UNIVERSITY OF CINCINNATI

Date: 11/26/2007

I, Sasthakumar Ramamurthy **,**

hereby submit this work as part of the requirements for the degree of:

Master of Science

in:

Computer Science

It is entitled:

Tracking Recurrent Concept Drift in Streaming data using Ensemble Classifiers

This work and its defense approved by:

Chair: Dr. Raj Bhatnagar

Dr. Anca Ralescu

Dr. Ali Minai

Tracking Recurrent Concept Drift in Streaming data using Ensemble Classifiers

by

Sasthakumar Rammaurthy

Bachelor of Engineering Electrical and Electronics

Bharathiar University, Coimbatore

A Masters thesis submitted to the faculty of

University of Cincinnati

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

University of Cincinnati

November 2007

ABSTRACT

Streaming data may consist of multiple drifting concepts each having its own underlying data distribution. We present an ensemble learning based approach to handle the data streams having multiple underlying modes. We build a *global* set of classifiers from sequential data chunks; ensembles are then selected from this *global set* of classifiers, and new classifiers created if needed, to represent the current concept in the stream. The system is capable of performing any-time classification and to detect concept drift in the stream. In streaming data historic concepts are likely to reappear so we dont delete any of the historic classifiers. Instead, we judiciously select only pertinent classifiers from the *global* set while forming the ensemble set for a classification task.

ACKNOWLEDGMENTS

I express my heartfelt gratitude to Dr.Raj Bhatnagar, my advisor, for being a constant source of motivation and guiding me through this thesis. I would also like to thank Dr. Anca Ralescu and Dr.Ali Minai for their presence on my thesis committee and providing guidance to improve my work.

I thank my parents and my brother Senthil kumar Ramamurthy for their unconditional love, support and motivation. I would like to thank my lab mates Abhishek Sharma, Amit Sinha, Giridhar Tatavarty, Kalyan Shencottah and Shriram Narayanaswamy for providing valuable suggestions and a wonderful environment to work. I would also like to thank my friends Arun Prasath, Thiagarajan Arumugam and Srivathsan Ranganathan for their motivation and making my stay at University of Cincinnati a memorable one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The oft-quoted example of what data mining can achieve is the case of a large US supermarket chain which discovered a strong association for many customers between a brand of babies nappies (diapers) and a brand of beer. Most customers who bought the nappies also bought the beer. The best hypothesisers in the world would find it difficult to propose this combination but data mining showed it existed, and the retail outlet was able to exploit it by moving the products closer together on the shelves."

in-The Financial Times of London (Feb. 7, 1996)

1.1 Introduction

Data mining is the process of extracting hidden and useful information from large data repositories. The term data repository may refer to flat files, databases or data streams. When retrieving information from a database, a pre-determined query is issued to the database. But in the case of data mining, we often do not know what we are looking for and hence, a query cannot be pre-determined. In the

corporate world this information can be used for customer profiling, targeted marketing or to design store layout. In order to achieve this complex task, techniques from statistics, artificial intelligence, machine learning and pattern recognition are used to design data mining algorithms.

In this thesis we study mining streaming data. Data mining algorithms can be a single pass or multi pass. In a single pass algorithm, the algorithm gets only one chance to pass through the data. But in the multi pass, data is scanned many times. Hence multi pass algorithms are more time consuming then single pass. In data streams, the rate at which data flows in is very high. Volume of the data flowing in is also very large to the extent that all the data cannot be buffered. Owing to these constraints, a stream data mining algorithm should parse the incoming data with as less time as possible. Since multi scan algorithm are time consuming, they are not applicable for stream data mining. The primary challenge in mining streaming data is the fact that algorithm has to be a single pass. Our objective is to design a one pass algorithm that will remember historic patterns.

1.2 Motivation

Streaming data environments are characterized by huge volumes of data flowing through a computer system. Examples of such streaming data include telephone call records, weather sensor data, credit card transaction data, surveillance video streams, data sent by network routers, etc. Typically, we don't have storage to retain all the data and we must learn its important characteristics and use them in future. In this environment, the data mining algorithm gets only one chance to look at the data. Necessary information has to be retrieved from the data and

the data as such has to be moved to a secondary storage due to space constraint.

Machine learning approaches assume a static underlying data distribution but this does not hold in streaming environments where data may span months and years and the generating sources may undergo periodic changes. For example, a customer's purchasing practices can change due to a seasonal factor like weather, economic factor like inflation or trend in the market.In general, the generating sources may drift from one mode of operation to another. This type of change in a system's operating mode is known as *concept drift*. If there is a concept drift in the data and a fixed classification system continues to do classification then this system is bound to perform erroneously. So it is very important for the classification system to trace this concept drift and evolve to learn, and also, use the new concept.

In many streaming environments historical concepts can reoccur. Information filtering techniques used to learn the news reading preferences of users [18] are an example. News reading preference of a user may change with time. A user can have different choices for mornings, evenings, weekdays, and weekends. In addition a user might surf astrology articles in the beginning of the year and financial articles at the beginning of each quarter. In general, different concepts can occur due to cyclic influences like seasons of a year or non-cyclic factors like inflation or market trend [10]. These distinct conditions can be termed as different *modes of operation* giving rise to different concepts embedded in the data streams.

Many concepts in data streams are likely to reoccur. If a classification system stores only the current mode's concept, the system has to relearn every time a new concept occurs. This significantly affects the performance of the system. Ideally, a classification system for stream data mining should be capable of 1) learning in one pass, 2) do any-time classification, 3) track the drift in the data over

time 4) remember historically learned concepts and 5) apply the best available classification scheme for the incoming data.

1.3 Contribution

We present an ensemble learning based approach to handle the data streams having multiple underlying modes. We build a *global* set of classifiers from sequential data chunks. To represent a current concept in the stream we select an ensemble set from this *global* set of classifiers. If the current set of classifiers are unable to adequately classify the current concept, a new classifier is created and added to the *global set*. This system is capable of performing any-time classification and to detect concept drift in the stream.

1.4 Organization of the thesis

Chapter 1 gives a brief introduction of data mining and stream data mining. It also describes the motivation of this thesis work. Chapter 2 discusses the relevant work in stream data mining. Chapter 3 describes our methodology to address the recurrent concepts in stream data mining. Chapter 4 discusses the experimental results obtained by applying our algorithm on the *Nursery* and *Car* data set derived from UCI repository [21]. Chapter 5 discusses conclusion, limitation and future work.

Chapter 2

Related Research

Since all data are not available simultaneously in a streaming environment, an incremental learning approach is used. Many researchers use a a decision tree based classification system for addressing stream data mining.

2.1 Decision Tree

Decision tress is a supervised learning method. Given a training set, a decision tree is constructed in which leafs carry a class label. Interior nodes in a decision tree are called decision nodes. A decision node is assigned one of the input attributes as its value. A decision node has as many children as there are possible values for the input attribute and each child is assigned one of the possible values. A decision node splits the incoming data to one of its children depending on the value of node attribute. Leaf nodes are labeled as one of the possible classes. If an input data point reaches a leaf marked with a class label C_a, the data point is labeled as Class C_a

ID-3 [22] decision tree learning system is a top-down strategy and it searches

only a part of search space. In this top-down approach a decision tree is built starting from the root. ID-3 is based on *information theory*. The attribute selection at a node in ID-3 depends on the amount of information each attribute provides towards distinguishing the data points. Information provided by each attribute at a node is evaluated using *information theory*:

$$\mathrm{i} = \sum (p_i ln(p_i))$$

where p_i is the probability that the data point belongs to class i. When this information is subtracted from the information provided by the data point, remainder is obtained. Remainder is an indication of how much more information is required to classify the data point accurately. So to minimize the remainder an input attribute which gives maximum information is selected at a node. One problem in this approach is for continuous attributes. Consider a numerical attribute *age* having possible values from 1-100. So if a decision node uses this attribute as splitting attribute, this node will have 100 children. This can have a very high memory and computational cost. C4.5 [15] extends domain of classification from categorical to numerical ones. In addition to above splitting function, other approaches like *gini* index [2], [19] is also used for decision tree construction. Jin *et. al.* [14] address the problem of numerical attributes in streaming data. In their *Numerical Interval Pruning* approach they divide the range of numerical values into equal sized intervals. This largely decreases the number of children a decision node can have.

2.2 Single Tree approach

Incremental learning can be seen from two different perspectives, namely, a) greedy approaches and b) non-greedy approaches. Utgoff *et. al.* [27] and Kalles *et. al.*

[16] use greedy method to construct a tree with the available information. As and when more data becomes available or as the concept in the data drifts, the tree is restructured. Domingos *et. al.* propose a non-greedy approach, VFDT [4] in which sufficient statistics about data is maintained at each node in a decision tree. This is updated as and when data streams through. In this approach decision tree is not restructured every time new data arrives. Alternatively, only when the statistics at a node reaches a limit restructuring is performed. Since the tree is not restructured for every data point, this approach minimizes the effect of ordering of incoming data. In this work a single tree is maintained for the entire data. As this does not take into consideration the drifting of concepts, CVDFT was developed by Hulten *et. al.* [12] which gives more importance to newer data than the older data. As in VFDT, sufficient statistics is maintained at each node. Periodically a splitting test is performed for all the nodes. If a new test-attribute is chosen in place of the older one, a subtree is grown for the new test-attribute. When the new subtree starts performing better than the older one, the old one is dropped. Since a single tree is grown, it takes time for the tree to evolve when a concept drift takes place.

Fan *et. al.* [5] use a two phase approach to address drifting data streams namely 1) class distribution replacement and 2) leaf node expansion. As stated earlier the underlying concept in a streaming data might drift with time. When such a drift happens, a leaf nodes labeling may no longer be correct and it can start performing erroneously. In that case, if the error exceeds a threshold level, probability at the leaf nodes are recalculated and leafs are relabeled with latest class distribution (class distribution replacement). In case if the error is still higher than the acceptable error, leaf node is converted to a decision node and new leaf nodes are formed as its children and labeled appropriately (leaf node expansion).

2.3 Ensemble approach

Street *et. al.* use an ensemble approach in their streaming ensemble algorithm (SEA) [25]. An ensemble of classifiers is built from sequential chunks of data and classification is performed by majority voting. Size of the ensemble is kept fixed. When a concept drift occurs in the data stream, older classifiers are dropped using aging criterion. This means a time stamp is maintained for each classifier and when the threshold limit of classifiers is reached, the oldest classifier (with least time stamp) is dropped to give space to new classifier. Hence SEA maintains only classifiers pertaining to concepts that occurred in the recent past. When a historic concept reoccurs, SEA relearns this concept as if it had never seen the concept earlier. During this relearning phase, ensemble is dominated by classifiers irrelevant to the current concept and this significantly deteriorates the performance of the system. Wang *et. al.* propose an algorithm for mining concept drift [28] in which they use weighed ensembles to produce more accurate results. Although this system produces more accurate representation of the current concept in the stream, this system also relearns historic concepts as if it has never seen the concept before. Since the system relearns from the scratch, this approach takes a considerable time before adjusting to a sudden change in the concept. Our approach addresses this problem by remembering all the historic concepts. When a historic concept reappears in the stream, we only reassemble the ensemble set to represent the current concept correctly. This approach allows our system to adjust to a sudden drift in concept.

In addition to above ensemble approach, other approaches like boosting and bagging are also used for building ensembles[26]. In Bagging a data sample of fixed size is selected from data set. A classifier is built for this sample and added

to the ensemble set. A new sample is selected from the same data set and another classifier is built. This procedure is continued until the size of ensemble is reached. Sampling is done randomly and hence some data points can be picked multiple times and some points can be ignored. Breiman [1] used bagging and experimentally shows that this approach improves classification accuracy significantly compared to conventional tree building strategy. Bagging does not assign weights to individual data points and there is an equal probability of selecting any data point from the data set. In boosting individual data points are weighed and the weights are changed after every sampling. In this approach more weight is given for those points which are getting wrongly classified. So hard data points get more weight and are more likely to get selected for next sample. This approach makes sure that a classifier is built for hard data points which otherwise would have been wrongly classified. After all the classifiers are formed, an ensemble approach is used to classify the data. Freund *et. al.* [6] used boosting to improve accuracy of the classification system.

Boosting and bagging do multiple scans of the dataset. There are a number of multiple-scan classification methods for large databases such as Sprint [24], RainForest [9], BOAT [8], etc. These approaches are not applicable for streaming data because multiple scan is not possible in this environment.

2.4 Option Tress

Some work uses option trees as an alternate to decision trees. Buntine [3] introduced option trees as a generalization of decision trees. Option trees allow option nodes in addition to decision nodes. While a decision node allows only one splitting attribute at a node, an option node allows multiple splitting attributes in the

same nodes. Classification is done similar to decision trees. In addition, at each option node, a rule is applied to combine the prediction of its children nodes.

Two important reasons for choosing an option trees over decision trees are lookahead and stability. While building a decision tree in a top-down approach, an attribute is selected solely depending on the information it provides to classify the data point correctly at that node. This is a one step lookahead. This approach would select an attribute that might perform well in isolation but there is no guarantee that they will perform better as a combination with another attribute. Instead of one step lookahead a two step or multiple step lookahead can be used. In a single step approach, the gain achieved by using an input attribute x_i at the current node alone is considered for tree construction. In a two step approach a pair of input attributes x_i, x_j is considered where x_i is chosen as current node attribute and x_j is used in the next level of the tree construction. Murthy *et. al.* [20] show that this multi-ply lookahead is computationally expensive in decision tree. Option trees by nature have multi-ply lookahead capability

Second problem with the decision tree is its stability. A minor change in one of the nodes close to the root will change the entire subtree below it. Option trees solve this problem by providing a single unified structure. Kohavi *et. al.* [17] shows experimentally that significant reduction in error can be achieved by restricting two levels of option nodes at the top. If an option tree is used for classifying stream data, this limitation of two levels at the top cannot be placed. If this limitation is kept in stream data environment, it will not be possible to represent multiple concepts. Both the above mentioned problem can be solved in decision tree using ensemble approach.

Holmes *et. al* [11] uses option trees for mining data streams. A tree is built for every data chunk. This tree is combined with a global model. At any point of time

only one global model is maintained. While this approach to maintain a single global model and to merge individual classifiers to it, can be used for decision trees also, it is computationally inefficient. Quinlan [23] shows that merging decision tress is multiplicative. But Holmes *et. al* [11] shows that merging option trees is additive. A single global model can represent multiple concepts. If one option trees is used for representing a continuously drifting data streams, the tree becomes very complex due to the fact that every decision node will become an option node. In addition to being computationally intense it become very tough for human interpretation.

Gama *at. al.* [7] present the Ultra Fast Forest of Trees (UFFT) for stream data. For multi class problems, UFFT grows a binary tree for each pair of classes.

Chapter 3

Methodology

3.1 Ensemble based approach

We use a decision tree as the base classifier. Decision trees are constructed using basic ideas of ID-3 algorithm [22]. Sometimes a system works in multiple modes, each mode having a different data distribution. This is a typical of streaming data. In this case individual data points do not give much information but a set of consecutive data points or a chunk of data points gives definitive information. This means that by analyzing a specific interval of the stream, information can be inferred to determine the mode, and also to classify data points in the context of this mode.

We represent a streaming data environment using an ensemble of classifiers (decision trees). Each individual classifier for this committee is built from a relatively small data chunk extracted from the sequential data. It has been shown that a committee of classifier can perform better than an individual classifier. Indeed, if individual classifiers are independent of each other, the ensemble behaves even better [13]. We use this as one of the basic features of our approach.

A new classifier is built when the concepts in the datastream drift. In our approach we don't delete the historic classifiers, but store all the classifiers in a *global set*. This is done so that we can reuse these classifiers to retrieve the historic concepts, if they reoccur. If not for this, the system has to relearn a reoccurring concept again and again. So the *global set* of classifiers, at any-time is likely to contain classifiers built to represent concepts discovered so for. It is not necessary for all the classifiers in the *global* set to classify the current stream correctly. If the entire *global set* is used as an ensemble for classification, the system performs erroneously, because irrelevant classifiers end up contributing to the classification of data. So, only those classifiers which are pertinent to the current concept in the data stream should be used for the purpose of classification.

3.1.1 Maximum mean square error

Our challenge is to select only relevant classifiers from the *global set* to form an ensemble set, and to create a new classifier when none is found. We use a filter which screens the existing classifiers and allows only relevant ones to participate in an ensemble. To be selected, the classification error of a classifier on the immediately preceding data points in the stream is taken into account. Wang *et. al.* use the error of a classifier that predicts randomly for this purpose [28]. A classifier is said to predict randomly, if the probability of data point x being classified to a class c depends solely on c's class distribution in the current data chunk.

If there are c possible classes in the data, classification error of a classifier that predict randomly is:

$$\text{MaxMSE} = \sum_{c} P(c) * (1 - P(c))^2$$

where P(c) = c's class distribution. *MaxMSE* stands for maximum mean square

error. Wang *et. al.* have used *MaxMSE* as the filtering criterion [28]. All the classifiers which perform better than a classifier predicting randomly (those producing less error than *MaxMSE*) are included in the ensemble set. Intuitive basis for this is that a classifier is performing poorly if its classification error is equal to or greater than *MaxMSE*. If *MaxMSE* is kept as the absolute filtering criterion, those classifiers which are minimally better than a classifier predicting randomly, also participates in the ensemble set. These are poor classifiers and affects the performance of the system.

3.1.2 Acceptance Factor

To overcome this problem, we use an *AcceptanceFactor* whose value is such that:

$$0 < \text{AcceptanceFactor} < 1$$

Instead of using *MaxMSE* as the ceiling to select a classifier from the *global set*, a fraction of it, *AcceptanceFactor* times *MaxMSE*, is used as the upper bound. Depending on the value of the *AcceptanceFactor*, classifiers which show minimal improvement to *MaxMSE* are blocked from participating in the ensemble set. For a low *AcceptanceFactor*, the product of *AcceptanceFactor* and *MaxMSE* will be significantly less than *MaxMSE*. Since this product is used as the threshold for the classifiers to participate in ensemble, a low *AcceptanceFactor* would mean that only those classifiers which show significant improvement to *MaxMSE* participates in the ensemble.

3.2 Algorithm

Our discussion is in two phases, *Training* and *Testing*. Here the assumption is that there is an oracle available which on request labels the data correctly. The

cost of labeling using this oracle is very high.

Primary objective of the *Training* phase is to check whether the system already has enough information to represent a current concept in the stream. If not, to build a new classifier and add it to the system so that the system is capable of doing correct classification. Input parameters for the *Training* phase are, 1) permitted error and 2) classifier precision. After a data chunk arrives, the *Training* phase constructs an ensemble of classifiers from *global* set using *AcceptanceFactor* and *MaxMSE*. If this ensemble set performs satisfactorily, this means the system already has the knowledge to evaluate the current concept in the stream. Here the satisfactory performance means classification error is less than the permitted error which is one of the input attributes.

If this ensemble formed does not perform satisfactorily, a new classifier is added to the system. For a building a new classifier, (decision tree in our case), in addition to the training data set, a classifier precision is given as input. Classifier precision for a decision tree is usually expressed as a percentage of the input dataset. A leaf node is not expanded if the number of data points reaching it falls below this limit. For instance, consider a data set of size 1000 and classifier precision of 3%. In this case, a leaf node is expanded to become a decision node only when the number of data points reaching it is greater than 30 (3% of 1000). This procedure makes sure that the decision tree is a generalization of the input data and it does not memorize the input data.

Algorithm *Ensemble Building*

Input: Data stream with class labels available intermittently,

τ = Permitted error

α = Classifier Precision

Output: A set of classifiers, Global set G,

G= {C_1, C_2, C_3,.., C_n }

Classification of Testing data

1. Global set ←G { };
2. Ensemble set ←E { };

Training

3. New_Classifier_Required ←true;
4. E ←{ };
5. Get data chunk T from input stream with class label
6. MaxMSE ←classification error for data set T using a classifier predicting randomly

$$\text{MaxMSE} = \sum_{c} P(c) * (1 - P(c))^2$$

where P(c) = c's class distribution.

7. **for** classifier C_i in G
8. CE_i ←classification error for data set T using classifier C_i
9. **if** CE_i < (AcceptanceFactor * MaxMSE)
10. E ←{E} Union { C_i }
11. Wi ←MaxMSE - CE_i
12. **endif**
13. **endfor**
14. Ensemble-error ←Calculate Classification error using ensemble set E with weights
15. **if** (Ensemble-error < τ)
16. New_Classifier_Required ←false;
17. GO TO *Testing*

18. **endif**

19. **if** (New_Classifier_Required = true)

20. $C_i \leftarrow$Build_Classifier (Data Chunk T, Classifier Precision α)

21. G $\leftarrow$\{G\} Union \{C_i \}

22. GO TO *Testing*

23. **endif**

End *Training*

Testing

24. Classify the incoming stream using ensemble set E until the next set of labeled data is available.

In line 5 of the *Ensemble Building* algorithm, data generated by the streaming source is stored in a data buffer and when the buffer reaches a fixed number, *training data chunk size*, data chunk T is formed.

For the above data chunk T, error produced by a classifier predicting randomly, *MaxMSE*, is calculated in line 6. This value is used for classifier selection to form the ensemble set and is considered as absolute maximum. *Global set* contains a list of all the historical classifiers. In line 8, data chunk T is classified by each of these classifiers and their performance is stored.

Performance of a classifier on the data chunk T is a measure of its capability to represent the current concept in the stream. If the underlying distribution in a stream had not changed, one or more of the classifiers in the *global set* would classify the data correctly. On the other hand, if there was a new concept in the data stream, none of the classifiers in the *global set* might evaluate the new data chunk correctly. This means that a new classifier has to be added to the system.

This is the basis on which a new classifier is added to the *global set*.

In line 9 error of the individual classifier C_i is compared with the product of *AcceptanceFactor* and *MaxMSE*. If C_i is less than the product, the ith classifier qualifies to participate in the ensemble and it is added to the ensemble set in line 10. Depending on the performance of the classifier on the current input data, a weight is assigned in step 11. More weight is assigned to classifiers which have less classification error. In other words, weight is inversely proportional to the error a classifier produces (C_i). From line 11 it can be seen that weight is the difference between *MaxMSE* and CE_i. A very good classifier would produce CE_i close to zero and it would get more weight. More weight means it has more say in classifying a data in ensemble. A poorer classifier will have a larger CE_i and would be assigned less weight and hence has less say in the ensemble.

Error produced by the above formed ensemble, *ensemble-error* is evaluated in line 14 and compared with *permitted error* in line 15. If *ensemble error* is better than *permitted error*, the system has the adequate knowledge to classify the current concept in the stream. Hence the control jumps to the *Testing* phase where the above built ensemble is used for classification.

If ensemble set fails to perform better than *permitted error*, a new classifier is built and added to the *global set* in line 20 and 21. A new classifier is built because the system has not seen this concept historically. So this new classifier is independent of all other classifiers already existing in the system. This independence makes the ensemble set approach stronger.

Whenever labeled data is available, *Training* phase is run. At the end of each *Training* phase an ensemble set is formed and this ensemble set is used for evaluating the current stream in *Testing* phase. The *Testing* phase continues until next set of labeled data arrives. At this point *Training* starts again.

Chapter 4

Experimental Results

4.1 Data Set

Two data sets from UCI repository [21] namely

1) *Nursery Data Set*

2) *Car Data Set*

has been used to demonstrate our approach.

Nursery data, a real life data set, contains data points belonging to 5 different classes (Appendix A). Amongst them, there are only 2 records belonging to class "recommend". Since they are outliers, we remove these two data points and hence it becomes a four-class problem. This data set contains 8 attributes besides class label. W. Peng *et. al.* use this data set for a concept drift problem and induce artificial drift in this data set [29]. Artificial concept drift is introduced by changing the value of an attribute throughout the data set in a consistent way. To achieve this an attribute value is shuffled, for example, an attribute x which has n possible values is changed as $x_1 \rightarrow x_2 \rightarrow \ ... \ \rightarrow x_n \rightarrow x_1$. For instance, *Nursery* data set has an attribute *parents* whose values are *usual, pretentious* and

great pret. Shuffling is achieved by changing all the tuples whose attribute value for *parents* is *usual* to *pretentious*. Also all tuples with *pretentious* is changed to *great pret* and *great pret* to *usual*. For instance three data tuples before the shift is shown in the table 4.1 and after the shift is shown in the table 4.1.

While doing so, the class label is kept intact. In addition to using this, we also shuffle the attributes in the opposite directions. But a particular attribute is either rotated in clock wise or anti-clock wise direction and not in both directions. We generate a data stream by repeating the *nursery* data set. In this stream, at regular intervals one of the 8 attributes is chosen and shuffled Depending on the attribute chosen, concept in the stream drifts mildly from the previous state or a new concept appears.

Once the stream is formed, first set of experiments are run with the entire stream. The next time for the same set of experiments, the stream starts at an offset x from the previous stream starting position. This is repeated for five iterations. The following experimental results are a consolidation of these five runs.

4.2 Nursery Data Set

4.2.1 Ensemble Approach

A single classifier is unlikely to learn the entire data set because there are multiple concepts present within the data set. But in order to observe the capability of our ensemble approach, comparison is made between our approach and that of building a single classifier for the entire data set. Training is done in three different ways. In *Method 1*, a classifier is built for the entire data set. In the *Method 2* a new classifier is built whenever a major concept drift takes place. There are

Table 4.1: Data Set before shift

S.No.	Parents	Has Nurse	Form	Children	Housing	Finance	Social	Health	admission
1	**usual**	proper	complete	1	convenient	convenient	nonprob	recommended	recommend
2	**pretentious**	proper	complete	1	convenient	convenient	nonprob	priority	priority
3	**great pret**	proper	complete	1	convenient	convenient	nonprob	priority	priority

Table 4.2: Data Set After shift

S.No.	Parents	Has Nurse	Form	Children	Housing	Finance	Social	Health	admission
1	**pretentious**	proper	complete	1	convenient	convenient	nonprob	recommended	recommend
2	**great pret**	proper	complete	1	convenient	convenient	nonprob	priority	priority
3	**usual**	proper	complete	1	convenient	convenient	nonprob	priority	priority

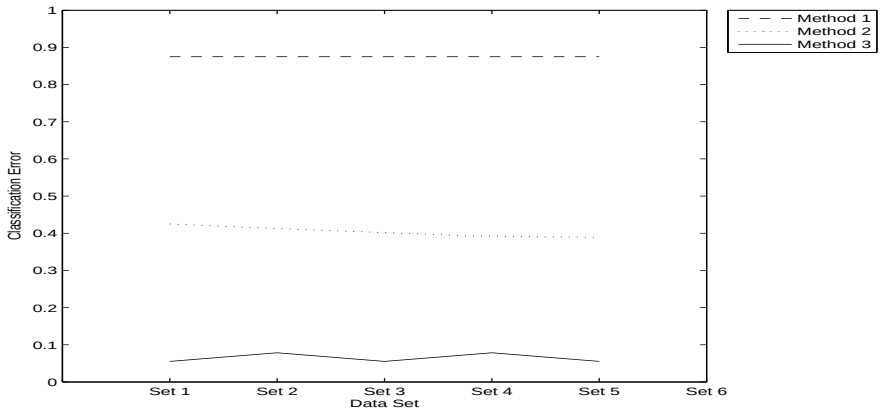

Figure 4.1: Ensemble Vs Single classifier

minor drifts present within this major drift. *Method 3* is our ensemble approach where a classifier is built on a relatively smaller data chunk and classification is done using an ensemble of classifiers rather then using a single classifier.

From Figure 4.1 it can be seen that *Method 1* classifies almost all the data points wrongly. This is because a single classifier cannot represent more than one concept. *Method 2* too performs poorly owing to the fact that it is not able to capture internal drifts within a major drift. *Method 3*, our ensemble approach, produces significantly less error. So, if a data contains multiple concepts, an ensemble approach can represent this data much better than a single classifier.

4.2.2 Data Chunk Length

Each individual classifier in the *global set* is built from a datastream chunk of size 400. If large data chunks are used, internal drift in the data gets lost. At the same time, if the data chunk size is too small, it does not get necessary information to build a classifier. Figure 4.2 shows classification error (performance measure) of the system for different data chunk sizes. Figure 2 shows that error is large for both very large data chunk size and for very small data chunk.

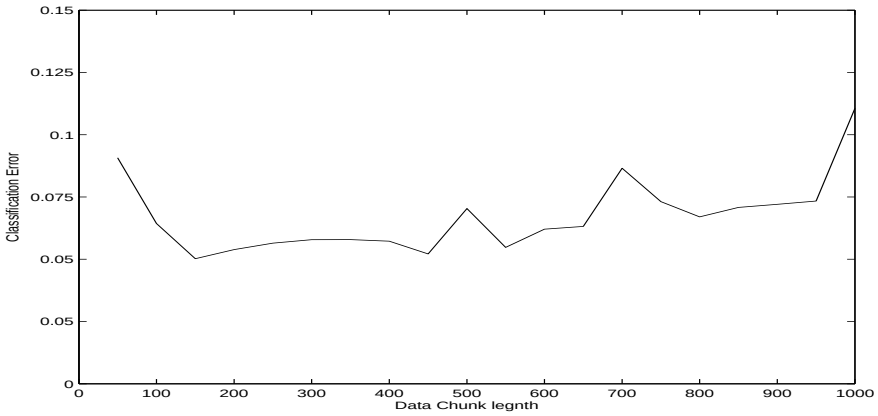

Figure 4.2: Data Chunk Length Vs Performance

4.2.3 Acceptance Factor

Figure 4.3 shows how the systems performance varies with *AcceptanceFactor*. As mentioned earlier, *AcceptanceFactor* takes a value between 0 and 1. A value of zero means only those classifiers which are perfectly classifying the data will participate in the ensemble and a value of one means all classifiers participate in the ensemble. If this value is very high, many erroneous classifiers are included in the system and the system would produce a poor performance. At the same time, if it is too less, either none or very few classifiers qualify which makes the ensemble set weak. Empirically we choose a value of 0.4 as *AcceptanceFactor* for *Nursery* data set. For *Nursery* data set, performance improvement achieved by using an *AcceptanceFactor* is very little. This is an indication that the set of classifiers produced for each distinct concept are independent of each other. So if there are three distinct concepts *A,B and C*, while selecting an ensemble for classifying concept *A*, classifiers belonging to concept *B and C* do not even compete to enter the ensemble. In other words, classifiers not directly built for this concept, produce error that is larger than *MaxMSE*. So the significance of *AcceptanceFactor* is not very evident.

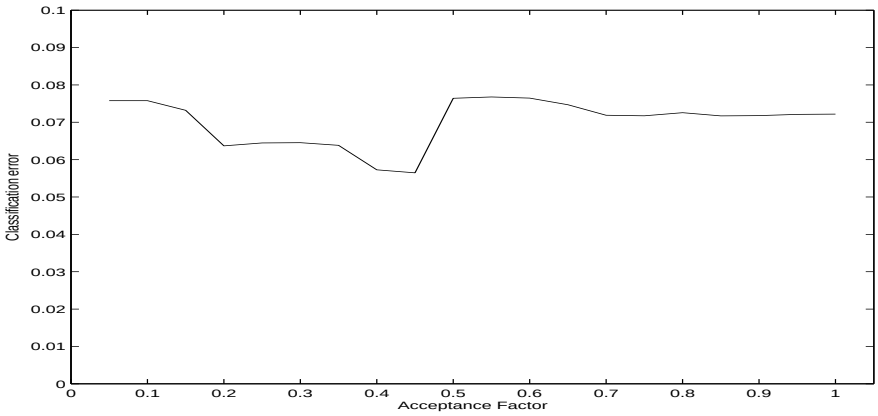

Figure 4.3: AcceptanceFactor Vs Performance

4.2.4 Permitted Error

Permitted error is inversely proportional to the drift tracking capability of the system. If it is very high, system is capable of tolerating a high value of error and hence the system does not sense minor drifts in the data. If this value is very low, the system cannot tolerate much error and unnecessarily new classifiers are added to the system for even modeling the noise. Figure 4.4 shows how classification error and number of classifiers built by the system vary with *Permitted error*. Although the system has very small classification error for low *Permitted Error*, it builds too many classifiers. This is an indication that the system does not learn the data stream but memorizes it. On the other hand, for larger values of *Permitted error*, greater than 0.15, number of classifiers is small, but the classification error is poor. This is because the system is not capable of tracking drifts in the concept. Using Figure 4.4, *Permitted error* for *Nursery* dataset is empirically chosen as 0.05 (5% error).

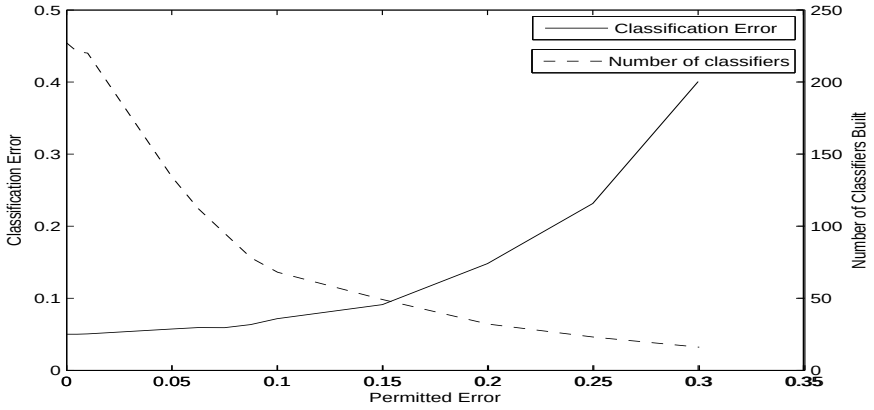

Figure 4.4: Effect of Permitted Error on Classification error and Number of classifiers built

4.2.5 Number of Classifiers built per concept drift

A vital feature of our systems is its ability to remember historic concepts and to use them when a concept reoccurs. Some concepts repeat, some are very close to an earlier concept and some are distinctly different from earlier concepts. In Figure 4.5, the X-axis represents the number of data points in the stream. Y-axis represents the number of classifiers built by the system for each concept drift. It can be observed from the Figure 4.5 that the number of classifiers built by the system for each concept drift is not the same. When a new concept appears in the stream, a relatively large number of classifiers are built; but when a concept mildly drifts from the existing concept or a concept reoccurs, fewer classifiers are built. This shows that our system does not relearn if a historic concept reappears.

4.2.6 Ensemble Weight

When building an ensemble set each classifier is assigned a distinct weight depending on its ability to classify the current concept. This gives more importance to good classifiers than the poor ones. Ensemble set is built with and without

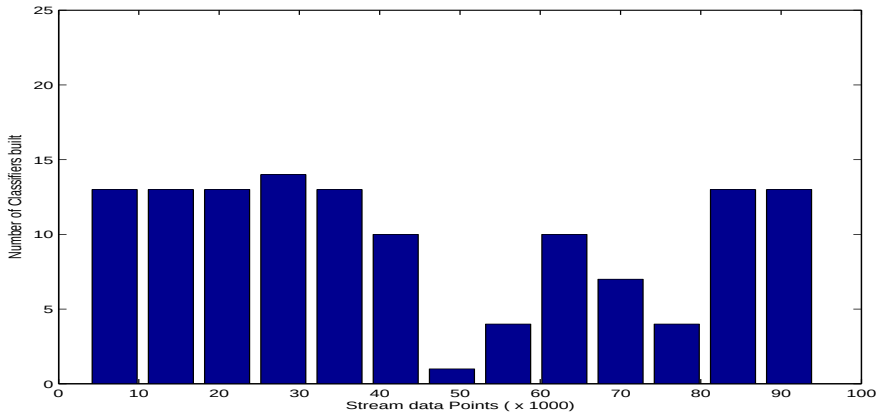

Figure 4.5: Number of classifiers built per concept drift

weights and the performance of the system is analyzed in Figure 4.6. Classification error without weight is indicated by dotted line and with weight is indicated by solid line. Clearly classification error is lower when classifiers are weighted. The performance improvement achieved by using weights is not very significant for *Nursery* data set.

We have generated synthetic streaming data from *Nursery* dataset. In this synthetic environment if a concept drift happens from concept A to concept B, this drift is sudden. But in real world environments like weather, drift from phase A to B might be gradual. For this intermediate phase, many intermediate classifiers will be built. Nature of these classifiers will be such that they can classify both concept A and B with some level of confidence but they cannot accurately classify either of them. Ensemble weights are very useful in this case to justly punish poor classifiers and rely more on good classifiers. Since out synthetic data does not have these mildly drifting property, effect of ensemble weight is not very evident.

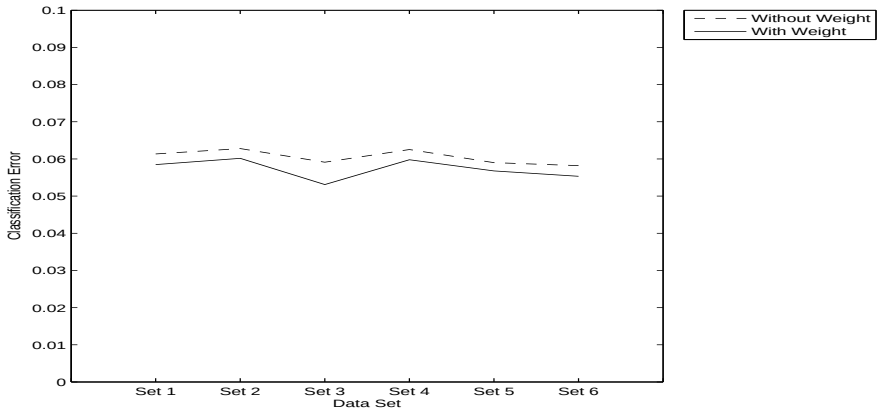

Figure 4.6: Effect of Ensemble weights

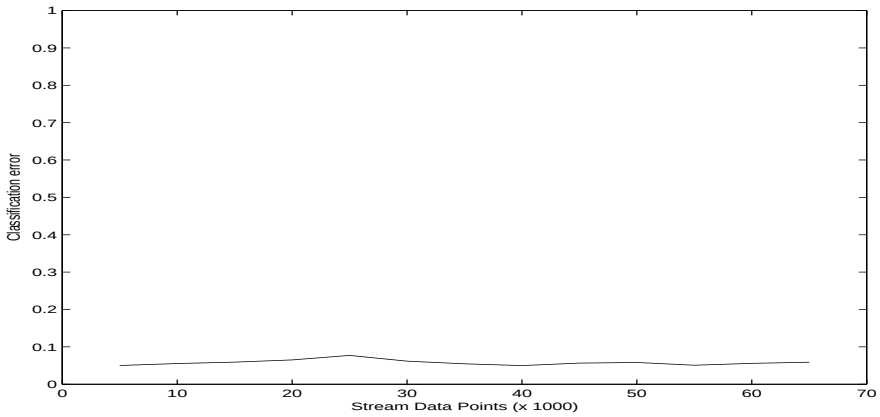

Figure 4.7: Performance of the system

4.2.7 System Performance

Performance of the system, (Classification error), produced by the system in a continuously concept drifting environment is represented in Figure 4.7. This stream includes both major and minor drift and also sudden concept drift. Figure 4.7 clearly shows that the performance of the system is consistent throughout the stream. This is an indication that the system is capable of evolving to learn new concepts and to perform consistently in environments with sudden concept drift.

4.2.8 Classifier Utilization

As mentioned earlier, *global set* contains classifiers built to represent multiple concepts. Not all the classifiers participate in the classification process at a time. If there are distinctly different concepts, the set of classifiers used for the classification of these two concepts might be mutually exclusive. During classification, each concept is represented by a set of classifiers from the *global set*. If two concepts are totally different, the set of classifiers representing them is distinct and they don't share any classifier. But if one concept is a minor drift from another concept then the two classifier set has many common classifiers. We show this by analyzing the utilization of *global set* during different concepts. This is shown in Figures 4.8, 4.9, 4.10 and 4.11. The classifiers in the *global set* are numbered sequentially and the number of times each classifier is used during the classification of a particular concept is represented as utilization. Concept 1, 3, and 5 are distinctly different concepts. From the Figures 4.8, 4.9 and 4.11 it can be seen that the classifiers numbered 0-12 participate for classifying concept 1, 13-54 for concept 4 and 55-114 for concept 5. So each of these concepts do not share classifiers amongst them. Concept 7 and concept 3 are closely related concepts. So it can be seen that, to represent concept 7 a lot classifiers used for concept 3 are used.

4.3 Car Data Set

Streaming data is generated from *Car* data set similar to *Nursery* data. *Car* data set is a four class problem (Appendix B).

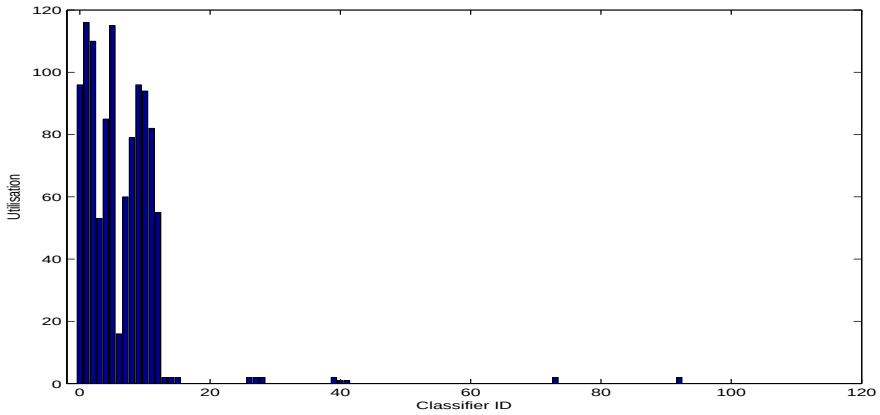

Figure 4.8: Concept 1

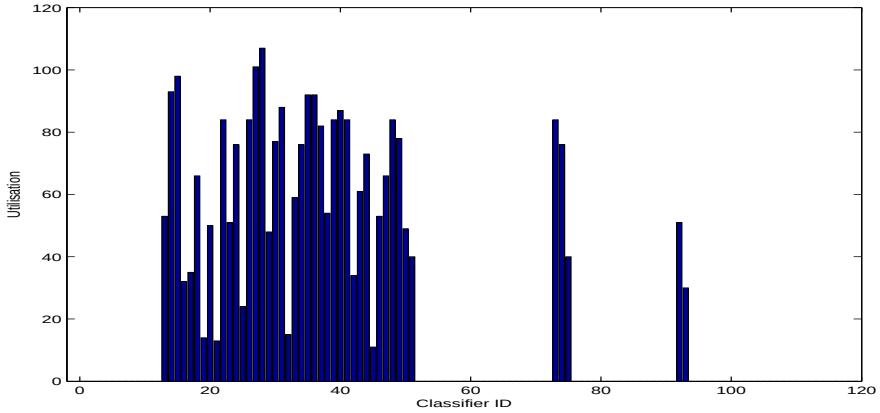

Figure 4.9: Concept 3

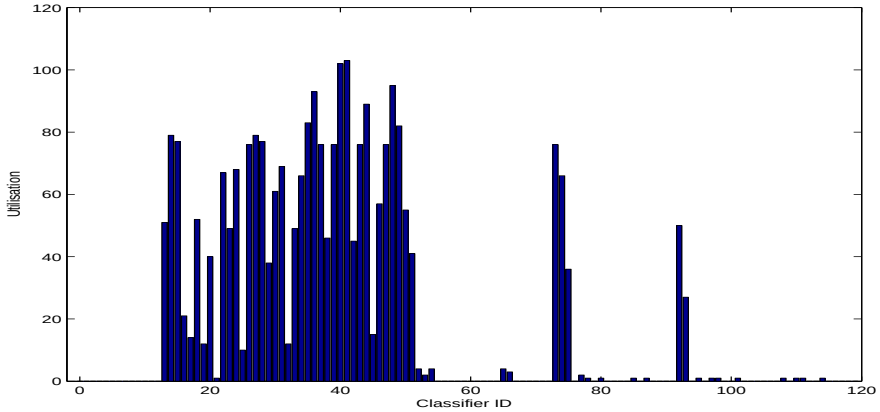

Figure 4.10: Concept 7

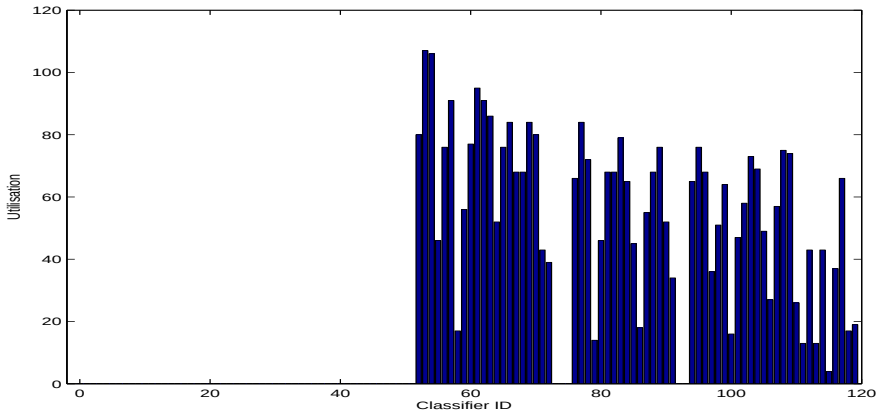

Figure 4.11: Concept 5

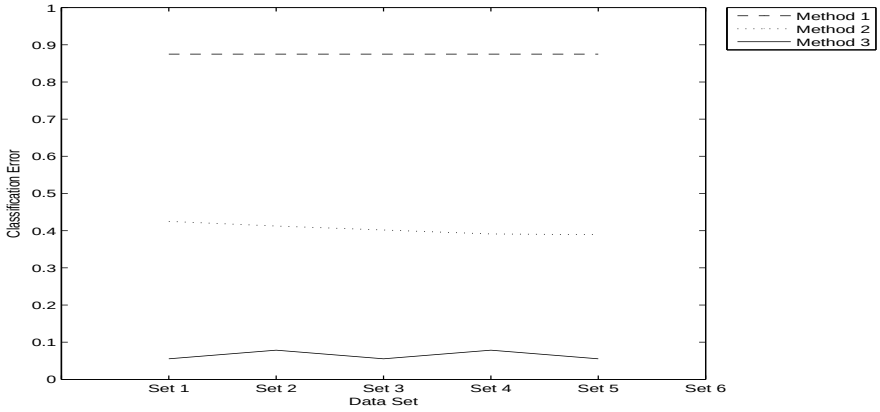

Figure 4.12: Ensemble Vs Single classifier

4.3.1 Ensemble Approach

Performance improvement achieved by using an ensemble of classifiers rather than a single classifier can be seen from the *Car data set* also. As stated earlier a single classifier cannot learn the entire stream because there are multiple concepts within the stream. As with *Nursery* data set training is performed in three different ways. A single classifier for entire data set in *Method 1*, a classifier per major concept drift in *Method 2* and our approach, namely ensemble of classifiers, in *Method 3*.

From Figure 4.12 it can be seen that single classifier approach, *Method 1*,

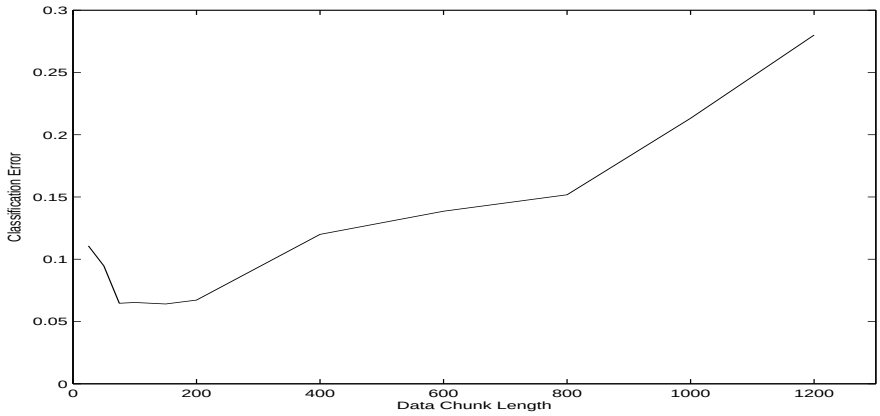

Figure 4.13: Data Chunk Length Vs Performance

almost classifies all the data points wrongly. *Method 2* too performs poorly owing to the fact that it is not able to capture internal drifts with a major drift. *Method 3*, our ensemble approach, produces significantly less error. This reemphasizes the point that an ensemble of classifiers perform better than a single classifier, if the data contains multiple concepts.

4.3.2 Data Chunk Length

As stated earlier the stream is divided into data chunks of equal size for the purpose training and testing. Effect of data chunk size on the performance of the classification system is more evident the *Car* data set. If the data chunk size is large the classification system can not capture minor drifts inside the stream. This can be observed from the Figure 4.13. As the data chunks size increases the performance of the system degrades. At the same time data chunk size cannot be very small also, classifier cannot generalize the information, otherwise it memorizes the stream rather than learning it. For *Car* data set an optimal size of 75 is chosen as data chunk length empirically.

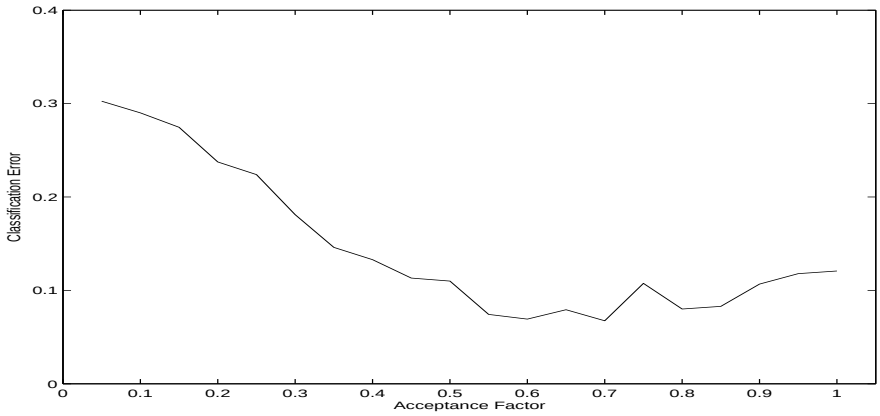

Figure 4.14: AcceptanceFactor Vs Performance

4.3.3 Acceptance Factor

The effect of *AcceptanceFactor* was not very evident in *Nursery* data set. By changing the value of *AcceptanceFactor* performance of the system on *Nursery* data set was not considerable. But for *Car* data set performance of the system can be highly improved by choosing a correct *AcceptanceFactor*. From the Figure 4.14 it can be seen that the system performs poorly for low acceptance factors. This is an indication that the ensemble is not formed with enough number of qualifying classifiers. We choose a value of 0.6 empirically for this data set. It can also be seen that the system starts performing poorly as the *AcceptanceFactor* is increased beyond a limit. This is because irrelevant classifiers start participating in the ensemble when the *AcceptanceFactor* is very high.

4.3.4 Permitted Error

Effect of *Permitted error* on the system's performance is similar for the *Car* data set as the *Nursery* data set. *Permitted error* controls both the classification error and the number of classifiers built by the system. From the Figure 4.15 it can be observed that for a low *Permitted error* the systems performs well in terms

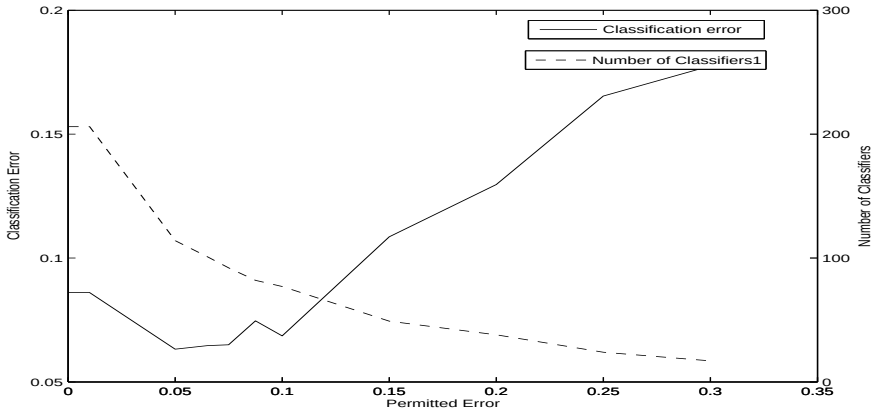

Figure 4.15: Effect of Permitted Error on Classification error and Number of classifiers built

of classification error. At the same time, there are too many classifiers built. Because even for a small drift in the data a new classifier is added to the system. On the other extreme when the permitted error is high, the systems error tracking capability is severely hampered. The reason is even a major drift is not detected by the system. Empirically a value of 0.06 is chosen as *Permitted error* for *Car* data set.

4.3.5 Number of Classifiers built per concept drift

Streaming data is characterized by recurring concepts. Our primary aim is to track these recurrent concepts without relearning from scratch. This means that the system should be capable of coping with a minor drift in the stream by adding only few extra classifiers to the *global list*. At the same time, when confronted with a new concept, the system must be capable of learning the new concept. This can be observed from the Figure 4.16. Whenever there is a minor drift in the stream, only a few classifiers are built, whereas when a new concept appears on the stream, the system builds as many classifiers as required to capture this

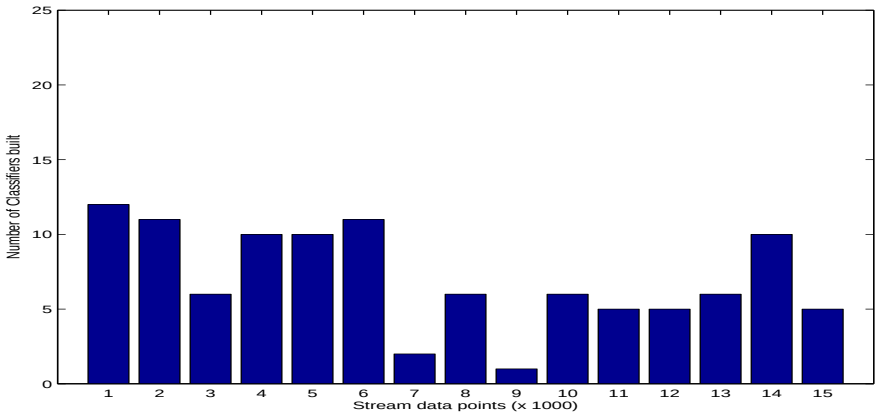

Figure 4.16: Number of Classifiers built per concept drift

new concept.

4.3.6 Ensemble Weight

As with *Nursery* data set, the classification systems performance is improved by using a weight to individual classifiers participating in the ensemble. Figure 4.17 depicts the performance of the system with and without using weights for classifiers. When all the classifiers inside the ensemble are given a uniform weight the system has a comparatively poorer performance which is represented by the dotted line in the Figure 4.17. When individual classifier are given a weight according to their performance of test data set, the system performs relatively better which is represented by the solid line.

4.3.7 System Performance

Performance of the classification system at regular intervals in a streaming environment is an indication of the systems capabilities to cope up with new concepts in the stream. Systems performance on the *Car* dataset is depicted in the Figure 4.18. Systems performance stays consistently around 7% error.

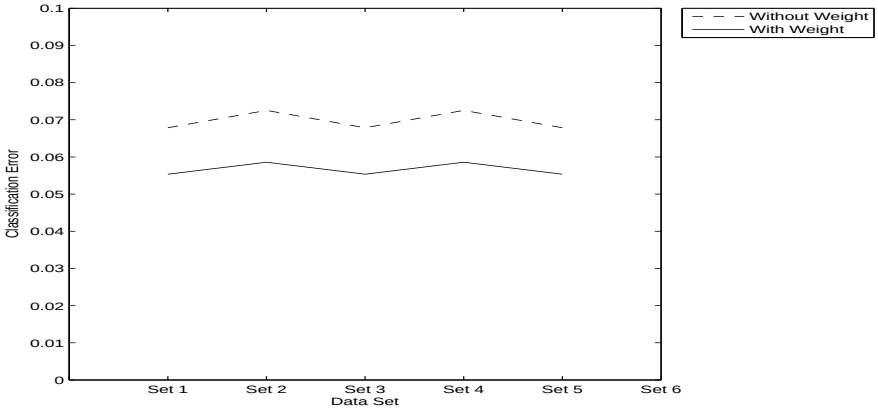

Figure 4.17: Effect of Ensemble Weight

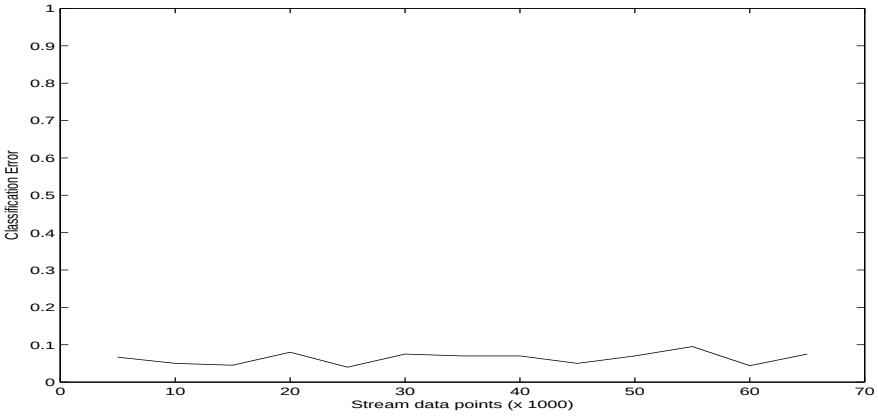

Figure 4.18: Performance of the system

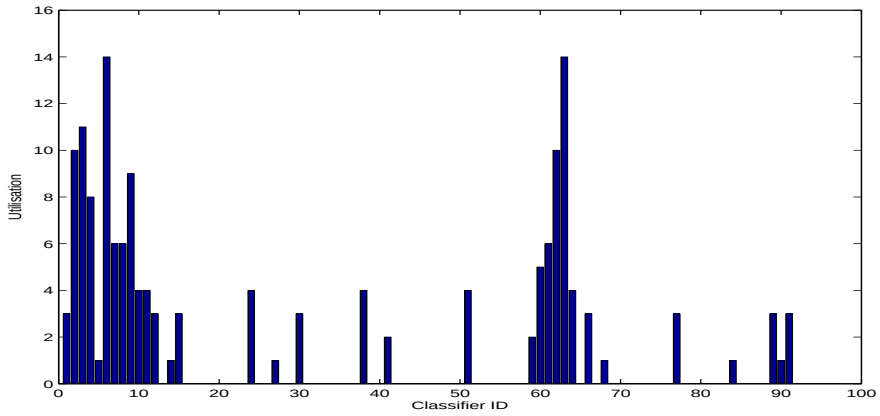

Figure 4.19: Concept 1

4.3.8 Classifier Utilization

Classifier utilization in the *Car* data set indicates the relationship between different concepts occurring in the stream. If two concepts are distinctly different then the classifiers they use from the *global* set might be mutually exclusive. Concepts 1,2 and 5 are different concepts in the *Car* data set and this is evident from the Figure 4.19,4.20 and 4.22. Since they are different from each other there is not much resemblance in the set of the classifiers they use. But the concepts 7 is a minor drift from concept 2. Hence the classifiers utilization pattern for the concepts 2 and 7 are very identical. this can be observed from 4.20 and 4.21

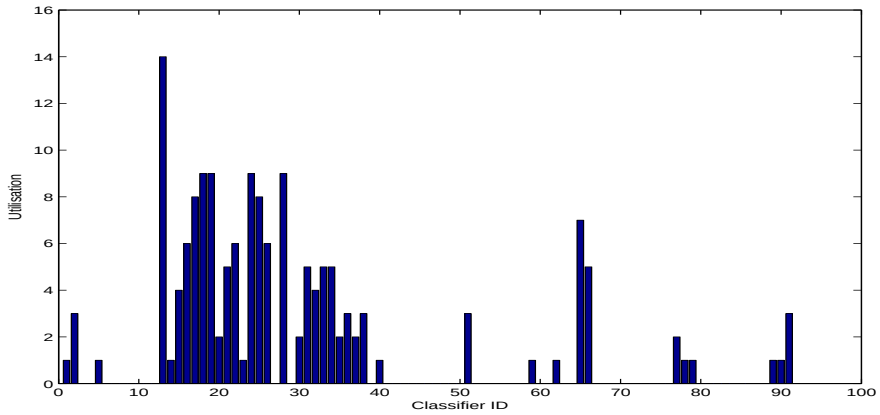

Figure 4.20: Concept 2

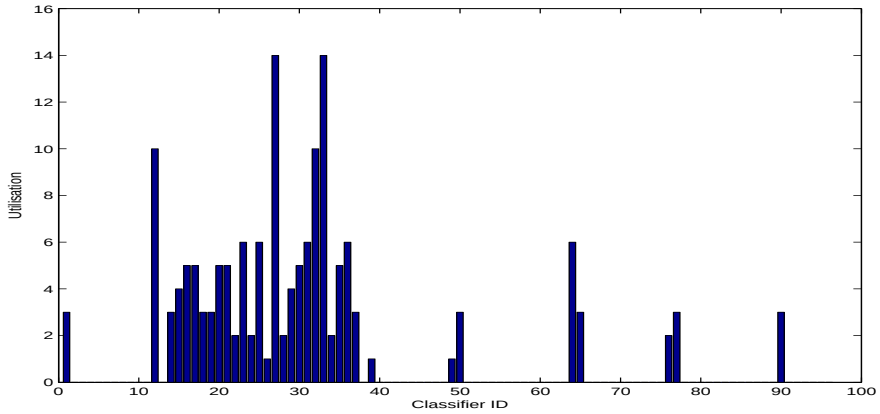

Figure 4.21: Concept 7

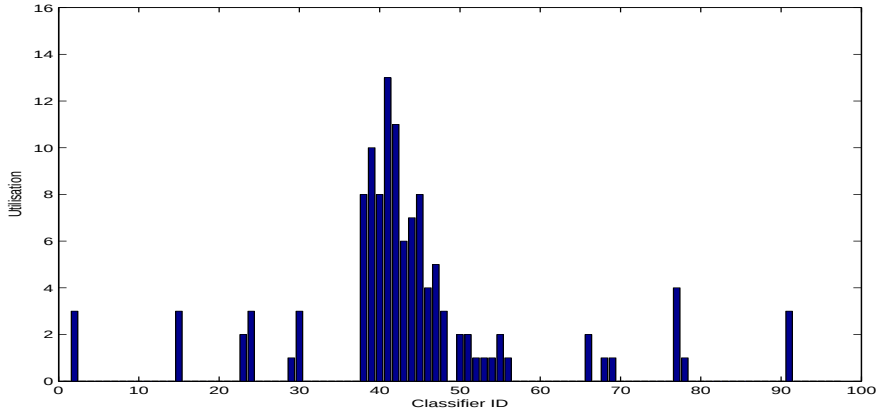

Figure 4.22: Concept 5

Chapter 5

Conclusion and Future work

5.1 Conclusion

We have proposed a classifier system for stream data mining using ensemble approach. This system is capable of performing any-time classification, learning in one scan and detecting drift in the underlying concept. The important issue of detecting recurring concept drifts has been solved by us for the situations similar to the dataset used. When there is a recurrent concept drift our system uses much of the already learnt information rather than learning it anew. Success of our approach has been demonstrated with the experimental results with the test dataset.

5.2 Future work

It is assumed that it is always possible to retrieve class label of the incoming data. In a domain like weather prediction, this assumption does not hold good. Even in the domains where this assumption holds well, it is too expensive to label the

data. This is the case with credit card fraud detection system. It is actually viable to do a deep investigation of a cards transaction to check whether the card is fraudulently used. But in practical condition it is very costly to do this check for every card. Hence, a mechanism has to be derived to determine the goodness of a classifier without labeled data.

Bibliography

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. The Wadsworth & Brooks/Cole, 1984.

[3] W. Buntine. Learning classification trees. *Statistics and Computing*, 2(2):63–73, 1993.

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

[5] W. Fan, Y. Huang, and P. S. Yu. Decision tree evolution using limited number of labeled data items from drifting data streams. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 379–382, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[7] J. Gama, P. Medas, and P. Rodrigues. Learning decision trees from dynamic data streams. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 573–577, New York, USA, 2005. ACM.

[8] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat - optimistic decision tree construction. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD inter-*

national conference on Management of data, pages 169–180, New York, NY, USA, 1999. ACM.

[9] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Mining Knowledgse Discovery*, 4(2-3):127–162, 2000.

[10] M. B. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32(2):101–126, 1998.

[11] G. Holmes, R. Kirkby, and B. Pfahringer. Mining data streams using option trees. In *First International Workshop on Knowledge Discovery in Data Streams*, 2004.

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

[13] R. A. Jacobs. Methods for combining experts' probability assessments. *Neural Computation*, 7(5):867–888, 1995.

[14] R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 571–576, New York, NY, USA, 2003. ACM.

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

[16] D. Kalles and T. Morris. Efficient incremental induction of decision trees. *Machine Learning*, 24(3):231–242, 1996.

[17] R. Kohavi and C. Kunz. Option decision trees with majority votes. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 161–169, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[18] K. Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

[19] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, pages 18–32, London, UK, 1996. Springer-Verlag.

[20] S. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1025–1031. Morgan Kaufmann, 1995.

[21] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.

[22] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106.

[23] J. Quinlan. Miniboosting decision trees. submitted to jair, 1998.

[24] J. C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 544–555, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[25] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, New York, NY, USA, 2001. ACM Press.

[26] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[27] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[28] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD '03: Proceedings of the ninth ACM SIGKDD*

international conference on Knowledge discovery and data mining, pages 226–235, New York, NY, USA, 2003. ACM Press.

[29] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi. On reducing classifier granularity in mining concept-drifting data streams. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 474–481, Washington, DC, USA, 2005. IEEE Computer Society.

Appendix A

Nursery Data Set

The purpose of the *Nursery* dataset is to determine whether a application for nursery school admission application will be accepted or not. There are eight inputs namely (1) Parents occupation, (2) Availability of Nurse (3) Family structure (4) Number of children's in the family (5) Housing condition (6) Financial standing (7) Social condition and (8) Health condition. Output class has five different values indicating the likely hood of the child being admitted.

Historically *Nursery* dataset was developed to rank applications for nursery school. During 1980s there was excessive enrollment to the schools in Ljubljana, Slovenia. Applicants who were rejected for admission wanted to know the reason for their rejection. An objective reason could be given to those applicants who got rejected by building a decision model on the nursery data set.

The input attributed and their possible values are as follows:

1) **parents** *usual, pretentious, great pret*

2) **has nurs** *proper, less proper, improper, critical, very crit*

3) **form** *complete, completed, incomplete, foster*

4) **children** *1, 2, 3, more*

Table A.1: Nursery data Set Class Distribution

Class	Number of instances	Percentage of the total %
not recom	4320	33.333
recommend	2	0.015
very recom	328	2.531
priority	4266	32.917
spec prior	4044	31.204

5) **housing** *convenient, less conv, critical*

6) **finance** *convenient, inconv*

7) **social** *non-prob, slightly prob, problematic*

8) **health** *recommended, priority, not recom*

As stated above the final class is an indication of how likely the child will get admission. There are five different values. Their class distribution is as in the table A

The 'recommend' class has only 2 data point in a data set of size 12960. We remove these two points as outliers before generating streaming data.

Appendix B

Car Evaluation Data Set

Car Evaluation data set is used to evaluate a car depending on various input parameters. There are six input attributes namely buying price, maintenance cost, number of doors, capacity (persons), luggage boot space and safety. Depending on these six values, car is evaluated to be unaccommodating, accommodating, good and very good.

Possible value of the input attributes are as follows:

1) **buying** *v-high, high, med, low*

2) **maint** *v-high, high, med, low*

3) **doors** *2, 3, 4, 5-more*

4) **persons** *2, 4, more*

5) **lug boot** *small, med, big*

6) **safety** *low, med, high*

There are 1728 tuples and the class distribution among the possible four classes is shown in the table B

Table B.1: Car data Set Class Distribution

Class	Number of instances	Percentage of the total %
unacc	1210	70.023
acc	384	22.222
good	69	3.993
v-good	65	3.762