MACHINE LEARNING

ANUJIT DATTA

WEEK 2

### warmUpExercise.m :

**function** A = warmUpExercise()

*%WARMUPEXERCISE Example function in octave*

*% A = WARMUPEXERCISE() is an example function that returns the 5x5 identity matrix*

A = [];

*% ============= YOUR CODE HERE ==============*

*% Instructions: Return the 5x5 identity matrix*

*% In octave, we return values by defining which variables*

*% represent the return values (at the top of the file)*

*% and then set them accordingly.*

A = eye(5); *%It's a built-in function to create identity matrix*

*% ===========================================*

**end**

### plotData.m :

**function** plotData(x, y)

*%PLOTDATA Plots the data points x and y into a new figure*

*% PLOTDATA(x,y) plots the data points and gives the figure axes labels of*

*% population and profit.*

figure; *% open a new figure window*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Plot the training data into a figure using the*

*% "figure" and "plot" commands. Set the axes labels using*

*% the "xlabel" and "ylabel" commands. Assume the*

*% population and revenue data have been passed in*

*% as the x and y arguments of this function.*

*%*

*% Hint: You can use the 'rx' option with plot to have the markers*

*% appear as red crosses. Furthermore, you can make the*

*% markers larger by using plot(..., 'rx', 'MarkerSize', 10);*

plot(x, y, 'rx', 'MarkerSize', 10); *% Plot the data*

ylabel('Profit in $10,000s'); *% Set the y-axis label*

xlabel('Population of City in 10,000s'); *% Set the x-axis label*

*% ============================================================*

**end**

### computeCost.m :

**function** J = computeCost(X, y, theta)

*%COMPUTECOST Compute cost for linear regression*

*% J = COMPUTECOST(X, y, theta) computes the cost of using theta as the*

*% parameter for linear regression to fit the data points in X and y*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta*

*% You should set J to the cost.*

*%%%%%%%%%%%%% CORRECT %%%%%%%%%*

*% h = X\*theta;*

*% temp = 0;*

*% for i=1:m*

*% temp = temp + (h(i) - y(i))^2;*

*% end*

*% J = (1/(2\*m)) \* temp;*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%%%%%%%%% CORRECT: Vectorized Implementation %%%%%%%%%*

J = (1/(2\*m))\*sum(((X\*theta)-y).^2);

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% =========================================================================*

**end**

### GradientDescent.m :

**function** [theta, J\_history] = gradientDescent(X, y, theta, alpha, num\_iters)

*%GRADIENTDESCENT Performs gradient descent to learn theta*

*% theta = GRADIENTDESCENT(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*

J\_history = zeros(num\_iters, 1);

**for** iter = 1:num\_iters

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Perform a single gradient step on the parameter vector*

*% theta.*

*%*

*% Hint: While debugging, it can be useful to print out the values*

*% of the cost function (computeCost) and gradient here.*

*%*

*%%%%%%%%% CORRECT %%%%%%%*

*%error = (X \* theta) - y;*

*%temp0 = theta(1) - ((alpha/m) \* sum(error .\* X(:,1)));*

*%temp1 = theta(2) - ((alpha/m) \* sum(error .\* X(:,2)));*

*%theta = [temp0; temp1];*

*%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%%%%% CORRECT %%%%%%%*

*%error = (X \* theta) - y;*

*%temp0 = theta(1) - ((alpha/m) \* X(:,1)'\*error);*

*%temp1 = theta(2) - ((alpha/m) \* X(:,2)'\*error);*

*%theta = [temp0; temp1];*

*%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%%%%% CORRECT %%%%%%%*

error = (X \* theta) - y;

theta = theta - ((alpha/m) \* X'\*error);

*%%%%%%%%%%%%%%%%%%%%%%%%%*

*% ============================================================*

*% Save the cost J in every iteration*

J\_history(iter) = computeCost(X, y, theta);

**end**

**end**

### computeCostMulti.m :

**function** J = computeCostMulti(X, y, theta)

*%COMPUTECOSTMULTI Compute cost for linear regression with multiple variables*

*% J = COMPUTECOSTMULTI(X, y, theta) computes the cost of using theta as the*

*% parameter for linear regression to fit the data points in X and y*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta*

*% You should set J to the cost.*

J = (1/(2\*m))\*(sum(((X\*theta)-y).^2));

*% =========================================================================*

**end**

### computeCostMulti.m :

**function** J = computeCostMulti(X, y, theta)

*%COMPUTECOSTMULTI Compute cost for linear regression with multiple variables*

*% J = COMPUTECOSTMULTI(X, y, theta) computes the cost of using theta as the*

*% parameter for linear regression to fit the data points in X and y*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta*

*% You should set J to the cost.*

J = (1/(2\*m))\*(sum(((X\*theta)-y).^2));

*% =========================================================================*

**end**

### gradientDescentMulti.m :

**function** [theta, J\_history] = gradientDescentMulti(X, y, theta, alpha, num\_iters)

*%GRADIENTDESCENTMULTI Performs gradient descent to learn theta*

*% theta = GRADIENTDESCENTMULTI(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*

J\_history = zeros(num\_iters, 1);

**for** iter = 1:num\_iters

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Perform a single gradient step on the parameter vector*

*% theta.*

*%*

*% Hint: While debugging, it can be useful to print out the values*

*% of the cost function (computeCostMulti) and gradient here.*

*%*

*%%%%%%%% CORRECT %%%%%%%%%%*

error = (X \* theta) - y;

theta = theta - ((alpha/m) \* X'\*error);

*%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% ============================================================*

*% Save the cost J in every iteration*

J\_history(iter) = computeCostMulti(X, y, theta);

**end**

**end**

WEEK 3

### plotData.m :

**function** plotData(X, y)

*%PLOTDATA Plots the data points X and y into a new figure*

*% PLOTDATA(x,y) plots the data points with + for the positive examples*

*% and o for the negative examples. X is assumed to be a Mx2 matrix.*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Plot the positive and negative examples on a*

*% 2D plot, using the option 'k+' for the positive*

*% examples and 'ko' for the negative examples.*

*%*

*%Seperating positive and negative results*

pos = find(y==1); *%index of positive results*

neg = find(y==0); *%index of negative results*

*% Create New Figure*

figure;

*%Plotting Positive Results on*

*% X\_axis: Exam1 Score = X(pos,1)*

*% Y\_axis: Exam2 Score = X(pos,2)*

plot(X(pos,1),X(pos,2),'g+');

*%To keep above plotted graph as it is.*

hold on;

*%Plotting Negative Results on*

*% X\_axis: Exam1 Score = X(neg,1)*

*% Y\_axis: Exam2 Score = X(neg,2)*

plot(X(neg,1),X(neg,2),'ro');

*% =========================================================================*

hold off;

**end**

### sigmoid.m :

**function** g = sigmoid(z)

*%SIGMOID Compute sigmoid function*

*% g = SIGMOID(z) computes the sigmoid of z.*

*% You need to return the following variables correctly*

g = zeros(size(z));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the sigmoid of each value of z (z can be a matrix,*

*% vector or scalar).*

g = 1./(1+exp(-z));

*% =============================================================*

**end**

### costFunction.m :

**function** [J, grad] = costFunction(theta, X, y)

*%COSTFUNCTION Compute cost and gradient for logistic regression*

*% J = COSTFUNCTION(theta, X, y) computes the cost of using theta as the*

*% parameter for logistic regression and the gradient of the cost*

*% w.r.t. to the parameters.*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

grad = zeros(size(theta));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta.*

*% You should set J to the cost.*

*% Compute the partial derivatives and set grad to the partial*

*% derivatives of the cost w.r.t. each parameter in theta*

*%*

*% Note: grad should have the same dimensions as theta*

*%*

*%DIMENSIONS:*

*% theta = (n+1) x 1*

*% X = m x (n+1)*

*% y = m x 1*

*% grad = (n+1) x 1*

*% J = Scalar*

z = X \* theta; *% m x 1*

h\_x = sigmoid(z); *% m x 1*

J = (1/m)\*sum((-y.\*log(h\_x))-((1-y).\*log(1-h\_x))); *% scalar*

grad = (1/m)\* (X'\*(h\_x-y)); *% (n+1) x 1*

*% =============================================================*

**end**

### predict.m :

**function** p = predict(theta, X)

*%PREDICT Predict whether the label is 0 or 1 using learned logistic*

*%regression parameters theta*

*% p = PREDICT(theta, X) computes the predictions for X using a*

*% threshold at 0.5 (i.e., if sigmoid(theta'\*x) >= 0.5, predict 1)*

m = size(X, 1); *% Number of training examples*

*% You need to return the following variables correctly*

p = zeros(m, 1);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Complete the following code to make predictions using*

*% your learned logistic regression parameters.*

*% You should set p to a vector of 0's and 1's*

*%*

*% Dimentions:*

*% X = m x (n+1)*

*% theta = (n+1) x 1*

h\_x = sigmoid(X\*theta);

p=(h\_x>=0.5);

*%p = double(sigmoid(X \* theta)>=0.5);*

*% =========================================================================*

**end**

### costFunctionReg.m :

**function** [J, grad] = costFunctionReg(theta, X, y, lambda)

*%COSTFUNCTIONREG Compute cost and gradient for logistic regression with regularization*

*% J = COSTFUNCTIONREG(theta, X, y, lambda) computes the cost of using*

*% theta as the parameter for regularized logistic regression and the*

*% gradient of the cost w.r.t. to the parameters.*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

grad = zeros(size(theta));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta.*

*% You should set J to the cost.*

*% Compute the partial derivatives and set grad to the partial*

*% derivatives of the cost w.r.t. each parameter in theta*

*%DIMENSIONS:*

*% theta = (n+1) x 1*

*% X = m x (n+1)*

*% y = m x 1*

*% grad = (n+1) x 1*

*% J = Scalar*

z = X \* theta; *% m x 1*

h\_x = sigmoid(z); *% m x 1*

reg\_term = (lambda/(2\*m)) \* sum(theta(2:**end**).^2);

J = (1/m)\*sum((-y.\*log(h\_x))-((1-y).\*log(1-h\_x))) + reg\_term; *% scalar*

grad(1) = (1/m)\* (X(:,1)'\*(h\_x-y)); *% 1 x 1*

grad(2:**end**) = (1/m)\* (X(:,2:**end**)'\*(h\_x-y))+(lambda/m)\*theta(2:**end**); *% n x 1*

*% =============================================================*

**end**

WEEK 4

### **lrCostFunction.m :**

**function** [J, grad] = lrCostFunction(theta, X, y, lambda)

*%LRCOSTFUNCTION Compute cost and gradient for logistic regression with*

*%regularization*

*% J = LRCOSTFUNCTION(theta, X, y, lambda) computes the cost of using*

*% theta as the parameter for regularized logistic regression and the*

*% gradient of the cost w.r.t. to the parameters.*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

grad = zeros(size(theta));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost of a particular choice of theta.*

*% You should set J to the cost.*

*% Compute the partial derivatives and set grad to the partial*

*% derivatives of the cost w.r.t. each parameter in theta*

*%*

*% Hint: The computation of the cost function and gradients can be*

*% efficiently vectorized. For example, consider the computation*

*%*

*% sigmoid(X \* theta)*

*%*

*% Each row of the resulting matrix will contain the value of the*

*% prediction for that example. You can make use of this to vectorize*

*% the cost function and gradient computations.*

*%*

*% Hint: When computing the gradient of the regularized cost function,*

*% there're many possible vectorized solutions, but one solution*

*% looks like:*

*% grad = (unregularized gradient for logistic regression)*

*% temp = theta;*

*% temp(1) = 0; % because we don't add anything for j = 0*

*% grad = grad + YOUR\_CODE\_HERE (using the temp variable)*

*%*

*%DIMENSIONS:*

*% theta = (n+1) x 1*

*% X = m x (n+1)*

*% y = m x 1*

*% grad = (n+1) x 1*

*% J = Scalar*

z = X \* theta; *% m x 1*

h\_x = sigmoid(z); *% m x 1*

reg\_term = (lambda/(2\*m)) \* sum(theta(2:**end**).^2);

J = (1/m)\*sum((-y.\*log(h\_x))-((1-y).\*log(1-h\_x))) + reg\_term; *% scalar*

grad(1) = (1/m) \* (X(:,1)'\*(h\_x-y)); *% 1 x 1*

grad(2:**end**) = (1/m) \* (X(:,2:**end**)'\*(h\_x-y)) + (lambda/m)\*theta(2:**end**); *% n x 1*

*% =============================================================*

grad = grad(:);

**end**

### **oneVsAll.m :**

**function** [all\_theta] = oneVsAll(X, y, num\_labels, lambda)

*%ONEVSALL trains multiple logistic regression classifiers and returns all*

*%the classifiers in a matrix all\_theta, where the i-th row of all\_theta*

*%corresponds to the classifier for label i*

*% [all\_theta] = ONEVSALL(X, y, num\_labels, lambda) trains num\_labels*

*% logistic regression classifiers and returns each of these classifiers*

*% in a matrix all\_theta, where the i-th row of all\_theta corresponds*

*% to the classifier for label i*

*% num\_labels = No. of output classifier (Here, it is 10)*

*% Some useful variables*

m = size(X, 1); *% No. of Training Samples == No. of Images : (Here, 5000)*

n = size(X, 2); *% No. of features == No. of pixels in each Image : (Here, 400)*

*% You need to return the following variables correctly*

all\_theta = zeros(num\_labels, n + 1);

*%DIMENSIONS: num\_labels x (input\_layer\_size+1) == num\_labels x (no\_of\_features+1) == 10 x 401*

*%DIMENSIONS: X = m x input\_layer\_size*

*%Here, 1 row in X represents 1 training Image of pixel 20x20*

*% Add ones to the X data matrix*

X = [ones(m, 1) X]; *%DIMENSIONS: X = m x (input\_layer\_size+1) = m x (no\_of\_features+1)*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: You should complete the following code to train num\_labels*

*% logistic regression classifiers with regularization*

*% parameter lambda.*

*%*

*% Hint: theta(:) will return a column vector.*

*%*

*% Hint: You can use y == c to obtain a vector of 1's and 0's that tell you*

*% whether the ground truth is true/false for this* [*class*](http://i.viglink.com/?key=38f231f0fe8f44ddbf74fb49f3afeea4&insertId=973fdb84d678cf6d&type=H&mid=39197&exp=60%3ACI1C55A%3A8&libId=k8irhkcw010326gn000DAekoz692s&loc=https%3A%2F%2Fwww.apdaga.com%2F2018%2F06%2Fcoursera-machine-learning-week-4.html&v=1&iid=973fdb84d678cf6d&opt=true&optExTitle=false&out=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fochemnmr%2F&ref=https%3A%2F%2Fwww.apdaga.com%2F2020%2F01%2Fcoursera-machine-learning-all-weeks-solutions-assignment-quiz.html&title=Coursera%3A Machine Learning (Week 4) %5BAssignment Solution%5D - Andrew NG - APDaga DumpBox %3A The Thirst for Learning...&txt=<span>class<%2Fspan>)*.*

*%*

*% Note: For this assignment, we recommend using fmincg to optimize the cost*

*% function. It is okay to use a for-loop (for c = 1:num\_labels) to*

*% loop over the different classes.*

*%*

*% fmincg works similarly to fminunc, but is more efficient when we*

*% are dealing with large number of parameters.*

*%*

*% Example Code for fmincg:*

*%*

*% % Set Initial theta*

*% initial\_theta = zeros(n + 1, 1);*

*%*

*% % Set options for fminunc*

*% options = optimset('GradObj', 'on', 'MaxIter', 50);*

*%*

*% % Run fmincg to obtain the optimal theta*

*% % This function will return theta and the cost*

*% [theta] = ...*

*% fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), ...*

*% initial\_theta, options);*

*%*

initial\_theta = zeros(n+1, 1);

options = optimset('GradObj', 'on', 'MaxIter', 50);

**for** c=1:num\_labels

all\_theta(c,:) = *...*

fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), *...*

initial\_theta, options);

**end**

*% =========================================================================*

**end**

### **predictOneVsAll.m :**

**function** p = predictOneVsAll(all\_theta, X)

*%PREDICT Predict the label for a trained one-vs-all classifier. The labels*

*%are in the range 1..K, where K = size(all\_theta, 1).*

*% p = PREDICTONEVSALL(all\_theta, X) will return a vector of predictions*

*% for each example in the matrix X. Note that X contains the examples in*

*% rows. all\_theta is a matrix where the i-th row is a trained logistic*

*% regression theta vector for the i-th* [*class*](http://i.viglink.com/?key=38f231f0fe8f44ddbf74fb49f3afeea4&insertId=973fdb84d678cf6d&type=H&mid=39197&exp=60%3ACI1C55A%3A8&libId=k8irhkcw010326gn000DAekoz692s&loc=https%3A%2F%2Fwww.apdaga.com%2F2018%2F06%2Fcoursera-machine-learning-week-4.html&v=1&iid=973fdb84d678cf6d&opt=true&optExTitle=false&out=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fochemnmr%2F&ref=https%3A%2F%2Fwww.apdaga.com%2F2020%2F01%2Fcoursera-machine-learning-all-weeks-solutions-assignment-quiz.html&title=Coursera%3A Machine Learning (Week 4) %5BAssignment Solution%5D - Andrew NG - APDaga DumpBox %3A The Thirst for Learning...&txt=<span>class<%2Fspan>)*. You should set p to a vector*

*% of values from 1..K (e.g., p = [1; 3; 1; 2] predicts classes 1, 3, 1, 2*

*% for 4 examples)*

m = size(X, 1); *% No. of Input Examples to Predict (Each row = 1 Example)*

num\_labels = size(all\_theta, 1); *%No. of Ouput Classifier*

*% You need to return the following variables correctly*

p = zeros(size(X, 1), 1); *% No\_of\_Input\_Examples x 1 == m x 1*

*% Add ones to the X data matrix*

X = [ones(m, 1) X];

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Complete the following code to make predictions using*

*% your learned logistic regression parameters (one-vs-all).*

*% You should set p to a vector of predictions (from 1 to*

*% num\_labels).*

*%*

*% Hint: This code can be done all vectorized using the max function.*

*% In particular, the max function can also return the index of the*

*% max element, for more information see 'help max'. If your examples*

*% are in rows, then, you can use max(A, [], 2) to obtain the max*

*% for each row.*

*%*

*% num\_labels = No. of output classifier (Here, it is 10)*

*% DIMENSIONS:*

*% all\_theta = 10 x 401 = num\_labels x (input\_layer\_size+1) == num\_labels x (no\_of\_features+1)*

prob\_mat = X \* all\_theta'; *% 5000 x 10 == no\_of\_input\_image x num\_labels*

[prob, p] = max(prob\_mat,[],2); *% m x 1*

*%returns maximum element in each row == max. probability and its index for each input image*

*%p: predicted output (index)*

*%prob: probability of predicted output*

*%%%%%%%% WORKING: Computation per input image %%%%%%%%%*

*% for i = 1:m % To iterate through each input sample*

*% one\_image = X(i,:); % 1 x 401 == 1 x no\_of\_features*

*% prob\_mat = one\_image \* all\_theta'; % 1 x 10 == 1 x num\_labels*

*% [prob, out] = max(prob\_mat);*

*% %out: predicted output*

*% %prob: probability of predicted output*

*% p(i) = out;*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%%%% WORKING %%%%%%%%%*

*% for i = 1:m*

*% RX = repmat(X(i,:),num\_labels,1);*

*% RX = RX .\* all\_theta;*

*% SX = sum(RX,2);*

*% [val, index] = max(SX);*

*% p(i) = index;*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% =========================================================================*

**end**

### **predict.m :**

**function** p = predict(Theta1, Theta2, X)

*%PREDICT Predict the label of an input given a trained neural network*

*% p = PREDICT(Theta1, Theta2, X) outputs the predicted label of X given the*

*% trained weights of a neural network (Theta1, Theta2)*

*% Useful values*

m = size(X, 1);

num\_labels = size(Theta2, 1);

*% You need to return the following variables correctly*

p = zeros(size(X, 1), 1); *% m x 1*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Complete the following code to make predictions using*

*% your learned neural network. You should set p to a*

*% vector containing labels between 1 to num\_labels.*

*%*

*% Hint: The max function might come in useful. In particular, the max*

*% function can also return the index of the max element, for more*

*% information see 'help max'. If your examples are in rows, then, you*

*% can use max(A, [], 2) to obtain the max for each row.*

*%*

*%DIMENSIONS:*

*% theta1 = 25 x 401*

*% theta2 = 10 x 26*

*% layer1 (input) = 400 nodes + 1bias*

*% layer2 (hidden) = 25 nodes + 1bias*

*% layer3 (output) = 10 nodes*

*%*

*% theta dimensions = S\_(j+1) x ((S\_j)+1)*

*% theta1 = 25 x 401*

*% theta2 = 10 x 26*

*% theta1:*

*% 1st row indicates: theta corresponding to all nodes from layer1 connecting to for 1st node of layer2*

*% 2nd row indicates: theta corresponding to all nodes from layer1 connecting to for 2nd node of layer2*

*% and*

*% 1st Column indicates: theta corresponding to node1 from layer1 to all nodes in layer2*

*% 2nd Column indicates: theta corresponding to node2 from layer1 to all nodes in layer2*

*%*

*% theta2:*

*% 1st row indicates: theta corresponding to all nodes from layer2 connecting to for 1st node of layer3*

*% 2nd row indicates: theta corresponding to all nodes from layer2 connecting to for 2nd node of layer3*

*% and*

*% 1st Column indicates: theta corresponding to node1 from layer2 to all nodes in layer3*

*% 2nd Column indicates: theta corresponding to node2 from layer2 to all nodes in layer3*

a1 = [ones(m,1) X]; *% 5000 x 401 == no\_of\_input\_images x no\_of\_features % Adding 1 in X*

*%No. of rows = no. of input images*

*%No. of Column = No. of features in each image*

z2 = a1 \* Theta1'; *% 5000 x 25*

a2 = sigmoid(z2); *% 5000 x 25*

a2 = [ones(size(a2,1),1) a2]; *% 5000 x 26*

z3 = a2 \* Theta2'; *% 5000 x 10*

a3 = sigmoid(z3); *% 5000 x 10*

[prob, p] = max(a3,[],2);

*%returns maximum element in each row == max. probability and its index for each input image*

*%p: predicted output (index)*

*%prob: probability of predicted output*

*% =========================================================================*

**end**

WEEK 5

### **sigmoidGradient.m :**

**function** g = sigmoidGradient(z)

*%SIGMOIDGRADIENT returns the gradient of the sigmoid function*

*%evaluated at z*

*% g = SIGMOIDGRADIENT(z) computes the gradient of the sigmoid function*

*% evaluated at z. This should work regardless if z is a matrix or a*

*% vector. In particular, if z is a vector or matrix, you should return*

*% the gradient for each element.*

g = zeros(size(z));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the gradient of the sigmoid function evaluated at*

*% each value of z (z can be a matrix, vector or scalar).*

g = sigmoid(z).\*(1-sigmoid(z));

*% =============================================================*

**end**

### **randInitializeWeights.m :**

**function** W = randInitializeWeights(L\_in, L\_out)

*%RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L\_in*

*%incoming connections and L\_out outgoing connections*

*% W = RANDINITIALIZEWEIGHTS(L\_in, L\_out) randomly initializes the weights*

*% of a layer with L\_in incoming connections and L\_out outgoing*

*% connections.*

*%*

*% Note that W should be set to a matrix of size(L\_out, 1 + L\_in) as*

*% the first column of W handles the "bias" terms*

*%*

*% You need to return the following variables correctly*

W = zeros(L\_out, 1 + L\_in);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Initialize W randomly so that we break the symmetry while*

*% training the neural network.*

*%*

*% Note: The first column of W corresponds to the parameters for the bias unit*

*%*

*% epsilon\_init = 0.12;*

epsilon\_init = sqrt(6)/(sqrt(L\_in)+sqrt(L\_out));

W = - epsilon\_init + rand(L\_out, 1 + L\_in) \* 2 \* epsilon\_init ;

*% =========================================================================*

**end**

### **nnCostFunction.m :**

**function** [J, grad] = nnCostFunction(nn\_params, *...*

input\_layer\_size, *...*

hidden\_layer\_size, *...*

num\_labels, *...*

X, y, lambda)

*%NNCOSTFUNCTION Implements the neural network cost function for a two layer*

*%neural network which performs classification*

*% [J grad] = NNCOSTFUNCTON(nn\_params, hidden\_layer\_size, num\_labels, ...*

*% X, y, lambda) computes the cost and gradient of the neural network. The*

*% parameters for the neural network are "unrolled" into the vector*

*% nn\_params and need to be converted back into the weight matrices.*

*%*

*% The returned parameter grad should be a "unrolled" vector of the*

*% partial derivatives of the neural network.*

*%*

*% Reshape nn\_params back into the parameters Theta1 and Theta2, the weight matrices*

*% for our 2 layer neural network*

*% DIMENSIONS:*

*% Theta1 = 25 x 401*

*% Theta2 = 10 x 26*

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)), *...*

hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):**end**), *...*

num\_labels, (hidden\_layer\_size + 1));

*% Setup some useful variables*

m = size(X, 1);

*% You need to return the following variables correctly*

J = 0;

Theta1\_grad = zeros(size(Theta1)); *%25 x401*

Theta2\_grad = zeros(size(Theta2)); *%10 x 26*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: You should complete the code by working through the*

*% following parts.*

*%*

*% Part 1: Feedforward the neural network and return the cost in the*

*% variable J. After implementing Part 1, you can verify that your*

*% cost function computation is correct by verifying the cost*

*% computed in ex4.m*

*%*

*% Part 2: Implement the backpropagation algorithm to compute the gradients*

*% Theta1\_grad and Theta2\_grad. You should return the partial derivatives of*

*% the cost function with respect to Theta1 and Theta2 in Theta1\_grad and*

*% Theta2\_grad, respectively. After implementing Part 2, you can check*

*% that your implementation is correct by running checkNNGradients*

*%*

*% Note: The vector y passed into the function is a vector of labels*

*% containing values from 1..K. You need to map this vector into a*

*% binary vector of 1's and 0's to be used with the neural network*

*% cost function.*

*%*

*% Hint: We recommend implementing backpropagation using a for-loop*

*% over the training examples if you are implementing it for the*

*% first time.*

*%*

*% Part 3: Implement regularization with the cost function and gradients.*

*%*

*% Hint: You can implement this around the code for*

*% backpropagation. That is, you can compute the gradients for*

*% the regularization separately and then add them to Theta1\_grad*

*% and Theta2\_grad from Part 2.*

*%*

*%%%%%%%%%%% Part 1: Calculating J w/o Regularization %%%%%%%%%%%%%%%*

X = [ones(m,1), X]; *% Adding 1 as first column in X*

a1 = X; *% 5000 x 401*

z2 = a1 \* Theta1'; *% m x hidden\_layer\_size == 5000 x 25*

a2 = sigmoid(z2); *% m x hidden\_layer\_size == 5000 x 25*

a2 = [ones(size(a2,1),1), a2]; *% Adding 1 as first column in z = (Adding bias unit) % m x (hidden\_layer\_size + 1) == 5000 x 26*

z3 = a2 \* Theta2'; *% m x num\_labels == 5000 x 10*

a3 = sigmoid(z3); *% m x num\_labels == 5000 x 10*

h\_x = a3; *% m x num\_labels == 5000 x 10*

*%Converting y into vector of 0's and 1's for multi-class classification*

*%%%%% WORKING %%%%%*

*% y\_Vec = zeros(m,num\_labels);*

*% for i = 1:m*

*% y\_Vec(i,y(i)) = 1;*

*% end*

*%%%%%%%%%%%%%%%%%%%*

y\_Vec = (1:num\_labels)==y; *% m x num\_labels == 5000 x 10*

*%Costfunction Without regularization*

J = (1/m) \* sum(sum((-y\_Vec.\*log(h\_x))-((1-y\_Vec).\*log(1-h\_x)))); *%scalar*

*%%%%%%%%%%% Part 2: Implementing Backpropogation for Theta\_gra w/o Regularization %%%%%%%%%%%%%*

*%%%%%%% WORKING: Backpropogation using for loop %%%%%%%*

*% for t=1:m*

*% % Here X is including 1 column at begining*

*%*

*% % for layer-1*

*% a1 = X(t,:)'; % (n+1) x 1 == 401 x 1*

*%*

*% % for layer-2*

*% z2 = Theta1 \* a1; % hidden\_layer\_size x 1 == 25 x 1*

*% a2 = [1; sigmoid(z2)]; % (hidden\_layer\_size+1) x 1 == 26 x 1*

*%*

*% % for layer-3*

*% z3 = Theta2 \* a2; % num\_labels x 1 == 10 x 1*

*% a3 = sigmoid(z3); % num\_labels x 1 == 10 x 1*

*%*

*% yVector = (1:num\_labels)'==y(t); % num\_labels x 1 == 10 x 1*

*%*

*% %calculating delta values*

*% delta3 = a3 - yVector; % num\_labels x 1 == 10 x 1*

*%*

*% delta2 = (Theta2' \* delta3) .\* [1; sigmoidGradient(z2)]; % (hidden\_layer\_size+1) x 1 == 26 x 1*

*%*

*% delta2 = delta2(2:end); % hidden\_layer\_size x 1 == 25 x 1 %Removing delta2 for bias node*

*%*

*% % delta\_1 is not calculated because we do not associate error with the input*

*%*

*% % CAPITAL delta update*

*% Theta1\_grad = Theta1\_grad + (delta2 \* a1'); % 25 x 401*

*% Theta2\_grad = Theta2\_grad + (delta3 \* a2'); % 10 x 26*

*%*

*% end*

*%*

*% Theta1\_grad = (1/m) \* Theta1\_grad; % 25 x 401*

*% Theta2\_grad = (1/m) \* Theta2\_grad; % 10 x 26*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%% WORKING: Backpropogation (Vectorized Implementation) %%%%%%%*

*% Here X is including 1 column at begining*

A1 = X; *% 5000 x 401*

Z2 = A1 \* Theta1'; *% m x hidden\_layer\_size == 5000 x 25*

A2 = sigmoid(Z2); *% m x hidden\_layer\_size == 5000 x 25*

A2 = [ones(size(A2,1),1), A2]; *% Adding 1 as first column in z = (Adding bias unit) % m x (hidden\_layer\_size + 1) == 5000 x 26*

Z3 = A2 \* Theta2'; *% m x num\_labels == 5000 x 10*

A3 = sigmoid(Z3); *% m x num\_labels == 5000 x 10*

*% h\_x = a3; % m x num\_labels == 5000 x 10*

y\_Vec = (1:num\_labels)==y; *% m x num\_labels == 5000 x 10*

DELTA3 = A3 - y\_Vec; *% 5000 x 10*

DELTA2 = (DELTA3 \* Theta2) .\* [ones(size(Z2,1),1) sigmoidGradient(Z2)]; *% 5000 x 26*

DELTA2 = DELTA2(:,2:**end**); *% 5000 x 25 %Removing delta2 for bias node*

Theta1\_grad = (1/m) \* (DELTA2' \* A1); *% 25 x 401*

Theta2\_grad = (1/m) \* (DELTA3' \* A2); *% 10 x 26*

*%%%%%%%%%%%% WORKING: DIRECT CALCULATION OF THETA GRADIENT WITH REGULARISATION %%%%%%%%%%%*

*% %Regularization term is later added in Part 3*

*% Theta1\_grad = (1/m) \* Theta1\_grad + (lambda/m) \* [zeros(size(Theta1, 1), 1) Theta1(:,2:end)]; % 25 x 401*

*% Theta2\_grad = (1/m) \* Theta2\_grad + (lambda/m) \* [zeros(size(Theta2, 1), 1) Theta2(:,2:end)]; % 10 x 26*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%%%%%%%%%%%% Part 3: Adding Regularisation term in J and Theta\_grad %%%%%%%%%%%%%*

reg\_term = (lambda/(2\*m)) \* (sum(sum(Theta1(:,2:**end**).^2)) + sum(sum(Theta2(:,2:**end**).^2))); *%scalar*

*%Costfunction With regularization*

J = J + reg\_term; *%scalar*

*%Calculating gradients for the regularization*

Theta1\_grad\_reg\_term = (lambda/m) \* [zeros(size(Theta1, 1), 1) Theta1(:,2:**end**)]; *% 25 x 401*

Theta2\_grad\_reg\_term = (lambda/m) \* [zeros(size(Theta2, 1), 1) Theta2(:,2:**end**)]; *% 10 x 26*

*%Adding regularization term to earlier calculated Theta\_grad*

Theta1\_grad = Theta1\_grad + Theta1\_grad\_reg\_term;

Theta2\_grad = Theta2\_grad + Theta2\_grad\_reg\_term;

*% -------------------------------------------------------------*

*% =========================================================================*

*% Unroll gradients*

grad = [Theta1\_grad(:) ; Theta2\_grad(:)];

**end**

WEEK 6

### **linearRegCostFunction.m :**

**function** [J, grad] = linearRegCostFunction(X, y, theta, lambda)

*%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear*

*%regression with multiple variables*

*% [J, grad] = LINEARREGCOSTFUNCTION(X, y, theta, lambda) computes the*

*% cost of using theta as the parameter for linear regression to fit the*

*% data points in X and y. Returns the cost in J and the gradient in grad*

*% Initialize some useful values*

m = length(y); *% number of training examples*

*% You need to return the following variables correctly*

J = 0;

grad = zeros(size(theta));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost and gradient of regularized linear*

*% regression for a particular choice of theta.*

*%*

*% You should set J to the cost and grad to the gradient.*

*%DIMENSIONS:*

*% X = 12x2 = m x 1*

*% y = 12x1 = m x 1*

*% theta = 2x1 = (n+1) x 1*

*% grad = 2x1 = (n+1) x 1*

h\_x = X \* theta; *% 12x1*

J = (1/(2\*m))\*sum((h\_x - y).^2) + (lambda/(2\*m))\*sum(theta(2:**end**).^2); *% scalar*

*% grad(1) = (1/m)\*sum((h\_x-y).\*X(:,1)); % scalar == 1x1*

grad(1) = (1/m)\*(X(:,1)'\*(h\_x-y)); *% scalar == 1x1*

grad(2:**end**) = (1/m)\*(X(:,2:**end**)'\*(h\_x-y)) + (lambda/m)\*theta(2:**end**); *% n x 1*

*% =========================================================================*

grad = grad(:);

**end**

### **learningCurve.m :**

**function** [error\_train, error\_val] = *...*

learningCurve(X, y, Xval, yval, lambda)

*%LEARNINGCURVE Generates the train and cross validation set errors needed*

*%to plot a learning curve*

*% [error\_train, error\_val] = ...*

*% LEARNINGCURVE(X, y, Xval, yval, lambda) returns the train and*

*% cross validation set errors for a learning curve. In particular,*

*% it returns two vectors of the same length - error\_train and*

*% error\_val. Then, error\_train(i) contains the training error for*

*% i examples (and similarly for error\_val(i)).*

*%*

*% In this function, you will compute the train and test errors for*

*% dataset sizes from 1 up to m. In practice, when working with larger*

*% datasets, you might want to do this in larger intervals.*

*%*

*% Number of training examples*

m = size(X, 1);

*% You need to return these values correctly*

error\_train = zeros(m, 1);

error\_val = zeros(m, 1);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to return training errors in*

*% error\_train and the cross validation errors in error\_val.*

*% i.e., error\_train(i) and*

*% error\_val(i) should give you the errors*

*% obtained after training on i examples.*

*%*

*% Note: You should evaluate the training error on the first i training*

*% examples (i.e., X(1:i, :) and y(1:i)).*

*%*

*% For the cross-validation error, you should instead evaluate on*

*% the \_entire\_ cross validation set (Xval and yval).*

*%*

*% Note: If you are using your cost function (linearRegCostFunction)*

*% to compute the training and cross validation error, you should*

*% call the function with the lambda argument set to 0.*

*% Do note that you will still need to use lambda when running*

*% the training to obtain the theta parameters.*

*%*

*% Hint: You can loop over the examples with the following:*

*%*

*% for i = 1:m*

*% % Compute train/cross validation errors using training examples*

*% % X(1:i, :) and y(1:i), storing the result in*

*% % error\_train(i) and error\_val(i)*

*% ....*

*%*

*% end*

*%*

*% ---------------------- Sample Solution ----------------------*

*%DIMENSIONS:*

*% error\_train = m x 1*

*% error\_val = m x 1*

**for** i = 1:m

Xtrain = X(1:i,:);

ytrain = y(1:i);

theta = trainLinearReg(Xtrain, ytrain, lambda);

error\_train(i) = linearRegCostFunction(Xtrain, ytrain, theta, 0); *%for lambda = 0;*

error\_val(i) = linearRegCostFunction(Xval, yval, theta, 0); *%for lambda = 0;*

**end**

*% -------------------------------------------------------------*

*% =========================================================================*

**end**

### **polyFeatures.m :**

**function** [X\_poly] = polyFeatures(X, p)

*%POLYFEATURES Maps X (1D vector) into the p-th power*

*% [X\_poly] = POLYFEATURES(X, p) takes a data matrix X (size m x 1) and*

*% maps each example into its polynomial features where*

*% X\_poly(i, :) = [X(i) X(i).^2 X(i).^3 ... X(i).^p];*

*%*

*% You need to return the following variables correctly.*

X\_poly = zeros(numel(X), p); *% m x p*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Given a vector X, return a matrix X\_poly where the p-th*

*% column of X contains the values of X to the p-th power.*

*%*

*%*

*% Here, X does not include X0 == 1 column*

*%%%% WORKING: Using for loop %%%%%%*

*% for i = 1:p*

*% X\_poly(:,i) = X(:,1).^i;*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

X\_poly(:,1:p) = X(:,1).^(1:p); *% w/o for loop*

*% =========================================================================*

**end**

### **validationCurve.m :**

**function** [lambda\_vec, error\_train, error\_val] = *...*

validationCurve(X, y, Xval, yval)

*%VALIDATIONCURVE Generate the train and validation errors needed to*

*%plot a validation curve that we can use to select lambda*

*% [lambda\_vec, error\_train, error\_val] = ...*

*% VALIDATIONCURVE(X, y, Xval, yval) returns the train*

*% and validation errors (in error\_train, error\_val)*

*% for different values of lambda. You are given the training set (X,*

*% y) and validation set (Xval, yval).*

*%*

*% Selected values of lambda (you should not change this)*

lambda\_vec = [0 0.001 0.003 0.01 0.03 0.1 0.3 1 3 10]';

*% You need to return these variables correctly.*

error\_train = zeros(length(lambda\_vec), 1);

error\_val = zeros(length(lambda\_vec), 1);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to return training errors in*

*% error\_train and the validation errors in error\_val. The*

*% vector lambda\_vec contains the different lambda parameters*

*% to use for each calculation of the errors, i.e,*

*% error\_train(i), and error\_val(i) should give*

*% you the errors obtained after training with*

*% lambda = lambda\_vec(i)*

*%*

*% Note: You can loop over lambda\_vec with the following:*

*%*

*% for i = 1:length(lambda\_vec)*

*% lambda = lambda\_vec(i);*

*% % Compute train / val errors when training linear*

*% % regression with regularization parameter lambda*

*% % You should store the result in error\_train(i)*

*% % and error\_val(i)*

*% ....*

*%*

*% end*

*%*

*%*

*% Here, X & Xval are already including x0 i.e 1's column in it*

m = size(X, 1);

*%% %%%%% WORKING: BUT UNNECESSARY for loop for i is inovolved %%%%%%%%%%%*

*% for i = 1:m*

*% for j = 1:length(lambda\_vec);*

*% lambda = lambda\_vec(j);*

*% Xtrain = X(1:i,:);*

*% ytrain = y(1:i);*

*%*

*% theta = trainLinearReg(Xtrain, ytrain, lambda);*

*%*

*% error\_train(j) = linearRegCostFunction(Xtrain, ytrain, theta, 0); % lambda = 0;*

*% error\_val(j) = linearRegCostFunction(Xval, yval, theta, 0); % lambda = 0;*

*% end*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%% %%%%%%% WORKING: BUT UNNECESSARY for loop for i is inovolved %%%%%%%%%%%*

*% for j = 1:length(lambda\_vec)*

*% lambda = lambda\_vec(j);*

*% for i = 1:m*

*% Xtrain = X(1:i,:);*

*% ytrain = y(1:i);*

*%*

*% theta = trainLinearReg(Xtrain, ytrain, lambda);*

*%*

*% error\_train(j) = linearRegCostFunction(Xtrain, ytrain, theta, 0); % lambda = 0;*

*% error\_val(j) = linearRegCostFunction(Xval, yval, theta, 0); % lambda = 0;*

*% end*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%% %%% NOT WORKING: BUT UNNECESSARY for loop inside learningCurve function is inovolved %%%%%%*

*% for j = 1:length(lambda\_vec)*

*% lambda = lambda\_vec(j);*

*%*

*% [error\_train\_temp, error\_val\_temp] = ...*

*% learningCurve(X, y, ...*

*% Xval, yval, ...*

*% lambda);*

*%*

*% error\_train(j) = error\_train\_temp(end);*

*% error\_val(j) = error\_val\_temp(end);*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%% %%%%% WORKING: OPTIMISED (Only 1 for loop) %%%%%%%%%%%*

**for** j = 1:length(lambda\_vec)

lambda = lambda\_vec(j);

theta = trainLinearReg(X, y, lambda);

error\_train(j) = linearRegCostFunction(X, y, theta, 0); *% lambda = 0;*

error\_val(j) = linearRegCostFunction(Xval, yval, theta, 0); *% lambda = 0*

**end**

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% =========================================================================*

**end**

week 7

### **gaussianKernel.m :**

**function** sim = gaussianKernel(x1, x2, sigma)

*%RBFKERNEL returns a radial basis function kernel between x1 and x2*

*% sim = gaussianKernel(x1, x2) returns a gaussian kernel between x1 and x2*

*% and returns the value in sim*

*% Ensure that x1 and x2 are column vectors*

x1 = x1(:); x2 = x2(:);

*% You need to return the following variables correctly.*

sim = 0;

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to return the similarity between x1*

*% and x2 computed using a Gaussian kernel with bandwidth*

*% sigma*

*%*

*%*

sim = exp(-1\*sum(abs(x1-x2).^2)/(2\*sigma^2));

*% =============================================================*

**end**

### **emailFeatures.m :**

**function** x = emailFeatures(word\_indices)

*%EMAILFEATURES takes in a word\_indices vector and produces a feature vector*

*%from the word indices*

*% x = EMAILFEATURES(word\_indices) takes in a word\_indices vector and*

*% produces a feature vector from the word indices.*

*% Total number of words in the dictionary*

n = 1899;

*% You need to return the following variables correctly.*

x = zeros(n, 1);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to return a feature vector for the*

*% given email (word\_indices). To help make it easier to*

*% process the emails, we have have already pre-processed each*

*% email and converted each word in the email into an index in*

*% a fixed dictionary (of 1899 words). The variable*

*% word\_indices contains the list of indices of the words*

*% which occur in one email.*

*%*

*% Concretely, if an email has the text:*

*%*

*% The quick brown fox jumped over the lazy dog.*

*%*

*% Then, the word\_indices vector for this text might look*

*% like:*

*%*

*% 60 100 33 44 10 53 60 58 5*

*%*

*% where, we have mapped each word onto a number, for example:*

*%*

*% the -- 60*

*% quick -- 100*

*% ...*

*%*

*% (note: the above numbers are just an example and are not the*

*% actual mappings).*

*%*

*% Your task is take one such word\_indices vector and construct*

*% a binary feature vector that indicates whether a particular*

*% word occurs in the email. That is, x(i) = 1 when word i*

*% is present in the email. Concretely, if the word 'the' (say,*

*% index 60) appears in the email, then x(60) = 1. The feature*

*% vector should look like:*

*%*

*% x = [ 0 0 0 0 1 0 0 0 ... 0 0 0 0 1 ... 0 0 0 1 0 ..];*

*%*

*%*

*%% WORKING: SOLUTION 1 %%%%%%*

*% for i = 1:length(word\_indices)*

*% x1 = ([1:n] == word\_indices(i));*

*% x = x | x1';*

*% end*

*%% WORKING: SOLUTION 2 %%%%%%*

**for** i = 1:length(word\_indices)

x(word\_indices(i)) = 1;

**end**

*% =========================================================================*

**end**

### **dataset3Params.m :**

**function** [C, sigma] = dataset3Params(X, y, Xval, yval)

*%DATASET3PARAMS returns your choice of C and sigma for Part 3 of the exercise*

*%where you select the optimal (C, sigma) learning parameters to use for SVM*

*%with RBF kernel*

*% [C, sigma] = DATASET3PARAMS(X, y, Xval, yval) returns your choice of C and*

*% sigma. You should complete this function to return the optimal C and*

*% sigma based on a cross-validation set.*

*%*

*% You need to return the following variables correctly.*

C = 1;

sigma = 0.3;

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to return the optimal C and sigma*

*% learning parameters found using the cross validation set.*

*% You can use svmPredict to predict the labels on the cross*

*% validation set. For example,*

*% predictions = svmPredict(model, Xval);*

*% will return the predictions on the cross validation set.*

*%*

*% Note: You can compute the prediction error using*

*% mean(double(predictions ~= yval))*

*%*

*%% %%%%%%%%%% WORKING: SOLUTION1 %%%%%%%%%%*

*% C\_list = [0.01 0.03 0.1 0.3 1 3 10 30]';*

*% sigma\_list = [0.01 0.03 0.1 0.3 1 3 10 30]';*

*%*

*% prediction\_error = zeros(length(C\_list), length(sigma\_list));*

*% for i = 1:length(C\_list)*

*% for j = 1: length(sigma\_list)*

*% C\_test = C\_list(i);*

*% sigma\_test = sigma\_list(j);*

*% model = svmTrain(X, y, C\_test, @(x1, x2) gaussianKernel(x1, x2, sigma\_test));*

*% predictions = svmPredict(model, Xval);*

*% prediction\_error(i,j) = mean(double(predictions ~= yval));*

*% end*

*% end*

*%*

*% % Finding row and col corresponding to min(prediction\_error)*

*% [values, row\_index]=min(prediction\_error);*

*% [~ ,col] = min(values);*

*% row = row\_index(col);*

*%*

*% % C and sigma corresponding to min(prediction\_error)*

*% C = C\_list(row);*

*% sigma = sigma\_list(col);*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%% %%%%%%%%%% WORKING: SOLUION 2 %%%%%%%%%%%%%%*

C\_list = [0.01 0.03 0.1 0.3 1 3 10 30]';

sigma\_list = [0.01 0.03 0.1 0.3 1 3 10 30]';

prediction\_error = zeros(length(C\_list), length(sigma\_list));

result = zeros(length(C\_list)+length(sigma\_list),3);

row = 1;

**for** i = 1:length(C\_list)

**for** j = 1: length(sigma\_list)

C\_test = C\_list(i);

sigma\_test = sigma\_list(j);

model = svmTrain(X, y, C\_test, @(x1, x2) gaussianKernel(x1, x2, sigma\_test));

predictions = svmPredict(model, Xval);

prediction\_error(i,j) = mean(double(predictions ~= yval));

result(row,:) = [prediction\_error(i,j), C\_test, sigma\_test];

row = row + 1;

**end**

**end**

*% Sorting prediction\_error in ascending order*

sorted\_result = sortrows(result, 1);

*% C and sigma corresponding to min(prediction\_error)*

C = sorted\_result(1,2);

sigma = sorted\_result(1,3);

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% =========================================================================*

**end**

### **processEmail.m :**

**function** word\_indices = processEmail(email\_contents)

*%PROCESSEMAIL preprocesses a the body of an email and*

*%returns a list of word\_indices*

*% word\_indices = PROCESSEMAIL(email\_contents) preprocesses*

*% the body of an email and returns a list of indices of the*

*% words contained in the email.*

*%*

*% Load Vocabulary*

vocabList = getVocabList();

*% Init return value*

word\_indices = [];

*% ========================== Preprocess Email ===========================*

*% Find the Headers ( \n\n and remove )*

*% Uncomment the following lines if you are working with raw emails with the*

*% full headers*

*% hdrstart = strfind(email\_contents, ([char(10) char(10)]));*

*% email\_contents = email\_contents(hdrstart(1):end);*

*% Lower case*

email\_contents = lower(email\_contents);

*% Strip all HTML*

*% Looks for any expression that starts with < and ends with > and replace*

*% and does not have any < or > in the tag it with a space*

email\_contents = regexprep(email\_contents, '<[^<>]+>', ' ');

*% Handle Numbers*

*% Look for one or more characters between 0-9*

email\_contents = regexprep(email\_contents, '[0-9]+', 'number');

*% Handle URLS*

*% Look for strings starting with http:// or https://*

email\_contents = regexprep(email\_contents, *...*

'(http|https)://[^\s]\*', 'httpaddr');

*% Handle Email Addresses*

*% Look for strings with @ in the middle*

email\_contents = regexprep(email\_contents, '[^\s]+@[^\s]+', 'emailaddr');

*% Handle $ sign*

email\_contents = regexprep(email\_contents, '[$]+', 'dollar');

*% ========================== Tokenize Email ===========================*

*% Output the email to screen as well*

fprintf('\n==== Processed Email ====\n\n');

*% Process file*

l = 0;

**while** ~isempty(email\_contents)

*% Tokenize and also get rid of any punctuation*

[str, email\_contents] = *...*

strtok(email\_contents, *...*

[' @$/#.-:&\*+=[]?!(){},''">\_<;%' char(10) char(13)]);

*% Remove any non alphanumeric characters*

str = regexprep(str, '[^a-zA-Z0-9]', '');

*% Stem the word*

*% (the porterStemmer sometimes has issues, so we use a try catch block)*

**try** str = porterStemmer(strtrim(str));

**catch** str = ''; **continue**;

**end**;

*% Skip the word if it is too short*

**if** length(str) < 1

**continue**;

**end**

*% Look up the word in the dictionary and add to word\_indices if*

*% found*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Fill in this function to add the index of str to*

*% word\_indices if it is in the vocabulary. At this point*

*% of the code, you have a stemmed word from the email in*

*% the variable str. You should look up str in the*

*% vocabulary list (vocabList). If a match exists, you*

*% should add the index of the word to the word\_indices*

*% vector. Concretely, if str = 'action', then you should*

*% look up the vocabulary list to find where in vocabList*

*% 'action' appears. For example, if vocabList{18} =*

*% 'action', then, you should add 18 to the word\_indices*

*% vector (e.g., word\_indices = [word\_indices ; 18]; ).*

*%*

*% Note: vocabList{idx} returns a the word with index idx in the*

*% vocabulary list.*

*%*

*% Note: You can use strcmp(str1, str2) to compare two strings (str1 and*

*% str2). It will return 1 only if the two strings are equivalent.*

*%*

*%% %%%%% WORKING: SOLUTION %%%%%%%%%%*

*% find index of the word in vocabList (if Exist)*

index = find(strcmp(str,vocabList),1);

*% Add the index in the vector word\_indices*

word\_indices = [word\_indices; index];

*%% =============================================================*

*% Print to screen, ensuring that the output lines are not too long*

**if** (l + length(str) + 1) > 78

fprintf('\n');

l = 0;

**end**

fprintf('%s ', str);

l = l + length(str) + 1;

**end**

*% Print footer*

fprintf('\n\n=========================\n');

**end**

week 8

### **pca.m :**

**function** [U, S] = pca(X)

*%PCA Run principal component analysis on the dataset X*

*% [U, S, X] = pca(X) computes eigenvectors of the covariance matrix of X*

*% Returns the eigenvectors U, the eigenvalues (on diagonal) in S*

*%*

*% Useful values*

[m, n] = size(X);

*% You need to return the following variables correctly.*

U = zeros(n);

S = zeros(n);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: You should first compute the covariance matrix. Then, you*

*% should use the "svd" function to compute the eigenvectors*

*% and eigenvalues of the covariance matrix.*

*%*

*% Note: When computing the covariance matrix, remember to divide by m (the*

*% number of examples).*

*%*

*% DIMENSIONS:*

*% X = m x n*

Sigma = (1/m)\*(X'\*X); *% n x n*

[U, S, V] = svd(Sigma);

*% =========================================================================*

**end**

### **projectData.m :**

**function** Z = projectData(X, U, K)

*%PROJECTDATA Computes the reduced data representation when projecting only*

*%on to the top k eigenvectors*

*% Z = projectData(X, U, K) computes the projection of*

*% the normalized inputs X into the reduced dimensional space spanned by*

*% the first K columns of U. It returns the projected examples in Z.*

*%*

*% You need to return the following variables correctly.*

Z = zeros(size(X, 1), K);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the projection of the data using only the top K*

*% eigenvectors in U (first K columns).*

*% For the i-th example X(i,:), the projection on to the k-th*

*% eigenvector is given as follows:*

*% x = X(i, :)';*

*% projection\_k = x' \* U(:, k);*

*%*

*% DIMENSIONS:*

*% X = m x n*

*% U = n x n*

*% U\_reduce = n x K*

*% K = scalar*

U\_reduce = U(:,[1:K]); *% n x K*

Z = X \* U\_reduce; *% m x k*

*% =============================================================*

**end**

### **recoverData.m :**

**function** X\_rec = recoverData(Z, U, K)

*%RECOVERDATA Recovers an approximation of the original data when using the*

*%projected data*

*% X\_rec = RECOVERDATA(Z, U, K) recovers an approximation the*

*% original data that has been reduced to K dimensions. It returns the*

*% approximate reconstruction in X\_rec.*

*%*

*% You need to return the following variables correctly.*

X\_rec = zeros(size(Z, 1), size(U, 1));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the approximation of the data by projecting back*

*% onto the original space using the top K eigenvectors in U.*

*%*

*% For the i-th example Z(i,:), the (approximate)*

*% recovered data for dimension j is given as follows:*

*% v = Z(i, :)';*

*% recovered\_j = v' \* U(j, 1:K)';*

*%*

*% Notice that U(j, 1:K) is a row vector.*

*%*

*% DIMENSIONS:*

*% Z = m x K*

*% U = n x n*

*% U\_reduce = n x k*

*% K = scalar*

*% X\_rec = m x n*

U\_reduce = U(:,1:K); *% n x k*

X\_rec = Z \* U\_reduce'; *% m x n*

*% =============================================================*

**end**

### **findClosestCentroids.m :**

**function** idx = findClosestCentroids(X, centroids)

*%FINDCLOSESTCENTROIDS computes the centroid memberships for every example*

*% idx = FINDCLOSESTCENTROIDS (X, centroids) returns the closest centroids*

*% in idx for a dataset X where each row is a single example. idx = m x 1*

*% vector of centroid assignments (i.e. each entry in range [1..K])*

*%*

*% Set K*

K = size(centroids, 1); *% K x 1 == 3 x 1*

*% You need to return the following variables correctly.*

idx = zeros(size(X,1), 1); *% m x 1 == 300 x 1*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Go over every example, find its closest centroid, and store*

*% the index inside idx at the appropriate location.*

*% Concretely, idx(i) should contain the index of the centroid*

*% closest to example i. Hence, it should be a value in the*

*% range 1..K*

*%*

*% Note: You can use a for-loop over the examples to compute this.*

*%*

*% DIMENSIONS:*

*% centroids = K x no. of features = 3 x 2*

**for** i = 1:size(X,1)

temp = zeros(K,1);

**for** j = 1:K

temp(j)=sqrt(sum((X(i,:)-centroids(j,:)).^2));

**end**

[~,idx(i)] = min(temp);

**end**

*% =============================================================*

**end**

### **computeCentroids.m :**

**function** centroids = computeCentroids(X, idx, K)

*%COMPUTECENTROIDS returns the new centroids by computing the means of the*

*%data points assigned to each centroid.*

*% centroids = COMPUTECENTROIDS(X, idx, K) returns the new centroids by*

*% computing the means of the data points assigned to each centroid. It is*

*% given a dataset X where each row is a single data point, a vector*

*% idx of centroid assignments (i.e. each entry in range [1..K]) for each*

*% example, and K, the number of centroids. You should return a matrix*

*% centroids, where each row of centroids is the mean of the data points*

*% assigned to it.*

*%*

*% Useful variables*

[m n] = size(X);

*% You need to return the following variables correctly.*

centroids = zeros(K, n);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Go over every centroid and compute mean of all points that*

*% belong to it. Concretely, the row vector centroids(i, :)*

*% should contain the mean of the data points assigned to*

*% centroid i.*

*%*

*% Note: You can use a for-loop over the centroids to compute this.*

*%*

*% DIMENSIONS:*

*% X = m x n*

*% centroids = K x n*

*%% %%%%%% WORKING: SOLUTION1 %%%%%%%%%*

*% for i = 1:K*

*% idx\_i = find(idx==i); %indexes of all the input which belongs to cluster j*

*% centroids(i,:)=(1/length(idx\_i))\*sum(X(idx\_i,:)); %calculating mean manually*

*% end*

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*%% %%%%%% WORKING: SOLUTION 2 %%%%%%%%*

**for** i = 1:K

idx\_i = find(idx==i); *%indexes of all the input which belongs to cluster j*

centroids(i,:) = mean(X(idx\_i,:)); *% calculating mean using built-in function*

**end**

*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*% =============================================================*

**end**

### **kMeansInitCentroids.m :**

**function** centroids = kMeansInitCentroids(X, K)

*%KMEANSINITCENTROIDS This function initializes K centroids that are to be*

*%used in K-Means on the dataset X*

*% centroids = KMEANSINITCENTROIDS(X, K) returns K initial centroids to be*

*% used with the K-Means on the dataset X*

*%*

*% You should return this values correctly*

centroids = zeros(K, size(X, 2));

*% ====================== YOUR CODE HERE ======================*

*% Instructions: You should set centroids to randomly chosen examples from*

*% the dataset X*

*%*

*% Randomly reorder the indices of examples*

randidx = randperm(size(X, 1));

*% Take the first K examples as centroids*

centroids = X(randidx(1:K), :);

*% =============================================================*

**end**

week 9

### **estimateGaussian.m :**

**function** [mu sigma2] = estimateGaussian(X)

*%ESTIMATEGAUSSIAN This function estimates the parameters of a*

*%Gaussian distribution using the data in X*

*% [mu sigma2] = estimateGaussian(X),*

*% The input X is the dataset with each n-dimensional data point in one row*

*% The output is an n-dimensional vector mu, the mean of the* [*data set*](http://i.viglink.com/?key=38f231f0fe8f44ddbf74fb49f3afeea4&insertId=0fb9c44119456567&type=H&mid=6768&exp=60%3ACI1C55A%3A6&libId=k8isck0n010326gn000DAejxhgd09&loc=https%3A%2F%2Fwww.apdaga.com%2F2018%2F06%2Fcoursera-machine-learning-week-9.html&v=1&iid=0fb9c44119456567&opt=true&optExTitle=false&out=https%3A%2F%2Fwww.pjtra.com%2Ft%2F1-6768-43737-6768%3Furl%3Dhttp%253A%252F%252Fwww.unlimitedcellular.com%252FMotorola-5180475B01-MC145428-DATA-SET-INTERFACE_p_219297.html&ref=https%3A%2F%2Fwww.apdaga.com%2F2020%2F01%2Fcoursera-machine-learning-all-weeks-solutions-assignment-quiz.html&title=Coursera%3A Machine Learning (Week 9) %5BAssignment Solution%5D - Andrew NG - APDaga DumpBox %3A The Thirst for Learning...&txt=<span>data <%2Fspan><span>set<%2Fspan>)

*% and the variances sigma^2, an n x 1 vector*

*%*

*% Useful variables*

[m, n] = size(X);

*% You should return these values correctly*

mu = zeros(n, 1);

sigma2 = zeros(n, 1);

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the mean of the data and the variances*

*% In particular, mu(i) should contain the mean of*

*% the data for the i-th feature and sigma2(i)*

*% should contain variance of the i-th feature.*

*%*

mu = ((1/m)\*sum(X))';

sigma2 = ((1/m)\*sum((X-mu').^2))';

*% =============================================================*

**end**

### **selectThreshold.m :**

**function** [bestEpsilon bestF1] = selectThreshold(yval, pval)

*%SELECTTHRESHOLD Find the best threshold (epsilon) to use for selecting*

*%outliers*

*% [bestEpsilon bestF1] = SELECTTHRESHOLD(yval, pval) finds the best*

*% threshold to use for selecting outliers based on the results from a*

*% validation set (pval) and the ground truth (yval).*

*%*

bestEpsilon = 0;

bestF1 = 0;

F1 = 0;

stepsize = (max(pval) - min(pval)) / 1000;

**for** epsilon = min(pval):stepsize:max(pval)

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the F1 score of choosing epsilon as the*

*% threshold and place the value in F1. The code at the*

*% end of the loop will compare the F1 score for this*

*% choice of epsilon and set it to be the best epsilon if*

*% it is better than the current choice of epsilon.*

*%*

*% Note: You can use predictions = (pval < epsilon) to get a binary vector*

*% of 0's and 1's of the outlier predictions*

cvPredictions = (pval < epsilon); *% m x 1*

tp = sum((cvPredictions == 1) & (yval == 1)); *% m x 1*

fp = sum((cvPredictions == 1) & (yval == 0)); *% m x 1*

fn = sum((cvPredictions == 0) & (yval == 1)); *% m x 1*

prec = tp/(tp+fp);

rec = tp/(tp+fn);

F1 = 2\*prec\*rec / (prec + rec);

*% =============================================================*

**if** F1 > bestF1

bestF1 = F1;

bestEpsilon = epsilon;

**end**

**end**

**end**

### **cofiCostFunc.m :**

**function** [J, grad] = cofiCostFunc(params, Y, R, num\_users, num\_movies, *...*

num\_features, lambda)

*%COFICOSTFUNC Collaborative filtering cost function*

*% [J, grad] = COFICOSTFUNC(params, Y, R, num\_users, num\_movies, ...*

*% num\_features, lambda) returns the cost and gradient for the*

*% collaborative filtering problem.*

*%*

*% Unfold the U and W matrices from params*

X = reshape(params(1:num\_movies\*num\_features), num\_movies, num\_features);

Theta = reshape(params(num\_movies\*num\_features+1:**end**), *...*

num\_users, num\_features);

*% You need to return the following values correctly*

J = 0;

X\_grad = zeros(size(X)); *% Nm x n*

Theta\_grad = zeros(size(Theta)); *% Nu x n*

*% ====================== YOUR CODE HERE ======================*

*% Instructions: Compute the cost function and gradient for collaborative*

*% filtering. Concretely, you should first implement the cost*

*% function (without regularization) and make sure it is*

*% matches our costs. After that, you should implement the*

*% gradient and use the checkCostFunction routine to check*

*% that the gradient is correct. Finally, you should implement*

*% regularization.*

*%*

*% Notes: X - num\_movies x num\_features matrix of movie features*

*% Theta - num\_users x num\_features matrix of user features*

*% Y - num\_movies x num\_users matrix of user ratings of movies*

*% R - num\_movies x num\_users matrix, where R(i, j) = 1 if the*

*% i-th movie was rated by the j-th user*

*%*

*% You should set the following variables correctly:*

*%*

*% X\_grad - num\_movies x num\_features matrix, containing the*

*% partial derivatives w.r.t. to each element of X*

*% Theta\_grad - num\_users x num\_features matrix, containing the*

*% partial derivatives w.r.t. to each element of Theta*

*%*

*%% %%%%% WORKING: Without Regularization %%%%%%%%%%*

Error = (X\*Theta') - Y;

J = (1/2)\*sum(sum(Error.^2.\*R));

X\_grad = (Error.\*R)\*Theta; *% Nm x n*

Theta\_grad = (Error.\*R)'\*X; *% Nu x n*

*%% %%%%% WORKING: With Regularization*

Reg\_term\_theta = (lambda/2)\*sum(sum(Theta.^2));

Reg\_term\_x = (lambda/2)\*sum(sum(X.^2));

J = J + Reg\_term\_theta + Reg\_term\_x;

X\_grad = X\_grad + lambda\*X; *% Nm x n*

Theta\_grad = Theta\_grad + lambda\*Theta; *% Nu x n*

*% =============================================================*

grad = [X\_grad(:); Theta\_grad(:)];

**end**