Simulated Annealing (SA)

Approach:

* We start with a feasible solution. Since with SVM, any (w,b) combination will be “feasible” since s\_i will be set as result of picking (w,b) that makes the constraint hold, specifically s\_i = max(0, 1-y(wx+b)). So this will be an application of unconstrained optimization
* We generate a random neighbour x’ and one of two things happens:
  + The obj value is better and we move to it
  + The obj value is worse, but we will move to it with probability p, stay put at x(k) with probability 1-p
    - P(x(k) -> x’) = p = exp((z(k)-z’)/T(k)), defined later in pseudocode
* Stopping condition: typically for SA we set the number of iterations, but for our project we want our final obj value to be within 5% of the optimal value.
  + In general for unconstrained optimization, if the problem (min problem) is convex (e.g. y = x^2) the min is the global min. If it’s non-convex (e.g. there’s a bunch of local mins), our solution may or may not be the global min and it depends on our starting point.
  + In SA, we allow a chance to escape a local region with a probability p which generally increases as we get to the end of a run (see pseudocode, T(k) decreases, p increases).
  + Apparently SVM is convex, so we should be getting a global min, but it depends on how we formalize the problem
  + Since the candidate point to move to is random we can’t set a tolerance for stopping, so we’ll experiment with different N, T(0)
  + Matlab has an SA function already, so I’ll use this to base off my methods and for testing

<https://www.mathworks.com/help/gads/how-simulated-annealing-works.html>

Pseudocode:

Function (x(0), T(0), alpha, N) (alpha e.g. 0.95, x(0) = 0 for all w,b)

x\* = x(0), z\* = z(0)

for k = 0:N

generate rand neighbour: from Matlab, candidate point x’ = x(k) + uT(k), u is unif[0,1], direction

if z’ < z(k) (z’ is better since min), or if z’ > z(k) and u < p (p = exp((z(k)-z’)/T(k)) < 1)

move: z(k+1) = z’, x(k+1) = x’

check against best sol so far: if z’ < z\*

z\* = z’, x\* = x’

else

stay put: z(k+1) = z(k), x(k+1) = x(k)

T(k+1) = alpha\*T(k)

Return x\*, z\*

Note: Matlab’s default stopping condition is when the AVERAGE change in z < tolerance (default 10^-6)

* To do this we would have to keep track of the changes and keep evaluating it for a while condition
* But matlab does a weird thing with k, so I won’t be using it, instead I’ll fine tune N, T(0)