Table of Contents

rungreedybanditlogistic.m

```
% Runs Greedy Bandit algorithm with logistic reward.
%
% This code implements the Greedy Bandit algorithm with logistic
 rewards.
% 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. Unlike the linear case, this
% version does not apply Sherman-Morrison rank on update.
% 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.
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.
contexts: A T*d matrix of contexts.
reward_vector: A T*k matrix of reward vectors.
verbose: Whether to print outputs or not.
```

Outputs:

```
regret: Cumulative regret as a running sum over regret terms.
fractions: Fractions of pulls of different arms.
```

Code:

```
function [regret, fractions] = rungreedybanditlogistic(k, T, d, b, ...
    random_initialization, contexts, reward_vector, verbose)

warning('off','all');

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.

for t=1:T
    x = contexts(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.

    bx = b*x;
    [largest_inner_product, ~] = max(bx);
    bestreward = exp(largest_inner_product) / ...
        (1 + exp(largest_inner_product));
    ourreward = exp(bx(arm_pulled)) / (1 + exp(bx(arm_pulled)));

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

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

    obs_filt = find(pull_ind(:, arm_pulled)==1);  % Filter
 observations.
    lsX = contexts(obs_filt, :); % Design matrix.
    lsY = reward_vector(obs_filt, arm_pulled);  % Observations.
    if (size(lsX,1)>=d && rank(lsX)>=d)
        mdl = fitglm(lsX, lsY, 'linear','Distribution', ...
            'binomial', 'Intercept', false);
        betahat(arm_pulled ,:) = mdl.Coefficients.Estimate';
    end
```

```
    if (verbose==1)
        if (mod(t,500)==0)
            fprintf('GLM-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('GLM-GB: Total parameter estimation error = %f. \n', ...
        norm(b - betahat, 'fro'));
    fprintf('GLM-GB: Fraction of pulls = %f. \n', fractions);
    fprintf('GLM-GB: Total regret occured = %f. \n', regret(end));
end
end
```

Published with MATLAB® R2015a