NYU | TANDON SCHOOL OF ENGINEERING

New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Written Homework 4.
Due Wednesday, April 16, 2025, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually.

Problem 1: Complexity of Hypothesis Classes (15pts)

(a) In the lecture on learning theory we saw how to bound the number of training examples required to PAC-learn certain functions (aka models, aka hypothesis classes) in the realizable setting. For the following functions, give as tight an upper bound as you can on how many samples are required for PAC-learning with accuracy ϵ and success probability $(1-\delta)$.

(i) A *decision list* is a boolean function mapping $\{x_1, \ldots, x_d\} \in \{0,1\}^d \to \{0,1\}$ of the following form:

$$\begin{aligned}&\text{If}(y_1)\text{ return } z_1\\ &\text{Else if}(y_2)\text{ return } z_2\\ &\vdots\\ &\text{Else if}(y_k)\text{ return } z_k,\\ &\text{Else return } z_{k+1}.\end{aligned}$$

Above $y_i \in \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_d, \bar{x}_d\}$ for each i and $z_i \in \{0,1\}$ for each i. Here $\bar{x}_i$ denotes the logical "not" of variable x_i. k is the number of terms in the decision list and can be as long as we want. Try to give a bound that depends polynomially on d but not on k at all.

(ii) A binary linear *threshold function* is a function mapping $\{x_1, \ldots, x_d\} \in \{-1,1\}^d \to \{0,1\}$ of the following form:

$$\mathbb{1}[w_1x_1 + w_2x_2 + \ldots + w_dx_d \geq \lambda]$$

where $w_i \in \{-1,1\}$ is a binary weight for variable x_i and $\lambda \in \mathbb{R}$ is an arbitrary threshold.

(b) Suppose I want to perform model selection. I have q different hypothesis classes $\mathcal{H}_1, \ldots, \mathcal{H}_q$ and I know that for some i, there is a function $h^* \in \mathcal{H}_i$ that perfectly fits my data. I.e. with $R_{pop}(h^*) = 0$. My goal is to find some h such that $R_{pop}(h) \leq \epsilon$. Give as tight an upper bound as you can on how many samples are required to solve this problem with success probability $(1-\delta)$.

Problem 2: Kernels for Shifted Images (15pts)

In class we discussed why the Gaussian kernel is a better similarity metric for MNIST digits than the inner product. Here we consider an additional modification to the Gaussian kernel that is very similar to state-of-the-art kernels for digit and character classification, which can achieve 99%+ accuracy on MNIST.

For illustration purposes we consider 5x5 black and white images: a pixel has value 1 if it is white and value 0 if it is black. For example, consider the following images of two 0s and two 1s:

$$I_1 = \begin{bmatrix}0&0&0&0&0\\0&1&1&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\end{bmatrix} \quad I_2 = \begin{bmatrix}0&0&0&0&0\\0&0&1&0&0\\0&0&1&0&0\\0&0&1&0&0\\0&0&0&0&0\end{bmatrix} \quad I_3 = \begin{bmatrix}0&0&0&0&0\\1&1&1&0&0\\1&0&1&0&0\\1&1&1&0&0\\0&0&0&0&0\end{bmatrix} \quad I_4 = \begin{bmatrix}0&0&0&0&0\\0&0&0&1&0\\0&0&0&1&0\\0&0&0&1&0\\0&0&0&0&0\end{bmatrix}$$

(a) Let $\mathbf{x}_i \in \{0,1\}^{25}$ denote the vectorized version of image I_i, obtained by concatenating the rows of the matrix representation of the image into a vector. Compute the 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using the standard Gaussian kernel $k_G(I_i, I_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}$ (this is just a simple calculation, but you might want to do it in Python to save time).

(b) Suppose I_1 and I_2 are in our training data and I_3 and I_4 are in our test data. Which training image is most similar to each of our test images according to Gaussian kernel similarity? Do you expect a kernel classifier (e.g., a 1-nearest neighbor method) to correctly or incorrectly classify I_3 and I_4?

(c) Consider a "left-right shift" kernel, which is a similarity measure defined as follows:

For an image I_i, let I_i^{right} be the image with its far right column removed and let I_i^{left} be the image with its far left column removed. Intuitively, I_i^{right} corresponds to the image shifted one pixel to the right and I_i^{left} corresponds to the image shifted one pixel left. Define a new similarity metric k_{shift} as follows:

$$k_{\text{shift}}(I_i, I_j) = k_G(I_i^{\text{right}}, I_j^{\text{right}}) + k_G(I_i^{\text{left}}, I_j^{\text{left}}) + k_G(I_i^{\text{right}}, I_j^{\text{left}}) + k_G(I_i^{\text{left}}, I_j^{\text{right}})$$

Intuitively this kernel captures similarity between images which are similar *after a shift*, something the standard Gaussian kernel does not account for.

Recompute the a 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using k_{shift}.

(d) Again I_1 and I_2 were in our training data and I_3 and I_4 were in our test data. Now which training image is most similar to each of our test images according to the "left-right shift" kernel? Do you expect a kernel classifier (e.g., a 1-nearest neighbor method) to correctly or incorrectly classify I_3 and I_4?

(e) Prove that k_{shift} is a positive semi-definite kernel function. **Hint:** Use the fact that k_G is positive semi-definite.

Problem 3. Randomized Coordinate Descent (8pts)

Randomized coordinate descent (RCD) is a close relative of stochastic gradient descent that is often used for minimizing loss functions in machine learning. See below for pseudocode. Note that for a bolded vector $\mathbf{a}$, we used a_i to denote the i^{th} entry.

- Choose starting vector $\boldsymbol{\beta} \in \mathbb{R}^d$ and positive learning rate η.
- For $i = 1, \ldots, T$
  - Compute $\mathbf{g} = \nabla L(\boldsymbol{\beta})$.
  - Choose j uniformly at random from $j \in \{1, \ldots, d\}$.
  - For index j, update $\beta_j \leftarrow \beta_j - \eta \cdot g_j$.

The method is similar to stochastic gradient descent in that it often allows for cheaper per-iteration cost than gradient descent. It does so by updating one randomly chosen entry (the j^{th} entry) of the parameter vector $\boldsymbol{\beta}$. The vector is updated by subtracting off a scaling of the j^{th} entry of the current gradient.

(a) Let $\boldsymbol{\beta}^{(i)}$ be the value of the parameter vector $\boldsymbol{\beta}$ at the end of the i^{th} iteration of the for loop above. Prove that $\mathbb{E}[\boldsymbol{\beta}^{(i)} - \boldsymbol{\beta}^{(i-1)}] = -c\nabla L(\boldsymbol{\beta}^{(i-1)})$ for some positive scalar constant c. I.e., like SGD, RCD moves in the direction of the negative gradient in expectation.

(b) Prove that, like the steepest descent methods considered in Problem 2 of Homework 3 $\lim_{\eta \to 0} \frac{L(\boldsymbol{\beta}^{(i-1)}) - L(\boldsymbol{\beta}^{(i)})}{\eta}$ is positive for stochastic coordinate descent. Interestingly, this same claim is *not true* for stochastic gradient descent. Even as the step size goes to zero, an SGD update is not guaranteed to decrease the objective value.

(c) **Optional Bonus (5pts).** Consider the least squares regression loss, $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ for a data matrix $\mathbf{X}$ with d columns. Recall that this loss has gradient $\nabla L(\boldsymbol{\beta}) = 2\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})$. Show that, after an upfront cost of $O(nd)$ on the first iteration, each subsequent iteration of coordinate descent on this loss can be implemented in $O(n)$ time. Write pseudocode showing an $O(n)$ time implementation. **Hint:** This problem requires some cleverness. Try to use the information from previous iterations to your advantage – you cannot be able to compute $\mathbf{g}$ from scratch at each iteration.