Table of Contents

rungreedybandit.m

```
% Runs Greedy Bandit algorithm and returns regret and fraction of
 pulls.
%
% This code implements the Greedy Bandit 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 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.
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.
```

Code:

```
function [regret, fractions] = rungreedybandit(k, T, d, b,
 sigma_e, ...
    sigma_x, xmax, random_initialization, verbose, varargin)

warning('off','all');


if nargin==9 % 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);

for t=1:T
    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
     if (verbose==1)
         if (mod(t,500)==0)
             fprintf('GB: t=%d, parameter estimation error = %f. \n',
 t, ...
                 norm(b - betahat, 'fro'))
         end
     end
end
fractions = mean(pull_ind);  % Fraction of times each arm is pulled.
if(verbose==1)
    fprintf('GB: Total parameter estimation error = %f. \n', ...
        norm(b - betahat, 'fro'));
    fprintf('GB: Fraction of pulls = %f. \n', fractions);
    fprintf('GB: Total regret occured = %f. \n', regret(end));
end
end
```

Published with MATLAB® R2015a