Chapter 1

Related Works

This chapter aims to present an overview of the work done in the fields of stream mining particularly focusing on ensemble learning over streams.

1.1 Stream Mining

Compared to the classical data mining approaches stream mining is relatively a newer topic to be addressed in literature. Even though for batched approaches both classification and clustering problems have been vastly studied, their stream adaption remains a challenge due the restrictions imposed by the stream data. Possibility of temporal locality makes the classification problem harder in a streaming environment. Algorithms needs to address the evolution of underlying data stream.

Domingos and Hulten introduced a strict one-pass adaptation of decision tree [Breiman et al., 1984, Quinlan, 1993] approach in streams. Classic approaches like ID3 and C4.5 learners assumes that all training examples can be stored in the main memory altogether. This is a significant limitation to the number of examples these algorithms can handle. Similarly, disk based decision tree learners (SLIQ [Mehta et al., 1996], SPRINT [Shafer et al., 1996], etc.) become very expensive when datasets are very large and the expected trees has many levels. Domingos and Hulten proposed Very Fast Decision Trees (VFDT) [Domingos and Hulten, 2000] that uses Hoeffding bound [Hoeffding, 1963] to build an anytime decision tree for constant memory and time. The primary assumption in this approach is that, to find the best attribute for a node in a decision tree, it may be sufficient to consider only a fraction of the training set that pass through that node. Hoeffding bound provides an statistical measure to determine how much data is needed to ensure a certain degree of certainty, i.e. error margin would be bounded by a given value [Catlett, 1991].

Like most statistical and machine leaning algorithms VFDT assumes that training data is randomly drawn from a stationary distribution. This assumption is not valid for large databases and data streams. Over time underlying method or environment could change that generates data. The shift is sometimes also referred as *concept drift* in literature and can be abrupt as well as very slow. Data related to weather forecast, economic condition prediction, mis-calibrated sensors, etc. are examples of concept drifting environment. A

concept-adaptive variant of VFDT, CVFDT [Hulten et al., 2001], can handle such scenarios. CVFDT updates its decision rules, essentially the tree structure, by detecting the concept drift in the data. It maintains alternate subtrees whenever an old subtree becomes questionable, and replaces the old one with the alternative when it become more accurate. CVFDT uses a sliding window and updates sufficient statistics by increasing the count of newly arrived examples and decreasing the count of old examples in the window. Essentially CVFDT achieves same accuracy that would be achieved if VFDT would have been run again with the new data. CVFDT does this in $O(1)$ with additional space requirement as compared to the VFDT's $O(w)$ where w is the window size. A extension of VFDT, VFDTc was proposed in [Gama et al., 2003] that improves VFDT by adding continuous numeric attribute handling and naive Bayes prediction at the leaves.

In a different approach to handle concept drift Castillo et al. [Castillo et al., 2003] used Shewhart P-Charts in an online framework based on the idea of Statistical Quality Control. Two alternatives of P-charts were used to monitor the stability of one or more quality characteristics in a drifting stream. The two alternatives only differed in the methods they estimate the target value to set the center. The group later introduced another drift detection scheme that monitors probability distribution of examples and maintains a online error rate to detect any concept drift [Gama et al., 2004a]. When distribution changes, error rate will increase. For stationary concept, the error rate will decrease. A new concept is said to be started if the error rate exceeds some predefined warning or threshold level. This approach has been used in Ultra Fast Forest Tree (UFFT) [Gama et al., 2004b, Gama et al., 2005] stream classification method. UFFT maintains naive Bayes statistics for every node. If at any node the error rate starts increasing, the node is pruned for drifting concept. UFFT uses similar approach and Hoeffding bound to control the growth of the tree. For each pair of classes a tree is maintained, hence it is called forest-of-trees.

Another decision tree based approach based on so-called Peano Count Tree (P-tree) has been developed by Ding et al. for spatial data streams [Ding et al., 2002]. The Peano Count Tree is a spatial data structure that facilitates a lossless compressed representation of spatial data. This structure is used for fast calculation of information gain for branching in decision trees.

Aggarwal et al. employs a slightly different idea in handling time-evolving data in their on demand classification approach of data streams [Aggarwal et al., 2004]. They used a modified micro-clusters concept introduced in [Aggarwal et al., 2003]. Micro-clusters are created from the training data stream only. Each micro-cluster corresponds to a set of points from the training data belonging to the same class. To maintain statistics over different time horizons and avoid storage of unnecessary data points a geometric time frame is used. In the classification task, the k nearest neighbor based approach is taken, where micro-clusters are treated as node weighted by their instance counts.

In [Ganti et al., 2002] two algorithms named GEMM and FOCUS have been introduced for streams under block evolution. GEMM is used for model maintenance and FOCUS is for change detection between two data stream models. These algorithms has been tested using decision trees and frequent item set models. FOCUS uses bootstrapping methods to

compute the distribution of deviation values when data characteristics remain the same. This distribution is then used to check whether the observed deviation value indicates a significant deviation. In another approach to handle concept drift, Last [Last, 2002] proposed an online classification system OLIN that would dynamically adjust the size of the training window and the number of new examples between model re-constructions to the current rate of concept drift. OLIN uses constant resources to produce models, and achieves nearly the same accuracy as the ones that would be produced by periodically re-constructing the model from all accumulated instances.

Later, Aggarwal proposed an concept drift technique based on velocity density estimation [Aggarwal, 2003]. Velocity density estimation is a technique to understand, visualize, and determine trends in the evolving data. The work presented a scheme to use velocity density estimation to create temporal velocity profiles and spatial velocity profiles at periodic instants in time. These profiles are then used to predict dissolution, coagulation, and shift in data. Proposed method could detect changes in trends in a single scan with linear order of number of data points. Additionally a batch processing techniques to identify combinations of dimensions which results the greatest amount of global evolution are also introduced. In [Kifer et al., 2004] authors tried to formally define and quantify the change so that existing algorithms can precisely specify when and how the underlying distribution has changed. They employed a two fixed-length window model, where a current one is updated every time a new example arrives and a reference one is only updated when a change has been detected. To compare the distributions of the windows $L1$ distance has been used [confirm!read again]. Another method to compare two distribution has been presented in [Dasu et al., 2004] where authors used Kullback-Leibler (KL) distance to compare two distributions. KL distance is known to be related to the optimal error in determining the similarity of two distributions. In this non-parametric method no assumptions on the underlying distributions is required.

1.2 Ensemble Learning

Traditional machine learning algorithms generally feature a single model or classifier such as Naïve Bayes or Multilayer Perceptron (MLP). The free parameters of these learners (e.g. weights of feed-forward neural network) are set by realizing the complete training set. These classifier provides a measurement of the generalization performance i.e. how well the classifier generalizes the training set. However, given a finite set of training example, it is rather reasonable to assume that the data might contain several different generalization. For example, a different setting of neural network classifier (weights, node layers, node counts, etc.) changes the final network to some extent. For stream environment, this assumption becomes primitive. Thus, choosing a single classifier is not always optimal. Using the best classifier among several classifiers where each are trained with same training set would be an alternative, however, information is still being lost by discarding sub-optimal options. A better alternative would be to build a classifier ensemble. Ensemble classifiers combine the prediction of multiple base level model built on traditional algorithm. A simple process for combining prediction could be to

choose the decision based on majority voting [Parhami, 1996]. As demonstrated in several works [Breiman, 1993, Schapire, 1990, Wolpert, 1992] ensemble methods (e.g. ensembles of neural networks) [Hansen and Salamo, 1990, Tumer and Ghosh, 1999] yield better performance.

Without proper selection and control over the training process of the base learners, ensemble classifiers could result in poorer performance. Simply choosing a base classifier and training it for several settings would surely produce highly correlated classifiers which would have adverse effect on the ensemble process. One solution of this issue is to train each classifier with its own training set generated by sampling the original one. However, with random sampling each classifier would receive a reduced number of training patterns, resulting a reduction in the accuracy of the individual base classifier. This reduction in the base classifier accuracy is generally not recovered by the gain of combining the classifier unless measures are taken to make the base classifiers diverse. Classifiers with complementary information would give the lowest correlation [Breiman, 1993, Tumer and Ghosh, 1999]. Many methods have been proposed to promote diversity among the base classifier: bagging [Breiman, 1994], boosting [Drucker et al., 1994, Freund and Schapire, 1997, Tumer and Oza, 1999], cross-validation partitioning [Krogh and Vedelsby, 1995, Tumer and Ghosh, 1999], etc. These methods mainly process the entire training set repeatedly and require at least one pass for each base model. This is not suitable for streaming scenarios. Stream adaptation of bagging and boosting methods has been introduced by Oza et al. [Oza and Russell, 2001, Oza, 2001].

1.2.1 Ensemble Learning in Streams

Learning algorithms in data streams require maintainance of a hypothesis based on the training instances seen thus far with the need for storage and reprocessing. Facilitating this requirement, Oza and Russell developed an online version [Oza and Russell, 2001, Oza, 2001] of traditional bagging and boosting. Bagging works by randomly sampling with replacement from the training set to form a given number of intermediate training sets which are used to train same number of classifiers. During testing a majority voting scheme is employed on the decisions of all classifiers to deduce the final decision. Boosting uses an iterative procedure to adaptively change distribution of training data by focusing more precisely on misclassified instances. Initially all instances have equal weights, and at the end of a boosting round weight of each instance is updated by increasing or decreasing if the instance was classified wrongly or correctly, respectively. For online variant of these algorithms, not knowing the size of the training data poses a problem in determining the size of training sets to build the base models. In [Oza and Russell, 2001] authors address this situation by training k models with each instances where k is a suitable Poisson random variable. Later on, [Pelossof et al., 2008] proposed the Online Coordinate Boosting algorithm where the number of weight updates of [Oza and Russell, 2001] is reduced using few simple alteration.

Online bagging and boosting method do not particularly give attention to the concept

drifting nature in the data. Accuracy Weighted Ensemble (AWE) [Wang et al., 2003] is one of the earliest work on concept-drifting stream data. AWE assumes that the stream is delivered in chunks of defined size. With each incoming chunk, AWE updates its k classifiers. Each classifier is associated a weight which is inversely proportional to the expected error of the respective classifier. To estimate this error, it is assumed that the distribution of test set is closest to the most recent chunks. Concept drift is adapted by effectively manipulating the number and the magnitude of the weights that are changing. An extension of AWE has been proposed in [Brzezinski and Stefanowski, 2011], namely Accuracy Updated Ensemble (AUE). AUE takes the weighting motivation from AWE, but improves the limitation of AWE. In AWE each classifier learns from the incoming chunks in a "batched" fashion. AUE employs an online scheme instead. AUE also adapts the weighting function to reduce the adverse effect in AWE of sudden drift in data. AWE weighting function is prone to suffer by rapid change in the stream and most or even all classifiers assuming they are "risky". This limitation has been addressed in AUE. Result shows that AUE performs marginally better than AWE, however, also requires slightly longer time and larger space.

As mentioned in the previous section, Hoeffding Tree (HT) e.g. VFDT [Domingos and Hulten, 2000], can be used to build classifiers for concept drifting streams. The Hoeffding Tree has the property that it adapts itself for the newer examples. The number of examples that a HT is build upon is determined by two numbers: (i) the be size of the tree, and (ii) the number of examples used to create a node. Thus, smaller trees adapt faster to the changes in the data, while larger trees tries to retain the rules that reflect longer time-frame, simply because they are built on more data. In other words tree bounded by size n would be reset twice as often as tree bounded by size 2n. Adaptive Size Hoeffding Tree (ASHT) [Bifet et al., 2009] uses this intuition to build an ensemble of classifiers of different sized Hoeffding trees. ASHT attempts to increase the diversity in the bagging approach. The maximum allowed size for n-th tree is twice the size of $(n-1)$-th, where the 1st tree has a size of 2. Additionally inverse of the squared error has been used as the weights for the trees. Diversity between the traditional bagging and ASHD bagging are compared using kappa statistic. If two claffier agree on every example then k = 0, and if they agree on the predictions purely by chance then k = 0. To test this approach authors have used Interleaved Test-then-Train method. At the leaf level of HT, Naive Bayes predictions are used. The experiments were performed on several different genereted dataset such as SEA Concepts Generator, STAGGER Concepts Generator, Rotating Hyperplane, Random RBF Generator, etc. Performance are compared with traditional Naive Bayes, HT, and boosting methods. Evaluation concluded that bagging provides best accuracy, however, with the higher cost in terms of running time and memory. Authors made an observation that even bagging using 5 trees of different size might be sufficient for gain higher accuracy, as error level for bagging with 10 trees does not drop much but takes twice time.

Same authors also proposed an adaptive window size bagging method- ADWIN [Bifet et al., 2009]. ADWIN automatically detects and adapts to the current rate

of change. To do so ADWIN adapts its window size to maximize the statistically consistently length that conforms following hypothesis "there has been no change in the average value inside the window". Window is not maintained explicitly rather using a variant of exponential histogram technique that takes $O(logw)$ memory and $O(logW)$ processing time where w is the length of the window. Experimental evaluation showed that ADWIN has better accuracy than ASHT, however, requires more time and memory.

ADWIN has later been used in leverage bagging [Bifet et al., 2010b]. Leverage bagging improves randomization by increasing resampling and using output detection codes. Resampling with replacement is done in Online Bagging using Poisson(1). Instead leverage bagging increases the weights of resampling using a larger value λ to compute the value of the Poisson distribution. The Poisson distribution is used to model the number of events occurring within a given time interval. In other improvement randomization is added at the output of the ensemble using output codes. Method works by assigning a binary string of length n to each class and building an ensemble of n binary classifiers. Each of the classifiers learns one bit for each position in the string. A new instance is classified to the class whose binary code is closest.

ADWIN has also been used in building ensemble of Restricted Hoeffding Trees [Bifet et al., 2010a]. A mechanism for setting the learning rate of perceptrons using ADWIN's change detection method is used to restrict the tree. Additionally, a mechanism for reseting the member Hoeffding trees is also been introduced when a particular member is no longer performing well. The method outperforms traditional bagging in terms of accuracy, but requires additional memory and time.

Chapter 2

Background

This chapter discusses the primitives of data and stream classification. First, a brief overview of traditional data classification methods are presented. Followed by a section on data stream classification where challenges and approaches for stream classification are introduces. Finally, overview of current state-of-the-art ensemble learning methods are discussed. These discussion are lays the foundation of the approach introduced in this thesis.

2.1 Learning Algorithms

The goal of data classification process is to predict a certain outcome based on some give data. In order to do so, data mining algorithms first processes a training set containing a set of attributes and corresponding outcome: a class or value. Algorithms develop hypotheses that best describe the relationship among the attributes and the outcome for the total instance set. Formally, learning are defined as follow: Given a data set of m instances $D \equiv \{\{\vec{x}_1, y_1\}, \{\vec{x}_2, y_2\}, \{\vec{x}_3, y_3\}, \ldots, \{\vec{x}_m, y_m\}\}$ for $i = \{1, 2, 3, \ldots, m\}$ and $\vec{x} = \{x_1, x_2, x_3, \ldots, x_n\}$, learning algorithms try to approximate $H \equiv y = f(\vec{x})$. Additional to being typical linear, polynomial, etc. functions, H can also be a set of IF-THEN rules. These rules or functions are then used to predict unseen instances where outcome class is not known. These algorithms are known as learning algorithms, and in last few decades, have widely been researched in the field of machine learning. This section briefly discusses few of the basic algorithms upon which the foundation of ensemble methods and this thesis are laid. The detailed discussion of these algorithms are out of the scope of this chapter. Thus we discuss only the core concept here.

2.1.1 k-Nearest Neighbors

K nearest neighbor algorithm, in short k-NN, is one the simplest learning available. It is a non-parametric instance-based lazy learning algorithm and works for both supervised and unsupervised learning. It uses similarity measure like distance functions to classify unknown instances. The decision is a majority voting of its k neighbors. For example, if $k = 1$ then an unknown is assigned the class label of its nearest neighbor. There are several

schemes to find the k nearest neighbours. For continuous data some well-known distance functions are Euclidean distance, Manhattan distance, Minkowski distance, etc.

- Euclidean distance $D_E = \sqrt{\sum_{ß=1}^{k}(x_i - y_i)^2}$
- Manhattan distance $D_M = \sum_{ß=1}^{k} |(x_i - y_i)|$
- Minkowski distance $D_{Min} \sqrt[q]{\sum_{ß=1}^{k}(|x_i - y_i|)^q}$

For categorical data, Hamming distance can be used. Hamming distance is defined as $D_H = \sum_{ß=1}^{k} |(x_i - y_i)|$ where $|x_i - y_i|$ is 0 if $x = y$, 1 otherwise.

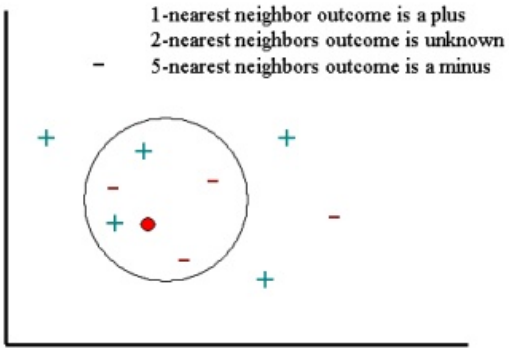

Figure 2.1: Concept of k-NN

Choosing an optimal k is the prime challange of $k-NN$ algorithm, which is extremely training data dependent. Changing position of few training points could led to a significant loss in performance. The method is particularly not stable in the class boundary. k should be large enough that k-NN could overcome the noises in data, and small enough that instances of other classes are not included. Generally, higher k reduces the overall noise and should be more precise. For most data set a value between 3-10 perfoms much better than 1-NN. Typically cross-validation is used to determine a good k.

2.1.2 Naïve Bayes

Naive Bayes classification is based on Bayes rule. Bayes rule says that the probability of event x conditioned on knowing event y, i.e. the probability of x given y is defined as

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(y|x)p(x)}{p(y)}$$

The naive Bayes classifier [Langley et al., 1992] assumes that all the explanatory variables are independent. Given a feature set $X = x_1, x_2, \ldots, x_n$ of n independent variables, naive Bayes assigns the instances to k possible outcome or classes, $p(C_k|X)$. Using Bayes rule, the conditional probability can be decomposed as

$$p(C_k|X) = \frac{p(C_k)p(X|C_k)}{p(X)}$$

In other words,

$$posterior = \frac{prior \times likelihood}{evidence}.$$

Using chain rule repetitively, it can be expressed as follows:

$$p(C_k|X) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

The predicted class is the one which maximizes the conditional probabilities $p(C_k|X)$, that is, classifier assigns the class label $\hat{y} = C_k$ for some k that satisfies:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{argmax}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

2.1.3 Decision Tree

Decision trees (DTs) are another popular genre of non-parametric supervised learning algorithms. Trees allow a way to graphically organize a sequential decision process. Decision tree, a directed acyclic graph, contains decision nodes, each with branches for all alternate decisions. Leaf nodes are labeled with respective class label. Each path from root to leaf is a decision rule, consisting of conditional part (unions of all internal nodes' conditions) and a decision (of leaf node).

Decision trees use *information gain* to select the splitting attribute that would maximize the total entropy of each of the subtrees resulting from the spit. Entropy is a measurement of purity of an attribute, typically ranged between 0 and 1. A pure attribute, attribute with definitive value-class relationship, would have a low entropy and is easy to predict. On the other hand attribute with highly mixed value-class relationship would yield high entropy. A common entropy function is-

$$entropy = -\sum_{i=1}^{n} p_i \lg p_i.$$

where $p_1, p_2, \ldots, p_n$ are the distributions of several class attributes and $\sum p_i = 1$. Information gain is the difference between old and new entropy after a split. This is good measurement of relevance of an attribute. However, in cases where an attribute can take on a large number of distinct values, information gain is not ideal. Presence of large set of values could uniquely identify the classes but also reduces chance to generalize unseen instance and should be avoided to be place near root.

Decision tree is, however, computationally feasible. Average cost of constructing a decision tree for n instances with m attributes is $O(mn \lg n)$, and querying time is $O(\lg n)$.

2.1.4 Neural Network

The Neural Network (NN) is a learning method inspired by biological neural network. A set of inter-connected nodes (also known as perceptrons and neurons) mimic biological neural network. Figure 2.2 shows a skeleton of a neural network. Neural network is a weighted

directed graph where a input layer is connected to the output layer via some layers of hidden layers. This is commonly known as Multi-Layer Perceptron (MLP). Decisions are made looking only into the output layer only. Hidden layers enlarge the space of hypotheses. If there is no hidden layer, then neural network becomes a simple regression problem i.e. output is a linear function of inputs. Each connection among these nodes has an associated weight. Given an input data set, these weights are updated accordingly to best fit the classification model. Learning is done by back-propagation algorithm [Rojas, 1996]. Errors are propagated back from the output layer to the hidden layer and weights are updated to minimize error.

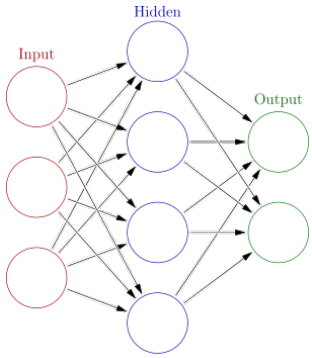

Figure 2.2: Neural network layers

Learning starts with assigning a small value as initial weights to all the connection weights. In case of a single perceptron, learning process is analogous to moving a parametric hyperplane around. Let w_i be the weight of i-th input, then after $t+1$ iteration weight $w_i(t+1) = w_i(t) + \nabla E_i(t)$, where ∇E_i is the gradient of the error function. For a input vector $\mathbf{x}$ and true output y, E is defined as the squared error:

$$E = \frac{1}{2}Err^2 = \frac{1}{2}(y - h_w(\mathbf{x}))^2$$

Thus gradient of E would be:

$$\begin{aligned}\nabla E = \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\sum_{j=0}^{n} w_j x_j)) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

Based on this, update rule would be:

$$w_j = w_j + \alpha \times Err \times g'(in) \times x_j$$

where α is learning rate coefficient. Back propagation algorithm for multi-layer perceptron also works in similar fashion. Weights in the output layer are updated using following equation:

$$w_{j,i} = w_{j,i} + \alpha \times a_j \times \nabla_i$$

where $\nabla_i = Err \times g'(in_i)$. Hidden layers propagates this error from the output layer using:

$$\nabla_j = g'(in_i) \sum_i w_{j,i} \nabla_i$$

$$w_{k,j} = w_{k,j} + \alpha \times a_k \times \nabla_j$$

MLP is a very expressive method. Only 1 hidden layer is sufficient to represent all continuous functions and 2 can represent any functions. MLP is prone to local minima, which is avoided by using different initial weight settings.

2.2 Data Stream Classification

Traditional data mining algorithms work in a memory bounded environment and requires multiple scan of the training data. In stream environment, one of the major assumptions is that new data samples are introduced in the system with such a high rate that repetitive analysis becomes infeasible. Thus, for stream classification, algorithms should be able to look into a instance only once and decide upon that. A bounded memory buffer can be used to facilitate some level of repetition. However, which instances are to remember and which are to forget would then become a decision choice. An alternate choice is to maintain sufficient statistics to have a representation of the data. The process of deletion or summarization of instances, however, means that some information are being lost over the time.

In this section, these challenges are first discussed in details. Then it presents the basis of some of the concepts arisen to handle these challenges. Finally, before moving onto the ensemble leaning, it discusses current state-of-the-art algorithms for stream mining.

2.2.1 Challenges

Challenges posed by the streaming environment can be categorized into two groups: (i) relating to runtime and memory requirements and (ii) relating to underlying concept identification. Speed of incoming data, unbounded memory requirement, single-pass learning fall into the first category. On the other hand, lack of labeled data, concept drifting, evolution and recurrence are examples of latter category.

Speed of data arrival As mentioned before, it is an inherent characteristic of data streams that it arrives with a high speed. The algorithm should be able to adapt to the high speed nature of streaming information. The rate of building the classifier model should be higher than the data rate. This gives a very limited amount of available time for classification as compared to the traditional batch classification models.

Memory requirements: To apply traditional batched approaches in streaming data, an unbounded memory would be needed. This challenge has been addressed using load shedding, sampling, aggregation, etc. Rather than storing all the instances, algorithms store a subset of the data set or some statistical values or a combination of both which

represents the data seen thus far. New instances can be classified only by looking into these stored information.

Single-pass learning: The premise of this requirement is two-fold. First, as mentioned above, data would not be available in the memory after a short period of time due to the volume of data. Secondly, even if the data remain available, running a batched-like approach for millions of data points would highly increase the running time of the algorithm. To attain faster processing time with limited storage, algorithm should access the data stream only once, or a small number of times. Mining models must posses the capability to learn the underlying nature of data in a single pass over the data.

Lack of labeled data: Unlike most data sets or settings of batched approaches, steam mining data sets are often poorly labeled. A large number of experimentations are done with generated data set where the data generation process can easily be controlled to have proper labeling. However, in data set collected from real world are often lack this. For example, to setup a supervised learning experimentation using a data set collected from social media, e.g. Twitter, data needs to be first categorized by human intervention. Manual labeling of such data is often costly, both in terms of resources and time. In practice, only a small fraction of data is labeled by human experts or automated scripts. A stream classification algorithm is thus required to be able to predict after observing a small number of instances, i.e. to be ready to predict anytime.

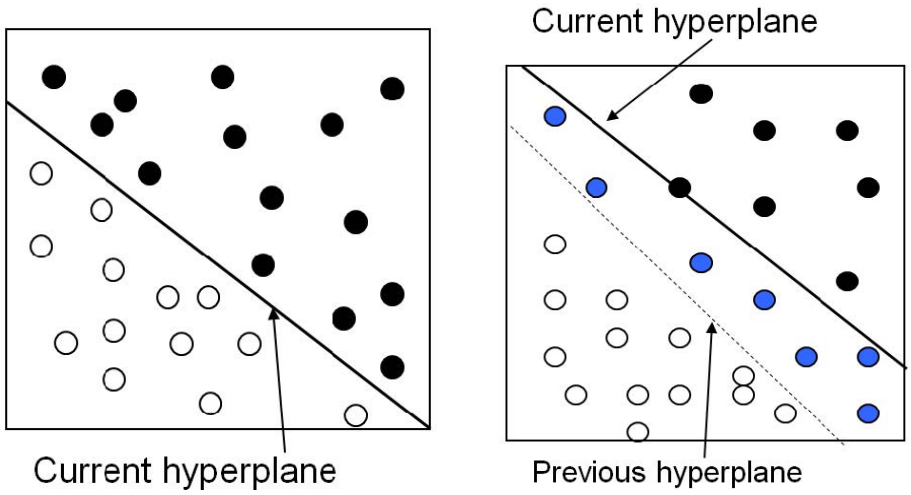

Figure 2.3: Concept drift in data streams.

Concept drift: Concept drift is a statistical property of data streams where the target variable drifts away from the model that is trying to predict it. In other words the underlying data distribution is changing over time. As a result accuracy of the classifier model decreases over time. For example, buying pattern of the customers in a store changes over time, mostly due to the seasonality. Electrical and mechanical devices wear off over time,

producing shifted result which would cause drift in the observing data. Learning models should adapt to these changes quickly and accurately. Let us consider the example in Figure 2.3. As a new chunk arrives (Figure 2.3a), a new classifier is learned. The decision boundary is denoted by the straight line. The positive examples are represented by unfilled circles while the negative examples are represented by filled circles. With time the concept of some of the examples may change. As shown in Figure 2.3b, due to concept drift some negative examples may have become positive. So, the previous decision boundary has become outdated and a new model has to be learned. Formally, concept drift is the change in the joint probability $P(X, y) = P(y|X) \times P(X)$. Thus, observing the change in y for given X, i.e. $P(y|X)$ is the key for detection.

One challenge posed here is to differentiate the noise in data and the actual shift of the concept. Often in streaming environment data contains the both. Rate of drift is also a factor in detection. Sudden drift, known as *concept shift*, is easier to detect than concept drift, which is considered to be gradual.

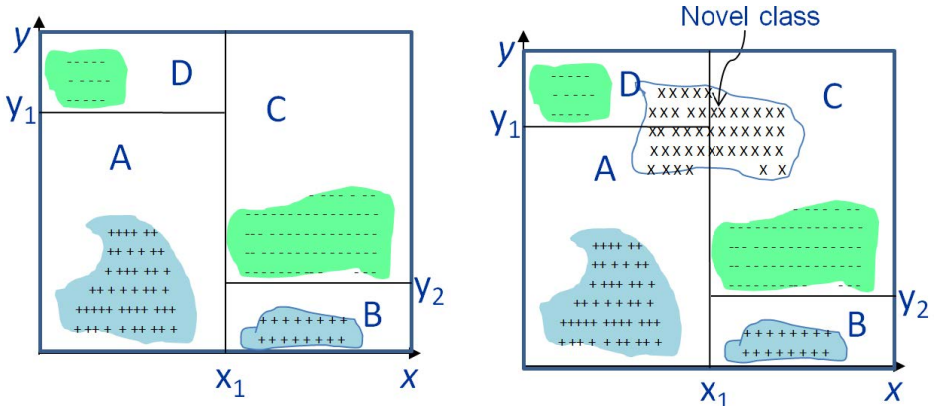

Figure 2.4: Concept evolution in data streams.

Concept evolution: Concept evolution is referred to the emergence of a new class or a set of classes in stream data as the time flows. Twitter stream is an ideal example where concept evolution is very easily identifiable. Twitter reacts, seemingly, very fast upon important news around the globe. Looking into the different hash tag usages in such social media currently trending topics can be identified. To present a clearer picture let's consider following example in Figure 2.4. At certain point of time four classes and their corresponding decision boundaries are shown in the Figure (2.4a). With the more incoming data a novel class emerges, and for that decision boundaries need updating. Emergence of new class can affect any number of decision boundary/ rule, from one to all.

Concept evolution is also prone to noise. Furthermore, clear distinction between drift and evolution might not always be possible, partially due the lack of unlabeled data.

Class recurrence: Class recurrence is a special case of concept drift and evolution. A this case, the model forgets a class due the drift, however, later the class reappears

(evolution) from the stream. Seasonality could be one cause of this situation. A intrusion in network traffic may reappear after a long time. Forgetting the earlier intrusions are not desired in such case. A fast recognition of the previously seen classes are desired in mining streams.

Following sections discuss the potential solutions to these challenges. First, how to address the limited resources and then the change detection schemes.

2.2.2 Maintaining Sufficient Statistics

In statistical evaluation, a statistic is sufficient for a family of probability distributions if the sample from which it is calculated gives no additional information than does the statistic, as to which of those probability distributions is that of the population from which the sample was taken [Fisher, 1922]. Mathematically, given a set $\mathbf{X}$ of independent identically distributed data conditioned on an unknown parameter θ, a sufficient statistic is a function $T(\mathbf{X})$ whose value contains all the information needed to compute any estimate of the parameter (e.g. maximum likelihood estimate). Using the factorization theorem (Theorem 2.2.1), for a sufficient statistic $T(\mathbf{X})$, the joint distribution can be written as $p(\mathbf{X}) = h(\mathbf{X})g(\theta, T(\mathbf{X}))$. From this factorization, it can easily be seen that the maximum likelihood estimate of θ will interact with $\mathbf{X}$ only through $T(\mathbf{X})$. Typically, the sufficient statistic is a set of function or random variables of the data.

Theorem 2.2.1 (Factorization Theorem) *Let $X_1, X_2, \ldots, X_n$ be a random sample with joint density $f(x_1, x_2, \ldots, x_n|\theta)$. A statistic $T = r(X_1, X_2, \ldots, X_n)$ is sufficient if and only if the joint density can be factored as follows:*

$$f(x_1, x_2, \ldots, x_n|\theta) = u(x_1, x_2, \ldots, x_n)v(r(x_1, x_2, \ldots, x_n), \theta)$$

where u and v are non-negative functions. The function u can depend on the full random sample $x_1, x_2, \ldots, x_n$, but not on the unknown parameter θ. The function v can depend on θ, but can depend on the random sample only through the value of $r(x_1, x_2, \ldots, x_n)$.

Bounds of Random Variable

A random variable is a variable that can take a set or range of values, each with an associated probability, and are subjected to change due to the alteration or randomness of the data. Random variables are of two types: (i) discrete, and (ii) continuous. Discrete random variable take a set of possible values (e.g. outcome of coin flipping), but a continuous random variable can take any value within a range (e.g. age of people in a randomly sampled group).

A function that is used to estimating a random variable is called an estimator. Estimator function is dependent on the observable sample data, and used for estimating unknown population within an interval with certain degree of confidence. For an interval of the true value of the parameter associates with a confidence of $1 - \delta$, interval can be defined as follows:

- Absolute approximation: $\bar{X} - \epsilon \leq \mu \leq \bar{X} + \epsilon$, where ϵ is the absolute error.
- Relative approximation: $(1 - \delta)\bar{X} \leq \mu \leq (1 + \delta)\bar{X}$, where is the relative error.

where μ and $\bar{X}$ represent actual and estimated mean. There are a number of theorems that provide bounds on the estimation, Chebyshev [!], Chernoff [!], Hoeffding [Hoeffding, 1963], etc. are few of them.

Theorem 2.2.2 (Chebyshev) *Let X be a random variable with standard deviation σ, the probability that the outcome of X is no less than $k\sigma$ away from its mean is no more that $1/k^2$:*

$$P(|X - \mu| \leq k\sigma) \leq \frac{1}{k^2}$$

In other words, it states that no more that $1/4$ of the values are more than 2 standard deviation away, no more than $1/9$ are more than 3 standard deviation away, and so on.

Theorem 2.2.3 (Chernoff Bound) *Let $X_1, X_2, \ldots, X_n$ be independent random variables from Bernoulli experiments. Assuming that $P(X_i = 1) = p_i$. Let $X_s = P_n \sum_{i=1} nX_i$ be a random variable with expected value $\mu_s = P \sum_{i=1} np_i$. Then for any $\delta > 0$:*

$$P[X_s > (1 + \delta)\mu_s] \leq (\frac{e^{\delta}}{(1 + \delta)^{1+\delta}})^{\mu_s}$$

and the absolute error is:

$$\epsilon \leq \sqrt{\frac{3\bar{\mu}}{n} \ln(2/\delta)}$$

Theorem 2.2.4 (Hoeffding Bound) *Let $X_1, X_2, \ldots, X_n$ be independent random variables. Assuming that each x_i is bounded, that is $P(X_i \in R = [a_i, b_i]) = 1$. Let $S = 1/n \sum_{i=1} nX_i$ whose expected value is $E[S]$. Then, for any $\epsilon > 0$:*

$$P[S - E[S] > \epsilon] \leq e^{\frac{2n^2\epsilon^2}{R^2}}$$

and the absolute error is:

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

Chernoff and Hoeffding bounds are independent of the underlying distribution of examples. They are more restrictive or conservative, and requires more observations as compared to the distribution dependent bounds. Chernoff bound is multiplicative and Hoeffding is additive. They are expressed as relative and absolute approximation, respectively.

These methods only take a finite number of values or a range. One of the well-known methods supports infinity is Poisson process. A random variable x is Poisson random variable with parameter λ if x takes values $0, 1, 2, \ldots, \infty$ with:

$$p_k = P(x = k) = e^{-\lambda} \frac{\lambda^k}{k!} \tag{2.1}$$

where, λ is both mean and variance, i.e. $E(X) = Var(X) = \lambda$

Recursive Mean, Variance, and Correlation

Fundamental equation of mean, variance, etc. are not usable for streams as past data points are lost as the time passes. However, their recursive version are also known. Equation 2.2, 2.3, and 2.4 can be used to recursively compute mean, variance, and correlation respectively.

$$\bar{x}_i = \frac{(i-1) \times \bar{x}_{i-1} + x_i}{i} \tag{2.2}$$

$$\sigma_i = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{i}}{i-1}} \tag{2.3}$$

$$corr(a,b) = \frac{\sum(x_i \times y_i) - \frac{\sum x_i \times \sum y_i}{n}}{\sqrt{\sum x_i^2 - \frac{\sum x_i^2}{n}}\sqrt{\sum y_i^2 - \frac{\sum y_i^2}{n}}} \tag{2.4}$$

As it can be seen from the equations, maintaining (i) number of observations, n; (ii) $\sum x_i$, sum of i data points; (iii) $\sum x_i^2$, sum of squares of i data points; and (iv) $\sum(x_i \times y_i)$, sum of cross product of X and Y are enough to recursively compute these statistics.

Windowing

Windowing the process of selecting a subset of the observed instances that would be remembered to be used in the computation of statistics. Where the data set is finite and of limited size, all the instances can be remembered. For streams, this is not possible. Furthermore, computing statistics over all the instances of the past, in streaming environment would wrongly introduce information of classes that are not needed in the present. Thus, information of recent past is more important than the entire set. Windowing is categorized in two basic types: (i) sequence based windowing, and (ii) time-stamp based windowing.

In sequence based windowing, sequence is based on the number of observations seen; in time-stamp based approach, it is elapsed time. Landmark windowing and sliding windowing are two most used sequence based windowing system.

Landmark Windowing: All observations after certain start point are remembered. Batched approaches can be thought of as examples of landmark windowing where every instance is remembered from the very first one. As new observations are seen, size of window increases. Landmark needs updating time to time to ensure recency of the statistics.

Sliding Windowing: A fixed length window is moved through the observation set. As a new observation is seen, oldest observation is forgotten, i.e., when j-th instance is pushed into the window, $(j-w)$-th instance is forgotten, where w is the size of the window. A limitation of sliding windowing is that it requires all elements within the window to be remembered, as it is needed to forget the oldest observation.

Often it is more useful to learn about most recent updates with fine granularity and older ones in a summarized fashion. With this motivation concept of tilted-time windowing

was introduced [Chen et al., 2002].

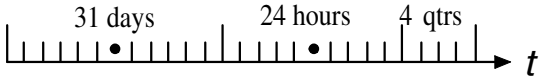

Figure 2.5: Natural Tilted Time Window

Natural Tilted Time Windowing: In natural tilted time windowing, units of time is distributed non-uniformly. Most recent time gets more units. For example, Figure 2.5 shows a natural tilted time windowing scheme, where for most recent hour 4 units stores quarterly updates, 24 units of time storing a day, and 31 units storing a months summary. That is, with 59 units of time information, this model stores more than 32 days' information.

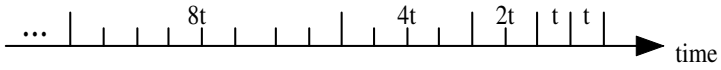

Figure 2.6: Logarithmic Tilted Time Window

Logarithmic Tilted Time Windowing: Concept for logarithmic tilted time windowing is same as natural tilted time windowing. The only deference here is that time scale grows in logarithmic order. Figure 2.6 shows an example.

2.2.3 Change Detection

An assumption of most machine learning methods is that data is generated from a stationary distribution. As discussed in the previous section, this assumption does not hold for streaming scenario. Thus stream mining requires algorithms to facilitate drift detection methods. Differentiating between *noise* and *change* makes the problem challenging. The difference between a new distribution and noise is *persistence*, where new examples consistently follows new distribution rather than old. This section discusses several methods to detect and adapt learning algorithms in presence of concept drift.

Detection

Detection methods can generally be classified into two categories. First approach is to monitoring the evolution of various performance indicators as done in [Klinkenberg and Renz, 1998, Zeira et al., 2004]. Another approach is to maintaining two (or more) distributions varying the window length. Typically one window would summarize past history while the other would summarize most recent information [Kifer et al., 2004].

Most methods follows the first approach. [Klinkenberg and Renz, 1998] monitors three performance indicators (accuracy, recall, and precision) over time, and uses their posterior

comparison to a confident interval of standard sample errors for a moving average value for each indicator.

A classical algorithm for change detection is Cumulative Sum (CUSUM) [Page, 1954]. CUSUM can detect that the mean of the input data is significantly different than zero. The test is as follows:

$$g_0 = 0$$

$$g_t = max(0, g_{t-1} + (r_t - v))$$

if $g_t > \lambda$ CUSUM triggers an alarm and set $g_t = 0$. This detects the changes in the positive direction, to detect the negative change min is used instead of max. CUSUM does not require any memory. Its accuracy depends on the choice of v and λ. Low v results in faster detection with more false alarms.

For latter category, [Kifer et al., 2004] uses Cherfnoff bound (Theorem 2.2.3) and examines examples drawn from two probability distribution and decides whether these distributions are different.

ADWIN Algorithm: As introduced in previous chapter, ADWIN (ADaptive sliding WINdow) is another change detection algorithm of latter type. ADWIN keeps a variable length window of recent items. It ensures the property *there has been no change in the average value inside the window* for maximally statistically consistent length. The core idea of ADWIN is that whenever two *large enough* sub-windows of W exhibit *distinct enough* averages, it is assumed that their corresponding expected values are different, and the older portion of the window is dropped. Essentially, this means that when the difference of means of the two windows is greater than a certain threshold ϵ_{cut}, the older portion should be dropped. Equation to compute ϵ_{cut} is as follows:

$$m = \frac{2}{1/|W_0| + 1/|W_1|}$$

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \ln \frac{4|W|}{\delta}}$$

where $\delta \in (0, 1)$ is confidence value, an input.

ADWIN does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length w using only $O(\lg w)$ memory and $O(\lg w)$ processing time per item.

Adaptation to Change

To improve accuracy of the decision model under concept drifting environment, decision model needs adaptation to the change. There are two types of approaches based on when to adapt:

- Blind or Periodic Methods: Models are updated on regular interval whether an changes have actually occurred or not.

Algorithm 1: ADWIN Algorithm

Data: Data Stream
Result: Window with most recent concept
begin
  Initialize window W
  foreach $t > 0$ **do**
    $W \leftarrow W \cup \{x_t\}$ // add x_t to the head of W
    repeat
      Drop element from the tail of W
    until $|\mu_{W_0} - \mu_{W_1}| < \epsilon_{cut}$ *holds for every split of* W
    $W = W_0.W_1$
  Output μ_W

- Informed Methods: Models are only updated when there are sufficient reasons to believe that changes in the concept have occurred.

It could a good idea to use periodic methods where duration of seasonality is known beforehand. Otherwise, chosen duration could be too large or too small to response to the change. In such cases, an informed decision is more desired. However, informed methods requires more resources (than blind methods) to be able to make such decision.

2.2.4 Naïve Bayes Adaptation

Naive Bayes algorithm is essentially a stream classification algorithm. One of the advantages of this classifier in the context of data stream is its low complexity for deployment. It only depends on the number of explanatory variables. Its memory consumption is also low since it requires only one conditional probability density estimation per variable.

2.2.5 Very Fast Decision Tree

Very Fast Decision Tree (VFDT), also known as Hoeffding Tree (HT), is the stream adaptation of decision tree generating from a stationary distribution. It is an anytime-ready algorithm and uses Hoeffding bounds to ensure the performance in terms of accuracy is asymptotically nearly identical to that of conventional tree algorithms. VFDT runs on constant time and memory per examples and can serve thousands of examples on a typical consumer system.

VFDT constructs the tree by recursively replacing leaves with decission nodes. Each leave stores sufficient statistics that are used to evaluate the merit of split-test and to decide the target class. For incoming instances, the tree is traversed from the root to a leaf (based on the incoming instance's values) and the statistics are updated accordingly. If a unanimous decision cannot reached based on observed instances at that particular leaf node, the node is tested to check the sufficiency for a split. If there is enough statistical support in favor of split, the node is splitted on the best attribute and stats are passed to the descendants (new leaves). Number of descendants of this new decision node is equal

Algorithm 2: VFDT: The Hoeffding Tree Algorithm

Input : S: Stream of examples
X: Set of nominal attributes
Y: Set of class labels $Y = \{y_1, y_2, \ldots, y_k\}$
$G(.)$: Split evaluation function
N_{min}: Minimum number of examples
δ: is one minus the desired probability
τ: Constant to resolve ties
Output: HT: is a decision tree

```
2.1  begin
2.2      Let HT ← Empty Leaf (Root) foreach example(x, y_k) ∈ S do
2.3          Traverse the tree HT from root till a leaf l
2.4          if y_k ==? then                                  // Missing class label
2.5              Classify with majority class in the leaf l
2.6          else
2.7              Update sufficient statistics
2.8              if Number of examples in l > N_min then
2.9                  Compute G_l(X_i) for all attributes
2.10                 Let X_a be the attribute with highest G_l
2.11                 Let X_b be the attribute with second highest G_l
2.12                 Compute ε = sqrt(R^2 ln(2/δ) / 2n)       // Hoeffding bound
2.13                 if G(X_a) − G(X_b) > ε || ε < τ then
2.14                     Replace l with a splitting test based on attribute X_a
2.15                     Add a new empty leaf for each branch of the split
2.16     Return HT
```

to the number of possible values of the chosen attribute. Thus, the tree is not necessarily a binary tree.

Deciding on whether to split a node or not is a unique contribution of VFDT. VFDT solves this difficult problem of deciding exactly how many examples are required to be observed by a leaf node before splitting by using Hoeffding bound (Theorem 2.2.4). Consider $G(.)$ be the heuristic measure of the attributes. This measure could be information gain as of C4.5 or Gini index of CART. For information gain range R in Hoeffding bound is $\lg(\#classes)$. Goal is to find n such that the attribute chosen for split after observing n instances, would, with high probability, be the same as it would be chosen after observing infinite instances. Assume that X_a is the attribute with the best $G(.)$, and X_b is the second best attribute after observing n instances. Then $\Delta G = G(X_a) - G(X_b)$ be the difference between their observed heuristics. From Hoeffding bound, we know if $\Delta G < \epsilon$, then with probability $1 - \delta$ X_a would be the attribute with highest value in the evaluation function in the universe. Otherwise, if $\Delta G < \epsilon$, then the sample size is not enough to make a stable split decision. As the assumption is that underlying generating distribution is stationary, thus as the sample size increases, ϵ decreases and heuristic value for most informative attribute goes up.

To fasten up the process, VFDT uses an extra parameter N_{min} to reduce the number of

$G(.)$ computation. Computing $G(.)$ when there is too few instances is run-time inefficient. Thus a user parameter N_{min} is used to indicate minimum number of instances needed to be observed before the evalution starts.

When multiple attributes continuously performs similar in heuristic evaluation after observing a large number of examples, ΔG might never be greater than ϵ. To break such tied cases, another user parameter *tau* is used, and when ϵ falls below τ (i.e. $\Delta G < \epsilon < \tau$), algorithm splits on the best attribute. Algorithm 2 summarizes the pseudocode of VFDT.

There are some property of VFDT that are different than C4.5. In contrast to the C4.5 algorithm number of examples that support a decision increases in VFDT. Typically VFDT also results a low variance model than that of C4.5. However, there is no mechanism avoid overfitting in VFDT as there is no room for pruning.

Algorithm 3: CVFDT: Concept-adapting VFDT

Input : S: Stream of examples
X: Set of nominal attributes
Y: Set of class labels $Y = \{y_1, y_2, \ldots, y_k\}$
$G(.)$: Split evaluation function
δ: is one minus the desired probability
τ: Constant to resolve ties
w: Size of the window
n_{min}: Number of examples between checks for growth
f: Number of examples between checks for drift

Output: HT: is a decision tree

```
3.1  begin
3.2      Let HT ← Empty Leaf (Root) l_0
3.3      Let Alt(l_0) ← ∅                                  // Alternate trees for l_0
3.4      foreach class y_k do
3.5          foreach x_ij ∈ X_i ∈ X do
3.6              Set n_ijk = 0
3.7      foreach example(x, y) ∈ S do
3.8          Traverse the tree HT including Alt({n : (x, y) passes through n in HT})
             trees root till a set of leaves L
3.9          Set id = max({L.id})
3.10         Add ((x, y), id) to the beginning of W
3.11         if |W| > w then
3.12             Let ((x_w, y_w), id_w) be the last element in W
3.13             FORGET(HT, n, (x_w, y_w), id_w))
3.14             W = W − ((x_w, y_w), id_w)
3.15         GROW(HT, n, G, (x, y)
3.16         if examples seen since last checking > f then
3.17             VALIDATE_SPLIT(n)
3.18     Return HT
```

2.2.6 Concept-adapting Very Fast Decision Tree

Concept-adapting Very Fast Decision Tree (CVFDT) is the extension of CFDT that adds the ability to adapt the model by detecting the changes in the underlying distribution that generates the examples. CVFDT does not require the model to be recomputed, instead it updates the sufficient statistics stored in each node that the new instances affects by maintaining a sliding window. It increases the counter of new instances and decreases count of oldest examples which now needs to be forgotten. If the underlying distribution is stationary, this would not have any effect. But in case of a concept drifting distribution, some decision nodes that previously had passed the Hoeffding bound test will no longer pass, rather an alternate attribute would now have higher or similar gain. CVFDT keeps stats to detect such situation and starts maintaining an alternate subtree with the new best attribute as its root. When this alternate subtree becomes more accurate on new data then the old subtree is replaced by the new one. Pseudocode of CVFDT has be shown in Algorithm 3. The sub-routine $GROW$ is essentially the Hoeffding Tree algorithm where instead of only keeping stats in the leave, every node keeps track of the instances it has seen. Two other sub-routines $FORGET$ and $VALIDATE\_SPLIT$ have been shown separately.

Algorithm: FORGET($HT, n, (x_w, y_w), id_w$)
begin
3.13.1 Sort (x_w, y_w) through HT while it traverses leaves with $id \leq id_w$
3.13.2 Let P is the set of sorted nodes
3.13.3 **foreach** *node* $l \in P$ **do**
3.13.4 **foreach** $x_{ij} \in x : X_i \in X_l$ **do**
3.13.5 Decrement $n_{ijk}(l)$
3.13.6 **foreach** *tree* $t_{alt} \in Alt(l)$ **do**
3.13.7 FORGET($t_{alt}, n, (x_w, y_w), id_w$)

$FORGET$ function is used to remove the effect of instances of older concepts. However, this is not straight-forward as HTs go through changes after the instance to be forgotten was initially added. Thus, CVFDT uses monotonically increasing ids for the nodes. When a new instance is added to the window (W), the maximum Id of the leaves it reaches in HT and all the alternate trees is recorded with it. To remove the effect of a old instance, counts are decremented at every node the example reaches in HT whose Id is less than the stored Id.

Lastly, there is $VALIDATE\_SPLIT$ sub-routine that periodically checks the internal nodes of HT where current split attribute would no longer have the highest $G(.)$ or ΔG would be less than ϵ or ΔG would be less than $\tau/2$. This is similar condition as to the original one with more restrictive tie condition. The added restriction ensures less alternate tree creation.

Now that related concept of basic learning algorithms and stream mining are discussed, in the next section, the concepts of ensemble learning are presented.

Algorithm: VALIDATE_SPLIT($HT, n, (x_w, y_w), id_w$)
begin
3.17.1 Let $HT \leftarrow$ Empty Leaf (Root) l_0
3.17.2 Let $Alt(l_0) \leftarrow \emptyset$ // Alternate trees for l_0
3.17.3 **foreach** *internal node* l *in* HT **do**
3.17.4 **foreach** $t_{alt} \in Alt(l)$ **do**
3.17.5 VALIDATE_SPLIT($t_{alt}, n, (x_w, y_w), id_w$)
3.17.6 **foreach** $x_{ij} \in X_i \in X$ **do**
3.17.7 Set $n_{ijk} = 0$
3.17.8 Let X_a be the current split attribute
3.17.9 Compute $\Delta(G) = G_l(X_n) - G_l(X_b)$ // X_n, X_b two current highest attributes
3.17.10 **if** $\Delta(G) \geq 0 \&\& X_n \notin \{rootsof Alt(l)\}$ **then**
3.17.11 Compute $\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}}$ // Hoeffding bound
3.17.12 **if** $\Delta(G) > \epsilon \;||\; \epsilon < \tau \&\& \Delta(G) \geq \tau/2$ **then**
3.17.13 Let l_{new} be an internal node that splits on X_n
3.17.14 $Alt(l) = Alt(l) + l_{new}$
3.17.15 **foreach** *branch of the split* **do**
3.17.16 Add a new leaf l_m to l_new
3.17.17 $X_m = X - X_n$
3.17.18 $Alt(l_m) =$
3.17.19 Compute $G_m(X_\emptyset)$ using most frequent class at l_m
3.17.20 **foreach** *node* $l \in P$ **do**
3.17.21 **foreach** $x_{ij} \in x : X_i \in X_l$ **do**
3.17.22 Decrement $n_{ijk}(l)$

2.3 Ensemble Learning

Ensemble learning, due to its intrinsic merits, focuses to get the best out of a collection of base learners. Ensemble learning can be thought of as a divide-and-conquer approach. However, it may not necessarily divides the tasks rather does more tasks. The motivation behind was presented in the previous chapter. In short, ensemble methods are interesting because of following possibilities:

- Avoiding worst classifier by averaging several classifier.
- Fusing multiple classifier to improve performance of the best classifier.
- Arranging the better classifiers (might include the best one) in a way to outperform the best one.
- Dividing streams into chunks, learn separate models from each and then combine their results (divide and conquer).

A number of methods has been developed in past couple of decades focusing on these motivations. They employ different approaches to improve the final classifier. Following is a summary of different approaches of generating ensembles of classifiers.

- Creating multiple training sets by resampling the original set and feeding those into different learners.
- Using different combination of features to learn multiple classifiers.
- Manipulating class labels. For example, transforming the classes into binary classification problem by partitioning the class labels into disjoint subsets.
- Manipulating the learning algorithms such that they result different outcomes for the same training set. For example, introducing certain randomness into tree growing algorithm.

In this section, we present these basic concepts of ensembling a collection of classifier. We start with primitive methods such as bagging and boosting. However, we are particularly interested in more sophisticated methods based on decision tree learning methods of stream data such as Adaptive Size Hoeffding Tree (ASHT) and ADaptive WINdow bagging (ADWIN bagging).

2.3.1 Bagging

Bagging, also known as bootstrap aggregating, is a meta algorithm that improves accuracy and reduces variance and chance of over-fitting. Bagging is typically used with tree based learners. Given a training set D of size n, bagging generates m new training sets D_i of size n_{new} where $i = \{1, 2, \ldots, m\}$ by sampling with replacement. If $n = n_{new}$ approximately two-thirds of the instances of D_i is expected to be unique examples of D while rest being duplicates. If K is the number of examples belonging from the original training set then $P(K = k) = \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k}$. These m sets are then used to learn m classifier. Final decision is given by a majority voting scheme over the decision of these m classifiers. Figure 2.8 shows an illustrative example of bagging method.

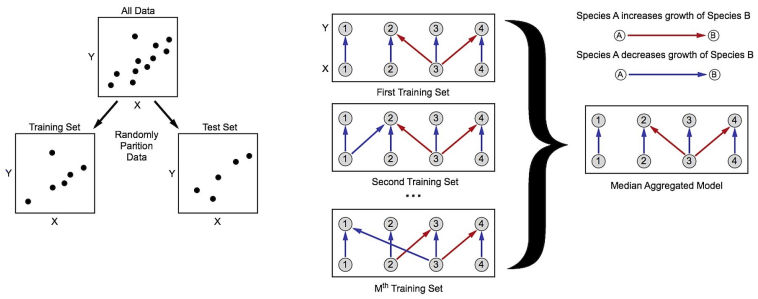

Figure 2.7: Bagging or bootstrap aggregation

For online bagging approach $n \to \infty$, and the Binomial distribution of $P(K = k)$ tends to a Poisson distribution with $P(K = k) = \exp(-1)/k!$. Using this justification, online bagging chooses a k $Poisson(1)$ for each incoming training examples (x_i, y_i), and updates the base learning learning algorithms k times. The classification is done as the same way of batched approach, by unweighted voting of m base classifiers. For similar distribution of training examples, online bagging produces similar approximated base models. If (i) used

based learner converges to the same same classifier with increasing training examples, and (ii) base learner produces same classifier given a fixed training set for both batched and online bagging methods; then online bagging will converge to the classifier obtained though batched learning methods.

2.3.2 Boosting

Boosting is another supervised learning algorithm to iteratively improve learning hypothesis. The motivation behind the boosting approach is that a set of weak classifier could create a single strong learner. It forces a weak classifier to update or generate new rules that make less mistakes on previously misclassified records. Initially all records are assigned same weights. At the end of a boosting round, weights might change due to the errors in classification. Weights of the instances are increased or decreased if they are classified wrongly or correctly, respectively. Thus successive classifiers depend upon their predecessors.

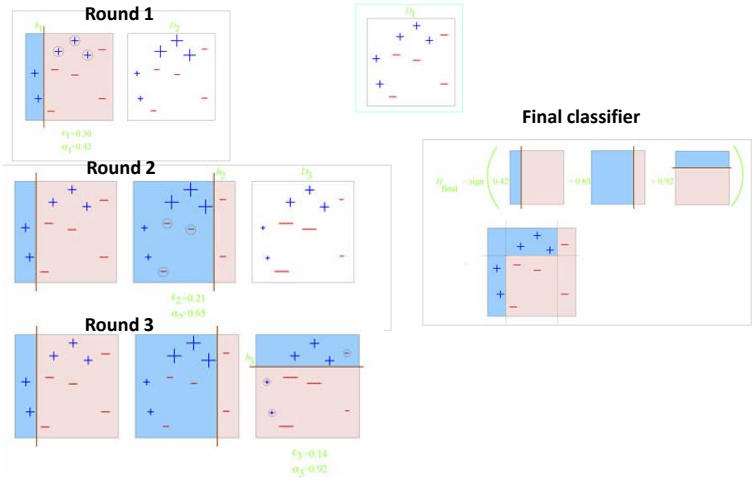

Figure 2.8: Boosting method example

Mathematically, let $H = \{h_1, h_2, \ldots, h_n\}$ be n weak hypotheses. The combined hypothesis $H(.)$ is a weighted majority vote of the n weak hypotheses where each hypothesis h_i has a weight of α_i for $i = \{1, 2, \ldots, n\}$:

$$H(.) = sign\left(\sum_{i=1}^{n} \alpha_i h_i(x)\right). \tag{2.5}$$

One of the earliest boosting approach was proposed in [Schapire, 1990]. It calls weak learners three times on three modified distributions and achieves slight improvement in accuracy. Later, [Freund and Schapire, 1997] proposed AdaBoost, an adaptive boosting method, that works with a principle of minimizing upper bound of empirical error. Let, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ are given examples where $x_i \in X, y_i \in Y = [-1, +1]$. Let, $D_t(i)$ be the weight of i-th example at t-th round. Then, AdaBoost initializes $D_1(i) = 1/n$. Iterative steps are as follows:

For each iteration $t = 1, 2, \ldots, T$:

- Train weak learner using distribution D_t
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = Pr_{i\ D_t}[h_t(x_i) \neq y_i]$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor

Finally, the output is calculated with the Equation 2.5 given above.

An online variant of AdaBoost uses a similar approach as online bagging method, simulating sampling with replacement using Poisson distribution. However, the Poisson parameter λ (Equation 2.1) associated with an example is increased if that particular example is misclassified. In case of correct classification λ is decreased. Similar to the AdaBoost, online boosting approach assigns total weights equally to the correctly and misclassified instances, i.e. half of the total weights each. Unlike AdaBoost, in online boosting weights are updated based on only the examples seen thus far rather than the complete training set. This is intuitively problematic as initial hypotheses are build on too few examples. Even with this limitation, online boosting shows good performance. Online boosting with naive Bayes base learner converges to the model achieved with AdaBoost as the number of training instances tends to infinity.

2.3.3 Adaptive-Size Hoeffding Tree (ASHT) Bagging

In previous section (Section 2.2.5), we have introduced Very Fast Decision Tree (VFDT) or Hoeffding Tree (HT). Hoeffding Tree is inspired by the fact that a small sample size could be enough to effectively choose an optimal splitting attribute. An upper bound of the error introduced because of such generalization is given using Hoeffding bound.

The Adaptive-Size Hoeffding Tree (ASHT) is, as the name suggests, extended from Hoeffding tree with deletion of nodes or reseting of the tree capabilities. Following are two significant differences of adaptive-size Hoeffding tree with Hoeffding tree:

- Maximum number of split nodes or the size of the tree is bounded in ASHT. There is no such limit on HT. HT grows indefinitely as the data and new concepts are introduced.
- When number of split nodes exceeds the maximum value (essentially after a new split), some nodes are deleted to retain the tree property (max size).

There are two different choices for the deletion of nodes for the second point. First option is to delete the oldest node. In Hoeffding tree, root is always the oldest node. All children of root except for the nodes that were further splitted are also deleted in this case. The new root would be the root of the child not being deleted. Another and more crude approach would be to delete the complete altogether i.e. reseting the tree.

Based on this modified Hoeffding tree, a bagging method is developed in [Bifet et al., 2009]. The intuition of the methods is as follows: smaller trees adapts

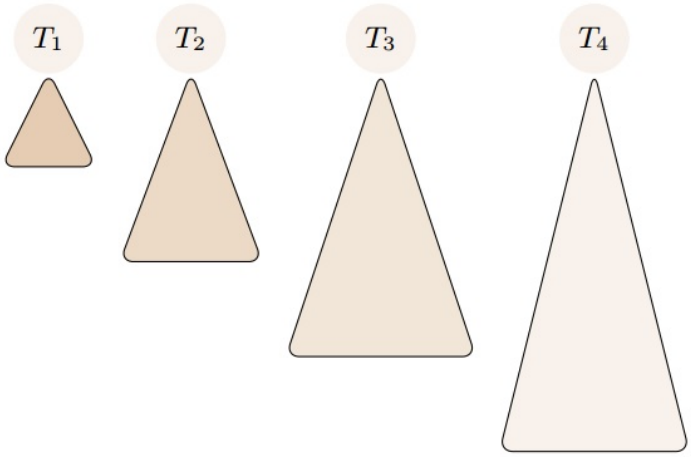

Figure 2.9: Adaptive Size Hoeffding Tree concept

to changes faster than the larger trees. However, larger trees perform better where data has only small changes or drifts because of the fact the it is built with more data. A tree of size s would expected to be reset twice as often as a tree with size $2s$. Using an ensemble of different sizes would thus give a set of trees with different reset speeds. Smaller trees would be updated for smalls changes in the streams, while larger ones will maintain a history for longer period. Polling over this set of classifier would thus expected to result in a classifier with finer granularity. It is to be noted that reset will occur even for stationary data, however, this should not have any negative influence on the ensemble's predictive performance.

The proposed bagging method in [Bifet et al., 2009] uses n adaptive-size Hoeffding trees. The n-th ASHT has twice the maximum size of $(n-1)$-th tree. Size of the smallest tree is 2. For the polling, each tree is associate with a weight, which is selected to be the inverse of squared error.

2.3.4 ADWIN Bagging

ADWIN Bagging combines three different powerful concepts together: (i) bagging (Section 2.3.1), (ii) adaptive window change detection (Section 2.2.3), and (iii) Hoeffding tree (Section 2.2.5).

Bagging using ADWIN is implemented as ADWIN Bagging where the Bagging method is the online bagging method of Oza and Rusell [Oza and Russell, 2001] with the addition of the ADWIN algorithm as a change detector. Hoeffding Trees are used as base classifier. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

Bibliography

[Aggarwal et al., 2003] Aggarwal, C., Han, J., Wang, J., and Yu, P. (2003). Clustream: A framework for clustering evolving data streams. In *VLDB*.

[Aggarwal et al., 2004] Aggarwal, C., Han, J., Wang, J., and Yu, P. (2004). On-demand classification of data stream. In *ACM KDD*, pages 503–508.

[Aggarwal, 2003] Aggarwal, C. C. (2003). A framework for diagnosing changes in evolving data streams. In *ACM SIDMOD*, pages 575–586.

[Bifet et al., 2010a] Bifet, A., Frank, E., Holmes, G., and Pfahringer, B. (2010a). Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. 13:225–240.

[Bifet et al., 2010b] Bifet, A., Holmes, G., and Pfahringer, B. (2010b). Leveraging bagging for evolving data streams. In *ECML PKDD*, pages 135–150.

[Bifet et al., 2009] Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., and Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *SIGKDD*, pages 139–148.

[Breiman, 1993] Breiman, L. (1993). Stacked regression.

[Breiman, 1994] Breiman, L. (1994). Bagging prediction.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and regression trees.

[Brzezinski and Stefanowski, 2011] Brzezinski, D. and Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In *HAIS*, pages 155–163.

[Castillo et al., 2003] Castillo, G., Gama, J., and Medas, P. (2003). Adaptation to drifting concepts. In *Progress in Artificial Intelligence*, pages 279–293.

[Catlett, 1991] Catlett, J. (1991). Megainduction: Machine learning on very large databases. In *PhD thesis, University of Sydney.*

[Chen et al., 2002] Chen, Y., Dong, G., Han, J., Wah, B. W., , and Wang, J. (2002). Multidimensional regression analysis of time-series data streams. In *VLDB*, pages 323–334.

[Dasu et al., 2004] Dasu, T., Krishnan, S., Venkatasubramanian, S., and Yi, K. (2004). An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Duke University Technical Report CS-2005-06*, pages 180–191.

[Ding et al., 2002] Ding, Q., Ding, Q., and Perrizo, W. (2002). Decision tree classification of spatial data streams using peano count trees. In *ACM Symposium on Applied Computing*, pages 413–417.

[Domingos and Hulten, 2000] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the ACM KDD*.

[Drucker et al., 1994] Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., and Vapnik, V. (1994). Boosting and other ensemble methods. 6(6):1289–1301.

[Fisher, 1922] Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. 222:309–368.

[Freund and Schapire, 1997] Freund, Y. and Schapire, R. (1997). A decision theoretic generalization of on-line learning and an application to boosting. 55(1):119–139.

[Gama et al., 2004a] Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004a). Learning with drift detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295.

[Gama et al., 2004b] Gama, J., Medas, P., and Rocha, R. (2004b). Forest trees for on-line data. In *Symposium on Applied computing*, pages 632–636.

[Gama et al., 2005] Gama, J., Medas, P., and Rodrigues, P. (2005). Learning decision trees from dynamic data streams. In *Symposium on Applied computing*, pages 573–577.

[Gama et al., 2003] Gama, J., Rocha, R., and Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In *SIGKKD*, pages 523–528.

[Ganti et al., 2002] Ganti, V., Gehrke, J., and Ramakrishnan, R. (2002). Mining data streams under block evolution. 3(2):1–10.

[Hansen and Salamo, 1990] Hansen, L. K. and Salamo, P. (1990). Neural network ensembles. 12(10):993–1000.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. 58:13–30.

[Hulten et al., 2001] Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time changing data stream. In *ACM KDD*, pages 97–106.

[Kifer et al., 2004] Kifer, D., Ben-David, S., and Gehrke, J. (2004). Detecting changes in data streams. In *VLDB*, pages 180–191.

[Klinkenberg and Renz, 1998] Klinkenberg, R. and Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drift. In *Learning for Text Categorization*, pages 33–40.

[Krogh and Vedelsby, 1995] Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation and active learning. pages 231–238.

[Langley et al., 1992] Langley, P., Iba, W., and K.Thompson (1992). An analysis of bayesian classifiers. In *National Conference of Artificial Intelligence*, pages 223–228.

[Last, 2002] Last, M. (2002). Online classification of non-stationary data streams. 6(2):127–147.

[Mehta et al., 1996] Mehta, M., Agrawal, A., and Rissanen, J. (1996). Sliq: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32.

[Oza, 2001] Oza, N. C. (2001). Online ensemble learning. In *Ph.D. thesis, Department of EECS, UC Berkeley*.

[Oza and Russell, 2001] Oza, N. C. and Russell, S. (2001). Online bagging and boosting. In *Artificial Intelligence and Statistics*, pages 105–112.

[Page, 1954] Page, E. S. (1954). Continuous inspection scheme. 41(1/2):100–115.

[Parhami, 1996] Parhami, B. (1996). Voting algorithms. 43(4):617–629.

[Pelossof et al., 2008] Pelossof, R., Jones, M., Vovsha, I., and Rudin, C. (2008). Online coordinate boosting.

[Quinlan, 1993] Quinlan, J. R. (1993). C4.5: Programs for machine learning.

[Rojas, 1996] Rojas, R. (1996). *Neural Networks*.

[Schapire, 1990] Schapire, R. (1990). Strength of weak learnability. 5(2):197–227.

[Shafer et al., 1996] Shafer, J. C., Agrawal, R., and Mehta, M. (1996). Sprint: A scalable parallel classifier for data mining. In *VLDB*, pages 544–555.

[Tumer and Ghosh, 1999] Tumer, K. and Ghosh, J. (1999). Linear and order statistics combiners for pattern classification. In *A. J. C. Sharkey, editor, Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, pages 127–162.

[Tumer and Oza, 1999] Tumer, K. and Oza, N. C. (1999). Decimated input ensembles for improved generalization. In *IJCNN*, pages 105–112.

[Wang et al., 2003] Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, pages 226–235.

[Wolpert, 1992] Wolpert, D. H. (1992). Stacked generalization. 5:244–259.

[Zeira et al., 2004] Zeira, G., Maimon, M., Last, M., and Rokach, L. (2004). Data mining in time series databases. 57:101–125.