python

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> rsd = pd.read_csv("data/random_sampled_data_300000.csv")
>>> rsl = pd.read_csv("data/random_sampled_labels_300000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(rsd,rsl,td,t1)
Building the model for decision trees...
2016-03-29 13:38:09.435716
2016-03-29 13:38:11.709235
Classification Score using Decision Tree:0.595505617978
>>>

```

FIGURE 4.2: Random Sampling with 300,000 records



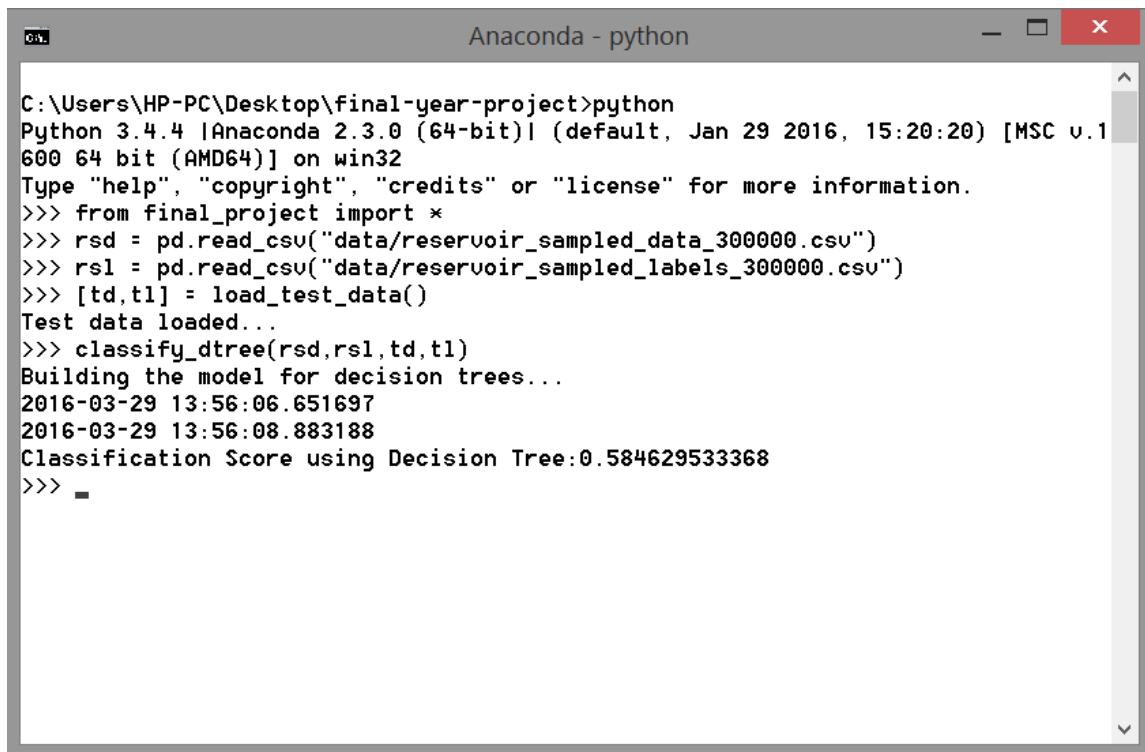
```

Anaconda - python

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> rsd = pd.read_csv("data/reservoir_sampled_data_100000.csv")
>>> rsl = pd.read_csv("data/reservoir_sampled_labels_100000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(rsd,rsl,td,t1)
Building the model for decision trees...
2016-03-29 13:55:10.752086
2016-03-29 13:55:11.353489
Classification Score using Decision Tree:0.556039825663
>>>

```

FIGURE 4.3: Reservoir Sampling with 100,000 records

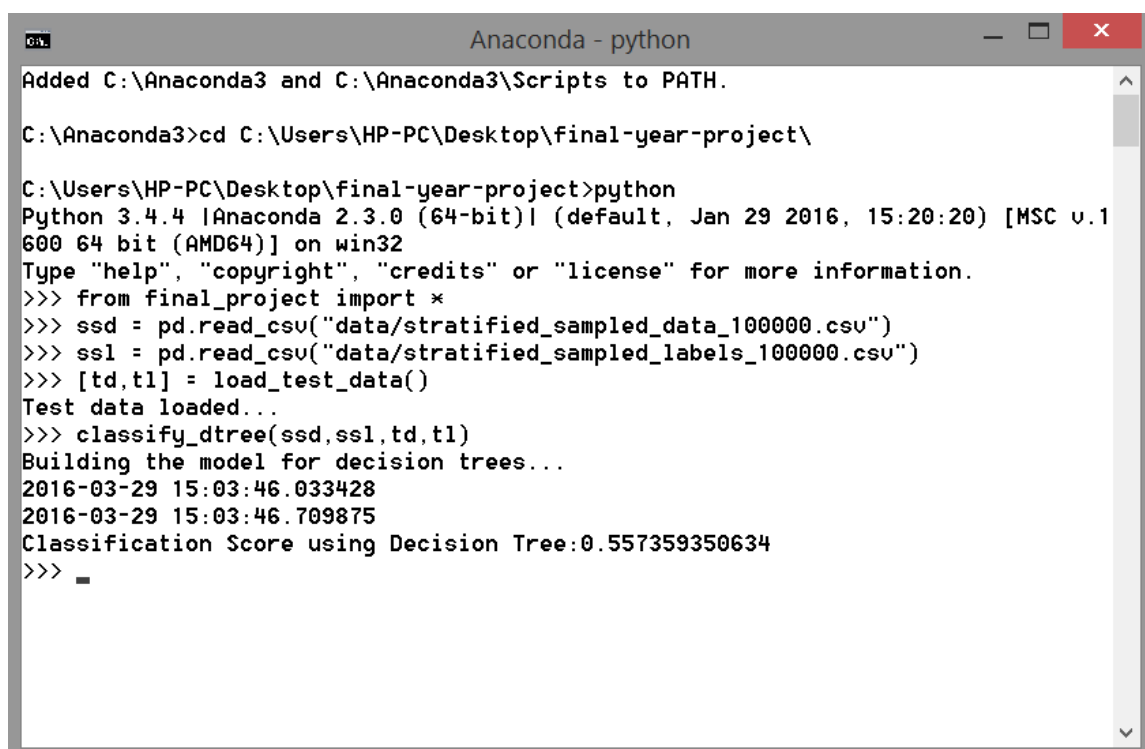


```

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> rsd = pd.read_csv("data/reservoir_sampled_data_300000.csv")
>>> rsl = pd.read_csv("data/reservoir_sampled_labels_300000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(rsd,rsl,td,t1)
Building the model for decision trees...
2016-03-29 13:56:06.651697
2016-03-29 13:56:08.883188
Classification Score using Decision Tree:0.584629533368
>>>

```

FIGURE 4.4: Reservoir Sampling with 300,000 records



```

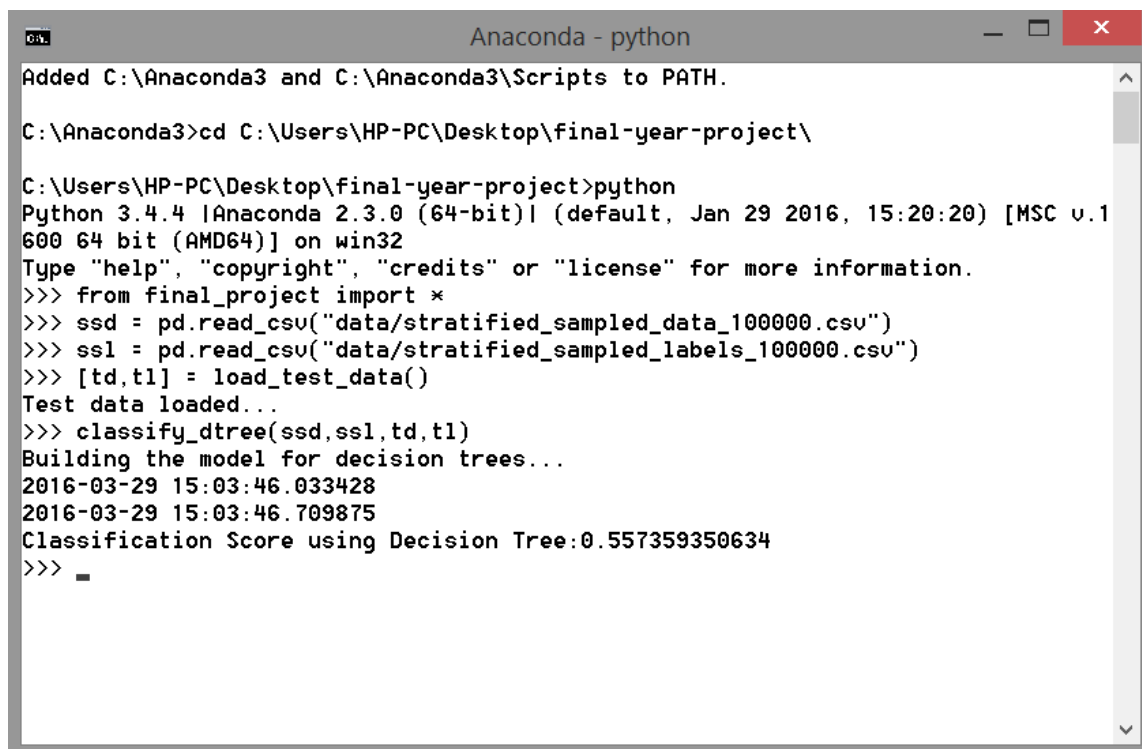
Added C:\Anaconda3 and C:\Anaconda3\Scripts to PATH.

C:\Anaconda3>cd C:\Users\HP-PC\Desktop\final-year-project\

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> ssd = pd.read_csv("data/stratified_sampled_data_100000.csv")
>>> ssl = pd.read_csv("data/stratified_sampled_labels_100000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(ssd,ssl,td,t1)
Building the model for decision trees...
2016-03-29 15:03:46.033428
2016-03-29 15:03:46.709875
Classification Score using Decision Tree:0.557359350634
>>>

```

FIGURE 4.5: Stratified Sampling with 100,000 records



```

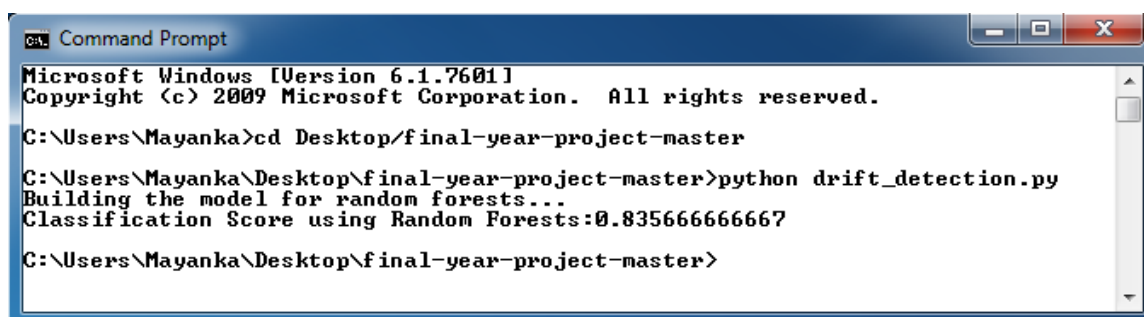
Anaconda - python
Added C:\Anaconda3 and C:\Anaconda3\Scripts to PATH.

C:\Anaconda3>cd C:\Users\HP-PC\Desktop\final-year-project\

C:\Users\HP-PC\Desktop\final-year-project>python
Python 3.4.4 |Anaconda 2.3.0 (64-bit)| (default, Jan 29 2016, 15:20:20) [MSC v.1
600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from final_project import *
>>> ssd = pd.read_csv("data/stratified_sampled_data_100000.csv")
>>> ssl = pd.read_csv("data/stratified_sampled_labels_100000.csv")
>>> [td,t1] = load_test_data()
Test data loaded...
>>> classify_dtree(ssd,ssl,td,t1)
Building the model for decision trees...
2016-03-29 15:03:46.033428
2016-03-29 15:03:46.709875
Classification Score using Decision Tree:0.557359350634
>>> _

```

FIGURE 4.6: Stratified Sampling with 300,000 records



```

Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

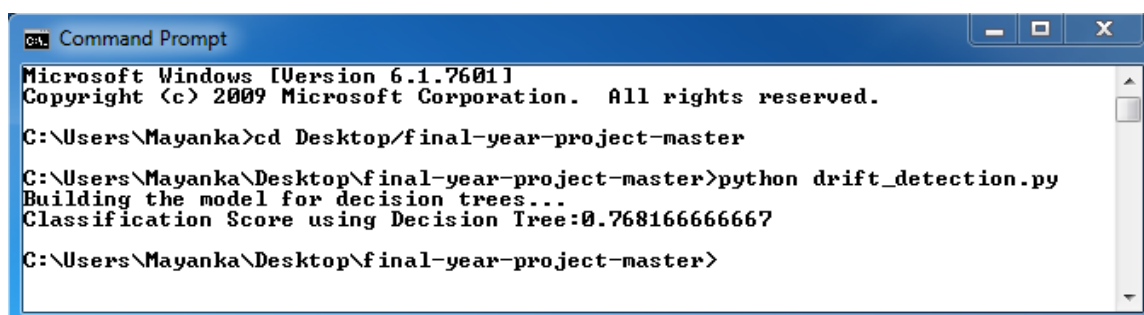
C:\Users\Mayanka>cd Desktop/final-year-project-master

C:\Users\Mayanka\Desktop\final-year-project-master>python drift_detection.py
Building the model for random forests...
Classification Score using Random Forests:0.835666666667

C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.7: Random Forest on SEA dataset



```

Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

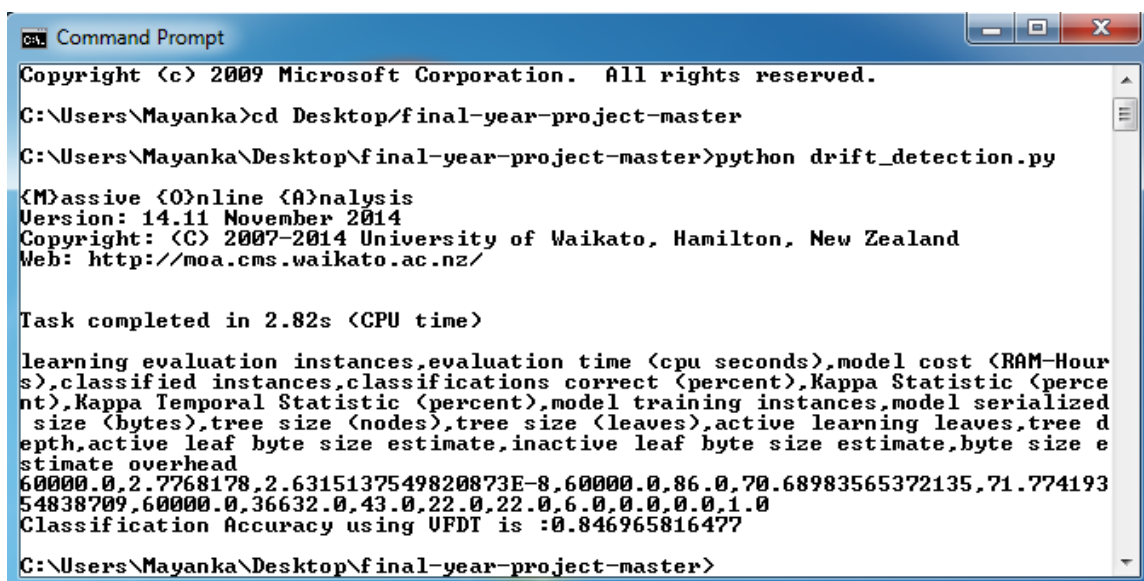
C:\Users\Mayanka>cd Desktop/final-year-project-master

C:\Users\Mayanka\Desktop\final-year-project-master>python drift_detection.py
Building the model for decision trees...
Classification Score using Decision Tree:0.768166666667

C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.8: CART on SEA dataset



```

Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Mayanka>cd Desktop/final-year-project-master
C:\Users\Mayanka\Desktop\final-year-project-master>python drift_detection.py

(M)assive (O)nline (A)nalysis
Version: 14.11 November 2014
Copyright: (C) 2007-2014 University of Waikato, Hamilton, New Zealand
Web: http://moa.cms.waikato.ac.nz/

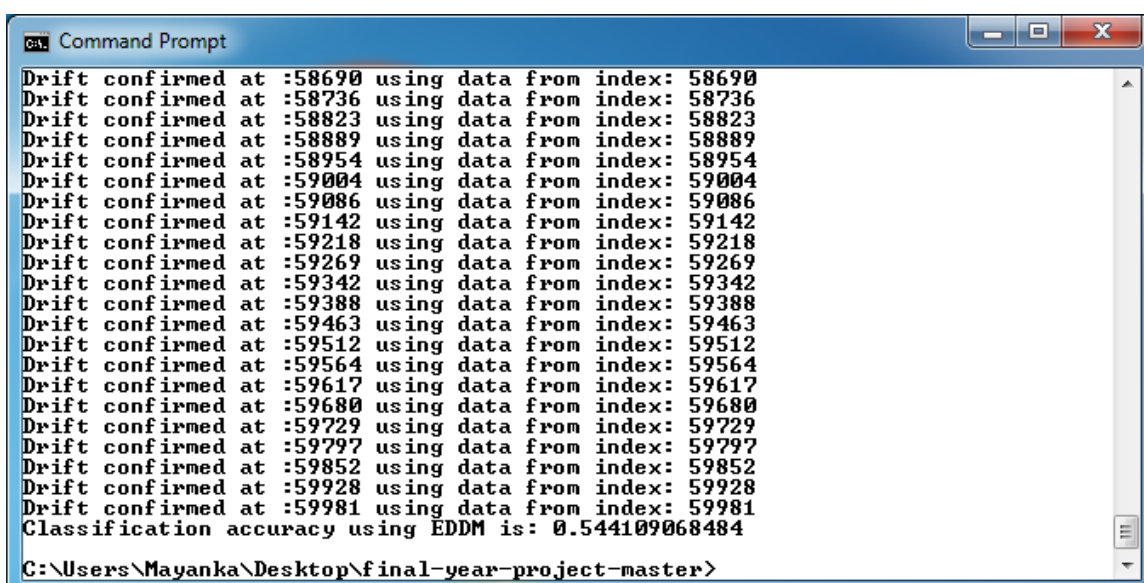
Task completed in 2.82s (CPU time)

learning evaluation instances,evaluation time (cpu seconds),model cost (RAM-Hour
s),classified instances,classifications correct (percent),Kappa Statistic (perce
nt),Kappa Temporal Statistic (percent),model training instances,model serialized
size (bytes),tree size (nodes),tree size (leaves),active learning leaves,tree d
epth,active leaf byte size estimate,inactive leaf byte size estimate,byte size e
stimate overhead
60000.0,2.7768178,2.6315137549820873E-8,60000.0,86.0,70.68983565372135,71.774193
54838709,60000.0,36632.0,43.0,22.0,22.0,6.0,0.0,0.0,1.0
Classification Accuracy using VFDT is :0.846965816477

C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.9: VFDT on SEA dataset



```

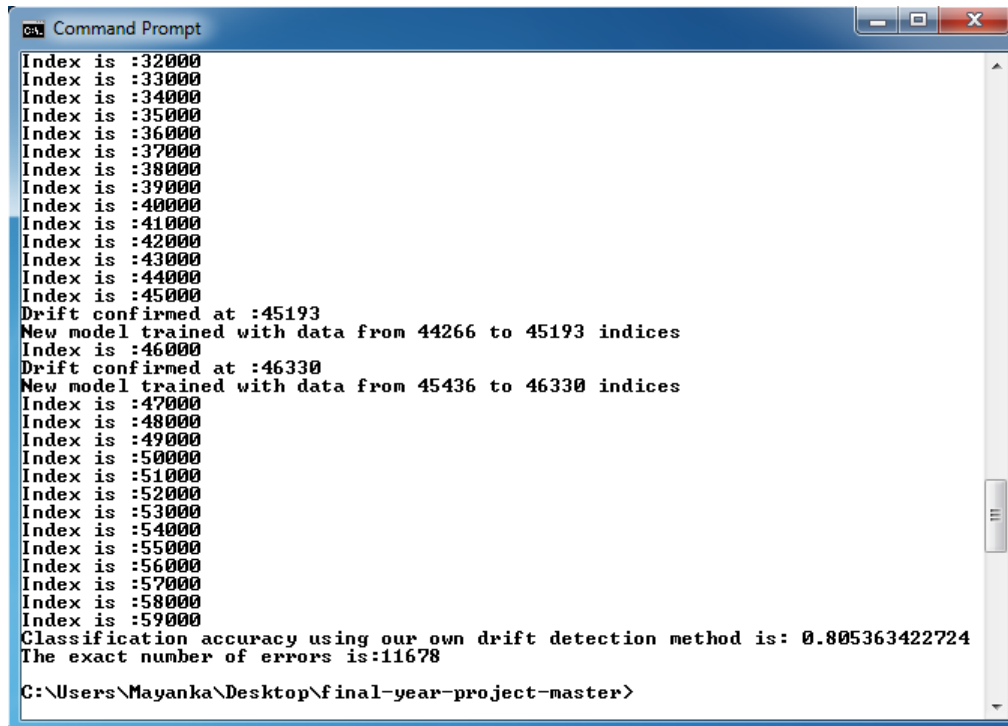
Command Prompt

Drift confirmed at :58690 using data from index: 58690
Drift confirmed at :58736 using data from index: 58736
Drift confirmed at :58823 using data from index: 58823
Drift confirmed at :58889 using data from index: 58889
Drift confirmed at :58954 using data from index: 58954
Drift confirmed at :59004 using data from index: 59004
Drift confirmed at :59086 using data from index: 59086
Drift confirmed at :59142 using data from index: 59142
Drift confirmed at :59218 using data from index: 59218
Drift confirmed at :59269 using data from index: 59269
Drift confirmed at :59342 using data from index: 59342
Drift confirmed at :59388 using data from index: 59388
Drift confirmed at :59463 using data from index: 59463
Drift confirmed at :59512 using data from index: 59512
Drift confirmed at :59564 using data from index: 59564
Drift confirmed at :59617 using data from index: 59617
Drift confirmed at :59680 using data from index: 59680
Drift confirmed at :59729 using data from index: 59729
Drift confirmed at :59797 using data from index: 59797
Drift confirmed at :59852 using data from index: 59852
Drift confirmed at :59928 using data from index: 59928
Drift confirmed at :59981 using data from index: 59981
Classification accuracy using EDDM is: 0.544109068484

C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.10: EDDM

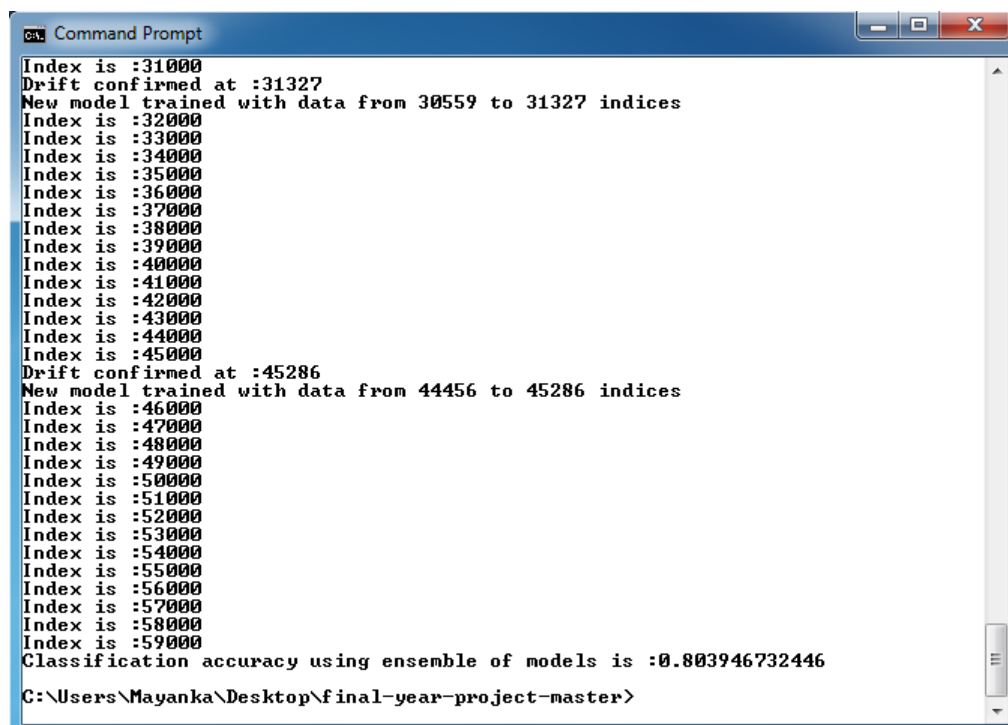


```

C:\> Command Prompt
Index is :32000
Index is :33000
Index is :34000
Index is :35000
Index is :36000
Index is :37000
Index is :38000
Index is :39000
Index is :40000
Index is :41000
Index is :42000
Index is :43000
Index is :44000
Index is :45000
Drift confirmed at :45193
New model trained with data from 44266 to 45193 indices
Index is :46000
Drift confirmed at :46330
New model trained with data from 45436 to 46330 indices
Index is :47000
Index is :48000
Index is :49000
Index is :50000
Index is :51000
Index is :52000
Index is :53000
Index is :54000
Index is :55000
Index is :56000
Index is :57000
Index is :58000
Index is :59000
Classification accuracy using our own drift detection method is: 0.805363422724
The exact number of errors is:11678
C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.11: UDDM

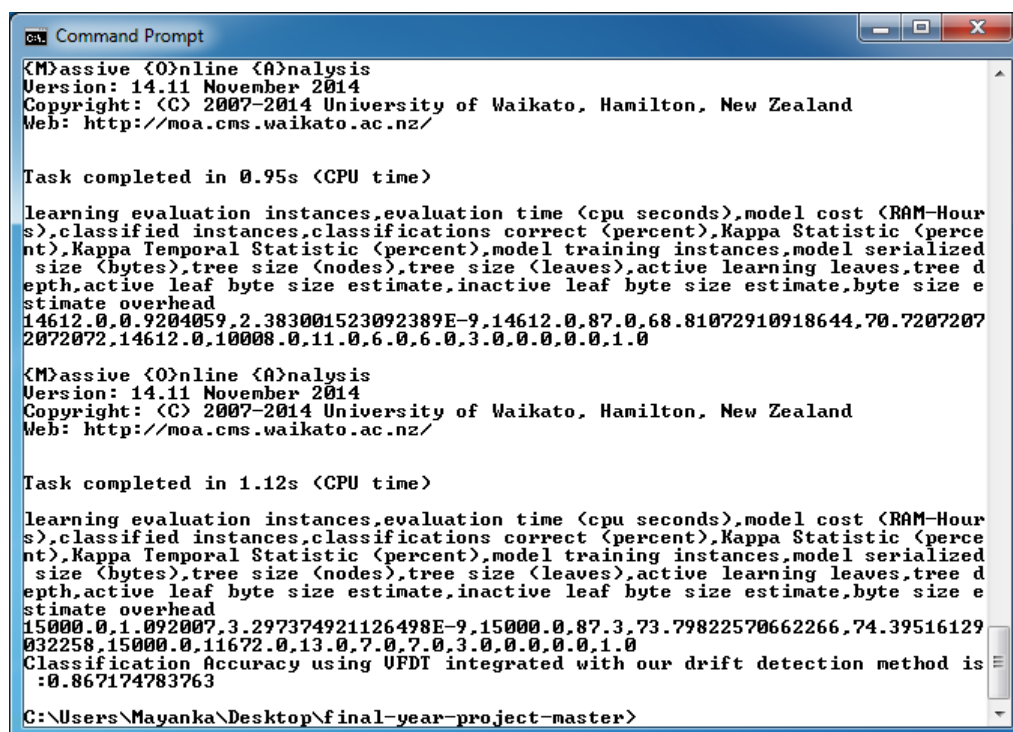


```

C:\> Command Prompt
Index is :31000
Drift confirmed at :31327
New model trained with data from 30559 to 31327 indices
Index is :32000
Index is :33000
Index is :34000
Index is :35000
Index is :36000
Index is :37000
Index is :38000
Index is :39000
Index is :40000
Index is :41000
Index is :42000
Index is :43000
Index is :44000
Index is :45000
Drift confirmed at :45286
New model trained with data from 44456 to 45286 indices
Index is :46000
Index is :47000
Index is :48000
Index is :49000
Index is :50000
Index is :51000
Index is :52000
Index is :53000
Index is :54000
Index is :55000
Index is :56000
Index is :57000
Index is :58000
Index is :59000
Classification accuracy using ensemble of models is :0.803946732446
C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.12: Ensembling with UDDM



```

C:\>assive <O>online <A>analysis
Version: 14.11 November 2014
Copyright: <C> 2007-2014 University of Waikato, Hamilton, New Zealand
Web: http://moa.cms.waikato.ac.nz/

Task completed in 0.95s <CPU time>

learning evaluation instances,evaluation time <cpu seconds>,model cost <RAM-Hour
s>,classified instances,classifications correct <percent>,Kappa Statistic <perce
nt>,Kappa Temporal Statistic <percent>,model training instances,model serialized
size <bytes>,tree size <nodes>,tree size <leaves>,active learning leaves,tree d
epth,active leaf byte size estimate,inactive leaf byte size estimate,byte size e
stimate overhead
14612.0,0.9204059,2.383001523092389E-9,14612.0,87.0,68.81072910918644,70.7207207
2072072,14612.0,10000.0,11.0,6.0,6.0,3.0,0.0,0.0,1.0

C:\>assive <O>online <A>analysis
Version: 14.11 November 2014
Copyright: <C> 2007-2014 University of Waikato, Hamilton, New Zealand
Web: http://moa.cms.waikato.ac.nz/

Task completed in 1.12s <CPU time>

learning evaluation instances,evaluation time <cpu seconds>,model cost <RAM-Hour
s>,classified instances,classifications correct <percent>,Kappa Statistic <perce
nt>,Kappa Temporal Statistic <percent>,model training instances,model serialized
size <bytes>,tree size <nodes>,tree size <leaves>,active learning leaves,tree d
epth,active leaf byte size estimate,inactive leaf byte size estimate,byte size e
stimate overhead
15000.0,1.092007,3.297374921126498E-9,15000.0,87.3,73.79822570662266,74.39516129
032258,15000.0,11672.0,13.0,7.0,7.0,3.0,0.0,0.0,1.0
Classification Accuracy using VFDT integrated with our drift detection method is
:0.867174783763

C:\Users\Mayanka\Desktop\final-year-project-master>

```

FIGURE 4.13: VFDT integrated with UDDM

CHAPTER 5

Conclusion and Future Work

In conclusion, this algorithm deals with classification of streaming data which not only handles the volume and velocity of the data, but also accurately detects concept drift. This is achieved using reservoir sampling, very fast decision trees and a novel drift detection method (UDDM). The efficiency is analysed by comparing with other existing techniques and shows promising results. The scalability of the algorithm is also confirmed using datasets of different sizes to show increase in volume of the arriving data and is shown in Table 4.6.

As part of our future work we intend to obtain a mathematical model to determine the allowed space for distance between two errors (d) and the number of errors to occur for a drift to be considered (n). These values were determined by trial and error for the dataset used. This may vary for each dataset and the method will be more generalised when determined mathematically by parameters defining the dataset. Furthermore, we aim to come up with a more optimized and efficient decision tree algorithm in place of VFDT which not only requires less data storage but also provides promising accuracy.

REFERENCES

1. Domingos, P., Hulten, G. (2000) 'Mining high-speed data streams', In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp.71-80.
2. G. Hulten, L. Spencer, and P. Domingos (2001) 'Mining time-changing data streams', In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp.97-106.
3. G. Widmer and M. Kubat (1996) 'Learning in the Presence of Concept Drift and Hidden Contexts', Machine Learning, Kluwer Academic Publishers, pp. 69-101.
4. Hilbert, Martin; Lpez, Priscila (2011) 'The World's Technological Capacity to Store, Communicate, and Compute Information', Science 332(6025), pp.60-65.
5. J. Gama , P. Medas, G. Castillo and P. Rodrigues (2004) 'Learning with drift detection', In Advances in artificial intelligenceSBIA 2004, Springer Berlin Heidelberg, pp.286-295.
6. J. Gehrke, R. Ramakrishnan and V. Ganti (1998) 'Rainforest-A framework for fast decision tree construction of large datasets', VLDB Vol.98, pp.416-427.
7. J. Shafer, R. Agrawal, and M. Mehta (1996) 'SPRINT: A scalable parallel classier for data mining', In Proceedings of Int. Conf. Very Large Data Bases, pp.544-555.
8. L. Cohen, G. Avrahami, M. Last, and A. Kandel, (2008) 'Info-fuzzy algorithms for mining dynamic data streams', Applied Soft Computing 8.4, pp.1283-1294.

9. M. Baena-Garcia Jose, J. Del Campo-vila, R. Fidalgo , A. Bifet, R. Gavald and R. Morales-bueno (2006) 'Early drift detection method', In Fourth international workshop on knowledge discovery from data streams Vol.6, pp.77-86.
10. M. Mehta, R. Agrawal, and J. Rissanen (1996) 'SLIQ: A fast scalable classifier for data mining', In Proceedings of Advances in Database Technology, Springer Berlin Heidelberg, pp.18-32.