Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Huge amounts of data are generated in an unprecedented rate now-a-days from different application domains like social networks, telecommunications, WWW, scientific experiments, e-commerce systems, health care systems, etc. These data flow in and out of systems continuously and with varying update rates. Banking system keeps registering each of their ATM and account transactions, telecommunication providers logs each of their call information, popular websites maintains their user logging details, organization like CERN produces petabytes of data during their experiments; these are known as stream data. Mining of stream data is relatively a newer topic as compared to the classical data mining. The volume and the rate of data incoming make is nearly impossible to mine these data with traditional data mining approaches. As alternatives, mining a subsample of available data or to mine for models much simpler than what the data might support can be performed. This, however, drastically limits the ability to extract information from the data. Moreover, in some cases, accumulating and storing the data in runtime and performing a sampling to apply mining algorithms itself is a challenge. For such reasons the notion of mining fixed-size database is slowly being replaced with the idea of open-ended data streams.

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 remain a challenge due the restrictions imposed by the stream data. Due to the volume and the nature of the stream data there are several restrictions that are needed to be considered while designing a stream mining algorithm. Most important limitations includes not being able to store the complete dataset in memory and hence only being able to use an example once to train the model; evolving nature of the data with time, etc. Thus streams mining requires different approach than the traditional batch learning. Possibility of temporal locality makes the classification problem hard in a streaming environment. Algorithms needs to address the evolution of underlying data stream.

Tradition machine learning algorithms generally feature a single model or classifier such as Naïve Bayes or multilayer perceptron (MLP) learned from the training set and use it to classify testing set. 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. Few of the assumptions these algorithms make are: (i) data are finite, (ii) underlying data regularities are stationary, (iii) stationary data sources generate independent and identically distributed data, (iv) data are invariant to the spatial or temporal changes, etc. None of these assumptions is valid for data streams. Stream data exhibit following characteristics:

- Data come as a continuous flow of unlimited stream, often at a very high speed.
- Underlying models in the data evolve over time.
- Data cannot be considered to be independent and identically distributed.
- Time and space can significantly influence data.

At the first glance, it may seem that some simple adaptation of current methods, should be sufficient to address these changed conditions in data. In reality, these changes challenge most trivial learning methods of machine learning. Consider the computation of entropy of data. For batched learning all the instances and their corresponding classes are known before-hand to compute entropy. However, for streaming case, data stream is no longer finite, number of classes are not known a priori, domain of variables are not necessarily small in size; and hence computation of batched-like entropy function is not possible. Similarly, maintaining a frequent item set in a hundreds of gigabytes of data is also cannot be easily be derived from traditional machine learning algorithms.

Typically adaptation of traditional algorithms needs to address continuous flow of data, dynamic environment of generating sources, unavailability of complete dataset, etc. Following are some of the new requirements that are needed to be considered while developing a knowledge discovery system for stream data:

- Algorithms should use limited resources in terms of power, memory, and processing time.
- Algorithms should only access the data certain limited number of times, and may only use limited bandwidth.
- Algorithms should be ready to predict *anytime*.
- Data gathering and processing might be distributed.

With traditional algorithms, a single model is learned, i.e. only one single generalization of the data is learned. 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 classifier where each are trained with same training set would be an alternative, however, information are 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. As demonstrated in several works [Breiman, 1994, Schapire, 1990] ensemble methods (e.g. ensembles of neural networks) yields 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 over 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.

Diversity is generally achieved by making the base classifiers minimally correlated. In recent years, various methods have been developed addressing this issue. Among others bagging, boosting, Hoeffding Tree based approaches are mentionable. Some of these approaches are suitable for label based classification, and some can be used to approximate the trend of data over a specific time granularity. It will, however, be nice if a method can approximate the trend of data over a set of time granularity. Moreover, investigating how ensemble methods performs where base classifiers are trained to predict one specific labels should be interesting. For example, given a twitter stream an ensemble of classifiers may contain base classifiers to classify sentiments for sports, politics, entertainment, etc. separately. It is expected this ensemble setting would out-perform the generic ensemble approach for complete stream altogether.

Ensemble methods build model outputs where abstract properties of the algorithms that produces the model are prioritized rather than the specifics of the algorithms. This allows a wide application across many fields of study. With precise use of ensemble methods, it would be possible to automatically exploit the strengths and weaknesses of different machine learning systems.

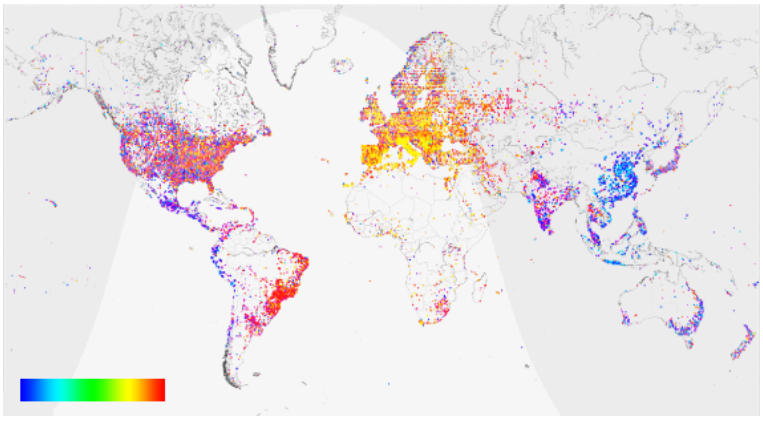
This thesis investigates currently available such ensemble approaches, and presents an empirical analysis of those methods. Moreover, this thesis presents a unique perspective relating to the underlying setup that generates the stream, that applies certain application domains such as social media. In following sections, we present this motivation and probable approaches to address such scenarios.

1.1 Motivation

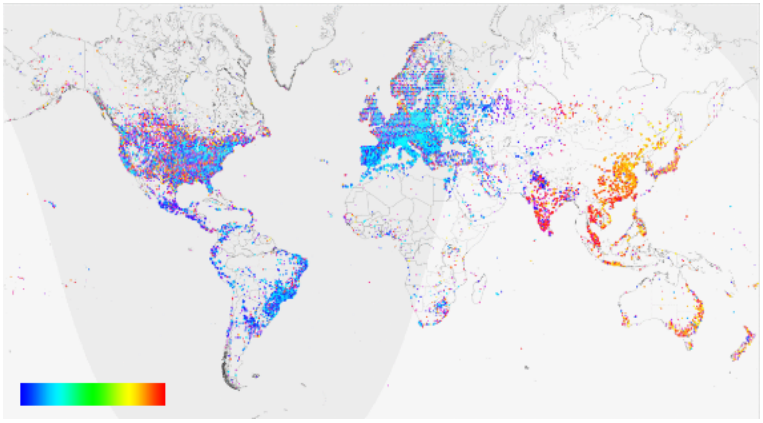
One of the major challenges faced in stream mining is the lack of labeled data. There is no such problem in batched leaning. Data are finite and only an insignificant portion might be unlabeled, if there is any. Streaming scenarios show a stark contrast. Most of real world data are mostly unlabeled. It needs human intervention, resources, and time to properly label a stream data set. Most often only a fraction of data set is labeled by

human expert or automated scripts are used. Owing to the fact of having such limitations, a large number of experimentations for stream mining algorithms are performed using synthesized data. It is much easier to control different parameters, concept drift, labeling, etc. in a generated dataset. In most of the processes, this data generation process is mostly randomized, and probability of data generating from a certain region remains the same for entire hyperspace. Temporal locality are sometimes created by adding bias to certain region, however, the rates at which these regions produces data points are mostly remain the same for all the regions in the hyperspace.

In reality many practical scenarios does not follow a uniform distribution for data generation. Some regions are more active than others. The term "more active" used here is in the sense that they produces more data. For example, Figure **??** shows the Internet traffic on an average day. A reference heat-map scale shows the blue color to be less than average and red to be higher then average. As the figure shows based on the time of the day, amount of network traffic in each region changes drastically even though the number of active nodes remains almost the same.



(a)



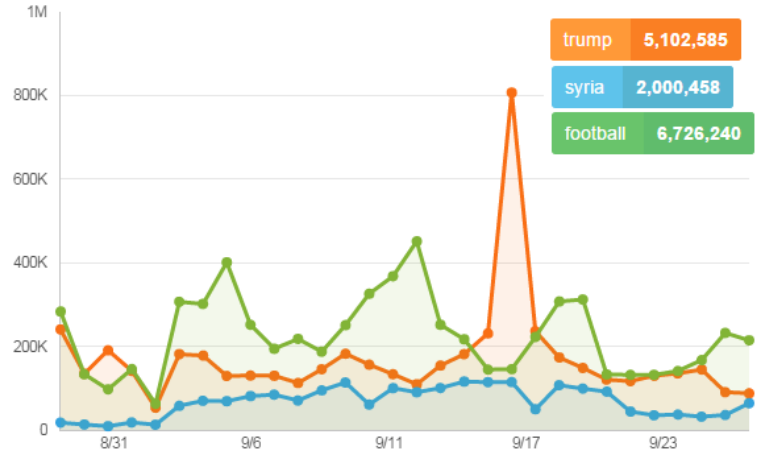
(b)

Figure 1.1: Worldwide Internet traffic. Day time in (a) western (b) eastern hemisphere [Botnet, 2012]

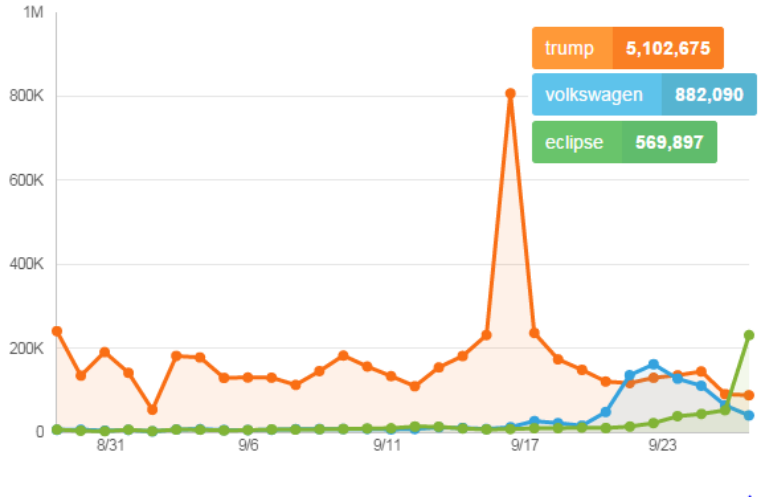
In another example, lets consider profiling of cell phone user based on the phone usages. Typically, young people are more inclined towards using data services than voice services

while the professional and elderly people mostly rely on voice services. Thus, if the profiling algorithm should consider this differences in rates in which different user groups are using the services.

To get a clearer picture of the locality of data activation and rate in which data are generated, let's consider the social media statistics. As of the second quarter of 2015, Facebook had 1.49 billion monthly active users. In the third quarter of 2012, the number of active Facebook users had surpassed 1 billion. Active users are those who have logged in to Facebook during the last 30 days. It was only on the August 28, 2015 that Facebook had 1 billion user on a single day. Let assume that we want to do a sentiment analysis over the data set collected from a week of data of Facebook. Even though the class distribution (positive and negative sentiments) may remain balanced, however, more active users will clearly overshadow the inputs given by the less active users. Similarly, about 300 million user produces 500 million tweets per day on Twitter. Figure 1.2 shows tweets for 5 different hashtags namely #Trump, #Syria, #football, #Volkswagen, and #eclipse for the month of September 2015. As it can be seen, #football has a weekly repetitive trend, most



(a)



(b)

Figure 1.2: Number of tweets for different hashtags [Topsy, 2015]

certainly because of weekend games. #Trump, on the other hand, has a steady rate with occasional spikes depending on some highlighted events e.g. debate. Topic like #Syria has lower rate, however, has a steady flow. Owing to the news of carbon emission from Volkswagen cars and lunar eclipse of September 28, 2015, topics like #Volkswagen and #eclipse got trending. Which will also soon disappear.

Based on the discussion so far, it is evidently be seen that these streams does not necessarily follow a uniform distribution. Rather these streams can be considered to be a collection of many sub-streams. The nature of these sub-streams could be as follows (but not limited to):

- A set of slow sub-streams generating low but relatively consistent number of data.
- A set of alternating sub-streams generating moderate number of data. Activation of sub-streams within the set are dependent to one other.
- A set of sub-streams that produces very large amount of data but only remain active for very short period of time.

Applying machine learning methods in such streams without considering the differences in the rate of data incoming for different sub-streams would lead to a set of decision rules dominated by the sub-streams with highest number of instances. Moreover, most stream mining algorithms only keep track of most recent incoming instances and forget older data. Such algorithms would thus forget important decision rules learned over longer times for slow but consistent sub-streams when a heavy burst of occasional sub-streams arrives. It would take long time to learn the same rules again. Similar burst could lead to the deletion of decision rules for recurring sub-streams too. This could significantly affect the performance measures. Even for slow sub-streams overall performance measures may not reflect poor decision performances for those slow sub-stream, as the number of instances belonging to those streams are not high. But ideally it is expected the mining algorithm should be equally effective for entire data space.

In this thesis, we address this scenario. To our best knowledge no previous work has specifically addressed this issue. The evaluation are always done with randomized streams. Concept drift, concept evolution, recurrence are addressed within the randomized streams. We extended one such data generation algorithm to implement the scenario present above. Empirical evaluation are presented to compare the performance of existing algorithms for such streams. Comparing the results with the results from general randomized streams, it is evident that existing algorithms do not perform at their best for streams with temporal locality. Thus, this thesis presents an ensemble algorithm devised from one of the state-of-the-art algorithm to improve performance. Extensive evaluation shows that new adaptation overcomes the challenges posed by the nature of the stream and retains all the positive factors of the original algorithm.

Next section discusses the basic idea of the algorithm without going into mathematical details. We presented the algorithm in great details once the related literatures and concepts are introduced in next couple of chapters.

1.2 Intuition

The central idea of our methods is based on the primal decision tree adaption by Domingos and Hulten [Domingos and Hulten, 2000] for streams named so-call Very Fast Decision Tree (VFDT), also commonly known as Hoeffding Tree (HT). Their method is based on the assumption that to find the best attribute for a split decision at any node in a decision tree, it would be sufficient to consider only a certain amount of training data that node. Hoeffding bound [Hoeffding, 1963] is a measurement of degree of certainty of such approach, which gives an error bound for a decision taken after seeing a certain amount of instances. This bound essentially states that any decision taken after observing a certain amount of instances would remain the same after seeing an infinite number of instances and the error margin can be computed with this bound. Detailed discussion of this bound and the algorithm is presented in Chapter 3.

With such approach, challenges posed by high volume of data can be avoided. It does not require to remember all the observed data instances. Rather the statistics of the instances are sufficient to effectively decide about split attributes. One of the limitations of Hoeffding Tree is that it grows linearly as the time passes or instances arrives at the system. Root remains the node that was splitted by observing the oldest examples. Similarly, decision rules at lower levels also becomes very old, while the leaves contains the information from most recent instances. To keep the rules updated numerous approaches such as reseting the tree, pruning bad performing nodes, etc have been proposed. Detailed discussion of adaptations for concept drift, evolution, recurrence, etc. are being discussed in Chapter 3.

With reset and pruning facility Hoeffding Tree exhibits a special property. It always adapts itself for the newer examples. Number of examples HT's decision rules depend on is directly related to the size of the tree and number of examples required to split a node. Number of decision or split nodes could a measurement of the size of the tree. A smaller tree would adapt faster to the changes in the data. A larger tree would take longer time to adapt. Using this rationale a bagging method based of different sized trees is proposed by Bifet et al. [Bifet et al., 2009]. In this method, a fixed number of Hoeffding Trees with different size (decision nodes) limits are used. Each time a tree exceeds the size threshold, the tree gets reset. Trivially, smaller trees would reset more often than the larger trees. And thus decision rules from smaller trees will based on the most recent data. Larger trees, however, would also have decision rules from older data.

We combined all these concepts to address our problem. First, we introduced a size restricted variant of Hoeffding Tree namely Size Restricted Hoeffding Tree (SRHT). Unlike its predecessor Adaptive Size Hoeffding Tree [Bifet et al., 2009], it does not get reset immediately after it reaches the size limit. Rather it stops growing i.e. no further split occurs, however, the weights in the leaf nodes are updated with incoming instances. It also combines two different concept introduces by its predecessors: (i) to reset once a size limit is reached and (ii) to maintain alternate trees or subtrees where necessary. Thus, even after reaching the size limit, a tree can switch to a alternate tree and start growing again till the limit is reached again.

We use this setup for a modified bagging scheme introduced in [Bifet et al., 2009]. Similar to the Hoeffding Tree, a smaller SRHT will have decision rules for most recent data, and a larger tree would contain rules for some older instances too. To put simply in our problem's context, smaller tree would have decisions for high speed sub-streams or burst of incoming streams. Whereas only the larger trees will have some decision rules for slow but consistent sub-streams, along with the decision rules for most recent streams. This is because, the recent data are always dominated by high speed sub-streams. Smaller does not get enough information about the slow stream or enough examples from slow streams to decide on them. Keeping this in mind, we control the reset of larger trees to keep hard-learned decision rules. In doing so, we maintain an alternate pool of trees that are to be reset soon, and start maintain a new tree with same size limit from scratch. For the transition time we consider votes from all trees, thus effectively increasing the weights for slower streams. In Chapter **??**, we present more elaborate description of the algorithms. In Chapter **??** we the experimental evaluation where we compare the results obtain by our approach with the current state-of-the-art method. [summarize the resutls here]

1.3 Contributions

In summary, the main contributions of this thesis in the domain are following:

- Detailed survey of the existing literature.
- As extensive experimental evaluation of decision tree based stream mining and ensemble approaches.
- A new look at the composition of various real-world streams e.g. social networks.
- Adaptation of Adaptive Size Hoeffding Tree into Size Restricted Hoeffding Tree.
- Combining the concepts of adaptation of concept by maintaining alternate tree and reseting of tree.
- Adaptation of bagging method to improve performance for slower streams, as well as to overcome limitation of ASHT.
- Adaptation of a data generation of approach to facilitate slow/fast, burst, recurrent behavior in a randomized data stream.
- Finally, presented justification of the new approach using real-world and synthesized data set.

1.4 Outline

This article is organized as follows: Chapter 2 presents a broad survey of present literature. Chapter 3 presents the base concepts required for the algorithm. ...

Chapter 2

Related Works

Compared to data mining, stream mining is particularly a very young area. Most of the methods date back only couple of decades. Similarly concept of the ensemble learning was first introduced in the traditional batched learning, however, ensemble adaptations for streams are fairly newer concepts. In this chapter, we list the most influential methods developed so far with particular focus on tree based learners. We follow a rudimentary narrating style starting directly with methods for general stream learners for stationary streams, and then for evolving data. After that, we list the base methods for ensemble learning, and finally presents ensemble based stream learners.

2.1 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].

In a recent approach, [Rutkowski et al., 2013] argued that using McDiarmid's bound instead of Hoeffding bound in VFDT is more appropriate for ensuring the approximation bound i.e. the split decisions made after seeing certain number of instances will, with high probability, remain the same for infinite number of instances. The authors also presented McDiarmid's bound for information gain of ID3 algorithm, and Gini index of CART algorithm. It is to be noted that Hoeffding bound is a special case of McDiarmid's bound. In another paper, [Matuszyk et al., 2013] showed usage of Hoeffding bound is mathematically flawed as (i) Hoeffding inequality only applies to arithmetic average, which information gain and Gini index are not, (ii) values obtained in sliding window methods are not independent, while Hoeffding inequality only applies to independent random variables. A revised bound showed that decision bound should be twice of the one given by Hoeffding bound, otherwise, error-likelihood should be updated accordingly.

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 should always gradually 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) dis-

tance 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.

2.2 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].

2.2.1 Ensemble Learning in Streams

Learning algorithms in data streams require maintenance 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)* i.e. 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 ASHT bagging are compared using kappa statistic. If two classifier 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 generated 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.

Another group of methods known as *random forests* uses bagging of decision trees. The

concept of random forest is introduced by Breiman [Breiman, 1999]. Random forest works in a similar fashion as bootstrap aggregation. However, for each split in the tree it only selects a subset of the features to be considered for splitting criterion. This *feature bagging* approach helps avoiding very strong predictors to get selected over and over. One method mentioned before, *Ultra Fast Forest Tree (UFFT)* [Gama et al., 2004b, Gama et al., 2005] uses concepts of tree bagging, however, works with all features the whole time. UFFT maintains statistical information on each node to detect drift and grows the tree in a similar approach to VFDT. Using the statistical information stored in nodes it detects concept drift with naive Bayes error-rate. Abdulsalam et al. proposed *Dynamic Streaming Random Forests (DSRF)* focusing on lowering the number of examples required to build up new model in a drifting environment [Abdulsalam et al., 2008, Abdulsalam et al., 2011]. Shannon entropy [Shannon, 2001] is used to detect concept drift. All these methods are empirically proven to be effective for generalized scenario and chosen data sets.

Chapter 3

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 introduced. 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.

3.1 Learning Algorithms

The goal of data classification process is to predict an 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 a value. Algorithms develop hypotheses that best describe the relationship among the attributes and the outcome for the total instance set. Formally, learning problems are defined as follow: Given a data set of m instances $D \equiv \{\{\vec{x}_1, y_1\}, \{\vec{x}_2, y_2\}, \{\vec{x}_3, y_3\}, \ldots, \{\vec{x}_m, y_m\}\}$ where $\vec{x} = \{x_1, x_2, x_3, \ldots, x_n\}$, 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 concepts here.

3.1.1 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 k neighbors where k is predefined. For example, if $k = 1$ then an unknown instance is assigned the class label of its near-

est neighbor. There are several schemes to find the k nearest neighbors. 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 $D_{Mink} \sqrt[q]{\sum_{ß=1}^{k}(|x_i - y_i|)^q}$

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.

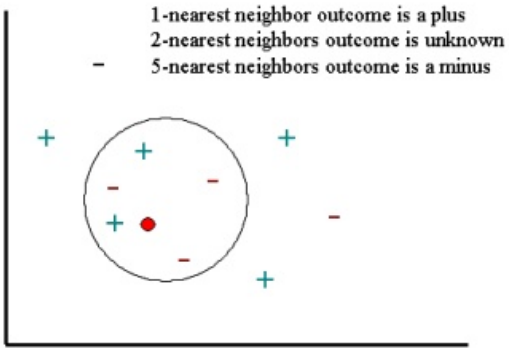


Figure 3.1: Concept of k-NN

Choosing an optimal k is the prime challenge of k-NN algorithm, which is extremely training data dependent. Changing position of few training points could lead 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 performs much better than 1-NN. Typically cross-validation is used to determine a good k.

3.1.2 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:

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

3.1.3 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 $p_1, p_2, \dots, p_n$ 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 $O(\lg n)$.

3.1.4 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) mimics biological neural network. Figure 3.2 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.

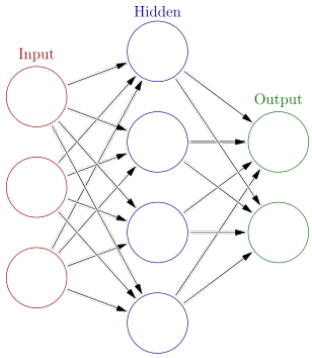


Figure 3.2: Neural network layers

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, weight 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 running MLP multiple times with different initial weight settings.

3.2 Data Stream Classification

Traditional data mining algorithms work in a memory bounded environment and require 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.

3.2.1 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 drift, 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.

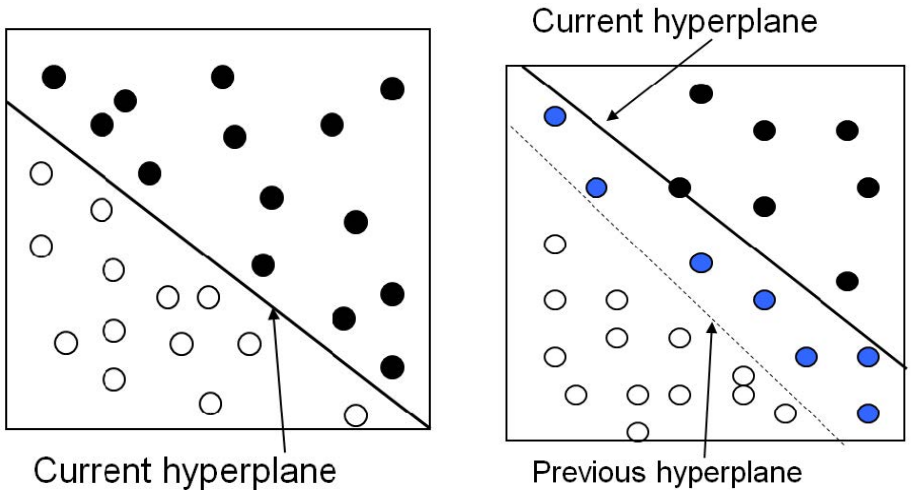


Figure 3.3: Concept drift in data streams

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 3.3. As a new chunk arrives (Figure 3.3a), 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 3.3b, 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.

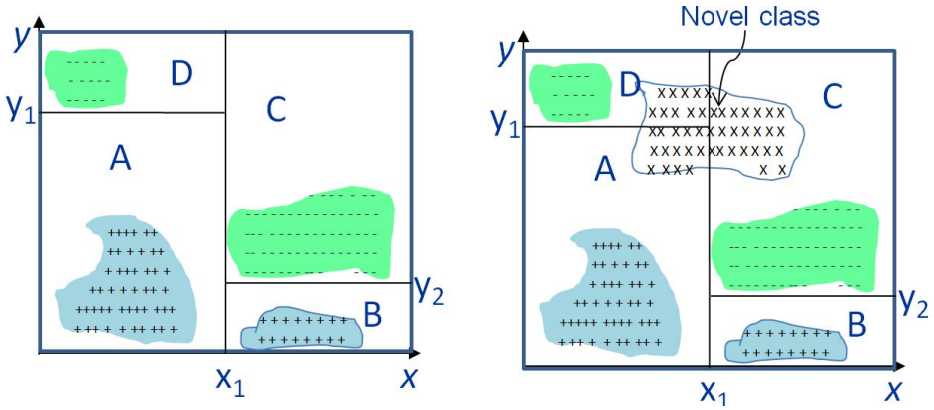


Figure 3.4: Concept evolution in data streams.

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 3.4. At certain point of time four classes and their corresponding decision boundaries are shown in the Figure (3.4a). 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.

3.2.2 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 3.2.1), 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 3.2.1 (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 bounds [Hoeffding, 1963], etc. are few of them.

Theorem 3.2.2 (Chebyshev Bound) *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 3.2.3 (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 3.2.4 (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:

$$p_k = P(x = k) = e^{-\lambda} \frac{\lambda^k}{k!} \tag{3.1}$$

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 can easily be derived. Equation 3.2, 3.3, and 3.4 can be used to recursively compute mean, variance, and correlation respectively.

$$\bar{x}_i = \frac{(i-1) \times \bar{x}_{i-1} + x_i}{i} \tag{3.2}$$

$$\sigma_i = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{i}}{i-1}} \tag{3.3}$$

$$corr(a,b) = \frac{\sum(x_i \times y_i) - \frac{\sum x_i \times \sum y_i}{n}}{\sqrt{\sum x_i^2 - \frac{\sum x_i^2}{n}}\sqrt{\sum y_i^2 - \frac{\sum y_i^2}{n}}} \tag{3.4}$$

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 is 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].

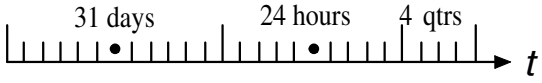


Figure 3.5: Natural Tilted Time Window

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 3.5 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 about 32 days' information.

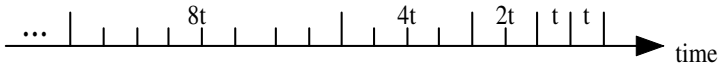


Figure 3.6: Logarithmic Tilted Time Window

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 3.6 shows an example.

3.2.3 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 3.2.3) and examines examples drawn from two probability distribution and decides whether these distributions are different.

ADWIN Algorithm: 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.

Algorithm 1: ADWIN Algorithm

Data: Data Stream
Result: Window with most recent concept

```
1.1 begin
1.2     Initialize window W
1.3     foreach t > 0 do
1.4         W ← W ∪ {x_t}                    // add x_t to the head of W
1.5         repeat
1.6             Drop element from the tail of W
1.7         until |μ_{W_0} − μ_{W_1}| < ε_cut holds for every split of W
1.8         W = W_0.W_1
1.9     Output μ_W
```

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 changes. There are two types of approaches based on when to adapt:

- Blind or Periodic Methods: Models are updated on regular interval whether any change has actually occurred or not.
- Informed Methods: Models are only updated when there are sufficient reasons to believe that changes in the concept have occurred.

It could be 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 changes. 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.

3.2.4 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.

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

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
```

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 3.2.4). 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 are 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 evaluation starts.

When multiple attributes continuously perform 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 pseudo-code 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 over-fitting 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 $HT \leftarrow$ Empty Leaf (Root) l_0
3.3 Let $Alt(l_0) \leftarrow \emptyset$ // Alternate trees for l_0
3.4 **foreach** *class* y_k **do**
3.5 **foreach** $x_{ij} \in X_i \in X$ **do**
3.6 Set $n_{ijk} = 0$
3.7 **foreach** $example(x, y) \in S$ **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

3.2.6 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 need 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. Pseudo-code 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.

3.3 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.

Algorithm: VALIDATE_SPLIT($HT, n, (x_w, y_w), id_w$)
begin

```
3.17.1  Let HT ← Empty Leaf (Root) l_0
3.17.2  Let Alt(l_0) ← ∅                                   // Alternate trees for l_0
3.17.3  foreach internal node l in HT do
3.17.4      foreach t_alt ∈ Alt(l) do
3.17.5          VALIDATE_SPLIT(t_alt, n, (x_w, y_w), id_w)
3.17.6      foreach x_ij ∈ X_i ∈ X do
3.17.7          Set n_ijk = 0
3.17.8  Let X_a be the current split attribute
3.17.9  Compute Δ(G) = G_l(X_n) − G_l(X_b)      // X_n, X_b two current highest attributes
3.17.10 if Δ(G) ≥ 0&&X_n ∉ {rootsofAlt(l)} then
3.17.11     Compute ε = sqrt(R^2 ln(2/δ) / 2n)                    // Hoeffding bound
3.17.12     if Δ(G) > ε || ε < τ&&Δ(G) ≥ τ/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_∅) using most frequent class at l_m
3.17.20             foreach node l ∈ P do
3.17.21                 foreach x_ij ∈ x : X_i ∈ X_l do
3.17.22                     Decrement n_ijk(l)
```

However, it may not necessarily divide the tasks rather might do 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 have 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 building an ensemble of 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).

3.3.1 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 3.8 shows an illustrative example of bagging method.

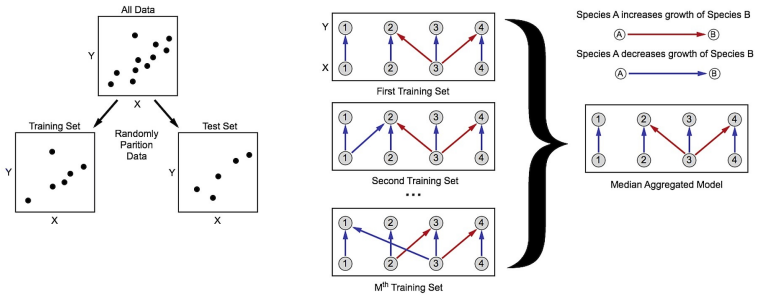


Figure 3.7: Bagging or bootstrap aggregation

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.

3.3.2 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.

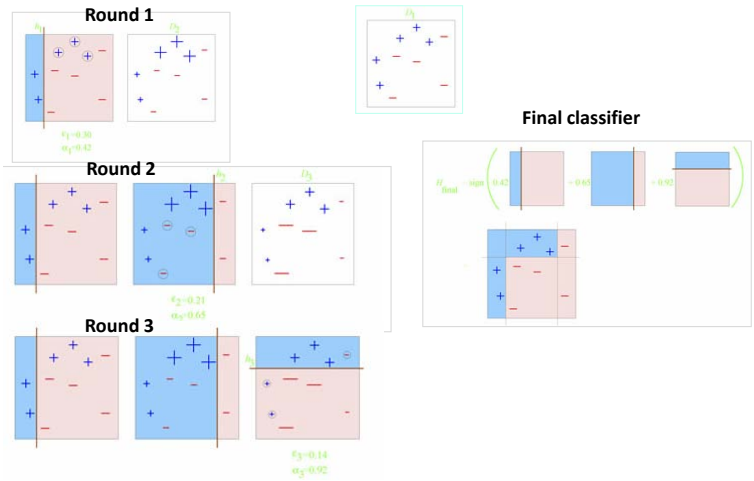


Figure 3.8: Boosting method example

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 $i = \{1, 2, \ldots, n\}$:

$$H(.) = sign\left(\sum_{i=1}^{n} \alpha_i h_i(x)\right). \tag{3.5}$$

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 3.5 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 3.1) 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.

3.3.3 Adaptive-Size Hoeffding Tree (ASHT) Bagging

In previous section (Section 3.2.5), 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 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

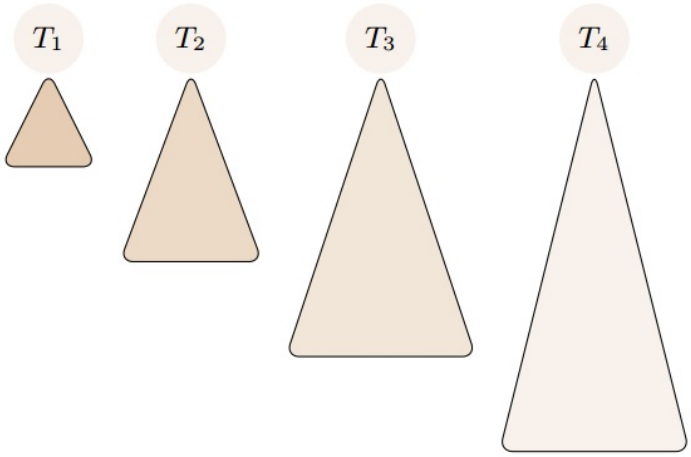


Figure 3.9: Adaptive Size Hoeffding Tree concept

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 for stationary cases.

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.

3.3.4 ADWIN Bagging

ADWIN Bagging combines three different powerful concepts together: (i) bagging (Section 3.3.1), (ii) adaptive window change detection (Section 3.2.3), and (iii) Hoeffding tree (Section 3.2.5).

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.

Chapter 4

Datasets

Due to the lack of labeled of real-world data streams, stream mining algorithms are most of the time tested with synthesized data. Synthesized data has several advantages over poorly labeled real-world data streams. First of all, it is rather easy to produce. It has significantly less storage overhead. It is easier to control synthesizing parameters to generate data sets with desired concept drifting, recurring, etc. scenarios.

In this chapter, we briefly discuss some of such generators. For our experimentations we used random RBF generator. Thus, we also discuss the generation process and generated stream's properties of random RBF generator in greater details. We then describe the modifications in the generation process to introduce variable speed RBF streams.

4.1 Stream Generators

A number of generators have been introduced in literature in past decades: SEA concept generator, STAGGER concepts generator, rotating hyperplane generator, random RBF generator, waveform generator are some of the commonly used ones.

4.1.1 SEA Concepts Generator

SEA concept generator is introduced in [Street and Kim, 2001] to generate synthetic dataset with abrupt concept drift. It uses three parameters valued between 0 and 10 inclusive. The points of the dataset are divided into blocks of different concepts. Classification within the each block is controlled using using the input parameters along with a threshold value.

4.1.2 STAGGER Concepts Generator

STAGGER is one of the early methods developed for stream generation by Schlimmer et al. [Schlimmer and Granger, 1986]. Binary concepts are generated from three attributes. Attributes are size, shape, and color. Each of the attributes has three possible values. Small, medium, and large are the sizes; circle, triangle, and rectangle are the shapes; and red, blue, and green are the colors. 27 combinations resulting from being mapped into a binary class by this generator.

4.1.3 Rotating Hyperplane Generator

Rotating Hyperplane generator is introduced in the evaluation of CVFDT [Hulten et al., 2001]. Hyperplanes can effectively be used to simulate concept drifting environments. A hyperplane in a d-dimensional space is a set of points x that satisfies following equation:

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

where x_i is the ith coordinate of x. Hyperplane works as the boundary between two of the binary classes. By controlling the orientation and position of the hyperplane concept drift can be controlled.

4.1.4 Wavefront Generator

Wavefront generator is a dataset available at UCI machine learning repository [Asuncion and Newman, 2007]. Two or three base waves are used to generated a data set of three different classes of waveforms. The prediction task is to predict the classes of waveforms. The optimal Bayes classification rate is known to be 86%. There are two variations of this generator. Wave21 uses 21 numeric noisy attributes, and wave40 uses 19 more irrelevant attributes along with original 21.

4.1.5 LED Generator

LED generator is also available at UCI machine learning repository [Asuncion and Newman, 2007]. The prediction task for this dataset is to predict the digit displayed on a seven segment LED display. Each segment has 10% chance of being inverted. An optimal classification using naive Bayes on this dataset is found to 74%.

4.2 Random Radial Basis Function Generator

All the generators mentioned above are relatively simple and easy to use. However, it is harder to model a complex scenario with those generators. Hypothesis space for these generator are not very large, except for rotating hyperplane generator. But with rotating hyperplane generator it is harder to overlapping space and outliers. Random radio basis function (RBF) generator is therefore introduced to generate complex concept type that is not straight forward to approximate, especially with decision tree model.

4.3 Generation of Variable Speed RBF Stream

Chapter 5

Algorithm

5.1 Problem Overview

5.2 Size Restricted Hoeffding Tree (SRHT)

5.3 Carry-Over Bagging with SRHT

Chapter 6

Experimental Study

6.1 Test Study-case with Adult Dataset

6.2 Structural Study of Decision Trees

6.3 Performance Evaluation

6.3.1 Performance Metrics

6.3.2 Traditional Streams

6.3.3 Social Network-like Streams

Chapter 7

Conclusion

7.1 Contribution

7.2 Future Works

7.3 Open Issues

Bibliography

[Abdulsalam et al., 2008] Abdulsalam, H., Skillicorn, D. B., and Martin, P. (2008). Classifying evolving data streams using dynamic streaming random forests. pages 643–651.

[Abdulsalam et al., 2011] Abdulsalam, H., Skillicorn, D. B., and Martin, P. (2011). Classification using streaming random forests. 23(1):22–36.

[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.

[Asuncion and Newman, 2007] Asuncion, A. and Newman, D. (2007). Uci machine learning repository. `http://archive.ics.uci.edu/ml/`.

[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.

[Botnet, 2012] Botnet, C. (2012). Port scanning using insecure embedded devices. `http://www.huffingtonpost.co.uk/2013/05/15/global-internet-gif_n_3277404.html`.

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

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

[Breiman, 1999] Breiman, L. (1999). Random forest.

[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.

[Matuszyk et al., 2013] Matuszyk, P., Krempl, G., and Spiliopoulou, M. (2013). Correcting the usage of the hoeffding inequality in stream mining. In *Advances in Intelligent Data Analysis*, pages 298–309.

[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.*

[Rutkowski et al., 2013] Rutkowski, L., Pietruczuk, L., Duda, P., and Jaworski, M. (2013). Decision trees for mining data streams based on the mcdiarmid's bound. 25(6):1272–1279.

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

[Schlimmer and Granger, 1986] Schlimmer, J. C. and Granger, R. H. (1986). Incremental learning from noisy data. 1(3):317–354.

[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.

[Shannon, 2001] Shannon, C. E. (2001). A mathematical theory of communication. 5(1):3–55.

[Street and Kim, 2001] Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm for large-scale classification. In *KDD*, pages 377–382.

[Topsy, 2015] Topsy (2015). Topsy social analytics. `http://topsy.com/analytics`.

[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.