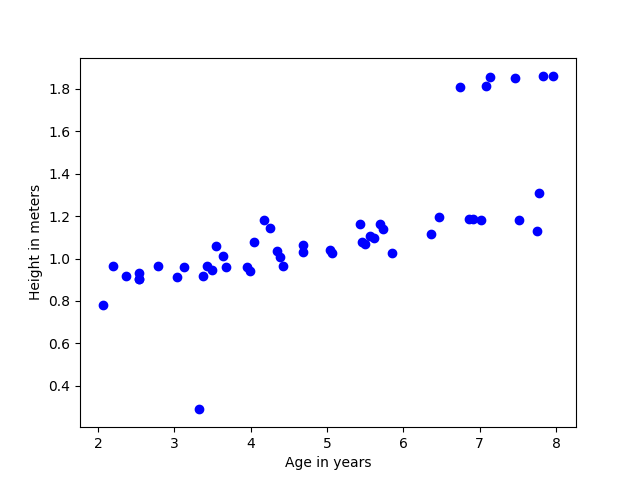
Machine Learning Assignment 1

Name: Anh Quoc Do

NetID: aqd14

# Part A

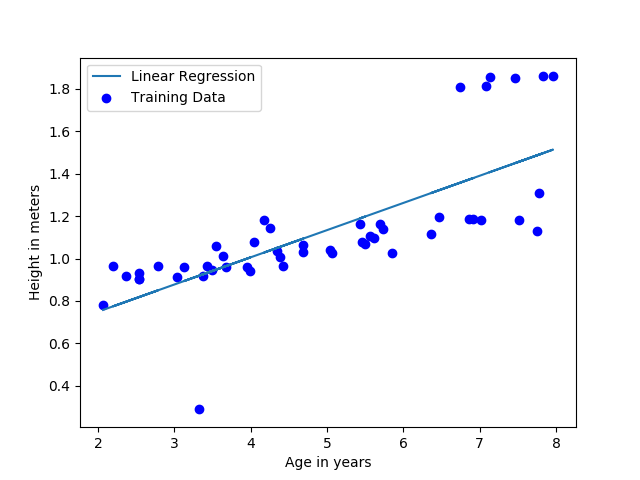
1. Plot of raw data



2. Converged value of θ

* First iteration:
  + θ = [ 0.0787538, 0.41494108]
* Converged after 803 iterations (set error threshold 1e-5)
  + θ = [0.49311086, 0.12820581]

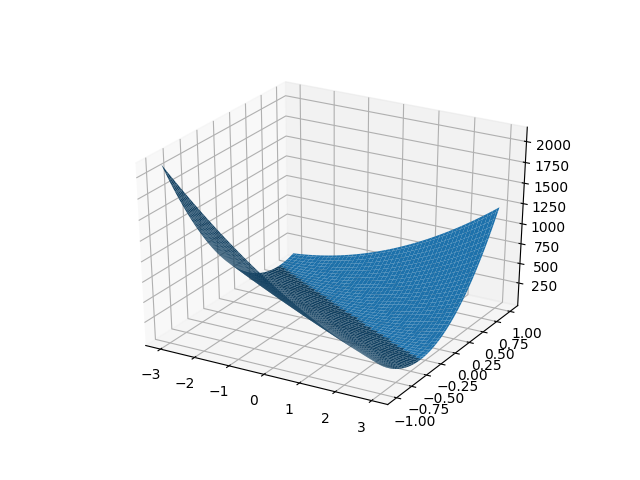
3. Plot of line that you fit



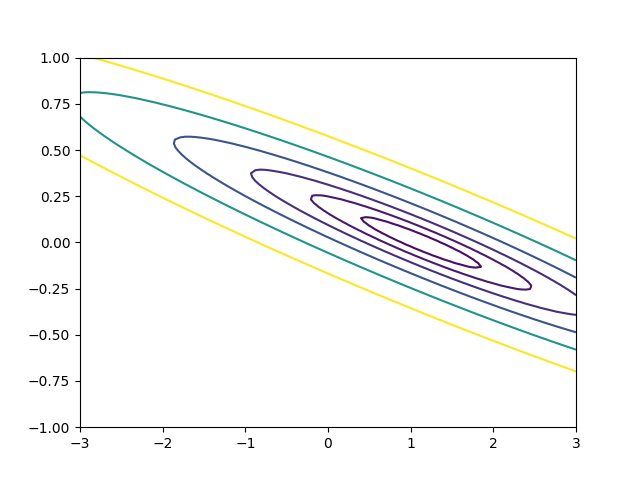
4. Prediction of your model for the height of two boys, age 3.5 and 7

* prediction = [0.94183121, 1.39055157]

5. Plot of 3D surface of J



6. Plot of contour of J



7. Answer to question about relationship between these plots and θ value found by your algorithm.

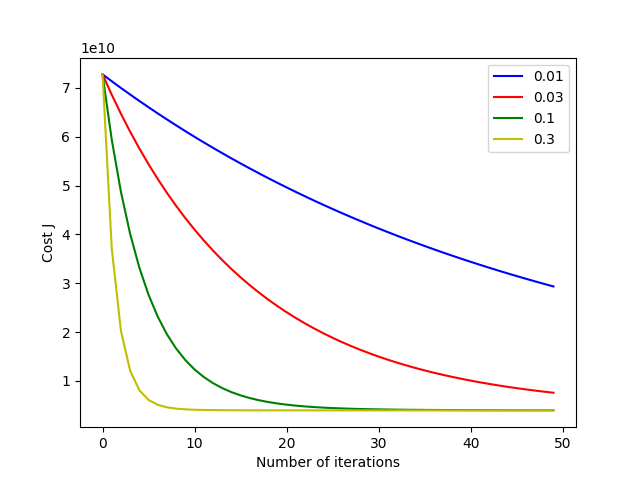
* These plots offer a visual view of how different values of θ affect to the value of cost function.
* There is exist an optimal vector θ that helps the cost function reaches minimum value. With gradient descent, in each step we update theta values that move toward optimal values

8. Include your code solution to Task A in your pdf submission

**See appendix**

# Part B

1. Plot(s) comparing different learning rates



2. Converged value of θ for best learning rate

* θ = [ 353178.61702128, 114973.94457945, -3702.28587976] with alpha = 0.3

3. Prediction of your model for the price of a house with 1650 square feet and 3 bedrooms

* Predicted price = $303,614

4. Value of θ computed using normal equations

* θ = [109185.1639846, 123.62723839, -3185.43247293]
* These values are different from those using gradient descent

5. Prediction of your normal equations θ on a house with 1650 square feet and 3 bedrooms.

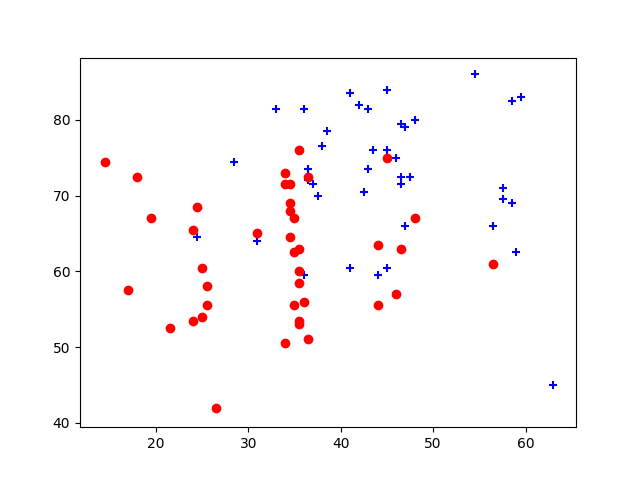
* Predicted price = $303,614

6. Include your code solution to Task b in your pdf submission

**See Appendix**

# Part C

1. Plot of raw data



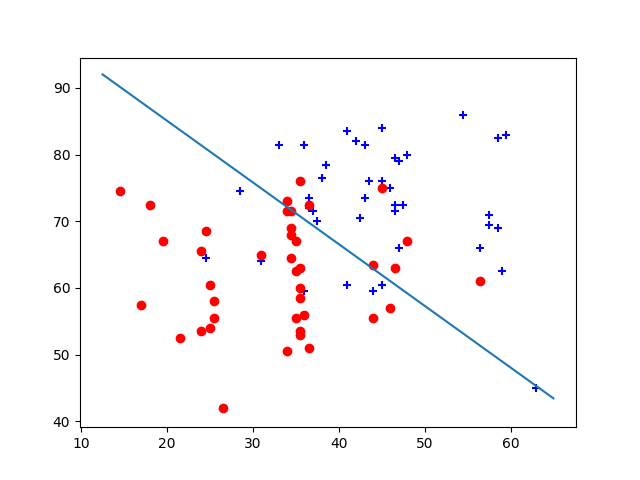
1. What values of θ did you get? How many iterations were required for convergence?

* Convergered at epoch 6
* θ = [-15.76822471, 0.14087783, 0.15219571]

1. What is the probability that a student with a score of 20 on Exam 1 and a score of 80 on Exam 2 will not be admitted?

* p = 0.684604

4. Plot of final decision boundary found



Appendix

# Part A & B

*# Implmentation of linear regession algorithm for only one feature dataset*

*# by using gradient descent to find optimum weights to the classification*

*'''*

*The data files ax.dat and ay.dat contain some example measurements of*

*heights for various boys between the ages of two and eights. The y-values*

*are the heights measured in meters, and the x-values are the ages of the boys*

*corresponding to the heights.*

*There are 50 training examples in total*

*'''*

*# usage python linear\_regression.py*

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

**from** **numpy.linalg** **import** inv

**class** **LinearRegression**(object):

*""" Linear regression model*

*Parameters*

*----------*

*alpha : float*

*Learning rate*

*iter : integer*

*Number of iteration*

*check\_conv: boolean*

*Check if model is check\_conv or not to stop iterations*

*"""*

**def** \_\_init\_\_(self, alpha=0.01, iter=2000, check\_conv=True):

self.alpha = alpha

self.iter = iter

self.check\_conv = check\_conv

*# self.normalized = normalized*

**def** fit(self, X, y):

*""" Fit traning dataset to the model.*

*Update theta simultaneously by using gradient descent.*

*For simplicity, ignore error handling.*

*X : integer*

*training input*

*y : integer*

*label output*

*"""*

self.theta = np.zeros((X.shape[1], 1)) *# Column vector*

err\_threshold = 1e-5 *# Stop when the cost function is smaller than threshold*

J = []

**for** i **in** range(self.iter):

cost = cost\_function(X, y, self.theta)

J.append(cost)

gra, \_ = self.gradient(X, y)

temp = self.theta - self.alpha \* gra

*# if np.array\_equal(temp, self.theta):*

**if** self.check\_conv **is** True **and** np.sum(abs(self.theta - temp)) < err\_threshold:

**break**

*# Simultaneously update theta*

self.theta = temp

*# print('Not convergered!')*

**return** self.theta, J

**def** hypothesis(self, X):

*# return np.dot(X, self.theta[1:]) + self.theta[0]*

**return** np.dot(X, self.theta)

**def** gradient(self, X, y):

*"""Calculate gradient descent*

*Parameters*

*----------*

*X : array-like*

*training data*

*Y : vector*

*label output*

*Returns*

*-------*

*gradient : float*

*gradient value of given data*

*mse : float*

*mean squared errors*

*"""*

err = y - self.hypothesis(X)

mse = (1.0/X.shape[0]) \* np.sum(np.power(err, 2))

gradient = -(1.0/X.shape[0]) \* X.T.dot(err)

**return** gradient, mse

**def** predict(self, X):

**return** self.hypothesis(X)

**def** cost\_function(X, y, theta):

*"""*

*Calculate cost function*

*"""*

m = X.shape[0] *# Training size*

err = y - np.dot(X, theta)

J = 1.0/(2\*m) \* np.sum(np.power(err, 2))

**return** J

*# ----------------------- PART A ----------------------- #*

**def** part\_a():

*# load data*

X = np.loadtxt('data/ax.dat') *# input data*

y = np.loadtxt('data/ay.dat') *# output data*

*# plot dataset*

plt.scatter(X, y, facecolors='blue')

plt.xlabel('Age in years')

plt.ylabel('Height in meters')

*# plt.show()*

*# Data preprocessing*

m = X.shape[0] *# Training size*

X = np.stack((np.ones(m), X), axis=-1)

y = y.reshape(m, 1)

model = LinearRegression(alpha=0.07)

theta, J = model.fit(X, y)

*# Plot straight line*

plt.plot(X[:,1],np.dot(X,theta))

plt.legend(['Linear Regression', 'Training Data'])

plt.show()

*# plt.savefig('plot1.png')*

*# Prediction*

test\_data = np.array([[1, 3.5], [1, 7]])

prediction = model.predict(test\_data)

**print**("Predicted height of kids age 3.5 and 7: ", prediction)

plot\_surface\_contour(X, y)

**def** plot\_surface\_contour(X, y):

*# Display Surface Plot of J*

t0 = np.linspace(-3, 3, 100).reshape(100, 1)

t1 = np.linspace(-1, 1, 100).reshape(100, 1)

T0, T1 = np.meshgrid(t0, t1)

J\_vals = np.zeros((len(t0), len(t1)))

**for** i **in** range(len(t0)):

**for** j **in** range(len(t1)):

t = np.hstack([t0[i], t1[j]])

J\_vals[i, j] = cost\_function(X, y, t)

*#Because of the way meshgrids work with plotting surfaces*

*#we need to transpose J to show it correctly*

J\_vals = J\_vals.T

*# print(J\_vals)*

fig = plt.figure()

ax = fig.gca(projection='3d')

ax.plot\_surface(T0,T1,J\_vals)

plt.show()

plt.close()

*#Display Contour Plot of J*

plt.contour(T0,T1,J\_vals, np.logspace(-2,2,15))

plt.show()

*# ----------------------- END PART A --------------------- #*

*# ----------------------- PART B ------------------------- #*

**def** part\_b():

*# Load data from files*

X = np.loadtxt('data/bx.dat')

y = np.loadtxt('data/by.dat')

sigma = np.std(X, axis=0) *# std*

mu = np.mean(X, axis=0) *# mean*

*# Data preprocessing*

m = X.shape[0]

y = y.reshape(m, 1)

X\_scaled = (X-mu) / sigma *# normalize(X)*

intercept = np.ones((m, 1))

X\_scaled = np.column\_stack((intercept, X\_scaled))

*# print(X\_scaled)*

*# y\_scaled = normalize(y)*

iterations = 50

alphas = [0.01, 0.03, 0.1, 0.3]

colors = ['b', 'r', 'g', 'y']

test\_data = np.array([[1, 1650], [1, 3]])

**for** i **in** range(len(alphas)):

model = LinearRegression(alpha=alphas[i], iter=iterations, check\_conv=False)

theta, J = model.fit(X\_scaled, y)

*# print('Learning rate %f' % alphas[i])*

*# print('Theta: %d -- %d' % (theta[0], theta[1]))*

*# print('Predict for the price of a house with 1650 square feet and 3 bedrooms: ', model.predict(test\_data))*

*# Now plot J*

plt.plot(range(iterations), J, color=colors[i], label=str(alphas[i]))

plt.xlabel('Number of iterations')

plt.ylabel('Cost J')

plt.legend(loc='upper right')

plt.show()

*# Find optimal theta vector with best learning rate*

*# Assume we already know that alpha = 0.3 is the best learning rate*

model = LinearRegression(alpha=0.3, check\_conv=True)

theta, \_ = model.fit(X\_scaled, y)

*# print(theta)*

test\_data = np.array([1650, 3])

test\_data\_scaled = (test\_data - mu) / sigma

*# print(test\_data\_scaled)*

test\_data\_scaled = np.append([1], test\_data\_scaled)

*# print(test\_data\_scaled)*

**print**('Predicted price using gradient descent: ', model.predict(test\_data\_scaled))

*# test\_data\_scaled.shape = (3, 1)*

*# print(theta.T.dot(test\_data\_scaled))*

X = np.column\_stack((intercept, X))

test\_data = np.append([1], test\_data)

theta\_normal\_equation = normal\_equation(X, y)

**print**('Predicted price using normal equation: ', theta\_normal\_equation.T.dot(test\_data))

**def** normalize(X):

*# Preprocess data to give std of 1 and mean of 0*

sigma = np.std(X, axis=0) *# std*

mu = np.mean(X, axis=0) *# mean*

X = (X-mu) / sigma *# adjustment*

**return** X

**def** normal\_equation(X, y):

theta = inv(X.T.dot(X)).dot(X.T).dot(y)

**return** theta

*# ----------------------- END PART B ------------------------- #*

part\_a()

part\_b()

# Part C

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **math** **import** log

**from** **numpy.linalg** **import** inv

**def** newton\_method(X, y, theta, tolerance=1e-5, max\_iters=15):

MAX\_ITERS = 1

epoch = 1

**for** \_ **in** range(max\_iters):

H = hessian(X, theta)

*# print(hes)*

g = gradient(X, y, theta)

*# print('Gradient shape: ', g.shape)*

temp = theta - np.dot(inv(H), g)

**if** np.sum(abs(theta - temp)) < tolerance:

**print**('Convergered at epoch **%d**' % epoch)

**break**

theta = temp

epoch += 1

**return** theta

**def** gradient(X, y, theta):

m = X.shape[0]

h = hypothesis(X, theta)

g = (1.0/m) \* X.T.dot(h-y)

**return** g

**def** hessian(X, theta):

m = X.shape[0]

h = hypothesis(X, theta)

h.shape = (len(h),)

H = (1.0/m) \* np.dot(np.dot(X.T, np.diag(h)), np.dot(np.diag(1-h), X))

**return** H

**def** sigmoid(z):

result = 1.0/(1.0+np.exp(-z))

**return** result

**def** cost\_function(X, y, theta):

m = X.shape[0]

h = hypothesis(X, theta)

J = (1.0/m) \* (-y.dot(np.log(h)) - (1-y).dot(np.log(1-h)))

**return** J

**def** hypothesis(X, theta):

*# print('Hypothesis: ', h.shape)*

**return** sigmoid(X.dot(theta))

**def** main():

X = np.loadtxt('data/cx.dat')

y = np.loadtxt('data/cy.dat')

*# Get positive and negative indices*

pos = np.nonzero(y)

neg = np.where(y==0)

*# Plot*

plt.scatter(X[pos,0], X[pos,1], color='b', marker='+')

plt.scatter(X[neg,0], X[neg,1], color='r', marker='o')

*# plt.show()*

m = X.shape[0]

X = np.column\_stack((np.ones((m, 1)), X))

y.shape = (y.shape[0], 1)

theta = np.zeros((X.shape[1], 1))

theta = newton\_method(X, y, theta)

**print**(theta)

*# predict if the student got 20 in exam 1 and 80 in exam 2*

test\_data = np.array([1, 20, 80])

p = 1 - hypothesis(test\_data, theta)

**print**('Probability that student is not admitted with a score of 20 on Exam 1 and a score of 80 on Exam 2 is **%f**' % p)

*# Plot decision boundary*

min\_X = np.min(X[:,1:3])

max\_X = np.max(X[:,1:3])

plot\_x = [min\_X-2, max\_X+2]

plot\_y = (-1/theta[2])\*(theta[1]\*plot\_x) + theta[0]

plt.plot(plot\_x, plot\_y)

plt.show()

main()

# Appendix

## Part A: