Note: These lecture notes are closely based on lecture notes by Sanjeev Arora [1] and the survey paper by Arora et al. [2]

1 Decision Making with Total Uncertainty

What if we need to make a decision and there is no a priori distribution on the outcomes. This type of decision making frequently occurs in the field of *online algorithms*, where the input appear piece by piece instead of all at once. The algorithm must take an action after receiving each piece of the input and may learn later, after seeing more pieces, that its past actions were suboptimal.

The book "Online Algorithms" by Borodin and El-Yaniv is a great introduction to this broad area of CS theory. In this lecture, we focus on one particular online algorithm called multiplicative weights update (MWU). It is one of the most successful online algorithms, because its minimal assumptions allow it be applied to many different problems.

2 Motivating Example: Weighted Majority

Imagine the process of picking the right time to buy a single stock. Assume that the daily price movement of the stock is just modeled by a single bit that indicates whether the stock goes up or down.

The stock's movement is arbitrary and even adversarial. However, we assume that we have access to the advice of n "experts" when making our predictions. Our goal is to limit cumulative loss (i.e. incorrect predictions) to roughly the same as the best expert.

At first, this seems impossible since the algorithm never learns who the best expert is till the end of the sequence of days, but the algorithm is required to make predictions at the beginning of each day. A naive algorithm is to always go with the majority opinion of the experts. But this fails since the majority could be consistently wrong, even though the best expert is consistently right.[1]

The *weighted majority algorithm* does better. It maintains a weighting of all experts, updating their weight based on their past accuracy, and then makes a decision in each turn by following the weighted majority.

Weighted Majority Set $0 < \eta \leq 1/2$. For each expert i, set $w_i^{(1)} \leftarrow 1$ For $t = 1, 2, \ldots T$

1. Make the prediction that is the weighted majority of the experts predictions based on the weights $w_1^{(t)}, \ldots w_n^{(t)}$. That is predict "up" or "down" depending on which prediction has higher total weight (breaking ties arbitrarily).

2. For every expert i who predicts wrongly, decrease their weight for the next round by multiplying by a factor of $1 - \eta$:
$$w_i^{(t+1)} \leftarrow (1 - \eta) w_i^{(t)}$$

2.1 Case: $\eta = 1/2$

Consider the case where $\eta = 1/2$. In other words, the experts that make a mistake have their weight halved. Let $M^{(t)}$ be the number of mistakes of our algorithm, and $m_i^{(t)}$ be the number of mistakes of expert i at time t. Let $W^{(t)}$ be the weight of all experts at time t, and for any expert i, let $w_i^{(t)}$ be the weight of expert i at time t. Note the following.

[1] In real life, going with the majority opinion of stock analysts is almost always a terrible strategy.

- Total weight of all experts at round T is no more than $n(3/4)^{M^{(T)}}$. To see this, consider the case where the algorithm makes a mistake at round t. Then the weight of all experts who predicted incorrectly at time t is at least $W^{(t)}/2$. These experts who predicted incorrectly will be penalized by having the weights decreased by $1/2$. Thus, at time $t+1$, $W^{(t+1)} \leq (1/2)W^{(t)} + (1/4)W^{(t)} \leq 3/4W^{(t)}$. Finally, note that $W^1 = n$. A simple induction shows that $W(t) \leq n(3/4)^{M^{(T)}}$.
- For any expert i, $w_i^{(t)} = (1/2)^{m_i^{(t)}}$
- At any time t, for any expert i, $W(t) \geq w_i^{(t)}$.

From these facts, we have that:

$$n(3/4)^{M^{(T)}} \geq (1/2)^{m_i^{(T)}}$$

Taking the logs of both sides of this inequality, we get that

$$\log n + M^{(T)} \log(3/4) \geq m_i^{(T)} \log(1/2)$$

Rearranging we get:

$$\log n + M^{(T)} \log(3/4) \geq -m_i^{(T)}$$

Multiplying by negative 1 and noting that $\log(3/4) = -\log(4/3)$, we get:

$$-\log n + M^{(T)} \log(4/3) \leq m_i^{(T)}$$

Rearranging to isolate $M^{(T)}$ and noting that $1/\log_2(4/3) = 2.4$, we get

$$M^{(T)} \leq 2.4 \left(m_i^{(T)} + \log n \right)$$

2.2 General Case

Theorem 1. *After T steps, let $m_i^{(T)}$ be the number of mistakes of the expert i, and let $M^{(T)}$ be the number of mistakes of our algorithm. Then we have the following for every expert i, $1 \leq i \leq n$.*

$$M^{(T)} \leq 2(1+\eta) m_i^{(T)} + \frac{2 \ln n}{\eta}$$

Proof: A simple induction shows that $w_i^{(t+1)} = (1-\eta)^{m_i^{(t)}}$. Consider the following potential function: $\Phi(t) = \sum_i w(i,t)$. Note that $\Phi(1) = n$. Each time we make a mistake, the weighted majority of experts also made a mistake, so at least half the total weight decreases by a factor of $1-\eta$. Thus, the potential function decreases by a factor of at least $1 - \eta/2$ since:

$$\Phi(t+1) \leq \Phi(t) \left(\frac{1}{2} + \frac{1}{2}(1-\eta) \right) = \Phi(t)(1 - \eta/2)$$

Then a simple induction gives that $\Phi(T+1) \leq n(1-\eta/2)^{M^{(T)}}$. Finally, note that $\Phi(T+1) \geq w(i, T+1) = (1-\eta)^{m_i^{(T)}}$. Putting these two inequalities together, we get that:

$$n(1-\eta/2)^{M^{(T)}} \geq (1-\eta)^{m_i^{(T)}}$$

Now we want to isolate $M^{(T)}$ in the above inequality. We can do this by taking ln of both sides to get:

$$\begin{aligned}
&\ln N + M^{(T)} \ln(1-\eta/2) \geq m_i^{(T)} \ln(1-\eta)) \\
\iff &-\ln N - \ln(1-\eta/2) M^{(T)} \leq -\ln(1-\eta)) m_i^{(T)} \\
\iff &-\ln N + (\eta/2) M^{(T)} \leq (\eta+\eta^2) m_i^{(T)} \\
\iff &M(T) \leq 2(1+\eta) m_i^{(T)} + \frac{2\ln N}{\eta}
\end{aligned}$$

The only tricky part in the above is going from the second to the third line, where we use two inequalities:

$$-\ln(1-x) \geq x$$
$$-\ln(1-x) \leq x + x^2, \quad \forall x \leq 1/2$$

To see the first inequality, note that $1-x \leq e^{-x}$, take ln of both sides, and multiply by -1.

To see the second inequality, note that it is equivalent to proving that $\ln(1-x) \geq -x - x^2$. Then, note that the Taylor expansion of $\ln(1-x) = -x - x^2/2 - x^3/3 - x^4/4 \ldots \geq -x - x^2$, for $x \leq 1/2$.

How to show that $-x - x^2/2 - x^3/3 - x^4/4 \ldots \geq -x - x^2$ for $x \leq 1/2$??? Idea: let $S = x^3/3 + x^4/4 + \ldots \leq (1/3)(x^3 + x^4 + \ldots)$. Note that $S' = \sum_{i=3}^{\infty} x^i$ is a geometric sum, and so it equals $x^3/(1-x) \leq 2x^3$ for $x \leq 1/2$. To see this, note that $xS' = \sum_{i=4}^{\infty} x^i$, and so $S' - xS' = (1-x)S' = x^3$, and so $S' = x^3/(1-x)$.

So we know that $S = x^3/3 + x^4/4 + \ldots \leq (1/3)(x^3 + x^4 + \ldots) \leq 2/3x^3 \leq 1/3x^2$. Hence, $-x - x^2/2 - x^3/3 - x^4/4 \geq -x - x^2/2 - x^2/3 \geq -x - x^2$. □

3 Multiplicative Weights Update

In the most general setting, we have a choice of n decisions in each round, from which we select 1. In each round, each decision incurs a certain *cost*, determined by nature (or adversary). All costs are revealed only after we chose our decision and incur the cost associated with it. For example, in the prediction from advice problem, each decision is a choice of an expert and the cost of an expert is 1 if the expert makes a mistake and 0 otherwise.

Let $t = 1, 2, \ldots, T$ denote the current round, and let i, $1 \leq i \leq n$ be a generic decision. In each round, we select a distribution $p^{(t)}$ over the set of decisions and select a specific decision i according to this distribution. Then, the costs of all decisions are revealed in the form of a vector $m^{(t)}$, where $m_i^{(t)}$ is the cost of decision i in round t. We assume that all costs lie in $[-1, 1]$, and that, again, these costs can be chosen by an adversary.

We next describe the algorithm. It is similar to weighted majority, with the following differences: (1) it is randomized; (2) it handles costs in $[-1, 1]$ rather than in $\{0, 1\}$.

Multiplicative Weights Algorithm Set $\eta \leq 1/2$. For each decision i, set $w_i^{(1)} \leftarrow 1$ For $t = 1, 2, \ldots T$

1. Choose decision i with probability proportional to its weight $w_i^{(t)}$. I.e. use the distribution $p_i^{(t)} = w_i^{(t)}/\phi(i)$, where $\phi(i) = \sum_i w_i^{(t)}$.

2. Get costs $m^{(t)}$ and update weights as follows:

$$w_i^{(t+1)} \leftarrow \left(1 - \eta m_i^{(t)}\right) w_i^{(t)}$$

Note that the expected cost (over distribution $p^{(t)}$) in round t for the algorithm is: $m^{(t)} \cdot p^{(t)}$. Thus, the total expected cost over all rounds is

$$\sum_{i=1}^{T} m_i^{(t)} \cdot p^{(t)}$$

Our goal is that this total expected cost is not too much more than the cost of the best decision in hindsight:

$$\min_i \sum_{i=1}^{T} m_i^{(t)}$$

We have the following theorem.

Theorem 2. *Multiplicative Weights guarantees that after T rounds, for any decision i:*

$$\sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}| + \frac{\ln n}{\eta}$$

Proof: The proof of this theorem is similar to that for the weighted majority algorithm. It is given in [2]. □

3.1 Convex Combinations of Decisions

Impressively, Multiplicative Weights (MWU) also tracks the best *probability distribution*, p over the decisions.

Theorem 3. *Multiplicative Weights guarantees that after T rounds, for any distribution p:*

$$\sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \leq \left(\sum_{t=1}^{T} m^{(t)} + \eta |m^{(t)}|\right) \cdot p + \frac{\ln n}{\eta}$$

Proof: Follows from Theorem 4 by taking a convex combination of the inequalities over all decisions i with distribution p. For example if we know that $x_1 \leq y_1$ and $x_2 \leq y_2$, it follows that $((1/2)x_1 + (1/2)x_2 \leq (1/2)y_1 + (1/2)y_2$. And more generally, for any $\alpha : 0 \leq \alpha \leq 1$, $\alpha x_1 + (1-\alpha)x_2 \leq \alpha y_1 + (1-\alpha)y_2$. □

3.2 Gains Instead of Lossees

Consider the case where the vector $m^{(t)}$ specifies gains for each decision, and we want to maximize the total gains. We can get an algorithm for this case by simply using the cost vector $-m^{(t)}$ in the MWU algorithm above. The following theorem follows.

Theorem 4. *MWU for gains guarantees that after T rounds, for any decision i:*

$$\sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \geq \left(\sum_{t=1}^{T} m_i^{(t)} - \eta \sum_{t=1}^{T} |m_i^{(t)}|\right) \cdot p + \frac{\ln n}{\eta}$$

The convex combination result similarly holds for the case of gains.

Theorem 5. *MWU for gains guarantees that after T rounds, for any distribution p:*

$$\sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \geq \left(\sum_{t=1}^{T} m^{(t)} - \eta |m^{(t)}| \right) \cdot p + \frac{\ln n}{\eta}$$

4 Learning a Linear Classifier: The Winnow Algorithm

Assume[2] we are give m labelled examples $(a_1, \ell_1), \ldots (a_m, \ell_m)$, where the $a_i \in \mathbb{R}^n$ are *feature vectors* and the $\ell_i \in \{-1, +1\}$ are their labels (i.e. classifications). We want to find non-negative weights such that, for any example, the sign of the weighted combination of the features matches its label. In particular, we want to find vector $x \in \mathbb{R}^n$, such that $\forall i, 1 \leq i \leq n$, $x[i] \geq 0$ and $\forall j, 1 \leq j \leq m$, $\text{sgn}(x \cdot a_j) = \ell_j$.

Equivalently, we require that for all j, $1 \leq j \leq m$, $\ell_j a_j \cdot x \geq 0$. Without loss of generality, we can assume that x is normalized so that it forms a distribution. I.e.. $\vec{1} \cdot x = 1$, where $\vec{1}$ is the all 1's vector.

For simplicity, we will redefine a_j to be $\ell_j a_j$.

Then the problem reduces to finding a solution to the following linear program:

$$\begin{aligned} \forall j : 1 \leq j \leq m : \quad & a_j \cdot x \geq 0 \\ & \vec{1} \cdot x = 1 \\ & \forall_i : x_i \geq 0 \end{aligned}$$

This is a pretty general form of a linear program. There is nothing to maximize, we just want to find if there is a solution vector x.

Now suppose that there is a *large-margin* solution to this linear program. In particular, there is an $\epsilon > 0$ and a distribution x^* such that for all j, we have $a_j \cdot x^* \geq \epsilon$. We now give a (very fast) algorithm based on MWU to solve this linear program. Run MWU in gain form (Section 3.2) as follows.

The Winnow Algorithm

1. **Setup:** $\rho \leftarrow \max_{j,i} |a_j[i]|$. I.e., ρ is the maximum absolute value of any entries in any attribute vector. $\eta \leftarrow \epsilon/(2\rho)$. The decisions/experts are the n features.

2. **Each Round:** In round t, find a misclassified example (or if no such example exists, terminate). In particular, find an example j such that $a_j \cdot p^{(t)} < 0$. Then the gain for decision i, $1 \leq i \leq n$, in round t is $a_j[i]/\rho$.

4.1 Small Example Run

If we have examples $(.5, -.3, -1)$, $(-2, .5, 1)$, $(1, -2, -1)$, $(-2, 4, 1)$, we first redefine to get the four attribute vectors: $(-.5, .3)$, $(-2, .5)$, $(-1, 2)$, $(-2, 4)$.

In this example $\rho = 4$. We know, for example, that the vector $x^* = (0, 1)$ is a .3-margin classifier, so let's set $\epsilon = .3$. Let's also set $\eta = 1/2$.

Now in the first round, all weights are 1, so $p^{(1)} = (1/2, 1/2)$. This misclassifies the first and second example vectors, so we pick one (breaking ties arbitrarily) in order to set the gains for the

[2] This section is from Section 3.1 of Arora survey

first round. Let's pick the first example, so that the gains for this round will be $(1/\rho)(-.5, .3) = (-.08, .075)$.

Next we plug these gains into MWU in order to update the weights for the next round. We will get

$$w_1^{(t+1)} \leftarrow (1 - 1/2(.08))1$$

or

$$w_1^{(t+1)} \leftarrow w_1^{(t+1)} \leftarrow (1 - 1/2(.08))1$$

or

$$w_1^{(t+1)} \leftarrow .96$$

and

$$w_2^{(t+1)} \leftarrow (1 + 1/2(.075))1$$

or

$$w_2^{(t+1)} \leftarrow 1.0375$$

This make sense, since the second attribute is "rewarded" since it was in the correct direction, and the first attribute is "punished" since it was in the wrong direction. If we continue this algorithm, we will converge to a correct classifier, as is proven in the next section.

Note that the next probability vector $p^{(2)}$ is just the normalized weights, i.e. $(.96/1.9665, 1.0375/1.9665)$ since $(1.004 + .9625 = 1.9665)$. So,

$$p^{(2)} = (.48, .51)$$

4.2 Analysis

Note that, by construction, the gain for Winnow in round t is $m^{(t)} \cdot p^{(t)} = \frac{a_j}{\rho} \cdot p^{(t)}$, where a_j is an incorrectly classified example in round t. Note that this is is less than 0 until Winnow finds a correct solution. In contract, the gain for the large margin classifier is at least $m^{(t)} \cdot x^* = \frac{a_j}{\rho} \cdot x^* \geq \frac{\epsilon}{\rho}$. The following proof uses Theorem 5 to bound the amount of time till the gains of the algorithm become positive.

Theorem 6. *Winnow finds a solution in* $\lceil \frac{4\rho^2 \ln n}{\epsilon^2} \rceil$ *iterations.*

Proof: Let T be the number of rounds until Winnow finds a correct classifier, and let $t \leq T$. Note that the expected cost of Winnow in such a round is $m^{(t)} \cdot p^{(t)} = \frac{a_j}{\rho} \cdot p^{(t)} < 0$.

Also, note that the cost of x^* is $m^{(t)} \cdot x^* = \frac{a_j}{\rho} \cdot x^* \geq \frac{\epsilon}{\rho}$.

To get a bound on the number of iterations until we find a solution, we apply Theorem 5, letting $p = x^*$. Recall that $\eta = \epsilon/(2\rho)$ in the following:

$$\begin{aligned} 0 &> \sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \\ &\geq \sum_{i=1}^{T} \left(m^{(t)} - \eta |m^{(t)}| \right) \cdot p - \frac{\ln n}{\eta} \\ &\geq \frac{\epsilon}{2\rho} T - \frac{2\rho \ln n}{\epsilon} \end{aligned}$$

In the above, the last line holds since $m^{(t)} \cdot x^* \geq \frac{\epsilon}{\rho}$, and $\eta|m^{(t)}| \cdot x^* \leq \eta = \frac{\epsilon}{2\rho}$. (Recall $|m^{(t)}|$ is the vector containing entry-wise absolute values of $m^{(t)}$, which are the entries of a vector a_j scaled by $\frac{1}{\rho}$. Hence the norm of $|m^{(t)}|$ is no more than 1.)

Now all we need to do is solve for T in the above inequality. This gives us:

$$T \leq \frac{4\rho^2 \ln n}{\epsilon^2}$$

□

5 Learning Theory and Boosting

Let $\mathcal{X}$ be some set (domain) and suppose we are trying to learn an unknown classification function $\mathcal{C}$ mapping from $\mathcal{X}$ to $\{0, 1\}$. Given a set of training examples $(x, c(x))$, where x is generated from distribution $\mathcal{D}$, the learning algorithm is required to output a hypothesis $h : \mathcal{X} \to \{0, 1\}$. The *error* of the hypothesis is defined to be the expected value over all examples x chosen according to distribution $\mathcal{D}$ of $|h(x) - c(x)|$.

A *strong learning algorithm* (for $\mathcal{C}$) is one that for every distribution $\mathcal{D}$, given any $\epsilon, \delta > 0$, and access to random examples drawn from $\mathcal{D}$, outputs with probability at least $1 - \delta$ a hypothesis whose error is at most ϵ on $\mathcal{D}$. Further the runtime is required to be polynomial in $1/\epsilon$ and $1/\delta$.

A *γ-weak* learning algorithm for some $\gamma > 0$, is an algorithm satisfying the same conditions, but where the error can be as high as $1/2 - \gamma$.

In this section, we'll prove that if a γ-weak learning algorithm exists for a concept class then a strong learning algorithm exists.

5.1 Boosting Algorithm

Assume we are given some training set S drawn from the distribution $\mathcal{D}$. The idea of the algorithm is to run the weak learning algorithm on different distributions defined by S. Our goal is that the final hypothesis has error ϵ under the uniform distribution on S.

Boosting Algorithm

1. **Setup:** Set $\eta \leftarrow \gamma$ and $T \leftarrow \lceil (2/\gamma^2) \ln(1/\epsilon) \rceil$ The decisions/experts correspond to the *examples* in S.

2. **Each Round:** In round $t \leq T$, the algorithm presents the distribution $p^{(t)}$ to the weak learning algorithm, and gets a hypothesis $h^{(t)}$. Then, we set the cost vector $m^{(t)}$ as follows: for example x, $m_x^{(t)} = 1 - |h^{(t)}(x) - c(x)|$.

3. **Final Hypothesis:** The final hypothesis, h_{final}, labels example x according to the majority vote among $h^{(1)}(x), \ldots, h^{(T)}(x)$.

Note that this algorithm is increasing the weights of those examples that are misclassified.

5.2 Example Run

Assume that we have the following set S:

$$(0, 0) = 0;\ (1, 0) = 1;\ (0, 1) = 1;\ (1, 1) = 0$$

Note that the underlying concept $\mathcal{C}$ for this example is the exclusive-or function, which no linear classifier can learn exactly. So the concept is a bit tricky. But assume that we have a good weak-learner such as Winnow that can always classify with error no more than $1/2 - \gamma$, where $\gamma = 1/4$; i.e. with error no more than $1/4$. We set $\eta = \gamma = 1/4$

So $p^{(1)}$ is uniform - all examples are weighted equally. Let's represent the hypothesis from our weak classifier as a vector which classifies the example as 1 if the dot product with the example is greater than or equal to 1, and as 0 otherwise.

Say that in the first round, our weak classifier returns $h^{(1)} = (1, 1)$. Then $(1, 1) \cdot (0, 0) = 0$, $(1, 1) \cdot (1, 0) = 1$, $(1, 1) \cdot (0, 1) = 1$ and $(1, 1) \cdot (1, 1) = 2$. Thus the examples are classified as: $0, 1, 1, 1$, for a total error of $1/4$ (only the last example is misclassified. So now we set the cost vector $m^{(1)} \leftarrow (1, 1, 1, 0)$. Then we update the weights as follows: $w_i^{(2)} \leftarrow (1 - \eta m_i^{(1)}) w_i^{(1)}$, to get $w^{(2)} = (3/4, 3/4, 3/4, 1)$.

Then $p^{(2)} = (3/13, 3/13, 3/13, 4/13)$. So we call the weak learner again with this **new** distribution. Note that the weak learning algorithm now needs to pay more attention to the last example in order to get an error less than $1/4$. So our next hypothesis from the weak learner might be, $h^{(2)} = (1, -1)$, note that this classifies the examples 1,2 and 4 correctly, so the weighted error is less than $1/4$. The new cost vector is $m^{(2)} \leftarrow (1, 1, 0, 1)$. So the new weight $w^{(3)} = (9/16, 9/16, 3/4, 3/4)$. Then, the new $p^{(3)} = (3/14, 3/14, 4/14, 4/14)$

So in the third round, the third and fourth examples are weighted slightly higher in our distribution. In this round, the weak learner may return $h^{(3)} = (-1, 1)$. This classifies examples 1, 3, 4 correctly, giving a weighted probability of error less than $1/4$. Let's say that $T = 3$, so that we end here.

What is our final *strong*-classifier? It's the classifier that takes the majority vote of $h^{(1)}$, $h^{(2)}$ and $h^{(3)}$. How does this classifier do? On every example, at least two of the hypothesis are correct, so this new classifier has error of 0.

5.3 Analysis

Our analysis of this algorithm will make use of the following additional result above the abilities of the MWU algorithm.

MWU with Relative Entropy. For two probability distributions p and q, the relative entropy between them is defined as

$$RE(p||q) = \sum_i p_i \ln \frac{p_i}{q_i}$$

where the $p_i \ln \frac{p_i}{q_i}$ term is defined to be 0 if $p_i = 0$ and infinite if $p_i \neq 0$ and $q_i = 0$.

We will make use of the following, which is proven in [2], using relative entropy as the potential function.

Theorem 7. *Assume that all costs are in $[-1, 1]$ and $\eta \leq 1/2$. Then multiplicative weights guarantees that after T rounds, for any distribution p, we have*

$$\sum_{i=1}^{T} m^{(t)} \cdot p^{(t)} \leq \left(\sum_{t=1}^{T} m^{(t)} + \eta |m^{(t)}| \right) \cdot p + \frac{RE(p||p^{(1)})}{\eta}$$

Boosting Result.

Theorem 8. *After $T = (2/\gamma^2)\ln(1/\epsilon)$ rounds, the Boosting algorithm will return an ϵ-strong learner.*

Proof: In each iteration, the Boosting algorithm presents distribution $p^{(t)}$ over the examples S to the weak learning algorithm, and gets a hypothesis $h^{(t)}$ whose error with respect to $p^{(t)}$ is no more than $1/2 - \gamma$. In other words, the expected cost in each iteration satisfies:

$$m^{(t)} \cdot p^{(t)} \geq 1/2 + \gamma$$

The algorithm is run for T rounds. Let E be the set of $x \in S$ incorrectly labelled by h_{final}. The total cost for each $x \in E$ is:

$$\sum_t m_x^{(t)} = \sum_t 1 - |h^{(t)}(x) - c(x)| \leq \frac{T}{2}$$

since the majority vote gives an incorrect label for it. Now applying Theorem 7 and letting p be the uniform distribution on E, we get

$$\begin{aligned}(1/2+\gamma)\,T &\leq \sum_{t\leq T} m^{(t)} \cdot p^{(t)} \\ &\leq (1+\eta)\sum_{t\leq T} m^{(t)} \cdot p + \frac{RE(p||p^{(1)})}{\eta} \\ &\leq (1+\eta)\,\frac{T}{2} + \frac{\log(n/|E|)}{\eta}\end{aligned}$$

Note that $p^{(1)}$ is the uniform distribution over all $n = |S|$ examples so $p_i^{(1)} = 1/n$ for all i. And $p_i = 1/|E|$ if $i \in E$ and 0 otherwise. Thus $RE(p||p^{(1)}) = \log(n/|E|)$.

Noting that $\eta = \gamma$, we get the following

$$(\eta/2)T \leq \frac{\ln(n/|E|)}{\eta}$$

Plugging in $T = (2/\eta^2)\ln(1/\epsilon)$ (recall $\gamma = \eta$), and cross multiplying, we get

$$\begin{aligned}&\ln(1/\epsilon) \leq \ln(n/|E|) \\ \iff &-\ln(1/\epsilon) \geq -\ln(n/|E|) \\ \iff &\ln(\epsilon) \geq \ln(|E|/n) \\ \iff &\epsilon \geq |E|/n\end{aligned}$$

□

The last line holds by taking each side of the previous inequality to the power of e.

References

[1] Sanjeev Arora. Advanced Algorithm Design Class, Princeton University, 2013. https://www.cs.princeton.edu/courses/archive/fall15/cos521/.

[2] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.