Table of Contents

rungreedyfirst.m

```
% Runs Greedy First algorithm and returns regret and fraction of
 pulls.
%
% This code implements the Greedy First algorithm. For more
 information,
% see our paper:

% - https://arxiv.org/abs/1704.09011.

% The algorithm proceeds as follows: first each arm is forced sampled
 by
% some number of time periods given by random_initialization). Then,
 at
% each time period the arm with the highest estimated mean is pulled.
% The algorithm keeps track of the %smallest value among minimum
% eigenvalues of the covariance matrices of arms and switches to OLS
 Bandit
% algorithm if this number drops below a linearly growing (in t)
 threshold.
% For more details on OLS bandit and discussion about its parameters
 see:

% - https://pubsonline.informs.org/doi/abs/10.1287/11-SSY032.
% - https://pubsonline.informs.org/doi/10.1287/opre.2019.1902.

% The algorithm uses Sherman-Morrison formula for fast updates of
 least
% squares estimations of arms. This fast update is applied once the
% covariance matrices become invertible. Before having invertible
% covariance matrices, the algorithm applies ridge regression for more
% accurate estimations.
%
```

Inputs:

```
k: Number of arms.
T: Time horizon.
d: Dimension of covariates.
b: A k*d matrix of arm parameters.
sigma_e: Standard deviation of noise (used only for reward
 generation).
```

```
    This parameter is unused if noise and contexts are provided.
sigma_x: Standard deviation of covariates
    (used only for context generation for Gaussian contexts).
    This parameter is unused if noise and contexts are provided.
xmax: Maximum of l2-norm of covariates
    (used only for context generation). This parameter is unused if
    noise and contexts are provided.
h: h parameter used in OLS bandit algorithm.
q: q parameter used in OLS bandit algorithm.
t0: Time at which the algorithm starts checking minimum eigenvalues
    from then onwards.
min_eig_threshold: Bound on the growth of smallest minimum eigenvalue
    among covariance matrices.
random_initialization: Number of rounds of forced sampling per arm in
    the beginning. If set to zero, the algorithm is purely greedy.
verbose: Whether to print outputs or not.
varargin: Additional arguments. In particular, if these are not
    provided the noise and contexts will be generated according to
    Gaussian and truncated Gaussian distributions.
    In case they are provided,
 there should exactly be THREE additional
    arguments. The first one is contexts. The second one is a binary
    input, called noise_input. If noise_input = 1, this means the last
    argument will be noise e=(Y-X*beta). On the other hand, if
    noise_input = 0, then the last argument will be Y or
    rewards. Note that the noise should be T*1 while rewards should be
    T*k.
```

Outputs:

```
regret: Cumulative regret as a running sum over regret terms.
fractions: Fractions of pulls of different arms.
switched: The time at which Greedy-
First switches to OLS bandit. If the
    algorithm does not switch, this is equal to -1.
```

Code:

```
function [regret, fractions, switched] = rungreedyfirst(k, T, d,
 b, ...
    sigma_e, sigma_x, xmax, h, ...
    q, t0, min_eig_threshold, random_initialization, ...
    verbose, varargin)

warning('off','all');

switched = -1;  % This only will be set if algorithm switches from
 greedy.

if nargin==13 % Context and noise are NOT provided, so generate those.
    % Noise is Gaussian with std sigma_e.
    e = randn(T, 1)*sigma_e;
    noise_input = 1;
```

```
    % Contexts follow truncated gaussian distributions with l-infinity
 norm
    % at most xmax.
    X = max(-xmax, min(xmax, mvnrnd(zeros(d, 1), sigma_x, T)));
else
    X = varargin{1};
    noise_input = varargin{2};
    if(noise_input==1)
        e = varargin{3};
    else
        rewards = varargin{3};
    end
end

reward_vector = zeros(T, k);  % Vector of all (potential) rewards.
pull_ind = zeros(T, k);  % Binary indicator whether each is pulled.

regret = zeros(1, T);
betahat = b * 0;  % Initialize all arm estimations with vector of
 zeros.

XtopX_inv = zeros(d, d, k);

% Whether direct LS calculation using normal equations is needed.
no_LS_calculations = zeros(k, 1);

% Minimum eigenvalue of sample covariance matrix of different arms.
lam_hat = zeros(k, 1);
lam0 = inf;

switch_to_olsbandit = 0;  % Binary, whether algorithm switches or not.

t = 1;

while (t<=T && switch_to_olsbandit==0)
    x = X(t,:)';
    %----- First, choose which arm to pull this round.
    if (t>random_initialization * k)
        z = betahat * x;
        opt_arms = find(z==max(z));
        % Break ties randomly.
        arm_pulled = opt_arms(randi(length(opt_arms)));
    else
        arm_pulled = mod(t-1, k) + 1;
    end
    pull_ind(t, arm_pulled) = 1;

    %------ Second, calculate the regret.

    if(noise_input==1)
        bx = b*x;
        ourreward = bx(arm_pulled);
        bestreward = max(bx);
```

```
    else
        ourreward = rewards(t, arm_pulled);
        bestreward = max(rewards(t,:));
    end

    if (t==1)
        regret(t) = bestreward - ourreward;
    else
        regret(t) = regret(t-1) + bestreward - ourreward;
    end

    %------ Third, update estimates.

    if(noise_input==1)
        reward_vector(t, arm_pulled) = ourreward + e(t);
    else
        reward_vector(t, arm_pulled) = rewards(t, arm_pulled);
    end

    if (no_LS_calculations(arm_pulled)==0)
       obs_filt = find(pull_ind(:, arm_pulled)==1);  % Filter
observations.
       lsX = X(obs_filt, :); % Design matrix.
       lsY = reward_vector(obs_filt, arm_pulled);  % Observations.
       if (rank(lsX)>=d)
           XtopX_inv(:, :, arm_pulled) = inv(lsX'*lsX);
           betahat(arm_pulled, :) = lsX\lsY; %
           no_LS_calculations(arm_pulled) = 1;
       else
           % Empirical estimator of \|x\|_2, used for ridge
regression.
           penalty = norm(lsX)/size(lsX,1);
           hat_beta = (lsX'*lsX + penalty^2 * eye(d)) \ (lsX' * lsY);
           betahat(arm_pulled,:) = hat_beta';
       end
    else
       [XtopX_inv(:, :, arm_pulled), betahat_vertical] =
rankoneupdate(...
           XtopX_inv(:, :, arm_pulled), betahat(arm_pulled, :)',
x, ...
           reward_vector(t, arm_pulled));
       betahat(arm_pulled,:) = betahat_vertical';
    end

    %------ Fourth, update lambda_min and check whether we should
    %switch
    if (t==t0)
        for i=1:k
            if(all(XtopX_inv(:, :, i))==0)
                lam_hat(i) = 0;
            else
                lam_hat(i) = 1 / max(eig(XtopX_inv(:, :, i)));
            end
        end
```

```
        lam0 = 0.5 * min(lam_hat) / t0;

        if (lam0 < min_eig_threshold)
            switch_to_olsbandit = 1;
            switched = t;
            if(verbose == 1)
                fprintf(...
                'GF: switched at t=t0=%d, lam_hat vector=%s. \n', ...
                    t, sprintf('%d ', lam_hat) ...
                    );
            end
        end
    elseif (t>(2*t0))
        lam_hat(arm_pulled) = 1/max(eig(XtopX_inv(:, :, arm_pulled)));
        [lmmin, bad_arm]=min(lam_hat);
        if (lmmin < (lam0*t/4));
            switch_to_olsbandit = 1;
            switched = t;
            if(verbose == 1)
                fprintf('GF: switched at t=%d, lam_hat vector=%s,
 lambdamin/t=%f, lam0/4=%f, bad_arm=%d. \n', ...
                    t, sprintf('%d ', lam_hat), lmmin/t, lam0/4,
 bad_arm);
            end
        end
    end
    if(verbose==1)
        if (mod(t,500)==0)
            if(noise_input==1)
            fprintf('GF: t=%d, parameter estimation error = %f.\n',
 t, ...
                norm(b - betahat, 'fro'));
            else
            fprintf('GF: t=%d, regret = %f.\n', t, ...
                regret(t));
            end
        end
    end
    t = t + 1;
end
% If t<T it means that the algorithm has switch to OLS bandit before
 the
% time horizon finishes. We need to execute OLS bandit for the
 remaining
% time periods.
if(t<=T)
    if(verbose==1)
        fprintf(...
            'GF: Running OLS bandit for the remaining %d periods.\n',
 T-t+1);
    end
    if(noise_input==1)
        [regret_OLS, ~, pull_ind_OLS] = runOLSbandit(...
```

```
            k, T-t+1, d, ...
            b, sigma_e, sigma_x, xmax, h, ...
            q, verbose, X(t:T,:), noise_input, e(t:T));
        regret(t:T) = regret(t-1) + regret_OLS;
        pull_ind(t:T, :) = pull_ind_OLS;

    else
        % This version will be mainly used when working with real
 datasets.
        % The main idea for improving the regret of Greedy-First
 (which is
        % quite effective when dimension d is large) is to reuse the
 first
        % random_initialization*k samples gathered at the beginning of
        % compiling greedy algorithm. As these samples are random
 samples,
        % they can be safely used as the initialization (or forced
        % sampling) samples of OLS. The way we do it here is that we
 feed
        % these samples again to the OLS, while we deduct the regret
 of
        % these first random_initialization*k rounds, when Greedy-
First
        % switches to OLS Bandit.

        ind = [1:random_initialization * k, t:T];
        X_aug = X(ind, :);
        rewards_aug = rewards(ind, :);
        [regret_OLS, ~, pull_ind_OLS] = runOLSbandit(...
            k, length(ind), d, ...
            b, sigma_e, sigma_x, xmax, h, ...
            q, verbose, X_aug, noise_input, rewards_aug);
            regret(t:T) = regret(t-1) + ...
                regret_OLS((random_initialization * k + 1):end) - ...
                regret_OLS(random_initialization*k);
        pull_ind(t:T, :) = pull_ind_OLS(...
            (random_initialization * k + 1):end, :);
    end

end

fractions = mean(pull_ind);  %fraction of times each arm is pulled
if(verbose==1)
    fprintf('GF: Fraction of pulls = %f. \n', fractions)
    fprintf('GF: Total regret occured = %f. \n', regret(end))
end

end
```

Published with MATLAB® R2015a