Algorithms and Bounds in Online Learning

A dissertation submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology

in

Computer Science

by

Ankit Sharma

Roll No : CS1429

Under the supervision of

Prof. C. A. Murthy
Machine Intelligence Unit

Indian Statistical Institute

203, BT Road, Kolkata
West Bengal - 700108
July 2016

Certificate

I hereby certify that the work which is being presented in the dissertation entitled *"Algorithms and Bounds in Online Learning"* in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science* submitted in Machine Intelligence Unit, Indian Statistical Institute, Kolkata is an authentic record of my own work carried out under the supervision of *Prof. C. A. Murthy* and refers other researcher's work which are duly listed in reference section.

The matter presented in the dissertation has not been submitted for award of any degree of this or any other University.

(Ankit Sharma)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Prof. C. A. Murthy)
Machine Intelligence Unit, Indian Statistical Institute, Kolkata, WB

Acknowledgement

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of the life.

This work would not have been possible without the encouragement and able guidance of my supervisor Prof. C. A. Murthy. I thank my supervisor for their time, discussions and valuable comments.

I will be failing in my duty if I do not express my gratitude to Dr. Pradipta Bandyopadhyay, Dean of studies, ISI Kolkata for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

I am also thankful to the entire faculty and staff members of ISI for their direct-indirect help, cooperation, love and affection, which made my stay at ISI memorable.

Last but not least, I would like to thank my family whom I dearly miss and without whose blessings none of this would have been possible. I would also like to thank my friends such as Arpan Losalka, Swati Singhal for their constant support.

Date: 13-July-2016
Place: Indian Statistical Institute, Kolkata, WB (Ankit Sharma)

Abstract

Online learning is the process of answering the sequence of questions based on the correct answers of the previous questions. The goal here is to make as little expected mistakes as possible over the entire sequence of questions. It is studied in many research areas such as game theory, information theory and machine learning where settings of online learning are similar to that of these areas.

There are two main components of online learning framework. First, the learning algorithm also known as the learner and second, the hypothesis class which is essentially a set of functions. Learner tries to predict answers (labels) to the asked questions using this set of functions.

This class may be finite or infinite. Sometimes, this class contains some functions which have the capability to provide correct answers to entire sequence of asked questions. In this case, the goal of learner becomes to identify these functions in the hypothesis class as early as possible during a learning round to avoid further mistakes for the remaining rounds. This setting, when function class contains some powerful functions which can provide correct answers to the entire sequence of questions, is called realizable case.

Sometimes, it may not contain any such powerful functions which can provide correct answers to the entire sequence of questions. In such a case, learner has to rely on all the available functions in the hypothesis class and use them intelligently to predict the answers. The goal of the learner, therefore, becomes to make as little mistakes as that could have been made by the most powerful functions among the available functions. This setting, when hypothesis class does not contain any powerful functions which can provide correct answers to the entire sequence of questions, is called unrealizable or agnostic case.

There are various learning algorithms for each of these settings. All learning algorithms are expected to make least possible mistakes in each setting. Performance of these algorithms is analyzed through the expected number of mistakes over all possible orderings of the sequence of questions.

This dissertation proposes three algorithms to improve the mistakes bound in the agnostic case. Proposed algorithms perform highly better than the existing ones in the long run when most of the input sequences presented to the learner are likely to be realizable.

Table of Contents

List of Figures

List of Tables

Table No. **Title** **Page No.**

Chapter 1
Introduction

This introduction provides an overview to online learning model, its basic settings and some applications. It also discusses objectives of the dissertation, overview of existing methods in the literature and statement of contributions from this dissertations to online learning model. These points are further described in depth in later chapters.

Notation : We denote by $\mathcal{X}$ the set of input points x_t. The associated label is denoted by $y_t \in \{0,1\}$. We use $\mathcal{H}$ to denote a hypothesis class. Each $h \in \mathcal{H}$ is a mapping from $\mathcal{X}$ to $\{0,1\}$. For a predicate π, we denote the indicator function by $\mathbb{1}[\pi]$ suct that

$$\mathbb{1}[\pi] = \begin{cases} 1 & \text{if } (h(x_t) \neq y_t) \\ 0 & \text{otherwise} \end{cases}$$

1.1 Online Learning

Online learning is the process of answering the sequence of questions based on the correct answers to the previous questions. This is performed in the sequence of rounds where in each round t, the learner is given a question x_t. The learner is required to predict the answer p_t to this question x_t. After the learner has predicted the answer p_t, learner is given the right answer y_t. Now depending on the discrepancy between the predicted and right answer, learner suffers a loss. If learner suffers a loss ie, it has made a wrong prediction, it is said to make a mistake. By the given right answer, learner tries to improve the prediction mechanism for further questions. Thus, the goal of the learner is to make as few mistakes as possible over the entire sequence of questions.

In general, $\mathcal{D}$ and $\mathcal{Y}$ can be different. But when $\mathcal{D} = \mathcal{Y} = \{0,1\}$, we call it online classification. And in this case, naturally, we use 0-1 loss function: $l(x_t, y_t) = |p_t - y_t|$.

Example : Suppose we want to predict whether it will be raining tomorrow or

Algorithm 1 Online Learning

```
for $i = 1, 2, \cdots$ do
    receive question(feature vector)$x_t \in \chi$
    predict $p_t \in \mathcal{D}$
    receive the true answer $y_t \in \mathcal{Y}$
    suffer loss $l(p_t, y_t)$
end for
```

not. On day t, x_t can be a vector of meteorological measurements. The learner is required to predict $p_t \in [0, 1]$ or $\{0, 1\}$. When p_t is in $\{0, 1\}$, it is interpreted as prediction of raining tomorrow. And if p_t is in $[0, 1]$, it is interpreted as prediction of probability of raining tomorrow. Here, again, the goal of predictor is to make as little cumulative loss as possible.

1.1.1 Applications of Online Learning

The following are the real life scenarios where online learning finds its applications. [1]

(i) **Online Ranking**: Here, the learner is required to rank the given list of elements. The learner is given the query $x_t \in \chi$, where x_t is a list of k elements (e.g. Documents). The learner is required to order these k elements. Clearly, in this case $\mathcal{D}$ is the set of all permutations of these k elements $\{1 \cdots k\}$. So learner predicts one permutation p_t out of the set D based on its knowledge deduced from the previous queries. Then, the learner is given the right answer $y_t \in \mathcal{Y} = \{1 \cdots k\}$. This right answer corresponds to the document which best matches the query. This is an application of online learning in web application where online learning is used to order the documents retrieved by the search system with respect to the user input query. Right answer in this case becomes the document (web page) which user clicks on finally.

(ii) **Prediction with expert advice** : In this learner has a set of hypotheses $\mathcal{H}$ (e.g. experts or functions) and at each round it uses one or some of these hypotheses to predict the answer. Here, challenge of learner is to use these experts intelligently so that it does not make more mistakes in the long run. For this, learner uses the “reward when correct and penalize when wrong“ policy to weigh the experts.

(iii) **Choosing best page replacement algorithm** : Operating system has many algorithm like FIFO, LRU, NRU etc to choose from for replacing the page at any instance of time. Using a particular algorithm may not be optimal all the time. Because the performance of different algorithms may vary depending on the current state of the system. Therefore, prediction with expert advice form of online learning can be used to choose best algorithm at t^{th} instance of time. The available algorithms can be considered as a set of experts. These experts should be chosen intelligently by the learner to reduce the page faults.

(iv) **Online email spam filtering** : This is another interesting application of online learning. In this, learner is given an email feature vector x_t and it is required to predict the label $\hat{y_t} \in \{0, 1\}$ of email as spam or non-spam. Then, learner is given the correct label y_t(marked spam or non-spam by the user) and thus learner updates its prediction mechanism for the next question.

1.2 Basic Settings and Terminologies

This section discusses basic settings and terminologies of the online learning framework.

(i) **Input Sequence :** Input sequence contains T points where T is finite. Another very important point is that the questions (input points) cannot be stored to be used in future. Once the algorithm has predicted the answer, point has to be discarded.

(ii) **Binary classification :** The algorithms given in this dissertation assumes that we have only two classes as class 0 and class 1 i.e. $\mathcal{Y} = \{0, 1\}$.

(iii) **No statistical assumption on input sequence :** Classical statistical theory requires strong assumptions on statistical properties on the data(e.g. Sampled i.i.d. according to some unknown distribution.) But online learning does not require any statistical assumptions over the input sequence. The sequence can be deterministic, stochastic, or even adversarial adaptive to the learner's prediction mechanism. Since learner tries to deduce the knowledge from the previous correct answers, there must be some correlation between past and present rounds (points). If not, an adversary can make all the predictions of the learner wrong by just giving the opposite answer to what the learner has predicted. Therefore, we restrict the adversary to decide the answer to input question before the learner predicts.

(iv) **Hypothesis Class ($\mathcal{H}$):** We assume that the learner is armed with a class of hypotheses (functions). Out of these, some or all are used to predict the label of the input point x_t. This class can be finite or infinite. But in this dissertation work, we assume that $\mathcal{H}$ is finite.

(v) **Realizable case :** The labels of the input sequence can always be assumed to be generated by a target hypothesis h^* such that $h^* : \mathcal{X} \to \mathcal{Y}$. When this $h^* \in \mathcal{H}$, we say that input sequence is realizable by the hypothesis class $\mathcal{H}$.

(vi) **Unrealizable case(Agnostic Case)**: When we no longer assume that $h^* \in \mathcal{H}$, we say that input sequence is unrealizable by the hypothesis class $\mathcal{H}$.

(vii) **Mistake Bound $M_A(H)$:** Mistake bound $M_A(H)$ is the maximal number of mistakes made by the algorithm A on a sequence of examples which is generated by some $h^* \in \mathcal{H}$. Now, in this case objective is to design an algorithm which has minimal mistake bound $M_A(H)$.

(viii) **Regret Bound :** In unrealizable case, where we no longer assume that all the input points are labelled by some $h^* \in \mathcal{H}$, the number of mistakes made by the algorithm is compared with some best hypotheses $h \in \mathcal{H}$. This is termed as regret because this captures the regret of the algorithm, which measures how sorry the learner is, in retrospect, not to have followed the predictions of some hypothesis $h \in \mathcal{H}$. Formally, the regret of the algorithm relative to some $h \in \mathcal{H}$ when running on a sequence of T points is defined as :

$$\text{Regret}_T(h) = \sum_{t=1}^{T} l(p_t, y_t) - \sum_{t=1}^{T} l(h(x_t), y_t)$$

And the regret of the algorithm relative to the hypothesis class $\mathcal{H}$ is

$$\text{Regret}_T(\mathcal{H}) = \max_{h \in \mathcal{H}} \text{Regret}_T(h)$$

In this case, objective becomes to design lowest possible regret algorithms. Low regret means $\text{Regret}_T(h)$ grows sub-linearly with T. i.e. $\text{Regret}_T(h) \to 0$ as $T \to \infty$. There are some other variations of these settings in online learning. For example, limited feedback [1], where after each round learner is given the loss value $l(p_t, y_t)$ but does not given the actual label y_t of point x_t. Discussion about the algorithms in this setting is out of the scope of this dissertation.

(ix) **Online Learnability of Hypothesis class $\mathcal{H}$:** Let $\mathcal{H}$ be a hypothesis class and let A be an online learning algorithm. Given any sequence $S = (x_1, h^*(y_1)), \cdots\cdots (x_T, h^*(y_T))$, where T is any integer and $h^* \in \mathcal{H}$, let $M_A(S)$ be the number of mistakes A makes on the sequence S. We denote by $M_A(\mathcal{H})$ the supremum of $M_A(S)$ over all sequences of the above form. A bound of the form $M_A(\mathcal{H}) \leq B \leq \infty$ is called a mistake bound. We say that a hypothesis class $\mathcal{H}$ is online learnable if there exists an algorithm A for which $M_A(\mathcal{H}) \leq B < \infty$.

(x) **Best Hypotheses :** $\mathcal{H}$ is the set of some hypotheses. Given an input sequence, if we use all of them one by one, some may make more mistakes than others. Then, those hypotheses which make least number of mistakes are called best hypotheses.

(xi) **Ldim($\mathcal{H}$) :** This is a dimension of hypothesis classes that characterizes the best possible achievable mistake bound for a particular hypothesis class. This measure was proposed by Nick Littlestone [6] and referred to as Ldim($\mathcal{H}$). Before explaining Ldim($\mathcal{H}$), one definition needs to be given.

Definition 1.1 [6] **($\mathcal{H}$ Shattered Tree):** A shattered tree of depth d is a sequence of instances $(v_1, v_2, \cdots, v_{2^d-1})$ in $\mathcal{X}$ such that for all labelling $(y_1, y_2, \cdots, y_d) \in \{0,1\}^d$, $\exists\, h^* \in \mathcal{H}$ such that $\forall\, t \in [d]$ we have $h(v_{i_t}) = y_t$, where $i_t = 2^{t-1} + \sum_{j=1}^{t-1} y_j 2^{t-1-j}$.

Definition 1.2 [6] **(Littlestone's dimension(Ldim($\mathcal{H}$))):** Ldim($\mathcal{H}$) is the maximal integer T such that there exist a shattered tree of depth T.

Table 1.1: Predictions of $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ on the sequence of examples v_1, v_2, v_3.

	h_1	h_2	h_3	h_4
v_1	0	0	1	1
v_2	0	1	*	*
v_3	*	*	0	1

Ldim($\mathcal{H}$) is very crucial combinatorial measure in online learning as VC dim($\mathcal{H}$) [2] is in PAC learning [3] because it provides the lower bound on the number of mistakes in the realizable case as the following lemma states:

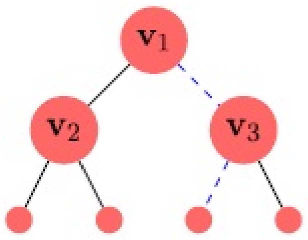

Figure 1.1: Shattered tree of depth 2

The dashed blue path corresponds to the sequence of examples $((v_1, 1), (v_3, 0))$. This tree is shattered by the hypothesis class $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ where the predictions of each hypothesis in $\mathcal{H}$ on the instances v_1, v_2, v_3 is given in the table 1.1. Here $*$ means it can be 0 or 1.

Lemma 1.1 : No algorithm can have a mistake bound strictly smaller than Ldim($\mathcal{H}$), namely, $\forall A$, $M_A(\mathcal{H}) \geq$ Ldim($\mathcal{H}$) [1].

We have the following relation among Ldim($\mathcal{H}$), VC-dim($\mathcal{H}$) and $\log_2(\mathcal{H})$.

Lemma 1.2 : VC-dim($\mathcal{H}$) $\leq$ Ldim($\mathcal{H}$) $\leq log_2(\mathcal{H})$ [1].

. (a) VC-dim($\mathcal{H}$) $\leq$ Ldim($\mathcal{H}$) and this gap can be arbitrarily large.
Suppose VC-dim($\mathcal{H}$) $= d$ and let $x_1, \cdots, x_d$ be a shattered set. We now construct a complete binary tree of instances $v_1, \cdots, v_{2^d 1}$, where all nodes at depth i are set to be x_i. Now, the definition of shattered sample clearly implies that we got a valid shattered tree of depth d, and we conclude that VC-dim($\mathcal{H}$) $\leq$ Ldim($\mathcal{H}$).

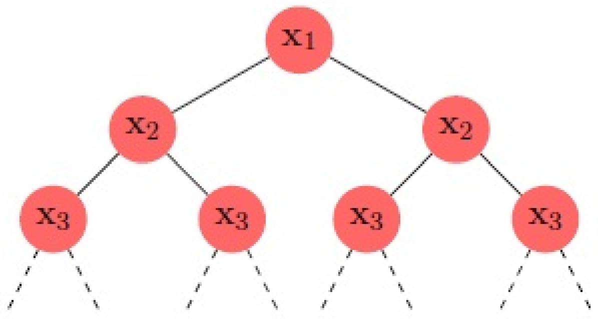

Figure 1.2: Constructing a shattered tree from a shattered sequence $(x_1, \cdots, x_d)$

Now the following example shows that the gap can be arbitrarily large.
Example 1.1 Let $\mathcal{X} = [0, 1]$ and $\mathcal{H} = \{x \to \mathbf{1}_{[x \geq a]} : a \in [0, 1]\}$, namely, $\mathcal{H}$ is

the class of threshold on the segment $[0, 1]$. Then, $\text{Ldim}(\mathcal{H}) = \infty$.
Here the gap between two quantities is infinity as $\mathcal{H}$ has $\text{Ldim} = \infty$ and VC-dim $= 1$.

(b). $\text{Ldim}(\mathcal{H}) \leq \log_2(\mathcal{H})$
Any tree that is shattered by $\mathcal{H}$ has depth at most $\log_2(\mathcal{H})$. Therefore

$$\text{Ldim}(\mathcal{H}) \leq \log_2(\mathcal{H})$$

.

□

1.3 Problem Statement

In the realizable case, we assume that all the labels of a sequnce of questions are generated from some $h^* \in \mathcal{H}$. i.e. these h^* makes 0 mistakes on the sequence of T points. In the unrealizable(agnostic) case, we do not assume this. However, we assume that there are some better hypotheses in $\mathcal{H}$ which make lesser mistakes than others.

In the literature, there are methods which are developed for both cases exclusively. It's true that the methods, which are developed for unrealizable case, will work for realizable case also. But existing methods do not have much better bound if the input sequence is found to be realizable before it ends.

So, we would like to devise some methods which perform extremely well in realizable case and do not perform that badly if sequence is remains unrealizable.

We begin with the following objectives :

(i) Devise some methods for the finite hypothesis class and unrealizable case which perform extremely well if input sequence is found to be realizable and do not perform that bad if it remains unrealizable. In other words, we would like to develop some methods which improves the mistake bound in realizable case greatly whereas loose very little in regret bound in unrealizable case.

(ii) We would also like to get hold of the best hypotheses at the end of the sequence. This might be required for various reasons. For example, an application of learning algorithm might be just to find the best hypotheses in the class for a input sequence.

1.4 Existing Methods in literature

We will be describing all the related methods to this dissertation work in the literature of online learning very briefly in this section. These will be explained in depth in chapter 2.

Section 1.1 clearly states the expectations from an online learning algorithm. Still, it would be useful to restate those requirements again informally here before describing the methods.

Let T be the number of rounds (points). Let $\mathcal{H}$ be the finite hypothesis class. Then mistake bound (in realizable case) or regret bound (in unrealizable case) should be sub linear with T. i.e. If A is any online learning algorithm then $\text{Mistake}_A(\mathcal{H})$ or $\text{Regret}_A(\mathcal{H}) = \text{o}(T)$.Here o is the small o of algorithmic complexity notations. It does not need to be a function of only T. In fact, it will also be some function of $|\mathcal{H}|$ as well. But with respect to T, it should be asymptotically sub linear with T. For example, if R denotes regret bound and $R = \sqrt{0.5\ \ln(|\mathcal{H}|)\,T}$, It is sub linear with T because $\sqrt{T} = o(T)$.

Based on the size of hypothesis class and realizability/unrealizability of the input sequence, all the available methods in online learning in standard settings can be divided into following categories:

(i) Finite hypothesis class and realizable case

(ii) Finite hypothesis class and unrealizable case

(iii) Infinite hypothesis class and both realizable and unrealizable cases

Now, we describe all the methods for each setting one by one.

1.4.1 Finite hypothesis class and realizable case

As this is the realizable case of input sequence we are assuming that all target labels are generated by some target hypotheses $h^* \in \mathcal{H}$ such that $y_t = h^*(x_t), \forall t$. We are also assuming that $|\mathcal{H}|$ is also finite i.e. $|\mathcal{H}| < \infty$.
Here are the following algorithms for this setting.

(i) **Consistent() [1]:** Consistent() algorithm is a very basic algorithm which uses very natural approach to find best hypothesis at any point. It chooses any hypothesis from the available hypothesis set to predict the label of the point. But for the future rounds, it carries forward only those hypotheses which have predicted right for the current round. By this way, if the algorithm makes a mistake in any round, it discards at least one hypothesis from the hypothesis class H. i.e. after making a mistake in t^{th} round; $|\mathcal{H}^{t+1}| = |\mathcal{H}^{t}| - 1$. Thus, its mistake bound is given as follows:

$$M_{consistent}(\mathcal{H}) \leq |\mathcal{H}| - 1$$

(ii) **Halving() [1]:** In consistent, we were just using arbitrary single hypothesis to predict the label. A better idea would be to take the majority vote and then decide. It will improve the chances of correct prediction and will also enable the learner to discard at least half of the hypotheses if algorithm makes a mistake in any round.

At each round, it partitions the hypothesis class into two sets. One partition consists of all those hypotheses which are predicting 0 and other contains which are predicting 1. Then, halving() chooses predictions of the partition which has larger cardinality. When correct answer is revealed to the learner, it discards the partition whose predictions are not same as correct answer. Since at any round if algorithm makes a mistake, we can safely discard at least half of the hypotheses.

Thus, Halving() enjoys the mistake bound equals to $\log_2(|\mathcal{H}|)$.

$$M_{halving}(\mathcal{H}) \leq \log_2(|\mathcal{H}|).$$

(iii) **Standard Optimal Algorithm or SOA () [1]:** This is the optimal algorithm in the realizable setting. The idea is same as that of halving(). It also partitions the hypothesis class into two sets. One partition consists of all those hypotheses which are predicting 0 and other contains which are predicting 1. Then, unlike halving(), it chooses predictions of the partition which has larger Ldim rather than partition with larger cardinality. When correct answer is revealed to the learner, it discards the partition whose predictions are not same as the correct answer.

Thus, similar to the bound for the halving(), SOA enjoys the mistake bound equals to the $\text{Ldim}(\mathcal{H})$.

$$M_{SOA}(\mathcal{H}) \leq \text{Ldim}(\mathcal{H})$$

1.4.2 Finite hypothesis class and unrealizable case

In the unrealizable case, we do not assume that all the labels are generated from some $h^* \in \mathcal{H}$. But we assume that there are some better hypotheses in $\mathcal{H}$ which make lesser mistakes than others. We analyse the mistake bound with respect to these hypotheses and term this mistake bound as regret bound.

The rationale behind all the described algorithms for realizable case was to find the best hypothesis and then continue prediction using that only for the future rounds. This gave the learner the liberty of discarding hypotheses which made mistakes even once. Learner could discard the hypotheses because once they make a mistake; they can never be the target hypothesis.

But in unrealizable case, there may not be such hypotheses; we cannot discard them just because they make a mistake in one or some rounds. We will have to keep track of mistake count of all the hypotheses and make our prediction based on the mistake count of each one so far.

Even if $|\mathcal{H}|$ is finite, it can be arbitrarily large. If it is too large; we cannot use all the hypotheses in consideration to make prediction. Therefore, based on this, there are two different algorithms for this setting.

(i) **Prediction with expert advice : Weighted Majority Algorithm(WM) when $|\mathcal{H}| < \infty$: [4]**

So far all the algorithms described were deterministic in their prediction. But this is a probabilistic algorithm which assigns some weights to each hypothesis and treats these weights as a probability vector.

Let $\mathcal{H} = \{h_1, h_2, \cdots, h_d\}$. Weighted majority (WM) algorithm treats this class as set of experts. It assigns a weight $w \in [0, 1]$ to each expert and keeps updating it based on the number of mistakes made by each expert so far. When a point $x_t$is received by the WM, it collects weights of all experts, which are predicting 1 for this x_t, in a variable p_t and then it predict 1 with probability p_t (Note that $p_t \in [0, 1]$). When the true answer $y_t$of x_t is revealed to the algorithm, it update the mistake count of each expert whichever made mistake on this x_t. This enjoys the asymptotically optimal regret bound [5] as follows:

$$Expected\_Regret\_Bound_{WM}(\mathcal{H}) = \sqrt{(0.5 \ln(|\mathcal{H}|)\, T)}$$

(ii) **Expert() algorithm when $|\mathcal{H}|$ is allowed to be ∞ but Ldim($\mathcal{H}$) $< \infty$: [1]**

When H is allowed to be ∞, we cannot use each hypothesis in each round for deciding the prediction. But we assume that Ldim($\mathcal{H}$) is finite so that we can use *SOA()* (described earlier in section 1.1.3) somehow. Since we are assuming that $\mathcal{H}$ can be arbitrarily large and we cannot use each hypothesis in each round, the challenge before us is how to define a set of experts that on one hand is not excessively large while on the other hand contains an expert that gives accurate prediction. Here, basic idea is to simulate each expert by running *SOA()* algorithm on a small sub-sequence of points. We define an expert for each sequence of length $L <=$ Ldim($\mathcal{H}$) and then use that constructed set of experts that sub-sequence.

$$Expected\_Regret\_Bound_{Expert()}(H) = \mathrm{Ldim}(\mathcal{H}) + \sqrt{(0.5\,\mathrm{Ldim}(\mathcal{H})\,T\,log(T))}$$

1.4.3 Infinite hypothesis class and realizable and unrealizable cases :

These cases are out of the scope of this dissertation work. However there are some algorithms like *Perceptron()* and *Winnow()* [6] in these settings which can be found in the literature [4], [7], [8], [9].

1.5 Contributions from this dissertation

This dissertation proposes three different methods to accomplish the objectives stated in the section 1.1.2. They are named as New_Consistent_WM(), New_Halving _WM(), and New_SOA_WM(). The best method among the three is New_SOA_WM(). All of the three proposed algorithms are given briefly as follows.

(i) **New_Consistent_WM**() :

This algorithm combines the Consistent() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.1) and Weighted_Majority() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.2) algorithms in a way to improve the regret bound in realizable case. This algorithm enjoys the following expected regret bound

(a) In realizable case :

$$M_{New\_Consistent\_WM}(\mathcal{H}, T) \leq |\mathcal{H}|$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq |\mathcal{H}| + \sqrt{(0.5 \ln(|\mathcal{H}|)\,(T - |\mathcal{H}|))}$$

(ii) **New_Halving_WM**() :

This algorithm combines the Halving() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.1) and Weighted_Majority() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.2) algorithms in a way to improve the regret bound in realizable case.

This algorithm enjoys the following expected regret bound

(a) In realizable case :

$$M_{New\_Halving\_WM}(\mathcal{H}, T) \leq \log_2(|\mathcal{H}|)$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5 \ln(|\mathcal{H}|)\,(T - \log_2 |\mathcal{H}|))} + \log_2(|\mathcal{H}|)$$

(iii) **New_SOA_WM**() :

This algorithm combines the SOA() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.1) and Weighted_Majority() (introduced in chapter 1, section 1.1.3 and described in detail in chapter 3, section 2.2) algorithms in a way to improve the regret bound in realizable case.

This algorithm enjoys the following expected regret bound

(a) In realizable case :

$$M_{New\_SOA\_WM}(\mathcal{H}, T) \leq \mathrm{Ldim}(\mathcal{H})$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}1[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbb{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5 \ \ln(|\mathcal{H}|) \, (T - \mathrm{Ldim}(\mathcal{H})))} +$$

$\mathrm{Ldim}(\mathcal{H})$

All of these three methods are described in depth in the chapter 3.

Chapter 2
Existing Methods

In this chapter, all the methods which were described in section 1.3 are explained in depth. As it was mentioned that based on the size of hypothesis class and realizability/unrealizability of the input sequence, all the available methods in online learning in standard settings can be divided into the following categories:

(i) Finite hypothesis class and realizable case

(ii) Finite hypothesis class and unrealizable case

(iii) Infinite hypothesis class and both realizable and unrealizable cases

Now, we describe all the methods for each setting one by one in the following sections.

2.1 Finite hypothesis class and realizable case

As this is the realizable case of input sequence we are assuming that all target labels are generated by some target hypotheses $h^* \in \mathcal{H}$ such that $y_t = h^*(x_t), \forall t$. We are also assuming that $|\mathcal{H}|$ is also finite i.e. $|\mathcal{H}| < \infty$.
Here are the following algorithms for this setting :

2.1.1 Consistent

This is given in Algorithm 2. [1]

As it was mentioned earlier that Consistent() algorithm is a very basic algorithm which uses very natural approach to find best hypothesis at any point. It chooses any hypothesis from the available hypothesis set to predict the label of the point. But for the future rounds, it carries only those hypotheses which predicted right for the current point. By this way, if the algorithm makes a mistake in any round, it discards at least one hypothesis from the hypothesis class $\mathcal{H}$. i.e. after making a mistake in t^{th} round $|H^{t+1}| = |H^t| - 1$.

Algorithm 2 Consistent

```
Input: A finite hypothesis class $\mathcal{H}$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots$ do
    receive $x_t$
    choose any $h \in V_t$
    predict $p_t = h(x_t)$
    receive true answer $y_t = h^*(x_t)$
    update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
end for
```

2.1.1.1 Analysis of Consistent()

The Consistent() algorithm maintains a set, V_t, of all the hypotheses which are consistent with $(x_1, y_1), \cdots, (x_{t1}, y_{t1})$. This set is often called the version space. It, then, picks any hypothesis from V_t and predicts according to this hypothesis. It is clear that whenever Consistent() makes a mistake; at least one hypothesis is removed from the V_t. So after making M mistakes, $|V_t| = |\mathcal{H}| - M$.

Note that $\mathcal{H}$ is never empty because of the realizability assumption that $h^* \in \mathcal{H}$. So in worst case Consistent() can make at most $|\mathcal{H}| - 1$ mistakes and in best case it will make only 1 mistake. This is the case when it gets hold of the best hypothesis in the very beginning itself. Therefore, based on this discussion, we have the following corollary stating the mistake bound of the Consistent().

Corollary 0.1. *: Let $\mathcal{H}$ be a finite hypothesis class. The consistent algorithm enjoys the mistake bound $M_{consistent}(\mathcal{H}) \leq |\mathcal{H}| - 1$ [1].*

2.1.2 Halving

This is given in Algorithm 3. [1]

Algorithm 3 Halving()

```
Input: A finite hypothesis class $\mathcal{H}$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots$ do
    receive $x_t$
    predict $p_t = \arg\max_{r \in \{0,1\}} |\{h \in V_t : h(x_t) = r\}|$
    (in case of a tie predict $p_t = 1$)
    receive true answer $y_t = h^*(x_t)$
    update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
end for
```

2.1.2.1 Analysis of Halving()

In consistent, we were just using arbitrary single hypothesis to predict the label. A better idea would be to take the majority vote and then decide. This will improve the chances of correct prediction and it will also enable us to discard at least half of the hypotheses if algorithm makes a mistake in any round.

At each round, it partitions the hypothesis class into two sets. One partition consists of all those hypotheses which are predicting 0 and other contains which are predicting 1. Then, halving() chooses predictions of the partition which has larger cardinality. When correct answer is revealed to the learner, it discards the partition whose predictions are not same as correct answer. Since at any round if algorithm makes a mistake, we can safely discard at least half of the hypotheses.

We have the following theorem analysing the mistake bound of Halving algorithm :

Theorem 1. *Let $\mathcal{H}$ be a finite hypothesis class. The Halving() algorithm enjoys the mistake bound $M_{halving}(H) \leq log_2(|\mathcal{H}|)$.[1]*

Proof. We simply note that whenever Halving makes a mistake we have $|V_{t+1}| \leq |V_t|/2$. Therefore, if M is the total number of mistakes then we have,

$$1 \leq |V_{t+1}| \leq |\mathcal{H}|/2^M$$

Now rearranging the terms, we have

$$1 \leq |\mathcal{H}|/2^M$$

$$2^M \leq |\mathcal{H}|$$

$$M \leq log_2|\mathcal{H}|$$

□

2.1.3 Standard Optimal Algorithm or SOA

This is given in Algorithm 4 .[1] :

2.1.3.1 Analysis of SOA()

This is the optimal algorithm in the realizable setting. The idea is same as that of halving(). It also partitions the hypothesis class into two sets. One partition consists of all those hypotheses which predict 0 and other contains which predict

Algorithm 4 SOA()

```
Input: A finite hypothesis class $\mathcal{H}$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots$ do
    receive $x_t$
    for $r \in \{0, 1\}$ let $V_t^{(r)} = \{h \in V_t : h(x_t) = r\}$
    predict $p_t = \arg\max_{r\in\{0,1\}} \text{Ldim}(V_t^{(r)})$
    (in case of a tie predict $p_t = 1$)
    receive true answer $y_t = h^*(x_t)$
    update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
end for
```

1. Then, unlike halving(), it chooses predictions of the partition which has larger Ldim rather than partition with larger cardinality. When correct answer is revealed to the learner, it discards the partition whose predictions are not same as the correct answer.

The following Lemma proves the optimality of SOA ().

Lemma 2. *SOA enjoys the mistake bound $M_{SOA}(\mathcal{H}) \leq Ldim(\mathcal{H})$.[1]*

Proof. It suffices to prove that whenever the algorithm makes a mistake, we have $\text{Ldim}(V_{t+1} \leq \text{Ldim}(V_t) - 1)$. We prove this claim by assuming the contrary, that is, $\text{Ldim}(V_{t+1} = \text{Ldim}(V_t))$. If this holds true, then the definition of pt implies that $\text{Ldim}(V_t^(r) = \text{Ldim}(V_t))$ for both $r = 1$ and $r = 0$. But, then we can construct a shattered tree of depth $\text{Ldim}(V_t) + 1$ for the class V_t, which leads to the desired contradiction. □

Combining Lemma 1.1 and Lemma 2, we obtain[1]:

Corollary 2.1. *Let $\mathcal{H}$ be any hypothesis class. Then, the standard optimal algorithm enjoys the mistake bound $M_{SOA}(\mathcal{H}) = Ldim(\mathcal{H})$ and no other algorithm Acan have $M_A(\mathcal{H}) < Ldim(\mathcal{H})$ [1].*

The following table summarizes all the available algorithms and their mistake bounds described above for finite hypothesis class and realizable case.

2.2 Finite hypothesis class and unrealizable case

In the unrealizable case, we do not assume that all the labels are generated from some $h^* \in \mathcal{H}$. But we assume that there are some better hypotheses in H which make lesser mistakes than others. We analyse the mistake bound with respect to these hypotheses and term this mistake bound as regret bound.

Table 2.1: Summary of mistake bounds of existing algorithms for finite hypothesis class and realizable case.

Seq	Algorithms	Mistake Bound	Optimal Mistake Bound
1.	Consistent()	$O(\lvert\mathcal{H}\rvert)$	$O(\mathrm{Ldim}(\mathcal{H}))$
2.	Halving()	$O(\log_2\lvert\mathcal{H}\rvert)$	
3.	SOA()	$O(\mathrm{Ldim}(\mathcal{H}))$	

The rationale behind all the described algorithms for realizable case was to find the best hypothesis and then predict using that only. This gave us the liberty of discarding hypotheses which made mistake even once. We could discard the hypotheses because once they make a mistake; they can never be the target hypothesis.

But in unrealizable case, there may not be such hypotheses; we cannot discard them just because they make a mistake in one or some rounds. We will have to keep track of mistake count of all the hypotheses and make our prediction based on the mistake count of each one.

Even if $\mathcal{H}$ is finite, it can be arbitrarily large. If it is too large we cannot use all the hypotheses in consideration to make prediction. Therefore, based on this, there are two different algorithms for this setting.

2.2.1 Prediction with expert advice: Weighted Majority Algorithm(WM) when $|\mathcal{H}|$ is finite

So far all the algorithms described were deterministic in their prediction. But this is a probabilistic algorithm which assigns some weights to each hypothesis and treats these weights as a probability vector.

Let $\mathcal{H} = \{h_1, h_2, \cdots, h_d\}$. Weighted_Majority (WM) algorithm treats this class as set of experts which help it predicting the answer. This is given in Algorithm 5.[1]:

2.2.1.1 Analysis of Weighted_Majority()

Basically, Weighted Majority assigns a weight $w \in [0, 1]$ to each expert and keeps updating it based on the number of mistakes made by each expert so far.

When a point $x_t$is received by the WM, it collects weights of all experts, which are predicting 1 for this x_t, in a variable p_t and then it predict 1 with probability

Algorithm 5 Weighted Majority : Learning with Expert Advice

```
Input: A finite hypothesis class $\mathcal{H}$ containing d experts. Number of rounds(input points) $T$
Initialize: $\eta = \sqrt{2\ln(d)/T}$ ; $\forall i \in [d], M_i^0 = 0$
for $t = 1, 2, \cdots$ do
    receive $x_t$
    receive expert advice $(h_1^t(x_t), h_2^t(x_t), \cdots, h_d^t(x_t)) \in \{0,1\}^d$
    Define $w_i^{t-1} = \frac{e^{-\eta M_i^{t-1}}}{\sum_{j=1}^d e^{-\eta M_j^{t-1}}}$
    Define $\hat{p_t} = \sum_{i:h_i^t(x_t)=1} w_i^{t-1}$
    Predict $\hat{y_t} = 1$ with probability $\hat{p_t}$
    receive true answer $y_t$
    update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
end for
```

p_t (Note that $p_t \in [0,1]$). When the true answer of x_t is revealed to the algorithm, it update the mistake count of each expert whichever has made mistake.

The following theorem analyses the regret bound for the Weighted Majority algorithm [1]:

Theorem 3. *Weighted Majority satisfies the following :*

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y_t} \neq y_t]] - \min_{i\in[d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \sqrt{0.5\ln(d)T}$$

Proof. The algorithm maintains the number of prediction mistakes each expert made so far,M_i^{t-1} , and assign a probability weight to each expert accordingly. Then, the learner sets $\hat{p_t}$ to be the total mass of the experts which predict 1. The definition of $\hat{y_t}$ clearly implies that

$$\mathbb{E}[\mathbb{1}[\hat{y_t} \neq y_t]] = \sum_{i=1}^{d} w_i^{t-1}\mathbb{1}[h_i^t(x_t) \neq y_t] \tag{2.1}$$

That is, the probability to make a mistake equals to the expected error of experts, where expectation is with respect to the probability vector $\mathbf{w}^t$.

Now, we begin the proof : Define $Z_t = \sum_i e^{-\eta M_i^t}$. We have

$$\ln\frac{Z_t}{Z_{t-1}} = \ln\frac{\sum_i e^{-\eta M_i^{t-1}} e^{-\eta\mathbb{1}[h_i^t(x_t)\neq y_t]}}{\sum_j e^{-\eta M_j^t}}$$

$$= \sum_{i=1}^{d} w_i^{t-1} e^{-\eta\mathbb{1}[h_i^t(x_t)\neq y_t]}$$

Note that $\mathbf{w}^t$ is a probability vector and $\mathbf{1}[h_i^t(x_t) \neq y_t] \in [0,1]$. Therefore, we can apply Hoeffdings inequality (see for example [2], Lemma 2.2) on the right-hand side of the above to get

$$\ln \frac{Z_t}{Z_{t-1}} \leq -\eta \sum_{i=1}^{d} w_i^{t-1} \mathbb{1}[h_i^t(x_t) \neq y_t]$$

where the last equality follows from Eq. (2.1). Summing the above inequality over t we get

$$\ln(Z_T) - \ln(Z_0) = \sum_{t=1}^{T} \ln \frac{Z_t}{Z_{t-1}} \leq -\eta \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] + \frac{T\eta^2}{8} \tag{2.2}$$

Next, we note that $\ln(Z_0) = \ln(d)$ and that

$$\ln Z_T = \ln(\sum_i e^{-\eta M_i^T}) \geq \ln(\max_i e^{-\eta M_i^T}) = -\eta \min_i M_i^T$$

Substituting the values of $\ln(Z_T)$ and $\ln(Z_0)$ in Eq.(2.2)

$$-\eta \min_i M_i^T - \ln(d) \leq -\eta \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] + \frac{T\eta^2}{8}$$

Dividing both sides by η and rearranging the terms, we get

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] - \min_{i \in [d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \frac{\ln(d)}{\eta} + \frac{\eta T}{8}$$

Putting $\eta = \sqrt{8\ln(d)/T}$, we get the desired result

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] - \min_{i \in [d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \sqrt{0.5\ln(d)T}$$

□

This dissertation work is based on constructing an algorithm to improve this bound in realizable case.

2.2.2 Expert algorithm: When $|\mathcal{H}|$ is allowed to be ∞ but Ldim($\mathcal{H}$) is finite

When H is allowed to be ∞, we cannot use each hypothesis in each round for deciding the prediction. But we assume that Ldim($\mathcal{H}$) is finite so that we can use

SOA() (described earlier in section 1.1.3) somehow.

Since we are assuming that $\mathcal{H}$ can be ∞ and we cannot use each hypothesis in each round, the challenge before us is how to define a set of experts that on one hand is not excessively large while on the other hand contains an expert that gives accurate prediction.

Here, basic idea is to simulate each expert by running *SOA()* algorithm on a small sub-sequence of points. We define an expert for each sequence of length $L <= \text{Ldim}(\mathcal{H})$ and then use that constructed set of experts that sub-sequence.

The algorithm for defining experts is given in Algorithm 9.[1]:

Algorithm 6 Expert$(i_1, i_2, \cdots, i_L)$

```
Input: A finite hypothesis class $\mathcal{H}$, Indices $i_1 < i_2 < \cdots < i_L$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots, T$ do
    receive $x_t$
    for $r \in \{0, 1\}$ let $V_t^{(r)} = \{h \in V_t : h(x_t) = r\}$
    predict $p_t = \arg\max_{r \in \{0,1\}} \text{Ldim}(V_t^{(r)})$
    (in case of a tie predict $p_t = 1$)
    receive true answer $y_t$
    if $y_t \neq \hat{y}_t$ and $t \in \{i_1, i_2, \cdots, i_L\}$ then
        Update $V_{t+1} = V_t^{(y_t)}$
    else
        Update $V_{t+1} = V_t$
    end if
end for
```

The following key lemma shows that there exists an expert whose performance is almost optimal[1].

Lemma 4. *Let $(x_1, y_1) \cdots (x_T, y_T)$ be a sequence of examples and let $\mathcal{H}$ be a hypothesis class with Ldim($\mathcal{H}$) $< \infty$. There exists L $\leq$ Ldim($\mathcal{H}$) and a subsequence $1 \leq i_1 \leq \cdots \leq i_L \leq T$, such that Expert($i_1, i_2, \cdots, i_L$) makes at most*

$$L + \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t]$$

mistakes on the sequence of examples.

Proof. To simplify our notation, let $M(h)$ be the number of mistakes a hypothesis h makes on the sequence of examples. Let $h^* \in \mathcal{H}$ be an optimal hypothesis, that is $M(h^*) = \min_h M(h)$. Let $j_1, \cdots, j_k$ be the set of rounds on which h^* does not err. Thus, $k = T - M(h^*)$. The sequence of examples $(x_{j_1}, y_{j_1}), \cdots (x_{j_1}, y_{j_1})$

is realizable for $\mathcal{H}$ (since $h^* \in \mathcal{H}$ never err on this sequence). Therefore, if we run SOA(Algorithm 4) on this sequence we have at most Ldim($\mathcal{H}$) mistakes. Choose $i_1, i_2, \cdots, i_L$ to be a sub-sequence of $j_1, \cdots, j_k$ that contains the indices of examples on which SOA errs, and note that $L \leq \text{Ldim}(\mathcal{H})$. Since the predictions of SOA on $j_1, \cdots, j_k$ are exactly the same as the predictions of Expert($i_1, i_2, \cdots, i_L$) on $j_1, \cdots, j_k$ we get that the total number of prediction mistakes of Expert($i_1, i_2, \cdots, i_L$) on the entire sequence is at most $L + M(h^*)$. □

After constructing the experts as above, our algorithm becomes the application of Weighted_Majority() algorithm and given in Algorithm 7.[1]:

Algorithm 7 Agnostic Online Learning Algorithm

```
Input: A finite hypothesis class $\mathcal{H}$ with $\text{Ldim}(\mathcal{H}) < \infty$; learning rate $\eta > 0$;
Number of rounds $T$
Initialize:
for each $L \leq \text{Ldim}(\mathcal{H})$ do
    for each sub-sequence $1 \leq i_1 < i_2 < \cdots, i_L \leq T$ do
        Construct an expert from $i_1, i_2, \cdots, i_L$ as in algorithm 4 Expert().
    end for
end for
```

To analyse this algorithm, we combine Lemma 4 with the upper bound on the number of experts

$$d = \sum_{L=0}^{\text{Ldim}(\mathcal{H})} \binom{T}{L} \leq T^{\text{Ldim}(\mathcal{H})}$$

to obtain the following [1]:

Theorem 5. *Let H be a hypothesis class with Ldim($\mathcal{H}$) < 1. If we run Algorithm 7 on any sequence $(x_1, y1), \cdots (x_T, y_T)$, we obtain the expected regret bound,*

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y_t} \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq \text{Ldim}(\mathcal{H}) + \sqrt{(0.5\,\text{Ldim}(\mathcal{H})\,T\,\ln(T))}$$

The above theorem implies that a class H that has a finite Ldim is agnostic online learnable.

The table 2.2 summarizes all the available algorithms and their regret bounds described above for finite Ldim hypothesis class in unrealizable case.

Table 2.2: Summary of regret bounds of existing algorithms for finite Ldim hypothesis class and unrealizable case.

Seq	Algorithms	Regret Bound	Optimal Regret Bound
1.	Weighted Majority() when $\|\mathcal{H}\| < \infty$ $\mathrm{Ldim}(\mathcal{H}) < \infty$	$=\sqrt{(0.5\ \ln(\mathcal{H})\,T\,)}$	$\mathrm{O}(\sqrt{\mathrm{Ldim}(\mathcal{H})\,T})$
2.	Experts() when $\|\mathcal{H}\|$ is allowed to be ∞, $\mathrm{Ldim}\|\mathcal{H}\| < \infty$	$\mathrm{O}(\mathrm{Ldim}(\mathcal{H})+\sqrt{(0.5\,\mathrm{Ldim}(\mathcal{H})\,T\ \ln(T))}\ )$	Same as above

2.3 Infinite hypothesis class and realizable and unrealizable case

This case is out of the scope of this dissertation work. However there are some algorithms like *Perceptron()* and *Winnow()* in this setting which can be found in the literature [1],[4], [7],[8],[9].

Chapter 3

Proposed Methodologies

3.1 Problem Statement and Objectives

As mentioned in the section 1.1.2, our settings are following:

In the realizable case, we assume that all the labels are generated from some $h^* \in \mathcal{H}$. i.e. these h^* makes 0 mistakes on the sequence of T points. In the unrealizable(agnostic) case, we do not assume this. However, we assume that there are some better hypotheses in $\mathcal{H}$ which make lesser mistakes than others.

In the literature, there are methods which are developed for both cases exclusively. It's true that the methods, which are developed for unrealizable case, will work for realizable case also. But existing methods do not have much better bound if the input sequence is found to be realizable.

So, we would like to devise some methods which perform extremely well in realizable case and do not perform that bad if sequence is found to be unrealizable.

To recapitulate, we began with the following objectives :

(i) Devise some methods for the finite hypothesis class and unrealizable case which perform extremely well if input sequence is found to be realizable and do not perform that bad if input sequence is nor found to be unrealizable. In other words, we would like to develop some methods which improves the mistake bound in realizable case greatly while loosing the regret bound in unrealizable case very slightly.

(ii) We would also like to get hold of the best hypotheses at the end of the sequence. This might be required for various reasons. For example, one application of learning algorithm might be just to find the best hypotheses in the class for a input sequence.

3.2 Approach

If we observe the algorithms described in the section 2.1 for the finite hypothesis class and realizable case, the mistake bound does not depend on the length T of the input sequence. It depends only on the size of the hypothesis class $|\mathcal{H}|$. Further, The *Weighted Majority()* algorithm described in section 2.2 enjoys the same regret bound which depends on both T and $|\mathcal{H}|$, no matter the input sequence is realizable or unrealizable. This is what that drives the idea of devising the proposed method. Therefore, the proposed algorithm combines the approaches of existing algorithms presented in section 1.1 and 1.2. In the following section, we present the new algorithms. These new algorithms couple the weighted majority algorithm with each of the algorithm described in section 1.2, namely, $Consistent(), Halving()$ and $SOA()$. This gives us the much better bound in the realizable case while does not loose that much in unrealizable case.

For all the proposed algorithms, we make following assumptions

(a) $|\mathcal{H}|$ is finite. i.e. $|\mathcal{H}| < \infty$

(b) Ldim($\mathcal{H}$) is finite. i.e. Ldim($\mathcal{H}$) $< \infty$

(c) Ldim($\mathcal{H}$) $<< T$

3.3 Proposed Algorithms

As already mentioned, the proposed algorithms couple the $weightedmajority()$ algorithm with each of the algorithm described in section 1.2, namely, $Consistent()$, $Halving()$, and $SOA()$. Out of these three algorithms, $New\_SOA\_WM()$ is the best algorithm. It is described in section 3.3.3.

All the proposed algorithms are described in detail in the following subsections.

3.3.1 Weighted Majority with Consistent

The following algorithm combines the Weighted Majority() and Consistent() algorithms. This is the most basic algorithm that can be produced by combining these two algorithms. Therefore, it does not provide much better bound. However, it is still better than the weighted majority in realizable case.

The proposed algorithm is given in Algorithm 8.

Algorithm 8 *New_Consistent_WM*()

```
Input: A finite hypothesis class $\mathcal{H}$ containing d experts.i.i $|\mathcal{H}| = d$. Number of rounds(input points) $T$
Initialize: $\eta = \sqrt{2\ln(d)/T}$ ; $\forall i \in [d], M_i^0 = 0$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots$ do
   Receive $x_t$
   if ($V_t$ is not empty) then
      choose any $h \in V_t$
      predict $p_t = h(x_t)$
      Receive true answer $y_t = h^*(x_t)$
      Update $V_{t+1} = \{h \in V_t : (h(x_t) = y_t)\}$
      Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
   else
      Receive expert advice $(h_1^t(x_t), h_2^t(x_t), \cdots, h_d^t(x_t)) \in \{0,1\}^d$
      Define $w_i^{t-1} = \frac{e^{-\eta M_i^{t-1}}}{\sum_{j=1}^d e^{-\eta M_j^{t-1}}}$
      Define $\hat{p}_t = \sum_{i:h_i^t(x_t)=1} w_i^{t-1}$
      Predict $\hat{y}_t = 1$ with probability $\hat{p}_t$
      Receive true answer $y_t$
      Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
   end if
end for
```

3.3.1.1 Analysis of New_Consistent_WM():

Let T be the number of input points and $|\mathcal{H}|$ be the size of the hypothesis class. We assume that $|\mathcal{H}| < \infty$ The basic idea here is to use Consistent() algorithm in the beginning of the sequence. Since consistent() algorithm discards all the hypothesis which predicts wrong on any input point, we keep using the predictions of Consistent() until its current hypothesis class V_t becomes empty. Depending on whether the sequence is realizable or not, there are following two cases to be analysed:

3.3.1.2 Case 1 : When input sequence is realizable by $\mathcal{H}$

From section 2.1, Algorithm 2, we know that mistake bound of Consistent() algorithm is $|\mathcal{H}| - 1$. Therefore in the worst case, instance of Consistent() will make as many as $|\mathcal{H}| - 1$ mistakes. Thereafter, it will not make any mistake.

Since in this case, even after processing $|\mathcal{H}| - 1$ points, V_t will not be empty and will contain at least one hypothesis. These are the those hypotheses which realizes the input sequence. Hence, Consistent() part will be continuing for the rest of the $T - |\mathcal{H}| - 1$ points and will not make any mistake further. Thus, in the realizable case the mistake bound of proposed algorithm is following :

$$\mathbf{M_{New\_Consistent\_WM}}(\mathcal{H}, \mathbf{T}) \leq |\mathcal{H}| \tag{3.1}$$

3.3.1.3 Case 2 : When input sequence is not realizable by $\mathcal{H}$

When the input sequence is not realizable by the hypothesis class, the Consistent() portion of the algorithm can make at most $|\mathcal{H}|$ mistakes . Hence, till or before $|\mathcal{H}|$ points, V_t will become empty and hereafter, Weighted Majority will starting predicting.

Note that, while Consistent() portion was predicting and whenever it was making mistakes, we were simultaneously updating the mistake count of each hypothesis. This leaves weighted Majority in the same state as if Weighted Majority() was being used from the beginning itself. Since Weighted Majority() receives the mistake count list containing mistake count of each hypothesis on the previous points, it makes predictions on the remaining points using this list. This helps Weighted majority() distribute the weights according to the mistakes count of hypothesis on the previous points.

This can be guaranteed that the point where V_t becomes empty, mistake count of each hypothesis is ≥ 1. If not so, algorithm would have been continued using consistent() with the hypothesis whose mistake count is 0. This also helps Weighted Majority() to initialize mistake count of each hypothesis in the beginning of the Weighted Majority() part in the *New_Consistent_WM* algorithm.

From the section 2.2, Theorem 3, we know that Weighted Majority() enjoys the following expected regret bound on a given input sequence of length T

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] - \min_{i \in [d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \sqrt{0.5\ \ln(|\mathcal{H}|)\, T} \tag{3.2}$$

After analyzing the realizable and unrealizable cases separately, we present the following theorem which presents the regret bound of the proposed *New_Consistent_WM*() algorithm.

Theorem 6. *Algorithm 8 enjoys the following expected regret bound*

(a) In realizable case :

$$M_{New\_Consistent\_WM}(\mathcal{H}, T) \leq |\mathcal{H}|$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5 \ln(|\mathcal{H}|)\,(T - |\mathcal{H}|))} + |\mathcal{H}|$$

Proof. Since the input sequence either be realizable or unrealizable, we have discussed the bounds in both of the cases separately and equation 3.1 and 3.2 clearly establish the above required bound.

□

In the following section, we couple the weighted majority algorithm with halving() algorithm described in the section 1.2.

3.3.2 Weighted Majority with Halving

The proposed algorithm is given in Algorithm 9.

Algorithm 9 *New_Halving_WM()*

```
Input: A finite hypothesis class $\mathcal{H}$ containing d experts. Number of rounds(input points) $T$
Initialize: $\eta = \sqrt{2\ln(d)/T}\,; \forall i \in [d], M_i^0 = 0$
Initialize: $V_1 = \mathcal{H}$
for $t = 1, 2, \cdots$ do
    Receive $x_t$
    if ($V_t$ is not empty) then
        Predict $p_t = \arg\max_{r \in \{0,1\}} |\{h \in V_t : (h(x_t) = r)\}|$
        (in case of a tie predict $p_t = 1$)
        Receive true answer $y_t = h^*(x_t)$
        Update $V_{t+1} = \{h \in V_t : (h(x_t) = y_t)\}$
        Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
    else
        Receive expert advice $(h_1^t(x_t), h_2^t(x_t), \cdots, h_d^t(x_t)) \in \{0,1\}^d$
        Define $w_i^{t-1} = \frac{e^{-\eta M_i^{t-1}}}{\sum_{j=1}^{d} e^{-\eta M_j^{t-1}}}$
        Define $\hat{p}_t = \sum_{i:h_i^t(x_t)=1} w_i^{t-1}$
        Predict $\hat{y}_t = 1$ with probability $\hat{p}_t$
        Receive true answer $y_t$
        Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
    end if
end for
```

3.3.2.1 Analysis of New_Halving_WM()

Let T be the number of input points and $|\mathcal{H}|$ be the size of the hypothesis class. We assume that $|\mathcal{H}| < \infty$ The basic idea here is to use Halving() algorithm in the beginning of the sequence. Since Halving() algorithm discards at least half of the hypothesis once it makes any mistake, we keep using the predictions of Halving() until its current hypothesis class V_t becomes empty. Depending on whether the sequence is realizable or not, there are following two cases to be analyzed:

3.3.2.2 Case 1 : When input sequence is realizable by $\mathcal{H}$

From section 2.1, Algorithm 3, we know that mistake bound of Halving() algorithm is $\log_2(|\mathcal{H}|)$. Therefore in the worst case, instance of Halving() will make as many as $\log_2(|\mathcal{H}|)$ mistakes. Thereafter, it will not make any mistake.

Since in this case, even after processing $\log_2(|\mathcal{H}|)$ points, V_t will not be empty and will contain at least one hypothesis. These are the those hypotheses which realizes the input sequence. Hence, Halving() part will be continuing prediction for the rest of the $T - \log_2(|\mathcal{H}|)$ points and will not make any mistake further. Thus, in the realizable case, the mistake bound of proposed algorithm is following :

$$\mathbf{M}_{\mathbf{New\_Halving\_WM}}(\mathcal{H}, \mathbf{T}) \leq |\log_{\mathbf{2}}(|\mathcal{H}|) \tag{3.3}$$

3.3.2.3 Case 2 : When input sequence is not realizable by $\mathcal{H}$

When the input sequence is not realizable by the hypothesis class, the Halving() portion of the algorithm can make at most $\log_2(|\mathcal{H}|)$ mistakes . Till or before $\log_2(|\mathcal{H}|)$ points, V_t will become empty and hereafter, Weighted Majority will starting predicting.

Note that, while Halving() portion was predicting and whenever it was making mistakes, we were simultaneously updating the mistake count of each hypothesis. This leaves Weighted Majority() in the same state as if the Weighted Majority() was being used from the very beginning. Since weighted majority() receives the mistake count list containing mistake count of each hypothesis on the previous points, Weighted Majority() makes predictions on the remaining points using this list. This helps Weighted Majority() distribute the weights according to the mistakes count of hypothesis on the previous points.

This can be guaranteed that the point where V_t becomes empty, mistake count of each hypothesis is ≥ 1. Because, otherwise, algorithm would have been continued using Halving() with the hypothesis whose mistake count is 0. This also helps Weighted Majority() to initialize mistake count of each hypothesis in the beginning of the Weighted Majority() part in the $New\_Halving\_WM()$ algorithm.

From the section 2.2, Theorem 3, we know that Weighted Majority() enjoys the following expected regret bound on a given input sequence of length T

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] - \min_{i \in [d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \sqrt{0.5 \ln(\mathcal{H})\, T} \tag{3.4}$$

After analysing the realizable and unrealizable cases separately, we present the following theorem which presents the regret bound of the proposed $New_Halving_WM()$ algorithm.

Theorem 7. *Algorithm 9 enjoys the following expected regret bound*

(a) In realizable case :

$$M_{New\_Halving\_WM}(\mathcal{H}, T) \leq \log_2(|\mathcal{H}|)$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5 \ln(|\mathcal{H}|)\,(T - \log_2 |\mathcal{H}|))} +$$

$log_2(|\mathcal{H}|)$

Proof. Since the input sequence either be realizable or unrealizable, we have discussed the bounds in both of the cases separately and equation 3.3 and 3.4 clearly establish the above required bound. □

3.3.3 Weighted Majority with SOA

As mentioned in the beginning of this section 3.3, this algorithm enjoys the least expected regret bound. This comes from the fact that unlike $Halving()$, it uses Ldim of the two different sets $V_t^{(0)}$ and $V_t^{(1)}$ and uses the set for prediction which has the larger Ldim. The proposed algorithm is given in the **Algorithm 10** :

3.3.3.1 Analysis of New_SOA_WM()

Let T be the number of input points and $|\mathcal{H}|$ be the size of the hypothesis class. We assume that $|\mathcal{H}| < \infty$ The basic idea here is to use $SOA()$ algorithm in

Algorithm 10 $New\_SOA\_WM()$

```
**Input:** A finite hypothesis class $\mathcal{H}$ containing d experts. Number of rounds(input points) $T$
**Initialize:** $\eta = \sqrt{2\ln(d)/T}\,; \forall i \in [d], M_i^0 = 0$
**Initialize:** $V_1 = \mathcal{H}$
**for** $t = 1, 2, \cdots$ **do**
  Receive $x_t$
  **if** ($V_t$ is not empty) **then**
    for $r \in \{0,1\}$ let $V_t^{(r)} = \{h \in V_t : h(x_t) = r\}$
    predict $p_t = \arg\max_{r\in\{0,1\}} \text{Ldim}(V_t^{(r)})$
    (in case of a tie predict $p_t = 1$)
    Receive true answer $y_t = h^*(x_t)$
    Update $V_{t+1} = \{h \in V_t : (h(x_t) = y_t)\}$
    Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
  **else**
    Receive expert advice $(h_1^t(x_t), h_2^t(x_t), \cdots, h_d^t(x_t)) \in \{0,1\}^d$
    Define $w_i^{t-1} = \frac{e^{-\eta M_i^{t-1}}}{\sum_{j=1}^d e^{-\eta M_j^{t-1}}}$
    Define $\hat{p}_t = \sum_{i:h_i^t(x_t)=1} w_i^{t-1}$
    Predict $\hat{y}_t = 1$ with probability $\hat{p}_t$
    Receive true answer $y_t$
    Update $M_i^t = M_i^{t-1} + \mathbb{1}[h_i^t(x_t) \neq y_t]$
  **end if**
**end for**
```

the beginning of the sequence. From the Lemma 1.2, we know that Ldim($\mathcal{H}$) $\leq$ $log_2(|\mathcal{H}|)$. Therefore, $SOA()$ algorithm discards at least half of the hypothesis whenever it makes a mistake. we keep using the predictions of $SOA()$ until its current hypothesis class V_t becomes empty. Depending on whether the sequence is realizable or not, there are following two cases to be analysed:

3.3.3.2 Case 1 : When input sequence is realizable by $\mathcal{H}$

From section 2.1, Lemma 2, we know that mistake bound of SOA() algorithm is Ldim($\mathcal{H}$). Therefore in the worst case, instance of SOA() will make as many as Ldim($\mathcal{H}$) mistakes. Thereafter, it will not make any mistake.

Since in this case, even after processing Ldim($\mathcal{H}$) points, V_t will not be empty and will contain at least one hypothesis. These are the those hypotheses which realizes the input sequence. Hence, SOA() part will be continuing prediction for the rest of the $T -$ Ldim($\mathcal{H}$) points and will not make any mistake further. Thus, in the realizable case the mistake bound of proposed algorithm is following :

$$\mathbf{M_{New\_SOA\_WM}}(\mathcal{H}, \mathbf{T}) \leq \mathbf{Ldim}(\mathcal{H}) \tag{3.5}$$

3.3.3.3 Case 2 : When input sequence is not realizable by $\mathcal{H}$

When the input sequence is not realizable by the hypothesis class, the SOA() portion of the algorithm can make at most Ldim($\mathcal{H}$) mistakes . Till or before Ldim($\mathcal{H}$) points, V_t will become empty and hereafter, Weighted Majority() will starting predicting.

Note that, while SOA() portion was predicting and whenever it was making mistakes, we were simultaneously updating the mistake count of each hypothesis. This leaves Weighted Majority() in the same state as if the Weighted Majority() was being used from the very beginning. Since weighted majority() receives the mistake count list containing mistake count of each hypothesis on the previous points, Weighted Majority() makes predictions on the remaining points using this list. This helps Weighted Majority() distribute the weights according to the mistakes count of hypothesis on the previous points.

This can be guaranteed that the point where V_t becomes empty, mistake count of each hypothesis is ≥ 1. Because, otherwise, algorithm would have been continued using SOA() with the hypothesis whose mistake count is 0. This also helps Weighted Majority() to initialize mistake count of each hypothesis in the beginning of the Weighted Majority() part in the *New_SOA_WM*() algorithm.

From the section 2.2, Theorem 3, we know that Weighted Majority() enjoys the following expected regret bound on a given input sequence of length T

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}[\hat{y}_t \neq y_t]] - \min_{i \in [d]} \sum_{t=1}^{T} \mathbb{1}[h_i^t(x_t) \neq y_t] \leq \sqrt{0.5 \ln(|\mathcal{H}|)\, T} \tag{3.6}$$

After analysing the realizable and unrealizable cases separately, we present the following theorem which presents the regret bound of the proposed New_SOA_WM() algorithm.

Theorem 8. *Algorithm 9 enjoys the following expected regret bound*

(a) In realizable case :

$$M_{New\_SOA\_WM}(\mathcal{H}, T) \leq Ldim(\mathcal{H})$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}1[\hat{y}_t \neq y_t]] - \min_{h\in\mathcal{H}} \sum_{t=1}^{T} \mathbb{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5\ \ln(|\mathcal{H}|)\,(T - Ldim(\mathcal{H})))} +$$

$$Ldim(\mathcal{H})$$

Proof. Since the input sequence either be realizable or unrealizable, we have discussed the bounds in both of the cases separately and equation 3.5 and 3.6 clearly establish the above required bound. □

Our second objective in section 3.1 was to get hold of the best function at the end of the sequence. It can be seen that it has also been achieved in all three proposed algorithms by keeping mistake count of each function. The functions which have least number of mistake count at the end; are the best functions. Note that, in realizable case, the functions which have mistake count equals to 0 are best functions.

In this section, we described several methods for the finite hypothesis class and both realizable or unrealizable case setting. In particular, We presented three methods to improve the mistake bound for this setting using Weighted Majority with other methods picked up from the different setting named as finite hypothesis class and realizable case. Proposed methods do not improve the regret bound in general but they greatly improve the mistake bound for the realizable case while loosing a little in unrealizable case.

3.4 Contributions

Firstly, table 3.1 summarize the regret bounds of proposed algorithms and compare them with that of existing ones.

Seq	Algorithms	Mistake Bound (Realizable case)	Regret Bound (Unrealizable case)	Optimal Regret Bound (Unrealizable case)
1.	Weighted_Majority()	$\sqrt{0.5\ \ln(\mathcal{H})\,T}$	$\sqrt{0.5\ \ln(\mathcal{H})\,T}$	$(\sqrt{\text{Ldim}(\mathcal{H})\,T})$
2.	New_Consistent_WM()	$\|\mathcal{H}\|$	$\|\mathcal{H}\| + \sqrt{0.5\ \ln(\|\mathcal{H}\|)\,(T - \|\mathcal{H}\|)}$	Same as above
3.	New_Halving_WM()	$\log_2(\mathcal{H})$	$\log_2(\mathcal{H}) + \sqrt{0.5\ \ln(\|\mathcal{H}\|)\,(T - \log_2(\|\mathcal{H}\|))}$	Same as above
4.	New_SOA_WM()	$\text{Ldim}(\mathcal{H})$	$\text{Ldim}(\mathcal{H}) + \sqrt{0.5\ \ln(\|\mathcal{H}\|)\,(T - \text{Ldim}(\mathcal{H}))}$	Same as above

Table 3.1: Comparison of regret bounds of existing and proposed algorithms for finite hypothesis class and unrealizable case.

It can be observed that the mistake and regret bounds of Weighted_Majority are same. Proposed algorithms reduces the mistake bound by a large factor while loosing very little in regret bound.

For example, In case of New_Consistent_WM() ;

$$\sqrt{T} \rightarrow \sqrt{T - |\mathcal{H}|}$$

In case of New_Halving_WM() ;

$$\sqrt{T} \rightarrow \sqrt{T - \log_2(|\mathcal{H}|)}$$

and in case of New_SOA_WM() ;

$$\sqrt{T} \rightarrow \sqrt{T - \mathrm{Ldim}(\mathcal{H})}$$

Chapter 4
Simulations

This chapter gives supplementary backing to the results claimed in proposed methodologies in chapter 3. Since implementation of all the three proposed algorithms is very much similar to that of one another and Algorithm 9 is easy to implement, we are giving the implementation of this algorithm only named New_Halving_WM() given in section 3.3.2.

Although, the simulations presented in this chapter are neither necessary nor sufficient to prove bounds for any proposed algorithm. In fact, the behaviour of any online learning algorithms is completely analysed by theoretical proofs for mistake and regret bounds. Still, the simulations are presented in this chapter with the following objectives:

- To show that the proposed algorithms indeed conform to the given regret bound.
- To present a simple implementation scenario for the online learning framework.

For implementing proposed algorithms we need the following :

(i) Input sequence of T labelled points (realizable and unrealizable by $\mathcal{H}$ based on the scenario being discussed)

(ii) Finite hypothesis class $\mathcal{H}$

4.1 Construction of hypothesis class $\mathcal{H}$ and input sequence S :

This section describes the process of construction of hypothesis class and input sequence. This constructed input sequence may be realizable or unrealizable depending on the hypothesis class $\mathcal{H}$.

For implementation purpose, we need to generate both types of input sequence. To analyse the realizable case, we need to generate it in such a way that $\exists\, h^* \in \mathcal{H}$ such that

$$h^*(x_t) = y_t\,;\ \forall t \in [T]$$

We generate points of sequence S satisfying the following :

$$x_t \in (-T/2+1, T/2)$$

and

$$y_t \in \{0,1\}$$

4.1.1 Generating hypothesis class $\mathcal{H}$ or function class

Since we have T many points and all $x_t \in (-T/2+1, T/2)$, the following is the way adopted for generating function such that neither all the functions make correct predictions on all the x_t (except the realizable case) nor all the functions make mistakes on all the x_t.

$$h_i(x_t) = \begin{cases} 0 & \text{if } x_t \leq i \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

That is, i^{th} function will assign label 0 to all the x_t which are $\leq i$ and assigns 1 to all those x_t which are > 1. Naturally, i should lie in $[d]$, where d is the number of functions in hypothesis class (ie $d = |\mathcal{H}|$) and for this example $d \leq T$

So our hypothesis class becomes the set of these h_i:

$$\mathcal{H} = \{h_i : i \in [d]\} \tag{4.2}$$

4.1.2 Generating input sequence S labeled by some h

Here is the following way adopted for generating realizable and Unrealizable sequence S of length T.

4.1.2.1 Constructing Realizable Sequence :

For realizable sequence, we need to generate the labels of points using such a function(hypothesis) h^* which can be easily included in the hypothesis class.(Here "easily" means that this function should be similar to other functions of the hypothesis class.)

To make the sequence realizable, labels of the sequence are generated by the following function.

$$h_0(x_t) = \begin{cases} 0 & \text{if } x_t \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{4.3}$$

That is, it assigns label 0 to all non positive points and label 1 to all positive points. The table 4.1 demonstrate an example of *a* sequence generated by the function in eq 4.3 with $T = 8$.

Table 4.1: Example sequence of length 8 which is realizable by the hypothesis class constructed according to function definition given by equation 4.3

t	x_t	y_t
1.	-3	0
2.	-2	0
3.	-1	0
4.	0	0
5.	1	1
6.	2	1
7.	3	1
8.	4	1

So the sequence is

$$S = \{(-3,0),(-2,0),(-1,0),(0,0),(1,1),(2,1),(3,1),(4,1)\}$$

Note that h_0 realizes this entire sequence.

4.1.2.2 Constructing Unrealizable Sequence :

To make the sequence unrealizable, labels are generated by the following function.

$$h(x_t) = 1 ; \ \forall x_t \tag{4.4}$$

That is, it assigns label 1 to all points. Since some x_t of the sequence are negative and some of them are positive and no single function provides the label 1 to all the points of the sequence; this constructed sequence will never be realizable by the below constructed hypothesis class $\mathcal{H}$.

Table 4.2 presents an example of a sequence generated by the function in eq 4.4 with $T = 8$.

Table 4.2: Example sequence of length 8 which is realizable by the hypothesis class constructed according to function definition given by equation 4.4.

t	x_t	y_t
1.	-3	1
2.	-2	1
3.	-1	1
4.	0	1
5.	1	1
6.	2	1
7.	3	1
8.	4	1

Table 4.3 shows the prediction of each function given by eq 4.2 with $d = 5$ over the following sequence(same as defined before).

$$S = \{(-3,0),(-2,0),(-1,0),(0,0),(1,1),(2,1),(3,1),(4,1)\}$$

Here $M(h_i)$ denotes the total number of mistakes made by the hypothesis i on the entire sequence S.(Red coloured number shows the wrong prediction.)

It can be seen that h_0 does not make any mistake. Hence it can realize the sequence S.

4.2 Simulation of New_Halving_WM Algorithm

As already mentioned that implementation of all the three proposed algorithms (**Algorithm 8, 9, 10**) is very much similar to that of one another, we are giving the implementation of only Algorithm 9 named New_Halving_WM() given in

Table 4.3: Showing predictions and total mistake count of each function on the entire sequence S.

t	x_t	y_t	$h_0(x_t)$	$h_1(x_t)$	$h_2(x_t)$	$h_3(x_t)$	$h_4(x_t)$
1.	-3	0	0	0	0	0	0
2.	-2	0	0	0	0	0	0
3.	-1	0	0	0	0	0	0
4.	0	0	0	0	0	0	0
5.	1	1	1	0	0	0	0
6.	2	1	1	1	0	0	0
7.	3	1	1	1	1	0	0
8.	4	1	1	1	1	1	0
$\mathbf{M(h_i)}$	$--$	$--$	0	1	2	3	4

section 3.3.2.

From theorem 7, algorithm 9 enjoys the following regret bound :

(a) In realizable case :

$$M_{New\_Halving\_WM}(\mathcal{H}, T) \leq \log_2(|\mathcal{H}|)$$

(b) In unrealizable case :

$$\sum_{t=1}^{T} \mathbb{E}[\mathbf{1}[\hat{y}_t \neq y_t]] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}[h(x_t) \neq y_t] \leq \sqrt{(0.5 \ln(|\mathcal{H}|)\,(T - \log_2 |\mathcal{H}|))} +$$

$log_2(|\mathcal{H}|)$

In realizable case, we analyse the "expected" mistake bound which can be described as follows :

$$\textbf{Expected\_Mistakes} = \frac{\textbf{Total\_number\_of\_mistakes}}{\textbf{All\_possible\_permutations\_of\_the\_input\_sequence}}$$

In unrealizable case, we analyse the "expected" regret bound which can be represented in terms of mistake bound as follows

$$\textbf{Expected\_Regret} = \textbf{Expected\_Mistakes} - \textbf{Mistakes\_by\_best\_functions}$$

Here we have "expected" mistake or regret bound. Then, ideally we need to count mistakes made by the algorithm over all possible permutations of an input sequence. it is extremely time consuming or in fact computationally infeasible to check the expected mistake count of any learning algorithm over all possible sequences of a long input sequence. Therefore, we either provide expected mistakes count on all possible permutations of very short sequence (e.g. $T = 8 \,\text{or}\, 9$) or we generate some random permutations of a long input sequence (when $T \sim 1000 \,\text{or}\, 10,000$).

4.3 Simulation results

This section presents the simulation results for the realizable and unrealizable cases separately. It, further, presents simulation results with randomization of realizable and unrealizable sequences. i.e. we will randomly pick any sequence from the set of realizable and unrealizable sequences.

4.3.1 Realizable case

Im realizable case, we assume that $\exists\, h^* \in \mathcal{H}$ such that

$$h^*(x_t) = y_t\,;\ \forall t \in [T]$$

hence,

$$M_{h^*}(S) = 0$$

But depending on the order in which points of input sequence are presented to the algorithm, algorithm can make any number of mistakes ranging from 0 to $\ln(\mathcal{H})$. i.e.

$$M_A(H, S) \in [\log_2(|\mathcal{H}|)]$$

These mistakes are essentially the mistakes made by the algorithm to find that best hypothesis (function) h^*.

Table 4.4 presents some simulation results in where S is realizable by the constructed hypothesis class $\mathcal{H}$.

4.3.1.1 Observations :

It can be seen that this difference is clearly very huge in realizable case and New_Halving_WM() algorithm really outperforms in the realizable case.

T	*Permuta−tions*	$\lvert\mathcal{H}\rvert$	$M(h_m)$	$WM()$ *total mistakes* (1)	$WM()$ *expected mistakes*	$WM()$ *expected mistake bound*	*Max mist−akes*	*Min mist−akes*	*New()* *total mista −kes* (2)	*New()* *expected mistakes*	*New()* *mistake bound*	*Max mist−akes*	*Min mist−akes*	*Diff* (1) − (2)
1000	100	500	0	3,576	35.76	55.74	46	23	325	3.25	8	5	1	3,251
1000	100	500	0	3,663	36.63	55.74	49	25	331	3.31	8	6	1	3,332
1000	100	500	0	3,636	36.36	55.74	46	25	318	3.18	8	6	1	3,318
8	40,320	4	0	53,136	1.31	2.35	3	0	36,964	0.91	2	2	0	16,172
8	40,320	4	0	53,305	1.32	2.35	3	0	36,999	0.91	2	2	0	16,306
8	40,320	4	0	53,052	1.31	2.35	3	0	37,003	0.91	2	2	0	16,049

Table 4.4: Showing comparison of mistake counts of existing and proposed algorithms in realizable case on all permutations of the sequence S given in table 4.1 and some randomly generated permutations of large sequence of 1000 points.

Due to space constraint in table, we rename new algorithm as : New_Halving_WM() → New().

In this table, h_m denotes the best function. i.e. which makes least number of misatkes over the entire input sequence.

The values in "$Diff$" column denotes the difference between the mistakes made by Weighted_Majority() algorithm and the New_Halving_WM()

We also observe that mistakes for both original and proposed one are conforming to the theoretical bounds.

4.3.2 Unrealizable case

In unrealizable case, we analyse the regret bound rather than the mistake bound. In other words, regret bound is essentially a mistake bound when mistakes are counted w.r.t the best hypothesis in the class.

Table 4.5 presents some simulation results in where S is not realizable by the constructed hypothesis class $\mathcal{H}$.

4.3.2.1 Observations :

If we observe the values in "$Diff$" column in table 4.5 and compare the "$Diff$" values from table 4.4, we see that we are still gaining very minimal in the unrealizable case. But, in general, we can not gain anything in the unrealizable case using the proposed algorithms. This unexpected gain shows that, in general, we will loose very little in the unrealizable case.

Similar to the table 4.4, table 4.5 also shows that the regrets for both original and proposed ones are conforming to their corresponding theoretical bounds.

T	*Permuta−tions*	$\vert\mathcal{H}\vert$	$M(h_m)$	*WM() total regret* (1)	*WM() expected regret*	*WM() expected regret bound*	*Max mist−akes*	*Min mist−akes*	*New() total regret* (2)	*New() expected regret*	*New() regret bound*	*Max mist−akes*	*Min mist−akes*	*Diff* (1) − (2)
1000	100	500	500	3,615	36.15	55.74	49	26	3593	35.93	64.49	47	0	22
1000	100	500	500	3,636	36.86	55.74	46	29	3,610	36.10	64.49	44	0	26
1000	100	500	500	3,584	35.84	55.74	51	28	3,584	35.84	64.49	44	0	0
8	40,320	4	4	3,187	1.31	2.88	7	4	1,908	1.28	4.03	3	0	1,279
8	40,320	4	4	3,461	1.32	2.88	7	4	2,106	1.29	4.03	3	0	1,355
8	40,320	4	4	3,462	1.32	2.88	7	4	1,703	1.28	4.03	3	0	1,759

Table 4.5: Showing comparison of regret counts of existing and proposed algorithms in unrealizable case on all permutations of the sequence S given in table 4.1 and some randomly generated permutations of large sequence of 1000 points.

Due to space constraint in table, we rename new algorithm as : New_Halving_WM() → New().

In this table, h_m denotes the best function. i.e. which makes least number of misatkes over the entire input sequence.

The values in "*Diff*" column denotes the difference between the regrets of Weighted_Majority() algorithm and the New_Halving_WM()

Note that regret values in the this table are obtained by subtracting mistakes of best expert from the mistakes made by each algorithm compared here.

Chapter 5
Conclusion, Discussion and Future Work

In this dissertation, we proposed three algorithms for the finite hypothesis class and both realizable and unrealizable cases. Proposed algorithms are designed by coupling the existing best algorithms available for realizable and unrealizable cases.

The motivation behind proposed algorithms was to reduce the mistakes which Weighted_Majority() makes in realizable case. Because, no matter the sequence is realizable or not, regret bound was same. This was exploited by running Weighted_Majority() in parallel. By this, if the input sequence is found to be realizable, algorithm will make very less mistake. If not, then we updated mistake count of each function which helped Weighted_Majority() take over later on and predict optimally thereafter.

The major contribution of this dissertation is to propose algorithms which perform really outstanding in realizable case but slightly worse in unrealizable case. This nature of performance of the proposed algorithms is very useful where we have some preliminary information about the realizability of the sequences. If input sequences are likely to be realizable than proposed algorithms will always be far better than the existing ones.

The way in which Weighted_Majority() runs in parallel indicates the scope of further improvement in theoretical bound of proposed algorithms in the unrealizable case. Further, the same approach can be applied in the other setting also of online learning; such as limited feedback model, stochastic noise model etc.

Bibliography

[1] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, Feb. 2012.

[2] V. Vapnik, E. Levin, and Y. Le Cun, "Measuring the vc-dimension of a learning machine," *Neural Computation*, vol. 6, no. 5, pp. 851–876, 1994.

[3] M. J. Kearns, R. E. Schapire, and L. M. Sellie, "Toward efficient agnostic learning," *Machine Learning*, vol. 17, no. 2-3, pp. 115–141, 1994.

[4] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games.* Cambridge university press, 2006.

[5] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, pp. 212–261, Feb. 1994.

[6] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Machine learning*, vol. 2, no. 4, pp. 285–318, 1988.

[7] L. Rosasco, J. Bouvrie, R. Rifkin, and T. Poggio, *Statistical Learning Theory and Applications.* Available at `http://www.mit.edu/~9.520/spring08/Classes/class01_2008.pdf`, [Accessed on June, 2016].

[8] A. Blum, "On-line algorithms in machine learning," in *Online algorithms*, pp. 306–325, Springer, 1998.

[9] A. Rakhlin and K. Sridharan, "Statistical learning theory and sequential prediction," *Lecture Notes in University of Pennsyvania*, 2012.