QUIZ 3

purdue university · cs 51500

David Gleich

matrix computations

June 27, 2023

Purdue Honor Code

The purpose of the Purdue University academic community is to search for truth and to endeavor to communicate with each other. Self-discipline and a sense of social obligation within each individual are necessary for the fulfillment of these goals. It is the responsibility of all Purdue students to live by this code, not out of fear of the consequences of its violation, but out of personal self-respect. As human beings we are obliged to conduct ourselves with high integrity. As members of the civil community we have to conduct ourselves as responsible citizens in accordance with the rules and regulations governing all residents of the state of Indiana and of the local community. As members of the Purdue University community, we have the responsibility to observe all University regulations.

To foster a climate of trust and high standards of academic achievement, Purdue University is committed to cultivating academic integrity and expects students to exhibit the highest standards of honor in their scholastic endeavors. Academic integrity is essential to the success of Purdue University's mission. As members of the academic community, our foremost interest is toward achieving noble educational goals and our foremost responsibility is to ensure that academic honesty prevails.

Exam Rules

AP____ This quiz is a *open to all class and internet resources* except talking with people.

AP____ The contents of this quiz are protected by copyright. Posting any piece of this quiz – including excepts that may be subject to fair-use rights – to any database, online resource, electronic medium, or other similar repository will be considered academic dishonesty (see below). Exams may be individually watermarked to identify violations.

AP____ You may not consult with anyone other than the professor and the TA.

AP____ Any behavior consistent with academic dishonesty (i.e. cheating) *will not be tolerated and may result in a 0 even a failing grade in the class*.

AP You have until the Gradescope or Blackboard submission closes to com plete and submit the exam. No exceptions.

Write your name, PUID, and sign below to indicate you agree with the statement:

The remainder of this exam represents my own work.

(Your Name) Aditya Patel (PUID) 0035573514

Aditya Patel
(Signature)

Problem 1 *(15 points)*

Recall the stochastic gradient descent iteration for a least squares problem.

```
for iter=1:maxiter
  s = rand(1:m)
  as = A[s,:]
  bs = b[s]
  g = 2(as'*x - bs).*as
  x -= step*g
end
```

Justify that stochastic gradient descent for least squares takes $O(n)$ work per step where the length of $\mathbf{x}$ is n.

One execution of the for-loop in the SGD is comprised of three stages, random sampling, gradient calculations, and gradient step.

RANDOM SAMPLING:

The random sampling stage is where the step value, as vector, and bs value are defined.

- s = rand(1:m): This is a single operation time, as it is generating a single integer.
- as = A[s,:]: Taking s = n in the worst case, defining as, the 1xn row vector, would take n operations.
- bs = b[s]: Identifying a single element from b would take a single operation.

Therefore the total time taken in the worst case for random sampling is O(n + 2) = O(n) work.

GRADIENT CALCULATION:

The gradient calculation step is where the gradient is calculated via three operations. Working from inside out:

- as' * x: This is the matrix dot product operation between a 1 x n vector and n x n matrix. This takes n operations in the worst case.
- 2 (as' * x – bs): This has two scalar operations, scalar multiplication by 2 and a scalar subtraction by bs. This takes 2 operations in the worst case.
- 2(as' * x – bs) .* as: this is elementwise multiplication of a 1xn vector by a 1xn vector. This takes n operations to complete.

Therefore, the total time taken in the worst case for the gradient calculation is O(2n+2) = O(n) work.

GRADIENT STEP:

The final step adjusts the matrix x by performing the gradient descent.

- x -= step * g: scalar multiplication on a 1xn matrix and updates each element of the matrix x by this gradient, which takes n operations.

Therefore, the total time taken in the worst case for the gradient step is O(n) work.

CONCLUSION:

For all steps, we see that the work performed in the worst case is:

- Random Sampling: O(n)
- Gradient Calculation: O(n)
- Gradient Step: O(n)

This means that one step for the gradient descent takes O(3n) = O(n) work overall.

Problem 2 *(20 points)*

Recall that a kernel matrix is constructed as follows:

$$K_{ij} = f(\mathbf{x}_i,\mathbf{x}_j)$$

for some function f such as the inner-product $f(\mathbf{x}_i,\mathbf{x}_j) = \mathbf{x}^T{}_i\,\mathbf{x}_j$ or $f(\mathbf{x}_i,\mathbf{x}_j) = e^{-\|\mathbf{x}_i-\mathbf{x}_j\|2/(2\sigma)}$. Suppose we have computed K for a set of N points: $\mathbf{x}_1,\ldots,\mathbf{x}_N$. In this example, the result is a rank N matrix. Now, suppose we change two data points. Without losing any generation, let's just suppose it's the first and second one. So the new set of points is $\mathbf{x}'_1, \mathbf{x}'_2, \mathbf{x}_3 \ldots, \mathbf{x}_N$ (where $\mathbf{x}_3,\ldots,\mathbf{x}_N$ are all the same). Let K' be the kernel matrix of the new set of points.

(10 points) Give an algorithm to compute K' given the values of K.

```
using LinearAlgebra

# Define the kernel function
function kernel(x, y)
    return dot(x, y)
end

# Function to compute the new kernel matrix K' given K
function update_kernel_matrix(K, x_new1, x_new2, x_old)
    N = size(K, 1)
    K_prime = copy(K)

    # Compute new kernel values involving the new points
    for i in 1:N
        if i > 2
            K_prime[1, i] = kernel(x_new1, x_old[i, :])
            K_prime[2, i] = kernel(x_new2, x_old[i, :])
            K_prime[i, 1] = kernel(x_old[i, :], x_new1)
            K_prime[i, 2] = kernel(x_old[i, :], x_new2)
        end
    end

    # Compute new kernel values for the updated points
    K_prime[1, 1] = kernel(x_new1, x_new1)
    K_prime[1, 2] = kernel(x_new1, x_new2)
    K_prime[2, 1] = kernel(x_new2, x_new1)
    K_prime[2, 2] = kernel(x_new2, x_new2)
```

```
    return K_prime
end
```

(10 points) Produce the tightest bound you can on the rank of the difference $K' - K$. (Note: There is an extremely trivial answer that will be worth 4 points.)

The trivial bound is 4, since the changes are based out of a 2x2 matrix.
The tightest bound is 2, since the difference in between x1 and x2, given by K' – K, can be represented by at most 2 linearly independent vectors.

Problem 3 *(15 points)*

Fill in the remainder of this julia function.

"""

‘solve_with_svd‘ solves a linear system of equations using the result of a singular value decomposition. --------
---

The input matrices U, V are given exactly by the output of svd(A).U and svd(A).V and the input s is given by the output of svd(A).s for a square n-by-n matrix A. This function returns a vector x that will solve a linear system of equations Ax = b but does not call Julia’s built-in "\" solver at all.
"""

```julia
function solve_with_svd(U, s, V, b)
    # Compute pseudo inverse of sigma
    sigma_inv = Diagonal(1 ./ s)

    # Solve for y in U * y = B
    y = U' * b

    # Solve for z in  sigma * z = y
    z = sigma_inv * y

    # solve for x in V' * x = z
    x = V * z

    return x
end
```