Table of Contents

runOFUL.m

```
% Runs OFUL algorithm and returns regret and fraction of pulls.
%
% This code runs the OFUL algorithm adapted to be used in our setting.
% For more information, on OFUL algorithm see the original paper:
%
% - https://papers.nips.cc/paper/4417-improved-algorithms-for-linear-
stochastic-bandits.pdf
%
% The idea is to map our contextual bandit problem into a linear
 bandit
% one. In particular, we can concatenate all the arm parameters to get
 a
% theta, parameter (of interest) in R^{k*d}. The k actions at time
% period t, with context x_t are given by:
% A_t = {[X_t, 0, 0, ..., 0], [0, X_t, 0, 0, ..., 0], ... ,
% [0, 0, ..., X_t]}. After having this mapping, the algorithm builds
% confidence sets around theta and picks the action (corresponds to
 the
% same arm) that maximizes the optimistic reward.

% Our implementation uses Sherman-Morisson formula for fast rank one
 update
% of arm parameters. As construction of confidence intervals requires
% the knowledge of noise parameters, if this parameter is not
 provided,
% we use observations to estimate such parameter.
%
```

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 (or subgaussianity parameter).
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.
```

```
lambda: Regularization penalty used in ridge estimation of algorithm.
delta: The probability that confidence intervals fail.
sigma_start: The noise parameter (or subgaussanity parameter) to start
with. If true sigma is provided, this parameter is not used.
use_true_sigma_e: Whether to use true noise parameter, sigma_e, for
    construction of confidence sets or not.
to_estimate_sigma_e: Whether to estimate sigma_e using observations.

 This is only effective if the true noise parameter is not provided.
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] = runOFUL(k, T, d, b, ...
    sigma_e, sigma_x, xmax, lambda, delta, ...
    sigma_start, use_true_sigma_e, to_estimate_sigma_e, verbose,
 varargin)

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.

% This is an upper bound on the l2-norm of contexts that will be
 updated as
% more samples are gathered.
L = 1;

% This is an upper bound on the l2-norm of the parameter (which is the
% concatenation of all arm parameters) and will be updated as more
 samples
% are gathered.
S = 1;

Vt_inverse = eye(k*d)/lambda;

if(use_true_sigma_e==1)
    sigmaHat = sigma_e;
else
    sigmaHat = sigma_start;
end

XtopY = zeros(k * d, 1);
residuals = zeros(1, T);

for t=1:T

    x = X(t,:)';
    % Radius of confidence set using confidence sets in Theorem 2 of
    % OFUL paper.  This is a slightly tighter version of the presented
 upp
    % upper bound. Using log(1+x)<= x we can deduce this.
    radius_t = sigmaHat * sqrt((k * d) * log(1 + (t-1) * L^2 / ...
        (lambda * k * d)) - 2 * log(delta)) + sqrt(lambda) * S;

    %------ First, choose which arm we should pull in this round
    % Calculate the optimistic reward for each action (arm).
    optimism_amount = zeros(k,1);
    for i=1:k
        optimism_amount(i) = x' * Vt_inverse(((i-1) * d + 1):(i *
 d), ...
            ((i-1) * d + 1):(i * d)) * x;
    end
    optimistic_reward = betahat*x +  radius_t * sqrt(optimism_amount);

    [~, imax] = max(optimistic_reward);
    arm_pulled = imax;  % Optimistic arm selected at time t.
```

```
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.

% First updating Vt_inverse.
new_concat_obs = zeros(k * d, 1);
new_concat_obs( ((arm_pulled-1) * d + 1):(arm_pulled * d) ) = x;

% This is the rank one update using Sherman-Morrison formula.
Vt_inverse_Ut = Vt_inverse * new_concat_obs;
Vt_inverse = Vt_inverse - (Vt_inverse_Ut * Vt_inverse_Ut') / ...
    (1 + new_concat_obs' * Vt_inverse_Ut);

% Observe the reward.
if(noise_input==1)
    reward_vector(t, arm_pulled) = ourreward + e(t);
else
    reward_vector(t, arm_pulled) = rewards(t, arm_pulled);
end

% Update the estimator.
XtopY = XtopY + new_concat_obs * reward_vector(t, arm_pulled);
betahat_concat = Vt_inverse*XtopY;
betahat = (reshape(betahat_concat,d,k))';

% update parameters L and S and sigmaHat
S = max(S, sqrt(sum(betahat_concat'.^2)));
L = max(L, norm(x, 2));
residuals(t) = reward_vector(t, arm_pulled) ...
    - betahat_concat'*new_concat_obs;

if(to_estimate_sigma_e == 1)
    if (t>=k*d+1)
        sigmaHat_est = sqrt(sum(residuals.^2)/(t-d));
        if(use_true_sigma_e == 1)
            sigmaHat = sigma_e;
        else
```

```
                sigmaHat = sigmaHat_est;
            end
        end
    end

    if(verbose==1)
        if (mod(t,500)==0)
            fprintf('OFUL: t=%d, sigma_e est. = %f, sigma_e = %f.
 \n', ...
                t, sigmaHat, sigma_e);
        end
    end

end

fractions = mean(pull_ind);  %fraction of times each arm is pulled
if(verbose == 1)
    fprintf('OFUL: Error in estimation = %f. \n', norm(b-betahat,2));
    fprintf('OFUL: Fraction of pulls = %f. \n', fractions);
    fprintf('OFUL: Total regret occured = %f. \n', regret(end));
end

end
```

Published with MATLAB® R2015a