Theory and Algorithm for Machine Learning 2020
National University of Singapore CS5339
Prof Lee Wee Sun

Homework 1 (Due date Sunday 8 March 11.59pm)

Please write the following on your homework:

- Name
- Collaborators (write none if no collaborators)
- Source, if you obtained the solution through research, e.g. through the web.

While you may collaborate, you *must write up the solution yourself*. While it is okay for the solution ideas to come from discussion, it is considered as plagiarism if the solution write-up is highly similar to your collaborator's write-up or to other sources.
You solution should be submitted to IVLE workbin. Scanned handwritten solutions are acceptable but must be legible.
Late Policy: A late penalty of 20% per day will be imposed (no submission accepted after 5 late days) unless prior permission is obtained.

---

1. **VC-dimension of Decision Trees**
   Consider using decision trees/forests to implement Boolean functions, i.e. classifiers from $\{0,1\}^d$ to $\{0,1\}$.

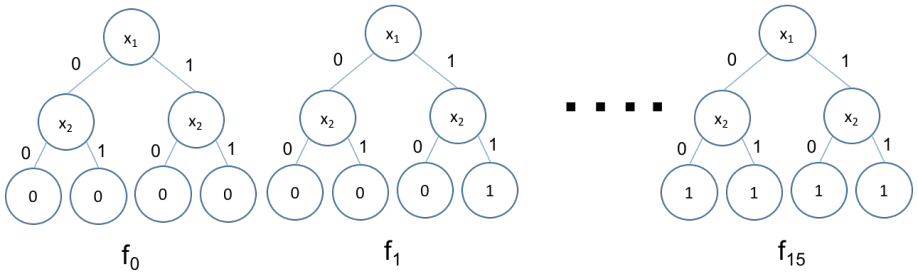
   (a) Argue that decision trees of height d has VC-dimension at least 2^d.

   Solution: Let the d variables be $x_1, \ldots, x_d$. To show that the VC-dimension is at least 2^d, it suffices to find a set of 2^d input points that is shattered by decision trees of depth d. Selecting all possible binary values for the d variables $(x_1, \ldots, x_d)$ will give 2^d points that can be shattered.

   To show that the set of points is shattered, we need to argue that for any possible labeling $f(x_1, \ldots, x_d)$ of the 2^d points, we can construct a decision tree that provides labeling. To do that, it suffices to construct a perfect binary tree where all the nodes at level i is split using variable x_i (in a perfect binary tree, all internal nodes have two children and all leaves are at the same depth). This tree has 2^d children, where each child corresponds to a particular assignment to the values of the variables $(x_1, \ldots, x_d)$. To achieve the labeling $f(x_1, \ldots, x_d)$, we simply assign the leaf corresponding to any $(x_1, \ldots, x_d)$ with the value $f(x_1, \ldots, x_d)$.

   For example, with $d = 2$, we have 4 possible input points $(0,0), (0,1), (1,0)$ and $(1,1)$. With 4 inputs points we have $2^4 = 16$ possible binary functions as shown below:

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1



(b) Give a good upper bound for the VC-dimension of binary decision trees with n nodes that maps from $\{0,1\}^d$ to $\{0,1\}$.
(Hint: Consider how many bits are used to encode such decision trees in Section 18.1 in SSBD.)

Solution: In Section 18.1 in SSBD, it is shown that we can encode all possible trees that have d nodes using $(n+1)\log_2(d+3)$ bits. This can be done by encoding the description of n nodes in a depth first scan of the tree followed by the encoding of the terminal symbol. The $d+3$ possible symbols for the description of each node are the d possible variables plus labels 0, 1, and the terminal symbol. Hence each node can be encoded using $\log_2(d+3)$ bits followed by encoding of the terminal symbol. For this question, we know the number of nodes, so the terminal symbol is not required. Hence, using $b = n\log_2(d+2)$ bits is sufficient, where we only need to encode $d+2$ possible symbols and do not need the additional terminal symbol at the end. The number of possible trees that can be described using b bits is bounded by 2^b. To bound the VC-dim, we find the largest m such that $2^m \leq 2^b$, giving $m \leq b = n\log_2(d+2)$.

(c) Give a good upper bound for the VC-dimension of a random forest with k trees where each tree has n leaves. Here we consider a random forest as a thresholded weighted average of the k decision trees.

Solution: We have k trees for the forest. First we consider the case where the k trees are fixed functions that are not learned. Treating the outputs of k trees as a k features $f_1(x), \ldots, f_k(x)$, we get a vector-valued mapping $x \mapsto (f_1(x), \ldots, f_k(x))$ of x to a feature vector of length k. This feature vector is then used as input to a linear threshold function. On m input points, the number of possible functions constructed from the linear threshold output unit with those k features is no more than $(em/(k+1))^{(k+1)}$ by Sauer's lemma as the VC-dimension of a linear threshold function in k dimension is $k+1$.

Let $b = n\log_2(d+2)$. We have previously shown that the number of distinct functions

that can be constructed by a single tree is no more than 2^b. As we have k trees that are used to construct the feature vector, there are no more than 2^{bk} possible mappings $x \mapsto (f_1(x), \ldots, f_k(x))$ of x to a feature vector of length k that can be constructed using the k trees. As just argued, for each fixed mapping, the number of possible functions constructed by the linear threshold function output is no more than $(em/(k+1))^{(k+1)}$. When all possible mappings can be used, we can upper bound the number of possible functions construct from the composition of the feature mapping and the linear threshold function by $2^{bk}(em/(k+1))^{k+1)}$. To bound the VC-dim, we find the largest value m such that $2^m \leq 2^{bk}(em/(k+1))^{(k+1)}$. Taking logs, we get $m \leq bk + (k+1)\log_2(em/(k+1))$. Substituting $b = n\log_2(d+2)$ and applying SSBD Lemma A.2, we get an upper bound of $O(kn\log d + k\log k)$ for the VC dimension.

2. **Kernels**
(Modified from SSBD 16.6.4) Let N be any positive integer. For every $x, x' \in \{1, \ldots, N\}$ define

$$K(x, x') = \min\{x, x'\}.$$

(a) Prove that K is a valid kernel; namely, find a mapping $\psi : \{1, \ldots, N\} \to H$ where H is some Hilbert space, such that

$$\forall x, x' \in \{1, \ldots, N\}, K(x, x') = \langle \psi(x), \psi(x') \rangle.$$

In this case, let H be a feature vector of length N.
(Hint: Consider using binary vectors (where each coefficient is either 0 or 1) of length N as the output of $\psi : \{1, \ldots, N\}$. What happens when you do inner product of two such vectors?)
Solution: Define $\psi : \{1, \ldots, N\} \to \mathbb{R}^N$ by

$$\psi(\mathbf{j}) = (\mathbf{1^j}; \mathbf{0^{N-j}}),$$

where $\mathbf{1^j}$ is the vector in $\mathbb{R}^\mathbf{j}$ with all elements equals to 1, and $\mathbf{0^{N-j}}$ is the zero vector in $\mathbb{R}^{\mathbf{N-j}}$. Then, assuming the standard inner product, we obtain that $\forall (i, j) \in [N]^2$,

$$\langle \psi(i), \psi(j) \rangle = \langle (\mathbf{1^i}; \mathbf{0^{N-i}}), (\mathbf{1^j}; \mathbf{0^{N-j}}) \rangle = \min\{i, j\} = K(i, j)$$

(b) Using the representer theorem, optimal solutions of regularized loss minimization can be represented using

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \psi(x_i),$$

where $x_1, \ldots, x_m$ are the training inputs and $\alpha_1, \ldots, \alpha_m$ are the learned parameters. Alternatively, we can represent $\mathbf{w}$ as a vector of real numbers. Which representation is smaller when:

i. m is much smaller than N

ii. N is much smaller than m

Solution: From part (a), the dimension of $\mathbf{w}$ is N. At the same time the dimension of $(\alpha_1, \ldots, \alpha_m)$ is m. Hence representation as $(\alpha_1, \ldots, \alpha_m)$ is smaller for (i) while representation as a vector of weights $\mathbf{w} \in \mathbb{R}^N$ is smaller for (ii).

(c) Consider the case where $N = 3$ and assume that the function is observed at points 1 and 3 with $f(1) = 0$ and $f(3) = 1$. From the representer theorem, the optimal solution of regularised loss minimization can be represented as

$$f(x) = \alpha_1 K(1, x) + \alpha_2 K(3, x).$$

i. Argue that it is not possible to represent the function

$$\begin{bmatrix} f(1) \\ f(2) \\ f(3) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

using $\alpha_1 K(1, x) + \alpha_2 K(3, x)$.

ii. On the other hand, if we allow do not restrict the kernels to be parameterized by the input data, show that the same function can be represented with $\alpha_1 K(2, x) + \alpha_2 K(3, x)$ for some value of α_1, α_2.

Solution:

i. The function

$$\begin{bmatrix} f(1) \\ f(2) \\ f(3) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

is not in the span of

$$\alpha_1 \begin{bmatrix} K(1,1) \\ K(1,2) \\ K(1,3) \end{bmatrix} + \alpha_2 \begin{bmatrix} K(3,1) \\ K(3,2) \\ K(3,3) \end{bmatrix} = \alpha_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

For it to be in the span, we need α_1 and α_2 to be the solution of

$$\alpha_1 + \alpha_2 = 0 \tag{1}$$

$$\alpha_1 + 2\alpha_2 = 0 \tag{2}$$

$$\alpha_1 + 3\alpha_2 = 1 \tag{3}$$

Subtracting (1) from (2) we get $\alpha_2 = 0$. Subtracting (2) from (3) we get $\alpha_2 = 1$ which gives a contradiction.

ii. Set $\alpha_1 = -1$ and $\alpha_2 = 1$. Then we have

$$\alpha_1 \begin{bmatrix} K(2,1) \\ K(2,2) \\ K(2,3) \end{bmatrix} + \alpha_2 \begin{bmatrix} K(3,1) \\ K(3,2) \\ K(3,3) \end{bmatrix} = -1 \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

(d) We are interested in doing text classification. Assume that the maximum number of any particular word in each file is N and there are a total of W possible words. A reasonable similarity function for two text files is the number of co-occurring words that they have in common. To count the number of co-occuring words, we first count how many common occurrences there are for each word i, then sum the common occurrences of words for all $i = 1, \ldots, W$. For example, if file x_1 contains "the quick brown fox jumps over the lazy dog" and file x_2 contains "the lazy dog slept through the whole event the entire time" the co-occuring words are "the"(2 co-occurrences), "lazy" (1 co-occurence), and "dog" (1 co-occurrence) and the similarity $K(x_1, x_2) = 4$. Argue that this similarity function forms a valid kernel.

Solution: Let $c_i(x)$ counts the number of occurrences of word i in x. Let $K_i(x, x') = \min\{c_i(x), c_i(x')\}$. This is a valid kernel that counts the number of common occurrences of word i in x and x'. Then $K(x, x') = \sum_{i=1}^{W} K_i(x, x')$. Since the sum of valid kernels is also a valid kernel, $K(x, x')$ is a valid kernel.

3. **Learn to do Forex Trading**

You are a currency trader and have K currencies, labeled $0, \ldots, K-1$, to trade. You use a simple strategy where each day, you either move all your money from the current currency to another currency, or keep your money in the same currency by not trading. The exchange rate for trading from currency i to j on day d is r_{ij}^d, where $r_{ii}^d = 1$ as no trade is done. If you start with \$1 in your home currency, which we assume to be currency 0, on day 1 and do a sequence of D trades obtaining a sequence $c_1, c_2, \ldots, c_{D-1}, c_D = 0$ of currencies, where the last trade on day D must be back to your home currency, then the final amount in your home currency after the trades is $r_{0c_1}^1 \ldots r_{c_{D-1}0}^D$.

(a) Assume that you know the exchange rates for each of the D days into the future r_{ij}^d. Describe a dynamic programming algorithm for computing $V(j, d)$, the optimal amount that can be obtained from \$1 in currency j at the start start of the d-th day and trading optimally for the following $D - d$ trades with the final trade on day d being constrained to be to currency 0. (Hint: Construct the equation for $V(j, d-1)$ in terms of r_{ji}^{d-1} and $V(i, d)$.)

Solution: Initialize $V(j, D)$ to r_{j0}^D. We can then update $V(j, d-1)$ for $d = D-1$ to 2 by $V(j, d-1) = \max_i r_{ji}^{d-1} V(i, d)$. $V(j, 1) = \max_{i=0}^{K-1} r_{01}^1 V(i, 2)$ gives the largest final amount possible after D days starting from currency j.

(b) You do not know the future exchange rates r_{ij}^d. However, you would like to learn a function $f(\mathbf{x}, \theta)$ for predicting $V(0, 1)$ based on a set of features $\mathbf{x}$ that you have extracted (e.g. current exchange rates, news from the different countries, etc.), where the features are used to learn the future exchange rates $r_{ij}^d(\mathbf{x}, \theta)$ with parameters θ that is learned.

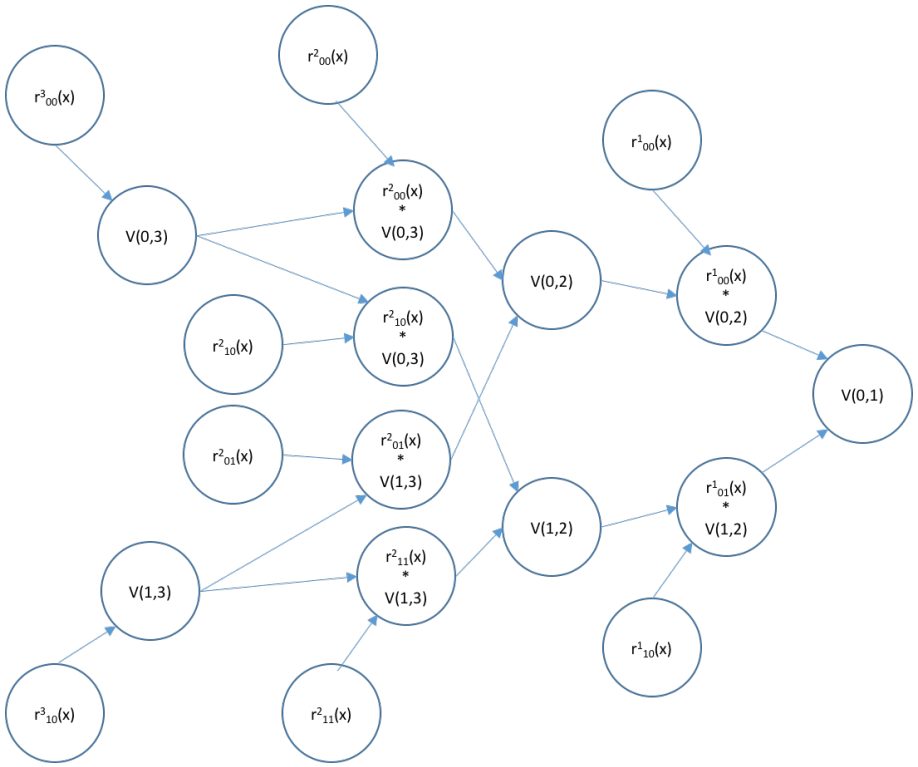
i. Describe how you can construct a deep neural network to simulate the dynamic programming computation and hence do end-to-end training for the function $f(\mathbf{x}, \theta)$

to predict $V(0,1)$ as a function of $\mathbf{x}$. Describe the functions used (e.g. max functions, product functions) and how they are connected together as a network.

ii. Assume that we would like to do regression with square loss to do the learning. Given observed exchange rates and feature vector $\mathbf{x}$ every day over a long period of time, describe how to construct the training examples $(\mathbf{x}_i, y_i)$.

Solution:

(i) The first layer consist of functions $V(j,d,\mathbf{x},\theta) = r^D_{j0}(\mathbf{x},\theta)$. The following d layers consist of the functions $V(j,d-1,\mathbf{x},\theta) = \max_{i=0}^{K-1} r^{d-1}_{ji}(\mathbf{x},\theta)V(i,d,\mathbf{x},\theta)$. Each function $V(j,d-1,\mathbf{x},\theta)$ is constructed using a max function whose arguments are K product functions $r^{d-1}_{ji}(\mathbf{x},\theta)V(i,d)$ for $i = 0,\ldots,K-1$. Finally the output is $f(\mathbf{x},\theta) = V(0,1,\mathbf{x},\theta) = \max_i r^1_{01}(\mathbf{x},\theta)V(i,2,\mathbf{x},\theta)$. The network for $K = 2$ and $D = 3$ is shown below.



(ii) For each day i in the historical dataset, we can gather the exchange rate for D days into the future and use the dynamic programming algorithm from part (a) to generate the target gain y_i from trading optimally for D days. This can be paired with the feature from day i, $\mathbf{x}_i$ to form the training data $(\mathbf{x}_i, y_i)$.

(c) What advantage does learning $f(\mathbf{x},\theta)$ as described have over doing supervised learning to learn $r^d_{ij}(\mathbf{x},\theta)$ as a function of $\mathbf{x}$, then using the dynamic programing algorithm in part (a) to compute the prediction for the optimal gain?

Solution: We are doing discriminative learning of the loss function that we are interested in. This is possibly more robust compared to learning $r_{ij}^d(\mathbf{x}, \theta)$ separately and then using it in the dynamic programming algorithm as we are aiming to get the best approximation for $f(\mathbf{x}, \theta)$ rather than assuming that we can learn the correct function for $r_{ij}^d(\mathbf{x}, \theta)$.