Master Engineering Systems

Big data
&
Small data

wk3 – linear regression assignment

linear regression assignment gradient descent

- Update thetas, loop over thetas

```
function [theta, cost_h] = mvgd_doubleloop2(X, y, theta, alpha, num_iters)
%MVGD Performs multi-variable gradient descent to learn theta
%   theta = MVGD(x, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

    %% Initialize some useful values
    m = length(y);                    %Number of training examples
    n = length(theta);
    cost_h = zeros(num_iters, 1);     %Preallocate for speed
    %% Perform gradient descent
    for iter = 1:num_iters           %loop for iterations
        hypothesis = X*theta;
        gradient = (1/m)*X'*(hypothesis-y); % compute gradient
        for index_theta = 1:n        %loop over thetas
            %update theta
            theta(index_theta) = theta(index_theta) - alpha*gradient(index_theta);
        end
        % Save the cost J in every iteration
        cost_h(iter) = 1/(2*m)*sum((X*theta-y).^2);
    end
end
```

linear regression assignment gradient descent

- Update thetas, loop over thetas
- Vectorized, update all thetas in one line

```
function [theta, cost_h] = mvgd(X, y, theta, alpha, num_iters)
%MVGD Performs multi-variable gradient descent to learn theta
%   theta = MVGD(x, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

    %% Initialize some useful values
    m = length(y);                  %Number of training examples
    cost_h = zeros(num_iters, 1);   %Preallocate for speed
    %% Perform gradient descent
    for iter = 1:num_iters
        gradient = (1/m)*X'*(X*theta-y);
        % Perform a single gradient step on the parameter vector
        theta = theta - alpha*gradient;
        % Save the cost J in every iteration
        cost_h(iter) = 1/(2*m)*sum((X*theta-y).^2);
    end
end
```

linear regression assignment gradient descent

- Update thetas, loop over thetas
- Vectorized, upda
- Stopping criteriu
  - Fixed number of it
  - Check for converge

```matlab
function [theta, cost_h] = mvgd2(X, y, theta, alpha, num_iters)
%MVGD Performs multi-variable gradient descent to learn theta
%   theta = MVGD(x, y, theta, alpha, num_iters) updates theta by
%   taking at most num_iters gradient steps with learning rate alpha
%   Stops when the improvement in the cost function is below eps = 10e-10

    %% Initialize some useful values
    m = length(y);                  %Number of training examples
    cost_h = zeros(num_iters, 1);   %Preallocate for speed
    %% Perform gradient descent
    iter=1;
    delta_cost=1;    % just an initial value to make sure it passes the first check
    eps = 10e-12;    % computer precission (~10^-16)
    while iter <= num_iters && delta_cost>eps
        gradient = 1/m*X'*(X*theta-y);
        % Perform a single gradient step on the parameter vector
        theta = theta - alpha*gradient;
        % Save the cost J in every iteration
        cost_h(iter) = 1/(2*m)*sum((X*theta-y).^2);
        % Check for improvement in the cost
        if iter>1
            delta_cost = cost_h(iter-1)-cost_h(iter);
        end
        iter = iter +1;
    end
    %remove trailing zeros from cost history
    cost_h = cost_h(1:iter-1);
end
```

gradient descent
weights and bias

```
function [weights, bias, cost_h] = mvgd_weights_bias(X_nobias, y, weights, bias, alpha, num_iters)
%MVGD Performs multi-variable gradient descent to learn theta
%   theta = MVGD(x, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

    %% Initialize some useful values

    m = length(y);                      %Number of training examples
    cost_h = zeros(num_iters, 1);       %Preallocate for speed
    %% Perform gradient descent
    for iter = 1:num_iters
        hypothesis = X_nobias*weights +bias*ones(m,1);

        % compute the gradient
        gradient_w = (1/m)*X_nobias'*(hypothesis-y);
        gradient_b = (1/m)*sum(hypothesis-y);

        % Perform a single gradient step on the parameter vector and the
        % bias
        weights = weights - alpha*gradient_w;
        bias = bias - alpha*gradient_b;

        % Save the cost J in every iteration
        hypothesis = X_nobias*weights +bias*ones(m,1);
        cost_h(iter) = 1/(2*m)*sum((hypothesis-y).^2);
    end
end
```

- Separate gradient and update for weights and bias

linear regression assignment normalization

- Shift with mean
- Scale with standard deviation or range
- A vectorized version
- What can go wrong?
  - Constant feature leads to division by zero
  - Do not include the bias vector in X

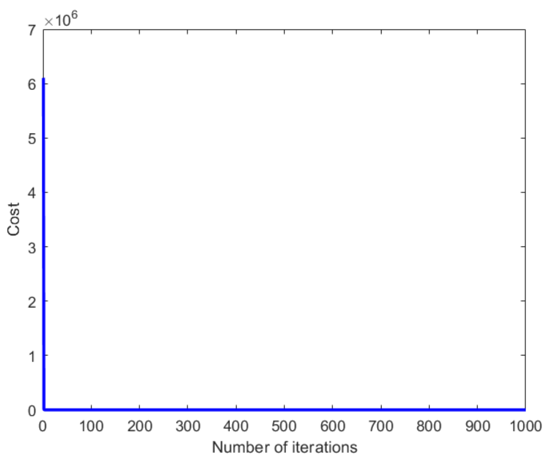
```
function [X,mu,sigma] = normalizeFeatures(X)

mu=zeros(1,size(X,2)); % creating a row of zeros
sigma=zeros(1,size(X,2)); % creating a row of zeros

tic
for i=1:size(mu,2)
    mu(1,i)=mean(X(:,i)); % iterative calculation of mu
    sigma(1,i)=std(X(:,i)); % iterative calculationof sigma
    X(:,i)=(X(:,i)-mu(1,i))/sigma(1,i); % iterative normaliz
end
toc
```

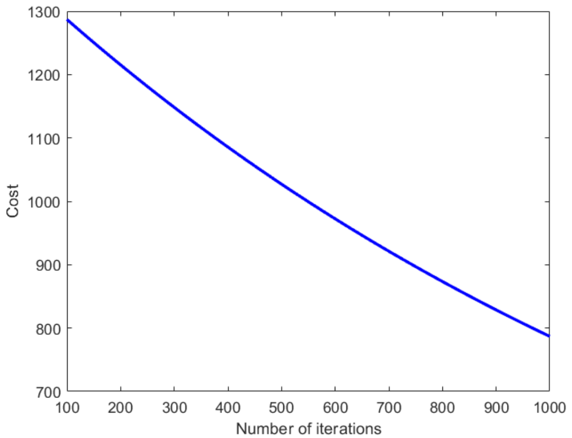
```
function [X, mu, sigma] = normalizeFeatures(X)
%Preparing parameters
mu = mean(X);          %
sigma=max(X);          %

%Data normalization
X = (X-mu)./sigma;     %
end
```

std (x)

linear regression assignment finding learning rate and number of iterations

- Non-normalized data
  - Features with ranges in different orders of magnitude
    - $X^T X$ eigenvalues $\sim 0.5 \ - 3.8 \cdot 10^9$
    - Slope very high in one direction, very low in other
    - Computational problem
  - Very small learning rate, many iterations
  - How to check for convergence?
    - Plot may be misleading





compare normalized and non-normalized

- normalized:
  - alpha = 0.3
  - iterations = ~1000
  - $\widehat{\theta}$ = (23.45, -0.8415, 2.082, -0.6524, -5.499, 0.2223, 2.766, 1.149)’
  - J($\widehat{\theta}$) = 5.424

- non-normalized
  - alpha <= $2 \cdot 10^{-7}$
  - iterations >= 10^7
  - $\widehat{\theta}$ = (0.9280, 0.2528, 0.0059, -0.0369, -0.0062, -0.0927, 0.5552, 1.0695)’
  - J($\widehat{\theta}$) = 5.774

direct vs gradient descent

- normalized data:
  - direct: 0.0005 s
  - gradient descent: 0.03 s
  - theta values ($\widehat{\theta}$) are nearly the same:
    - $\widehat{\theta}_d = (23.446, -0.842, 2.082, -0.653, -5.499, 0.222, 2.766, 1.149)'$
    - $\widehat{\theta}_g = (23.446, -0.841, 2.081, -0.652, -5.499, 0.222, 2.766, 1.149)'$

- non-normalized data:
  - direct: 0.0005 s
  - gradient descent: ~100 s (10^7 iterations)
  - theta values ($\widehat{\theta}$) are completely different:
    - $\widehat{\theta}_d = (-17.2, -0.49, 0.020, -0.017, -0.0065, 0.081, 0.75, 1.43)'$
    - $\widehat{\theta}_g = (0.93, 0.25, 0.0059, -0.037, -0.0062, -0.093, 0.56, 1.07)'$