CMSC651 Project

Yancy Liao, Fan Yang, Bowen Zhi

May 2018

Problem 5.2

Assume that there exists an $a \in \mathbb{R}$ s.t. $err_{\mathcal{D}}(f_a) = 0$, which we know exists if the data is separable by some threshold function. Define the set $G \subset \mathbb{R}$ as $\{a' \in \mathbb{R} \mid err_{\mathcal{D}}(f_{a'}) \leq \epsilon\}$. Also define the set $T(S) \subset \mathbb{R}$ as $\{a' \in \mathbb{R} \mid err_S(f_{a'}) = 0\}$. Note that our desired result (with probability $\geq 1-\delta$, every a' s.t. $err_S(f_{a'}) = 0$ has $err_{\mathcal{D}}(f_{a'}) \leq \epsilon$) can be thus equivalently expressed as: with probability $\geq 1-\delta$, $T(S) \subset G$. Also, let $p_{\mathcal{D}}$ be the probability density function of distribution $\mathcal{D}$, and define the difference operator $\Delta(x, y) = 1$ if $x = y$, and 0 otherwise. Now, noting the definition of $err_{\mathcal{D}}$, we see that for any a' we have:

$$\begin{aligned} err_{\mathcal{D}}(f_{a'}) &= \int_{\mathbb{R}} p_{\mathcal{D}}(x) \left(\Delta(f_{a'}(x), c^*(x))\right) \mathrm{d}x = \int_{\mathbb{R}} p_{\mathcal{D}}(x) \left(\Delta(f_{a'}(x), f_a(x))\right) \mathrm{d}x \\ &= \begin{cases} \int_{a'}^{a} p_{\mathcal{D}}(x)\,\mathrm{d}x = \Pr\left(X \in (a', a)\right), & \text{if } a' \leq a, \\ \int_{a}^{a'} p_{\mathcal{D}}(x)\,\mathrm{d}x = \Pr\left(X \in (a, a')\right), & \text{if } a' > a, \end{cases} \end{aligned}$$

where the last equalities are derived from the definition of the PDF (here, X is a $\mathcal{D}$-distributed random variable). Thus, let us define values $c, d \in \mathbb{R}$, $c \leq a \leq d$, s.t.:

$$c \text{ is the smallest value satisfying } \int_c^a p_{\mathcal{D}}(x)\,\mathrm{d}x = \frac{\epsilon}{2}, \text{ or, if } \int_{-\infty}^a p_{\mathcal{D}}(x)\,\mathrm{d}x \leq \frac{\epsilon}{2}, \text{ let } c = -\infty,$$

$$d \text{ is the largest value satisfying } \int_a^d p_{\mathcal{D}}(x)\,\mathrm{d}x = \frac{\epsilon}{2}, \text{ or, if } \int_a^{\infty} p_{\mathcal{D}}(x)\,\mathrm{d}x \leq \frac{\epsilon}{2}, \text{ let } d = \infty.$$

Note then that the interval (c, d) is a subset of G. Next, note that $T(S)$ is the interval between the largest point in S that is $\leq a$ (or $-\infty$ if there are none) and the smallest point in S that is $\geq a$ (or ∞ if there are none), as any f'_a with a' outside this interval would incorrectly classify at least one point in S. There are three cases to consider based on whether c, d are finite:

Case 1: c, d are both finite. Thus, $\int_c^a p_{\mathcal{D}}(x)\,\mathrm{d}x = \int_a^d p_{\mathcal{D}}(x)\,\mathrm{d}x = \epsilon/2$. Note that if at least one of the points in S lies in (c, a), and at least one lies in (a, d), then clearly $T(S) \subset (c, d) \implies T(S) \subset G$. We now compute the probability that S satisfies this. Let E_c be the event that at least one point in S lies in (c, a), and E_d be the event that at least one in S lies in (a, d). Hence, we want to compute $\Pr(E_c \cap E_d)$. Using the union bound on the complement, we see this is:

$$\begin{aligned} \Pr(E_c \cap E_d) &= 1 - \Pr\left(\overline{E_c} \cup \overline{E_d}\right) \geq 1 - \left(\Pr(\overline{E_c}) + \Pr(\overline{E_d})\right) \\ &= 1 - \left(\prod_{x \in S} \Pr(x \notin (c, a))\right) - \left(\prod_{x \in S} \Pr(x \notin (a, d))\right) \\ &= 1 - 2(1-\epsilon/2)^{|S|} \geq 1 - 2e^{-|S|\epsilon/2}, \end{aligned}$$

where the last inequality is derived from $(1 + x) \leq e^x$. Note then that for $|S| = 2(\ln(2/\delta))/\epsilon$, the above inequality gives $\Pr(E_c \cap E_d) \geq 1 - \delta$. Moreover, since $E_c \cap E_d \implies T(S) \subset G$, we thus have $\Pr(T(S) \subset G) \geq \Pr(E_c \cap E_d) \geq 1 - \delta$.

Case 2: one of c, d is finite. For c finite, we have $\int_c^{\infty} p_{\mathcal{D}}(x)\,\mathrm{d}x \leq \epsilon$. Observe that for any $a' \in (c, \infty)$, $err_{\mathcal{D}}(f_{a'}) \leq \int_c^{\infty} p_{\mathcal{D}}(x)\,\mathrm{d}x \leq \epsilon \implies (c, \infty) \subset G$. Thus, if at least one point in S lies in (c, a), then $T(S) \subset (c, \infty) \implies T(S) \subset G$. This has probability:

$$\Pr(E_c) = 1 - \Pr\left(\overline{E_c}\right) = 1 - \left(\prod_{x \in S} \Pr(x \notin (c, a))\right) = 1 - (1-\epsilon/2)^{|S|} \geq 1 - e^{-|S|\epsilon/2}.$$

For $|S| = 2(\ln(1/\delta))/\epsilon$, we have $\Pr(T(S) \subset G) \geq \Pr(E_c) \geq 1 - \delta$. Similarly, for d finite, we can use the same logic to show that for the same value of $|S|$, we have $\Pr(T(S) \subset G) \geq \Pr(E_d) \geq 1 - \delta$.

Case 3: c, d are both infinite. In this case, note that $T(S) \subset G$ is trivially satisfied, as we have $\int_{-\infty}^{a} p_{\mathcal{D}}(x)\,\mathrm{d}x + \int_{a}^{\infty} p_{\mathcal{D}}(x)\,\mathrm{d}x \leq \epsilon \implies \epsilon \geq 1 \implies G = \mathbb{R}$.

In all three cases, note that $|S| = 2(\ln(2/\delta))/\epsilon = 2(\ln(1/\delta) + \ln(2))/\epsilon$, which is $\mathcal{O}(\log(1/\delta)/\epsilon)$, ensures that $T(S) \subset G$ with probability $\geq 1 - \delta$, and thus completes the proof.

Problem 5.3

The Perceptron algorithm does not always make the same number of mistakes. For simplicity, suppose our feature space is $\mathbb{R}^2$, and our training set is $S = \{s_i\}_{i=0}^{2}$ with $s_0 = \left(\frac{1}{\sqrt{2}}\left[1,1\right]^T, +\right)$, $s_1 = \left(\left[0,-1\right]^T, -\right)$, $s_2 = \left(\left[1,0\right]^T, +\right)$, where each s_i is a (feature, label) tuple. Also note that S is indeed separable by a hyperplane. Denote the weight at iteration k of the algorithm as $w^{(k)}$, and $w^{(0)} = \left[0,0\right]^T$ as per algorithm initialization. If the examples are fed in order s_0, s_1, s_2, we see that $w^{(1)} = w^{(2)} = w^{(3)} = \frac{1}{\sqrt{2}}\left[1,1\right]^T$, with 1 mistake. Now consider the case where the examples are fed in order s_1, s_2, s_0. In this case, we have $w^{(1)} = \left[0,1\right]^T, w^{(2)} = w^{(3)} = \left[1,1\right]^T$, with 2 mistakes. Thus we have shown that the number of mistakes is not invariant under example order.

Problem 5.6

(1) Let N be the dimension of our feature space (i.e. x, $w \in \mathbb{R}^N$). Associate with each example's label a value $y \in \{1, -1\}$ that is 1 for positive examples and -1 for negative examples. Recall that the hinge loss for predictor $f_w(x) = w^T x = \langle w, x\rangle$ is $L(y, f_w(x)) = \max(0, 1 - y f_w(x))$. Also recall the SGD update formula $w \leftarrow w - \lambda_t \nabla_w L$. For $y f_w(x) < 1$ we have:

$$\nabla_w L = \begin{bmatrix} \partial L/\partial w_0 \\ \partial L/\partial w_1 \\ \vdots \\ \partial L/\partial w_{N-1} \end{bmatrix} = \begin{bmatrix} -yx_0 \\ -yx_1 \\ \vdots \\ -yx_{N-1} \end{bmatrix} = -yx.$$

Otherwise, (for $y f_w(x) \geq 1$), $\nabla_w L = 0$. Note that in this latter case w does not change, which corresponds exactly to our modified Perceptron algorithm's behavior: it does not update w when $y f_w(x) \geq 1$. In the former case, we see that with $\lambda_t = 1$, the SGD update formula simplifies to $w \leftarrow w + yx$, which exactly coincides with the Perceptron algorithm's update formula. Thus, as SGD with a learning rate of $\lambda_t = 1$ behaves the same as our modified Perceptron algorithm in both cases, the two algorithms are equivalent.

(2) Let w^* be the weight vector that separates our examples by margin $\gamma = 1/|w^*|$, $w^{(k)}$ be the weight vector at iteration k of our algorithm, and $x^{(k)}$ be the example we process on the same iteration. Suppose on some iteration k, the algorithm updates the weight vector. Then, note that:

$$\langle w^*, w^{(k)}\rangle = \langle w^*, w^{(k-1)} + yx^{(k)}\rangle = \langle w^*, w^{(k-1)}\rangle + \langle w^*, yx^{(k)}\rangle \geq \langle w^*, w^{(k-1)}\rangle + 1,$$

where the inequality is derived by noting that if the data is separable by a hyperplane, then $\langle w^*, yx\rangle \geq 1$ by construction of w^*. Hence if we updated the weight vector U times during the algorithm to obtain the final weight vector w, we see that w satisfies $\langle w^*, w\rangle \geq \langle w^*, w^{(0)}\rangle + U$. By algorithm initialization of $w^{(0)} = \left[0\right]$, the previous inequality simplifies to $\langle w^*, w\rangle \geq U$. Next, note that:

$$\begin{aligned} \left|w^{(k)}\right|^2 = \langle w^{(k)}, w^{(k)}\rangle = \langle w^{(k-1)} + yx^{(k)}, w^{(k-1)} + yx^{(k)}\rangle &= \left|w^{(k-1)}\right|^2 + 2\langle w^{(k-1)}, yx^{(k)}\rangle + \left|yx^{(k)}\right|^2 \\ &= \left|w^{(k-1)}\right|^2 + 2y f_{w^{(k-1)}}\left(x^{(k)}\right) + y^2\left|x^{(k)}\right|^2 < \left|w^{(k-1)}\right|^2 + 2 + 1 = \left|w^{(k-1)}\right|^2 + 3, \end{aligned}$$

where the inequality is derived using the fact that w is only updated in the algorithm if $y f_w(x) < 1$. Hence, after running the algorithm, the final weight vector w also satisfies $|w|^2 < \left|w^{(0)}\right|^2 + 3U = 3U$. Since $\langle w^*, w\rangle/|w^*| \leq |w|$, we require that $U/|w^*| \leq \sqrt{3U} \implies U^2 \leq 3U|w^*|^2 \implies U \leq 3|w^*|^2 = 3/\gamma^2$.

(3) Since we update w at most $3/\gamma^2$ times, with $|w|^2$ increasing by at most 3 with each update, our final weight satisfies $|w|^2 \leq 3 \cdot 3/\gamma^2 = 9/\gamma^2 \implies 1/|w|^2 \geq \gamma^2/9 \implies 1/|w| \geq \gamma/3$. Hence w has margin of separation of at least $\gamma/3$.

Problem 5.7

We first note that a decision tree is a full binary tree, and thus $L(T) = I(T) + 1$, where $I(T)$ is the number of interior nodes of T, and $L(T)$ the number of leaves. Next, note that to minimize the regularized objective function:

$$err_S(h) + \sqrt{\frac{\text{size}(h)\ln(4) + \ln(2/\delta)}{2|S|}},$$

it suffices to minimize $err_S(h)$ over pruned trees h with a fixed number of leaves $\ell \in \{1, 2, \ldots, L(T)\}$ (as the value under the radical remains fixed for fixed ℓ), and then pick the minimum among these.

To accomplish this, we first copy over the interior nodes of T to a new tree T', whose edges have weights defined as follows: for every non-root node v with direct parent u, assign the edge (u, v) with weight $w(u, v) = \min(n_+(v), n_-(v)) - n_e(v)$, where $n_+(v)$ and $n_-(v)$ are the number of positive and negative samples from S that get fed to v from its ancestors, and $n_e(v)$ is the number of these samples that v would classify incorrectly if it were a decision stump (i.e. if its children were leaves). Moreover, note that $\min(n_+(v), n_-(v))$ is the number of samples that would be incorrectly classified by a leaf at v. Thus, the weight $w(u, v)$ can be interpreted as the number of additional samples from S that are correctly classified if we keep v in our pruned tree (as opposed to turning it into a leaf). Clearly $n_e(v) \leq \min(n_+(v), n_-(v))$ (since a decision stump at worst performs the same as a leaf) and thus all weights are non-negative. This step can be easily done in $\mathcal{O}(|S|I(T))$ time.

Note that minimizing $err_S(h)$ for a fixed number of leaves ℓ is equivalent to finding the maximum-weight rooted sub-tree T'_k in T' with size $k = \ell - 1$ (we define size as the number of vertices), and then adding leaves to T'_k to obtain h. We solve this separately for each $k \in \{0, 1, \ldots, L(T) - 1\}$ with dynamic programming:

At each node v, store a list of k 3-tuples (l_i, r_i, w_i), $i \in \{1, \ldots, k\}$ which will be used to keep statistics for the maximum-weight subtrees U_i of size i rooted at v (or of smaller size, if v has $< i - 1$ descendants). Here, l_i and r_i track the number of nodes in the left and right subtrees of v present in U_i (hence $l_i + r_i + 1 \leq i$), and w_i is the total weight of U_i. Initially, we fill the tuples of all leaf nodes of T' with zeros, and also initialize the first tuple in all nodes with zeros (since a subtree of size 1 rooted at a vertex is just that vertex). We propagate up our tree recursively as follows: for a parent u whose child nodes v_L and v_R have been filled, for each $i \in \{2, 3, \ldots, k\}$, we find the statistics for the maximal-weight subtree U_i of size i rooted at u, and update the tuples in u to these values. More specifically, for each i we find the values l, r that maximize $F(l, r) = p_l + q_r + b_l w(u, v_L) + b_r w(u, v_R)$ subject to $l + r + 1 = i$. Here, p are the weights from the tuples stored in v_L (i.e. $p_i = w_i$ from v_L), q the weights from the tuples stored in v_R (i.e. $q_i = w_i$ from v_R), and b_l, b_r are defined as 1 if $l, r > 0$ and 0 otherwise. Thus, $F(l, r)$ represents the weight of the subtree constructed by combining the max-weight subtree of size l from v_L and the max-weight subtree of size r from v_R. Once the optimal l, r are found, we update the i-th tuple of u with $l_i = l$, $r_i = r$, $w_i = F(l, r)$. The algorithm continues until the root node is processed, thereby filling in the tuple list at every node. To recover T'_k, we inspect the values l_k, r_k at the root node, assign these respectively to i, j, and then recursively inspect l_i, r_i at the left child if $i > 0$, and l_j, r_j at the right child if $j > 0$. T'_k is then the subtree consisting of the inspected nodes. Finally, we obtain h from T'_k via a tree traversal that adds leaves to T'_k s.t. all original nodes in T'_k have 2 children.

We now analyze the complexity of the above dynamic programming algorithm. First, note that maximizing $F(l, r)$ with constraint $l + r + 1 = i$ (and $l \geq 0, r \geq 0 \in \mathbb{Z}$) can be done in $\mathcal{O}(i)$ time by iterating over the possible pairs $(l, r) \in \{(0, i-1), (1, i-2), (2, i-3), \ldots, (i-1, 0)\}$. Hence, maximizing this for each i takes $\sum_{i=1}^{k} \mathcal{O}(i) = \mathcal{O}(k^2)$ time. As we do this for each node in T', which are the internal nodes in T, computing T'_k takes $\mathcal{O}(I(T)k^2)$ time. Next, to find the optimal pruning, we need to evaluate $err_S(h)$ for h constructed from T'_k, $k \in \{0, 1, \ldots, L(T) - 1\}$, and thus requires $\sum_{k=0}^{L(T)-1} \mathcal{O}(I(T)k^2) = \mathcal{O}(I(T)L(T)^3) = \mathcal{O}(L(T)^4)$ time, which is polynomial in $L(T)$.

Problem 5.8

Part 1

We associate a pruning h of T with a subset of T's sleeping experts as follows: for each leaf of h, take the sleeping expert at the corresponding node in T with the same label (i.e., if the leaf is labeled positive, take the sleeping expert which has positive label at the corresponding node in T). We see that this subset has the property that exactly one expert makes a prediction for each sample. This is because all experts in the subset correspond to nodes which are leaves in a particular pruning of T. As leaves in a pruning of T, exactly one of them will be hit in any given sample's path through T. Then the exact one that a sample hits, will be the sleeping expert that gives a prediction.

Part 2

Let a_i be a leaf node of h_L, y_i be the label of a_i, and S_i be the subset of examples that arrive at a_i. We know by definition that the S_i's are disjoint and the union of them is S.

Let m_i denote the sleeping expert associated with a_i that predicts y_i; in other words, m_i is the sleeping expert that predicts y_i on S_i. There are L such m_i's; the collection of them corresponds to h_L as shown in Part 1. Let M denote the collection.

Run the Combining Sleeping Experts Algorithm A with all of the $\mathcal{O}(L(T))$ sleeping experts. Define $mistakes(A, S_i)$ as the number of mistakes of A on S_i. By Theorem 5.22 in the book, we know the following inequality holds for every m_i (since by the theorem, this inequality holds for every sleeping expert):

$$\mathrm{E}\left(mistakes(A, S_i)\right) \leq (1+\epsilon) \cdot mistakes(m_i, S_i) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon}\right).$$

Summing up the L inequalities yields:

$$\sum_{i=1}^{L} \mathrm{E}\left(mistakes(A, S_i)\right) \leq (1+\epsilon) \cdot \sum_{i=1}^{L} mistakes(m_i, S_i) + L \cdot \mathcal{O}\left(\frac{\log(L(T))}{\epsilon}\right).$$

Since the S_i's are disjoint and the union of them is S, we have:

$$\sum_{i=1}^{L} mistakes(A, S_i) = mistakes(A, S),$$
$$\sum_{i=1}^{L} mistakes(m_i, S_i) = mistakes(M, S),$$

where $mistakes(A, S)$ is the total number of mistakes of A on S, and $mistakes(M, S)$ is the total number of mistakes of the m_i's on S. Note that $mistakes(M, S)$ is also the number of mistakes of h_L on S because h_L is equivalent to M.

So, using linearity of expectation, the above sum of inequalities becomes:

$$\mathrm{E}\left(mistakes(A, S)\right) \leq (1+\epsilon) \cdot mistakes(M, S) + L \cdot \mathcal{O}\left(\frac{\log(L(T))}{\epsilon}\right).$$

It follows that

$$\begin{aligned}
\mathrm{E}\left(err_S(A)\right) &= \frac{\mathrm{E}\left(mistakes(A, S)\right)}{|S|} \\
&\leq (1+\epsilon) \cdot \frac{mistakes(M, S)}{|S|} + \frac{L}{|S|} \cdot \mathcal{O}\left(\frac{\log(L(T))}{\epsilon}\right) \\
&= (1+\epsilon) \cdot err_S(h_L) + \mathcal{O}\left(\frac{L \log(L(T))}{\epsilon \cdot |S|}\right).
\end{aligned}$$

Note that $\epsilon^2 = \frac{L\log(L(T))}{|S|}$. Plug it into the above inequality:

$$\begin{aligned}
\mathrm{E}\left(err_S(A)\right) &\leq (1+\epsilon)\cdot err_S(h_L) + \mathcal{O}\left(\frac{\epsilon^2}{\epsilon}\right) \\
&= err_S(h_L) + \epsilon \cdot err_S(h_L) + \mathcal{O}(\epsilon) \\
&\leq err_S(h_L) + \epsilon + \mathcal{O}(\epsilon) \\
&= err_S(h_L) + \mathcal{O}(\epsilon) \\
&= err_S(h_L) + \mathcal{O}\left(\sqrt{\frac{L\log(L(T))}{|S|}}\right),
\end{aligned}$$

where the second inequality is derived from $err_S(h_L) \leq 1$.

This completes the proof.

Part 3

A copy of the algorithm discussed in Part 2 can itself be considered a sleeping expert (one that predicts all the time). After instantiating $L(T)$ copies of the algorithm which we denote as A_L, $L \in \{1, 2, \ldots, L(T)\}$, we can use them together with the Combining Sleeping Experts Algorithm, which we call A. By Theorem 5.22 we know:

$$\forall\, L \in \{1, 2 \cdots L(T)\},\ \forall\, S$$
$$\mathrm{E}\left(mistakes(A,S)\right) \leq (1+\epsilon)\cdot mistakes(A_L,S) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon}\right).$$

Dividing both sides by $|S|$ yields:

$$\forall\, L \in \{1, 2 \cdots L(T)\},\ \forall\, S$$
$$\mathrm{E}\left(err_S(A)\right) \leq (1+\epsilon)\cdot err_S(A_L) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon\cdot|S|}\right).$$

In Part 2 we have proved that the expectation of $err_S(A_L)$ is bounded by $err_S(h_L) + \mathcal{O}(\sqrt{L\log(L(T))/|S|})$. We will now show that we can replace $err_S(A_L)$ with its expectation in the above inequality.

Assume S is randomly selected from a distribution $\mathcal{D}$ and let $\mathcal{D}(S)$ denote the probability of selecting it. Sum up the above inequalities over $\mathcal{D}$:

$$\sum_S \left(\mathcal{D}(S)\cdot \mathrm{E}\left(err_S(A)\right)\right) \leq (1+\epsilon)\cdot \sum_S \left(\mathcal{D}(S)\cdot err_S(A_L)\right) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon\cdot|S|}\right)$$

We know the following:

$$\sum_S \left(\mathcal{D}(S)\cdot \mathrm{E}\left(err_S(A)\right)\right) = \mathrm{E}\left(err_S(A)\right) \qquad \text{because } \mathrm{E}\left(err_S(A)\right) \text{ is a constant}$$
$$\sum_S \left(\mathcal{D}(S)\cdot err_S(A_L)\right) = \mathrm{E}\left(err_S(A_L)\right) \qquad \text{by definition}$$

Therefore, the inequality becomes:

$$\begin{aligned}
\mathrm{E}\left(err_S(A)\right) &\leq (1+\epsilon)\cdot \mathrm{E}\left(err_S(A_L)\right) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon\cdot|S|}\right) \\
&\leq (1+\epsilon)\cdot\left(err_S(h_L) + \mathcal{O}\left(\sqrt{\frac{L\log(L(T))}{|S|}}\right)\right) + \mathcal{O}\left(\frac{\log(L(T))}{\epsilon\cdot|S|}\right).
\end{aligned}$$

Plugging in $\epsilon = \sqrt{\frac{L\log(L(T))}{|S|}}$ yields:

$$\begin{aligned}
\mathrm{E}\left(err_S(A)\right) &\leq (1+\epsilon)\cdot\left(err_S(h_L)+\mathcal{O}(\epsilon)\right)+\mathcal{O}\left(\frac{\epsilon}{L}\right)\\
&= err_S(h_L)+\mathcal{O}(\epsilon)+\epsilon\cdot err_S(h_L)+\mathcal{O}(\epsilon^2)+\frac{\mathcal{O}(\epsilon)}{L}\\
&= err_S(h_L)+\mathcal{O}(\epsilon).
\end{aligned}$$

The last equality follows because $err_S(h_L) \leq 1$, $L \geq 1$, and $\epsilon \in (0,1)$.

Note that the above bound holds for all $L \in \{1, 2 \cdots L(T)\}$, i.e., there are $L(T)$ such bounds. We can pick the strictest one and get:

$$\begin{aligned}
\mathrm{E}\left(err_S(A)\right) &\leq \min_L \left(err_S(h_L)+\mathcal{O}(\epsilon)\right)\\
&= \min_L \left(err_S(h_L)+\mathcal{O}\left(\sqrt{\frac{L\log(L(T))}{|S|}}\right)\right).
\end{aligned}$$

This completes the proof.