CSE 251A: AI - Learning Algorithms **Winter 2026**

Project 2 — Coordinate Descent

Overview

In this project we consider a standard unconstrained optimization problem:

$$\min L(w)$$

where $L(\cdot)$ is some cost function and $w \in \mathbb{R}^d$. In class, we looked at several approaches to solving such problems—such as gradient descent and stochastic gradient descent—under differentiability conditions on $L(w)$. We will now look at a different, and in many ways simpler, approach:

- Initialize w somehow.
- Repeat: pick a coordinate $i \in \{1, 2, \ldots, d\}$, and update the value of w_i so as to reduce the loss.

Two questions need to be answered in order to fully specify the updates:

(a) Which coordinate to choose?

(b) How to set the new value of w_i?

Give answers to these questions, and thereby flesh out a coordinate descent method. For (a), your solution should do something more adaptive than merely picking a coordinate at random.

Then implement and test your algorithm on a *logistic regression* problem. First download the `wine` data set from:

`https://archive.ics.uci.edu/ml/datasets/Wine`

This contains 178 data points in 13 dimensions, with 3 classes. Just keep the first two classes (with 59 and 71 points, respectively) so as to create a **binary** problem.

What to turn in

On the due date, upload (to **Gradescope**) a **pdf** report containing the elements listed below (each labeled clearly). We require that your pdf report be produced using Latex with standard conference style files. We suggest cloning the the same Overleaf template you used in Project 1. Finally, be sure to also **uploaded a zip file of your code** for this project to the separate Gradescope entry for code.

1. *A short, high-level description of your coordinate descent method.*

   In particular, you should give a concise description of how you solve problems (a) and (b) above. Do you need the function $L(\cdot)$ to be differentiable (and maybe even have continuous second-order derivatives), or does it work with any cost function?

2. *Convergence.*

   Under what conditions do you think your method converges to the optimal loss? There's no need to prove anything: just give a few sentences of brief explanation.

3. *Experimental results.*

   (Remember that all training must take place on just classes 1 and 2.)

   - Begin by running a standard logistic regression solver (e.g., from `scikit-learn`) on the training set. It should not be regularized: if the solver forces you to do this, just set the regularization constant suitably to make it irrelevant. Make note of the final loss L^*.
   - Then, implement your coordinate descent method and run it on this data.
   - Finally, compare to a method that chooses coordinates i uniformly at random and then *updates* w_i *using your method* (we'll call this "random-feature coordinate descent").
   - Produce a clearly-labeled graph that shows how the loss of your algorithm's current iterate—that is, $L(w_t)$—decreases with t; it should asymptote to L^*. On the same graph, show the corresponding curve for random-feature coordinate descent.

4. *Critical evaluation.*

   Do you think there is scope for further improvement in your coordinate descent scheme in (1); if so, how?

5. *Sparse coordinate descent.*

   Now, suppose we want a k*-sparse* solution w: that is, one that has at most k nonzero entries.

   - Propose a modified version of your method for this task. Assume k is part of the input, along with the data.
   - Do you think this method always find the best k-sparse solution when $L(\cdot)$ is convex?
   - Try this out on the `wine` data. Make a table of loss values obtained for a few selected values of k.