# CS480/680: Introduction to Machine Learning

Homework 2

Due: 11:59 pm, November 01, 2022, submit on LEARN.

**NAME**

**student number**

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TA can easily run and verify your results. Make sure your code runs!
[Text in square brackets are hints that can be ignored.]

## Exercise 1: Graph Kernels (7 pts)

One cool way to construct a new kernel from an existing set of (base) kernels is through graphs. Let $\mathcal{G} = (V, E)$ be a directed acyclic graph (DAG), where $V$ denotes the nodes and $E$ denotes the arcs (directed edges). For convenience let us assume there is a source node $s$ that has no incoming arc and there is a sink node $t$ that has no outgoing arc. We put a base kernel $\kappa_e$ (that is, a function $\kappa_e : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$) on each arc $e = (u \to v) \in E$. For each path $P = (u_0 \to u_1 \to \cdots \to u_d)$ with $u_{i-1} \to u_i$ being an arc in $E$, we can define the kernel for the path $P$ as the product of kernels along the path:

$$\forall \mathbf{x}, \mathbf{z} \in \mathcal{X}, \ \kappa_P(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{d} \kappa_{u_{i-1} \to u_i}(\mathbf{x}, \mathbf{z}). \tag{1}$$

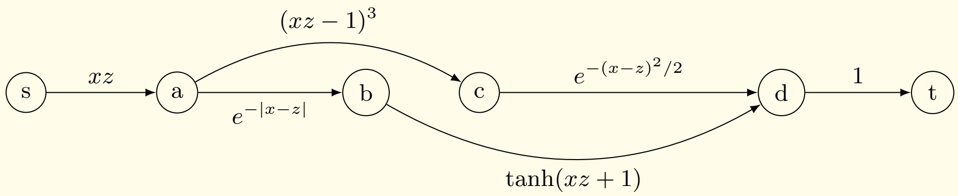
Then, we define the kernel for the graph $\mathcal{G}$ as the sum of all possible $s \to t$ *path kernels*:

$$\forall \mathbf{x}, \mathbf{z} \in \mathcal{X}, \ \kappa_{\mathcal{G}}(\mathbf{x}, \mathbf{z}) = \sum_{P \in \mathrm{path}(s \to t)} \kappa_P(\mathbf{x}, \mathbf{z}). \tag{2}$$

1. (1 pt) Prove that $\kappa_{\mathcal{G}}$ is indeed a kernel. [You may use any property that we learned in class about kernels.]
   Ans:

2. (1 pt) Let $\kappa_i, i = 1, \ldots, n$ be a set of given kernels. Construct a graph $\mathcal{G}$ (with appropriate base kernels) so that the graph kernel $\kappa_{\mathcal{G}} = \sum_{i=1}^{n} \kappa_i$. Similarly, construct a graph $\mathcal{G}$ (with appropriate base kernels) so that the graph kernel $\kappa_{\mathcal{G}} = \prod_{i=1}^{n} \kappa_i$.
   Ans:

3. (1 pt) Consider the subgragh of the figure below that includes nodes $s, a, b, c$ (and arcs connecting them). Compute the graph kernel where $s$ and $c$ play the role of source and sink, respectively. Repeat the computation with the subgraph that includes $s, a, b, c, d$ (and arcs connecting them), where $d$ is the sink now.




Ans:

4. (2 pts) Find an efficient algorithm to compute the graph kernel $\kappa_{\mathcal{G}}(\mathbf{x}, \mathbf{z})$ (for two fixed inputs $\mathbf{x}$ and $\mathbf{z}$) in time $O(|V| + |E|)$, assuming each base kernel $\kappa_e$ costs $O(1)$ to evaluate. You may assume there is always at least one $s - t$ path. State and justify your algorithm is enough; no need (although you are encouraged) to give a full pseudocode.

   [Note that the total number of paths in a DAG can be exponential in terms of the number of nodes $|V|$, so naive enumeration would not work. For example, replicating the intermediate nodes in the above figure $n$ times creates $2^n$ paths from $s$ to $t$.]

   [Hint: Recall that we can use topological sorting to rearrange the nodes in a DAG such that all arcs go from a "smaller" node to a "bigger" one.]

   Ans:

5. (2 pts) Let $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a kernel whose mixed partial derivative exists:

$$\frac{\partial^2 k}{\partial s \partial t}(s, t) =: k'(s, t). \tag{3}$$

   For example, if $k(s, t) = e^{-\frac{1}{2}(s-t)^2}$ is the Gaussian kernel, then the mixed derivative $k'(s, t) = e^{-\frac{1}{2}(s-t)^2}[1 - (s - t)^2]$.

   Prove that $k'$ is also a kernel.

   [*Hint:* $k'(s, t) = \lim_{\delta s \to 0, \delta t \to 0} \frac{k(s+\delta s, t+\delta t) - k(s, t+\delta t) - k(s+\delta s, t) + k(s,t)}{\delta s \cdot \delta t}$.]

Ans:

## Exercise 2: Automatic Differentiation (4 pts)

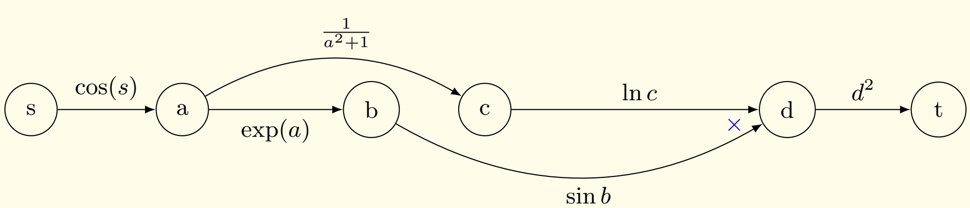
[**FYI**: Compare this exercise with the previous one. See any similarity?]

Recall that in a computational graph $\mathcal{G}$, each node $v$ executes some function $f_v$ on its input nodes $\mathcal{I}_v$ and then sends the output, which we also denote as $v$, i.e. $v = f_v(\mathcal{I}_v)$. Take two arbitrary nodes $u$ and $v$ in $\mathcal{G}$, we claim the following formula:

$$\frac{\partial u}{\partial v} = \sum_{P \in \mathrm{path}(v \to u)} \prod_{(v_i, v_{i+1}) \in P} \frac{\partial v_{i+1}}{\partial v_i}, \tag{4}$$

where $\frac{\partial v_{i+1}}{\partial v_i} = \frac{\partial f_{v_{i+1}}}{\partial v_i}$ and by definition node $v_i$ is an input to node $v_{i+1}$.

1. (1 pt) <u>Compute</u> the following function (i.e., express the sink node $t$ as a function of the source node $s$):




The function implemented by each node is depicted on the incoming edge. For example, we have $a = \cos s$. For node $d$, we also multiply the two incoming edges.

Ans:

2. (1 pt) <u>Compute</u> the derivative of $t$ as a function of $s$. You may use the explicit function $t(s)$ that you derived in Ex 2.1.

Ans:

3. (1 pt, forward mode) Without loss of generality let us order the nodes as $v_0 := s, v_1 = a, v_2 = b, v_3 = c, v_4 = d, t =: v_5$. Going from left to right, and compute sequentially

$$\frac{\partial v_i}{\partial s}, \quad i = 0, 1, \ldots, 5. \tag{5}$$

For instance $\frac{\partial v_0}{\partial s} \equiv 1$. You should not start from scratch for each $i$. Instead, build on results that you have already computed. Do you need to traverse the graph once, twice, or more? [Hint: your final result $\frac{\partial t}{\partial s}$ should match the one in Ex 2.2, provided that you didn't mess things up in calculus...]

Ans:

$$\frac{\partial v_0}{\partial s} = 1 \tag{6}$$
$$\frac{\partial v_1}{\partial s} = \tag{7}$$
$$\frac{\partial v_2}{\partial s} = \tag{8}$$
$$\frac{\partial v_3}{\partial s} = \tag{9}$$
$$\frac{\partial v_4}{\partial s} = \tag{10}$$
$$\frac{\partial t}{\partial s} = \tag{11}$$

As shown above, traversing the graph is enough to compute $\frac{\partial t}{\partial s}$.

4. (1 pt, backward mode) Similar as above, but this time from right to left and compute sequentially

$$\frac{\partial t}{\partial v_i}, \quad i = 5, 4, \ldots, 0. \tag{12}$$

For instance $\frac{\partial t}{\partial v_5} \equiv 1$. You should not start from scratch for each $i$. Instead, build on results that you have already computed. Do you need to traverse the graph once, twice, or more? [Hint: your final result $\frac{\partial t}{\partial s}$ should match the one in Ex 2.2 and Ex 2.3.]

Ans:

$$\frac{\partial t}{\partial v_5} = 1 \tag{13}$$

$$\frac{\partial t}{\partial v_4} = \tag{14}$$

$$\frac{\partial t}{\partial v_3} = \tag{15}$$

$$\frac{\partial t}{\partial v_2} = \tag{16}$$

$$\frac{\partial t}{\partial v_1} = \tag{17}$$

$$\frac{\partial t}{\partial s} = \tag{18}$$

As shown above, traversing the graph is enough to compute $\frac{\partial t}{\partial s}$.

## Exercise 3: Adaboost (9 pts)

In this exercise we will implement Adaboost on a synthetic dataset. Recall that Adaboost aims at minimizing the exponential loss:

$$\min_{\mathbf{w}} \sum_i \exp\left(-y_i \sum_j w_j h_j(\mathbf{x}_i)\right), \tag{19}$$

where $h_j$ are the so-called weak learners, and the combined classifier

$$h_{\mathbf{w}}(\mathbf{x}) := \sum_j w_j h_j(\mathbf{x}). \tag{20}$$

Note that we assume $y_i \in \{\pm 1\}$ in this exercise, and we simply take $h_j(\mathbf{x}) = \mathrm{sign}(\pm x_j + b_j)$ for some $b_j \in \mathbb{R}$. Upon defining $M_{ij} = y_i h_j(\mathbf{x}_i)$, we may simplify our problem further as:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \mathbf{1}^\top \exp(-M\mathbf{w}), \tag{21}$$

where exp is applied component-wise and $\mathbf{1}$ is the vector of all 1s.

Recall that $(s)_+ = \max\{s, 0\}$ is the positive part while $(s)_- = \max\{-s, 0\} = |s| - s_+$.

**Algorithm 1:** Adaboost.

**Input:** $M \in \mathbb{R}^{n \times d}$, $\mathbf{w}_0 = \mathbf{0}_d$, $\mathbf{p}_0 = \mathbf{1}_n$, $\mathsf{max\_pass} = 300$

**Output:** $\mathbf{w}$

**for** $t = 0, 1, 2, \ldots, \mathsf{max\_pass}$ **do**
- $\mathbf{p}_t \leftarrow \mathbf{p}_t / (\mathbf{1}^\top \mathbf{p}_t)$ `// normalize`
- $\boldsymbol{\epsilon}_t \leftarrow (M)_-^\top \mathbf{p}_t$ `//` $(\cdot)_-$ `applied component-wise`
- $\boldsymbol{\gamma}_t \leftarrow (M)_+^\top \mathbf{p}_t$ `//` $(\cdot)_+$ `applied component-wise`
- $\boldsymbol{\beta}_t \leftarrow \frac{1}{2}(\ln \boldsymbol{\gamma}_t - \ln \boldsymbol{\epsilon}_t)$ `//` $\ln$ `applied component-wise`
- choose $\boldsymbol{\alpha}_t \in \mathbb{R}^d$ `// decided later`
- $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \boldsymbol{\alpha}_t \odot \boldsymbol{\beta}_t$ `//` $\odot$ `component-wise multiplication`
- $\mathbf{p}_{t+1} \leftarrow \mathbf{p}_t \odot \exp(-M(\boldsymbol{\alpha}_t \odot \boldsymbol{\beta}_t))$ `// exp applied component-wise`

1. (2 pts) We claim that Algorithm 1 is indeed the celebrated Adaboost algorithm if the following holds:
   - $\boldsymbol{\alpha}_t$ is 1 at some entry and 0 everywhere else, i.e., it indicates which weak classifier is chosen at iteration $t$.
   - $M \in \{\pm 1\}^{n \times d}$, i.e., if all weak classifiers are $\{\pm 1\}$-valued.

With the above conditions, prove that (a) $\boldsymbol{\gamma}_t = \mathbf{1} - \boldsymbol{\epsilon}_t$, and (b) the equivalence between Algorithm 1 and the Adaboost algorithm in class. [Note that our labels here are $\{\pm 1\}$ and our $\mathbf{w}$ may have nothing to do with the one in class.]

Ans:

2. (3 pts) Let us derive each week learner $h_j$. Consider each feature in turn, we train $d$ linear classifiers that each aims to minimize the weighted training error:

$$\min_{b_j \in \mathbb{R}, s_j \in \{\pm 1\}} \sum_{i=1}^{n} p_i [\![ y_i(s_j x_{ij} + b_j) \leq 0 ]\!], \tag{22}$$

where the weights $p_i \geq 0$ and $\sum_i p_i = 1$. Find (with justification) an optimal value for each $b_j$ and $s_j$. [If multiple solutions exist, you can use the middle value.] Apply your algorithm to the default dataset (available on course website), where $d = 23$. Report the feature (i.e. weak learner) that results in the best and worst test error, respectively, along with its training error.

Ans:

Feature achieves the best test error at , with training error and $s_j =$ , $b_j =$

Feature achieves the worst test error at , with training error and $s_j =$ , $b_j =$

3. (2 pts) [Parallel Adaboost.] Implement Algorithm 1 with the following choices:

- $\boldsymbol{\alpha}_t \equiv \mathbf{1}$
- pre-process $M$ by dividing a constant so that for all $i$ (row), $\sum_j |M_{ij}| \leq 1$.

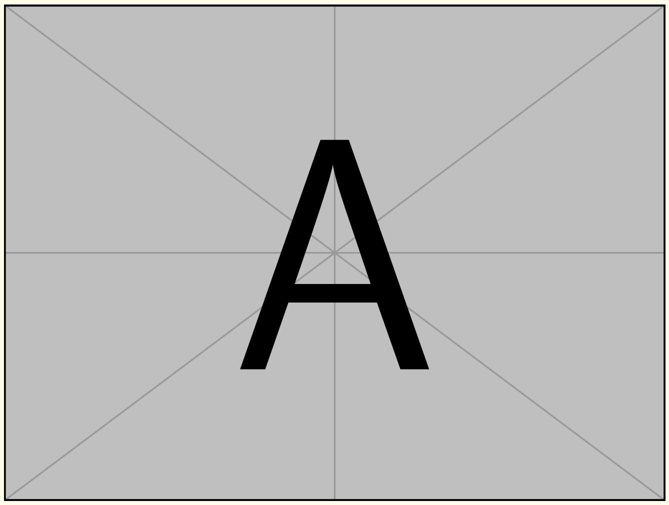
Run your implementation on the default dataset (available on course website), and report the training loss in (21), training error, and test error w.r.t. the iteration $t$, where

$$\mathrm{error}(\mathbf{w};\mathcal{D}) := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} [\![ y h_{\mathbf{w}}(\mathbf{x}) \leq 0 ]\!]. \tag{23}$$

[Recall that $h_{\mathbf{w}}(\mathbf{x})$ is defined in (20) while each $h_j$ is decided in the previous question. In case you fail to determine $h_j$, you may simply use $h_j(\mathbf{x}) = x_j$ in Ex 2.3 to Ex 2.5.]

[Note that $\mathbf{w}_t$ is dense (i.e. using all weak classifiers) even after a single iteration.]

Ans: We report all 3 curves in one figure, with clear coloring and legend to indicate which curve is which.




4. (2 pts) [Sequential Adaboost.] Implement Algorithm 1 with the following choice:

   - $j_t = \mathrm{argmax}_j \, |\sqrt{\epsilon_{t,j}} - \sqrt{\gamma_{t,j}}|$ and $\boldsymbol{\alpha}_t$ has 1 on the $j_t$-th entry and 0 everywhere else.

   Run your implementation on the default dataset (available on course website), and report the training loss in (21), training error, and test error in (23) w.r.t. the iteration $t$.

   [Note that $\mathbf{w}_t$ has at most $t$ nonzeros (i.e. weak classifiers) after $t$ iterations.]

   Ans: We report all 3 curves in one figure, with clear coloring and legend to indicate which curve is which.