1. (20 points) When talking about non-uniform voting in aggregation, we mentioned that $\boldsymbol{\alpha}$ can be viewed as a weight vector learned from any linear algorithm coupled with the following transform:

$$\boldsymbol{\phi}(\mathbf{x}) = \Big(g_1(\mathbf{x}), g_2(\mathbf{x}), \cdots, g_T(\mathbf{x})\Big).$$

When studying kernel methods, we mentioned that the kernel is simply a computational short-cut for the inner product $(\boldsymbol{\phi}(\mathbf{x}))^T(\boldsymbol{\phi}(\mathbf{x}'))$. In this problem, we mix the two topics together using the decision stumps as our $g_t(\mathbf{x})$.

Assume that the input vectors contain only integers between (including) L and R, where $L < R$. Consider the decision stumps $g_{s,i,\theta}(\mathbf{x}) = s \cdot \text{sign}\Big(x_i - \theta\Big)$, where

$$\begin{array}{ll} i & \in \{1, 2, \cdots, d\}, \\ d & \text{ is the finite dimensionality of the input space,} \\ s & \in \{-1, +1\}, \\ \theta & \in \{\theta_1 = L + 0.5, \theta_2 = L + 1.5, \ldots, \theta_k = R - 0.5\}. \end{array}$$

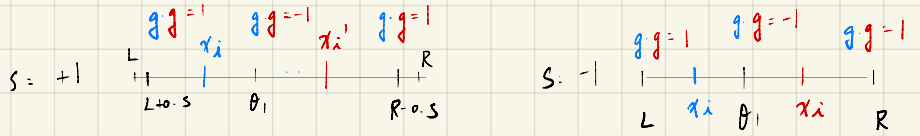
Define $\boldsymbol{\phi}_{ds}(\mathbf{x}) = \Big(g_{+1,1,\theta_1}(\mathbf{x}), g_{+1,1,\theta_2}(\mathbf{x}), \ldots, g_{+1,1,\theta_k}(\mathbf{x}), \ldots, g_{-1,d,\theta_k}(\mathbf{x})\Big)$ as a column vector.

What is $K_{ds}(\mathbf{x}, \mathbf{x}') = (\boldsymbol{\phi}_{ds}(\mathbf{x}))^T(\boldsymbol{\phi}_{ds}(\mathbf{x}'))$? Prove your answer.

(Hint: This result shows that aggregation learning with SVMs is possible. Those who are interested in knowing how perceptrons and decision trees can be used instead of decision stumps during kernel construction can read the following early work)

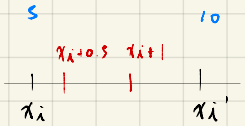
`https://www.csie.ntu.edu.tw/~htlin/paper/doc/infkernel.pdf` .

$\phi_{ds} = (g_{+1,1,\theta_1}(x) \cdot g_{+1,1,\theta_2}(x) \cdots g_{+1,2,\theta_1}(x) \cdots g_{-1,1,\theta_1}(x) \cdots g_{-1,d,\theta_k}(x))$



$s = +1$ $\quad K_{ds}(x, x') = g_{+1,1,\theta_1}(x) \cdot g_{+1,1,\theta_1}(x') + \cdots$

θ_n between x_i, x_i' : $g g' = -1$



$(R - 0.5) - (L + 0.5) + 1 = R - L$

$x_i + 0.5,\ x_i + 1 \cdots x_i' - 0.5$

$\text{total} = |(x_i' - 1) - x_i + 1|$

$= |x_i' - x_i|$

$$\sum_{i=0}^{d} \Big(\underbrace{(R-L) \times 1}_{+1} - \underbrace{2|x_i' - x_i|}_{-1} \Big) = d(R-L) - 2\sum_{i=0}^{d} |x_i' - x_i|$$

$s = -1$ $\quad K_{ds}(x, x') = g_{-1,1,\theta_1}(x) \cdot g_{-1,1,\theta_1}(x') + \cdots$

θ_n between x_i, x_i' : $g g' = -1$

$\Rightarrow$ same as $s = +1$

$$\Rightarrow K_{ds}(x, x') = 2d(R-L) - 4\sum_{i=0}^{d} |x_i' - x_i|$$

2. (20 points) For a given valid kernel K, consider a new kernel $\tilde{K}(\mathbf{x}, \mathbf{x}') = uK(\mathbf{x}, \mathbf{x}') + v$ for some $u > 0$. Note that v can be any real value. When $v \geq 0$, it is easy to prove that $\tilde{K}$ is still a valid kernel; when $v < 0$, however, $\tilde{K}$ may not always be a valid kernel. Prove that for the dual of soft-margin support vector machine, using $\tilde{K}$ along with a new $\tilde{C} = \frac{C}{u}$ instead of K with the original C leads to an equivalent g_{SVM} classifier. That is, the optimal $(\boldsymbol{\alpha}, b)$ obtained leads to the same decision boundary.

(Note: This result can be used to shift and scale the kernel function derived in the previous problem to a simpler form (even not as a valid kernel) when coupling it with the soft-margin SVM.)

$$\tilde{K} = uK(x, x') + v$$

$$\tilde{C} = \frac{C}{u}$$

$$b = y_s - \sum_{sv} \alpha_n y_n K(x_n, x_s) \qquad 0 < \alpha_n < C \qquad \sum_{n=1}^{N} \alpha_n y_n = 0$$

$$\tilde{K} = \min_{\alpha} \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \left(uK(x_m, x_n) + v\right) - \sum_{n=1}^{N} \alpha_n$$

subject to $0 \leq \alpha_n < \frac{C}{u}$ $\quad \alpha_n' = \frac{\alpha_n}{u}$

$$g_{SVM}(x) = \text{sign}\left(\sum_{sv} \frac{\alpha_n}{u} y_n \tilde{K}(x_n, x) + b \right)$$

$$= \text{sign}\left(\sum_{sv} \frac{\alpha_n}{u} y_n (u \cdot K(x_n, x) + v) + b \right)$$

$$= \text{sign}\left(\sum_{sv} \frac{\alpha_n}{u} y_n u K(x_n, x) + \sum_{sv} \frac{\alpha_n}{u} y_n v + y_s - \sum_{sv} \frac{\alpha_n}{u} y_n u K(x_n, x_s) - \sum_{sv} \frac{\alpha_n}{u} y_n v \right)$$

(the $\sum_{sv} \frac{\alpha_n}{u} y_n v$ term → 0; the $\sum_{sv} \frac{\alpha_n}{u} y_n v$ term → 0)

$$= \text{sign}\left(\sum_{sv} \alpha_n y_n K(x_n, x) + y_s - \sum_{sv} \alpha_n y_n K(x_n, x_s) \right)$$

$$= \text{sign}\left(\sum_{sv} \alpha_n y_n K(x_n, x) + b \right)$$

this lead to an equivelant g_{SVM}

3. (20 points) Consider an aggregated binary classifier G that is constructed by uniform blending on 17 binary classifiers $\{g_t\}_{t=1}^{17}$. That is,

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{17} g_t(\mathbf{x})\right)$$

Assume that each g_t is of test 0/1 error $E_{\text{out}}(g_t) = e_t > 0$. Define the total error $E = \sum_{t=1}^{17} e_t$. What is the largest value of $\frac{E_{\text{out}}(G)}{E}$? Prove your result.

(Note: The ratio roughly shows how G reduces the total error made by the hypotheses.)

$G(x) = \text{sign}\left(\sum_{t=1}^{17} g_t(x)\right)$. for $y = +1$ there must be more g_t that gives wrong prediction than g_t that predict correctly. in this case, to get the largest value, we have to have smallest $E = \sum_{t=1}^{17} e_t$. and $E_{out}(G) = 1$ so at least 9 g_t classify as -1 to make $E = \sum_{t=1}^{17} E_{out}(g_t) \geq 9$

the largest value of $\frac{E_{out}(G)}{E} = \frac{1}{9}$ #

4. (20 points) If bootstrapping is used to sample exactly $\frac{3}{4}N$ examples out of N, what is the probability that an example is *not* sampled when N is very large? List your derivation steps.

(Note: This is just an "easy" exercise of letting you think about the amount of OOB data in bagging/random forest.))

$$\lim_{N\to\infty}\left(1-\frac{1}{N}\right)^{\frac{3}{4}N} = \lim_{N\to\infty}\left(\frac{N-1}{N}\right)^{\frac{3}{4}N} = \lim_{N\to\infty}\frac{1}{\left(\frac{N}{N-1}\right)^{\frac{3}{4}N}} = \lim_{N\to\infty}\frac{1}{\left(1+\frac{1}{N-1}\right)^{\frac{3}{4}N}} = \frac{1}{e^{\frac{3}{4}}} \#$$

5. (20 points) Consider applying the AdaBoost algorithm on a binary classification data set where 98% of the examples are positive. Because there are so many positive examples, the base algorithm within AdaBoost returns a constant classifier $g_1(\mathbf{x}) = +1$ in the first iteration. Let $u_n^{(2)}$ be the individual example weight of each example in the second iteration. What is

$$\frac{\sum_{n:\, y_n>0} u_n^{(2)}}{\sum_{n:\, y_n<0} u_n^{(2)}}?$$

Prove your answer.

(Note: This is designed to help you understand how AdaBoost can deal with "imbalanced" data immediately after the first iteration.)

$y_n > 0$: predict all right ⇒ weighted incorrect rate 0.02

$y_n < 0$: predict all wrong ⇒ weighted correct rate 0.98

assume that $u_n^{(1)} = \frac{1}{N}$ for all examples

$$\frac{\sum_{n:\, y_n>0} u_n^{(2)}}{\sum_{n:\, y_n<0} u_n^{(2)}} = \frac{\sum_{n:\, y_n>0} u_n^{(1)} \times 0.02}{\sum_{n:\, y_n<0} u_n^{(1)} \times 0.98} = \frac{0.98\, u_n^{(1)} \times 0.02}{0.02\, u_n^{(1)} \times 0.98} = 1 \quad \#$$

6. (20 points) For the AdaBoost algorithm introduced in Lectures 208 and 211, let $U_t = \sum_{n=1}^{N} u_n^{(t)}$. That is, $U_1 = 1$ (you are very welcome! ;-)). Assume that $0 < \epsilon_t < \frac{1}{2}$ for each hypothesis g_t. Express $\frac{U_{T+1}}{U_1}$ in terms of $\epsilon_1, \epsilon_2, \ldots, \epsilon_T$. Prove your answer.

(Hint: Consider checking $\frac{U_{t+1}}{U_t}$ first. The result is the backbone of proving that AdaBoost will converge within $O(\log N)$ iterations.)

in iteration t. $\quad U_t \cdot \varepsilon_t = \sum u_n^{(t)} [\![y_n \neq g_t(x_n)]\!]$

$U_t(1-\varepsilon_t) = \sum u_n^{(t)} [\![y_n = g_t(x_n)]\!]$

$$U_{T+1} = U_t \cdot \varepsilon_t \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (U_t - U_t \cdot \varepsilon_t) \cdot \sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}}$$

$$= U_t \left(\varepsilon_t \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (1-\varepsilon_t) \sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} \right) = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}\, U_t$$

$$\frac{U_{t+1}}{U_t} = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$$

$$\Rightarrow \frac{U_{t+1}}{U_1} = 2^T \prod_{t=1}^{T} \sqrt{\varepsilon_t(1-\varepsilon_t)} \quad \#$$

7. (20 points) For the gradient boosted decision tree, after updating all s_n in iteration t using the steepest η as α_t, prove that $\sum_{n=1}^{N}(y_n - s_n)g_t(\mathbf{x}_n) = 0$.

(Note: This special value may tell us some important physical property on the relationship between the vectors $[y_1 - s_1, y_2 - s_2, \ldots, y_N - s_N]$ *and* $[g_t(\mathbf{x}_1), g_t(\mathbf{x}_2), \ldots, g_t(\mathbf{x}_N)]$.

$$\min_{\eta}\left(\frac{1}{N}\sum_{n=1}^{N}(y_n - s_n - \eta g_t(x_n))^2\right) \Rightarrow \frac{\partial err}{\partial \eta} = \frac{1}{N}\sum_{n=1}^{N}(2(y_n - s_n) - 2\eta g_t(x_n))\, g_t(x_n) = 0$$

$$\Rightarrow \eta = \frac{\sum_{n=1}^{N}(y_n - s_n)g_t(x_n)}{\sum_{n=1}^{N} g_t(x_n)^2}$$

$$s_n' = s_n + \eta g_t(x_n)$$

$$\sum_{n=1}^{N}(y_n - s_n - \eta g_t(x_n))\, g_t(x_n)$$

$$\Rightarrow \sum_{n=1}^{N} y_n g_t(x_n) - \sum_{n=1}^{N} s_n g_t(x_n) - \sum \eta g^2(x_n)$$

$$= \sum_{n=1}^{N}(y_n - s_n)g_t(x_n) - \sum_{n=1}^{N}(y_n - s_n)g_t(x_n) = 0$$

#

8. (20 points) For a Neural Network with at least one hidden layer and $\tanh(s)$ as the transformation functions on all neurons (including the output neuron), when all the initial weights $w_{ij}^{(\ell)}$ are set to 0, what gradient components are also 0? Prove your answer.

(Note: This result tells you that all-0 initialization may make it impossible for back-propagation to update some weights.)

we update $w_{ij}^{(l)}$ by $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta\, x_i^{(l-1)}\, \delta_j^{(l)}$

also, we compute δ_j by $\delta_j^{(l)} = \sum_{k=1} \delta_k^{(l+1)} (w_{jk}^{(l+1)}) (\tanh'(s_j^{(l)}))$

$\delta_j^{(l)} = 0$ if w_{ij}^{+1} are initially set to 0. we can't update $w_{ij}^{(l)}$

and all $\frac{\partial e_n}{\partial w_{ij}^{l}} = \delta_j^{(l)} (x^{l-1}) = 0$ #

9. (20 points, *) First, let's implement a simple C&RT algorithm without pruning using the squared error as the impurity measure, as introduced in the class. For the decision stump used in branching, if you are branching with feature i, please sort all the $x_{n,i}$ values to form (at most) $N+1$ segments of equivalent θ, and then pick θ within the median of the segment. If multiple (i, θ) produce the best split, pick the one with the smallest i (and if there is a tie again, pick the one with the smallest θ).

Please run the algorithm on the following set for training:

`http://www.csie.ntu.edu.tw/~htlin/course/ml23fall/hw6/hw6_train.dat`

and the following file as our test data set for evaluating E_{out}:

`http://www.csie.ntu.edu.tw/~htlin/course/ml23fall/hw6/hw6_test.dat`

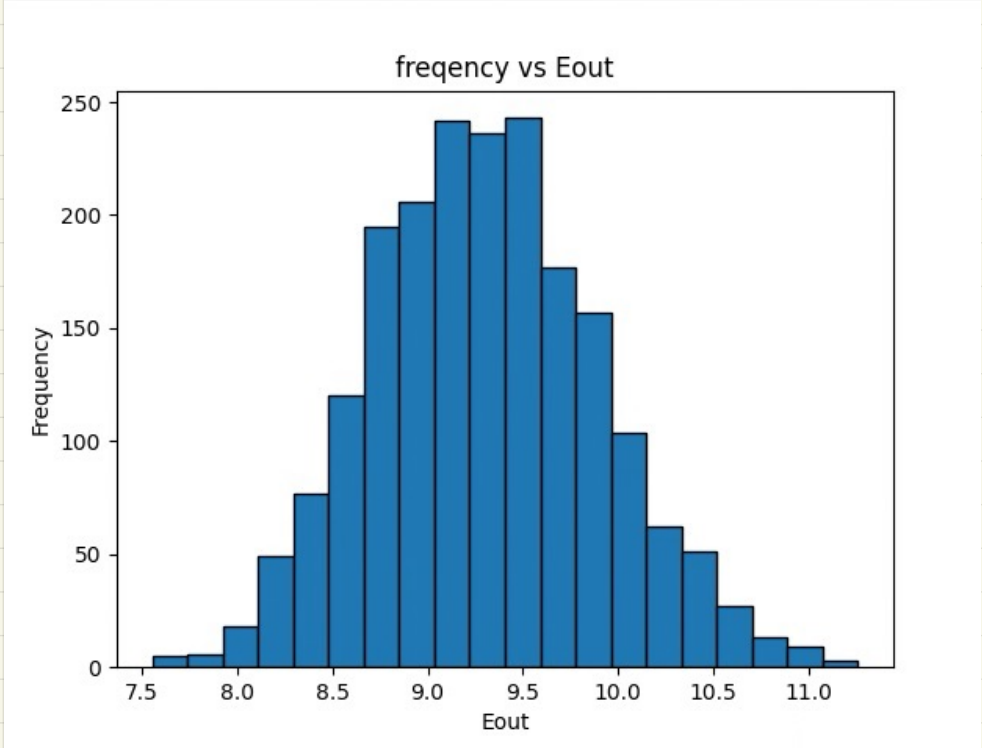
The datasets are in LIBSVM format and are processed from

`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/abalone`

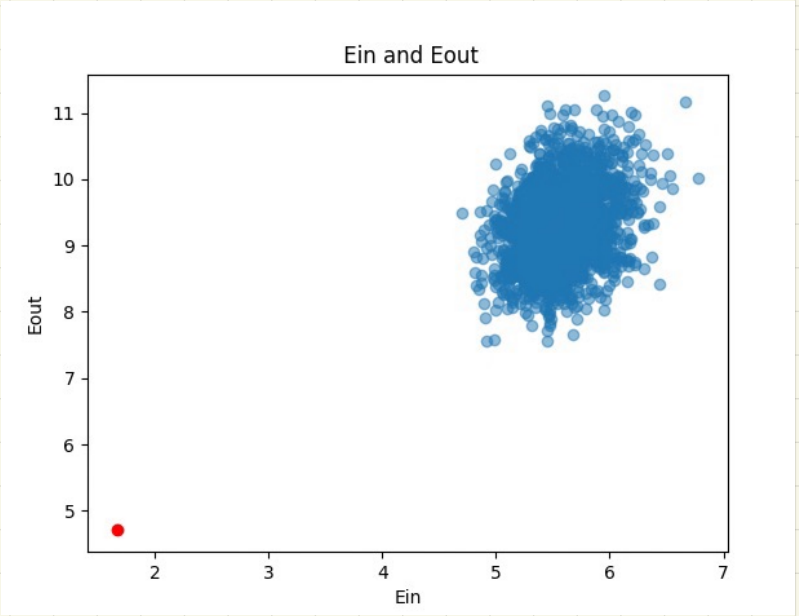
What is the $E_{\text{out}}(g)$, where g is the unpruned decision tree returned from your C&RT algorithm and E_{out} is evaluated using the squared error?

```
E_out(g): 8.793244626407 37
```

10. (20 points, *) Next, we implement the random forest algorithm by coupling bagging (by sampling with replacement) with $N' = 0.5N$ with your unpruned decision tree in the previous problem. You do not need to do random feature selection or expansion. Produce $T = 2000$ trees with bagging. Let $g_1, g_2, \ldots, g_{2000}$ denote the 2000 trees generated. Plot a histogram of $E_{\text{out}}(g_t)$, where E_{out} is evaluated using the squared error.

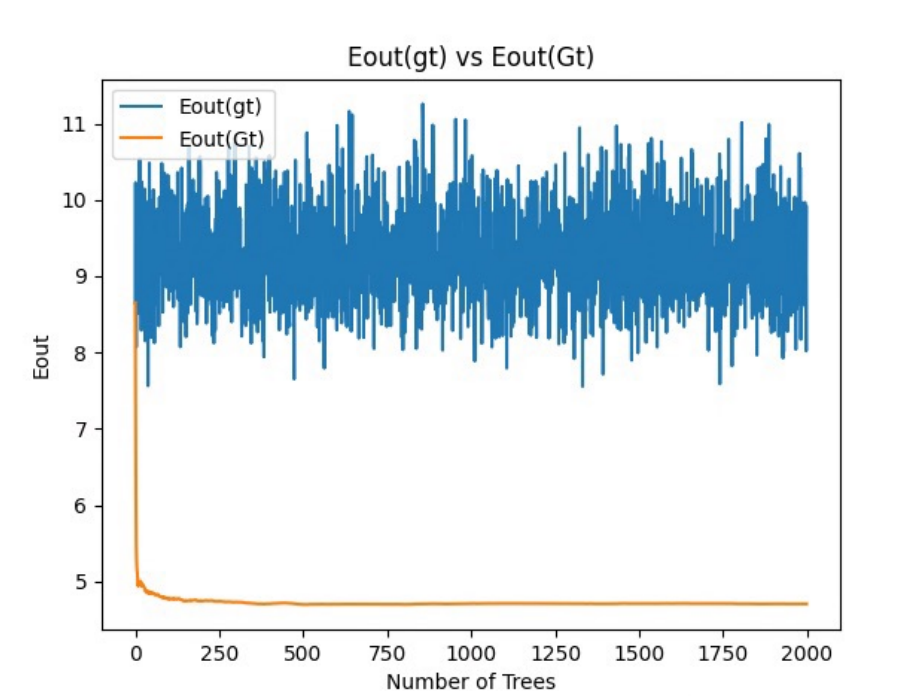


11. (20 points, *) Let $G(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{x})$ be the random forest formed by the trees above. Plot a scatter plot of $(E_{\text{in}}(g_t), E_{\text{out}}(g_t))$ along with a clear mark of where $(E_{\text{in}}(G), E_{\text{out}}(G))$ is. Describe your findings.



the Eout, Ein of G is much lower than Ein, Eout (g)
although many g peforms not that good, the G
can still perform well.

12. (20 points, *) Let $G_t(\mathbf{x}) = \frac{1}{t}\sum_{\tau=1}^{t} g_\tau(\mathbf{x})$ be the random forest formed by the first t trees. Plot the $E_{\text{out}}(g_t)$ as a function of t, and plot $E_{\text{out}}(G_t)$ as a function of t on the same figure. Describe your findings.



at the beginning Gt performs as worse as gt with the tree become larger. Eout (G) becomes smaller and more stable. although gt didn't performs well.

13. (Bonus 20 points) Construct a d-d-1 feed-forward neural network with sign(s) as the transformation function (such a neural network is also called a Linear Threshold Circuit) to implement $\texttt{XOR}\big((x)_1, (x)_2, \ldots, (x_d)\big)$. Then, prove that it is impossible to implement $\texttt{XOR}\big((x)_1, (x)_2, \ldots, (x_d)\big)$ with any d-$(d-1)$-1 feed-forward neural network with sign(s) as the transformation function.

for $d=2$

X_1	X_2	XOR(X_1, X_2)
0	0	0
0	1	1
1	0	1
1	1	0

XOR(g_1, g_2) = OR(AND($-g_1$, g_2), AND(g_1, $-g_2$))

$d-1$ neurons in the hidden layer cannot represent all possible combinations of inputs to simulate XOR behavior of d input. with d input, we need to check C_2^d of combination. and thats impossible to accomplish by $d-1$ neurons