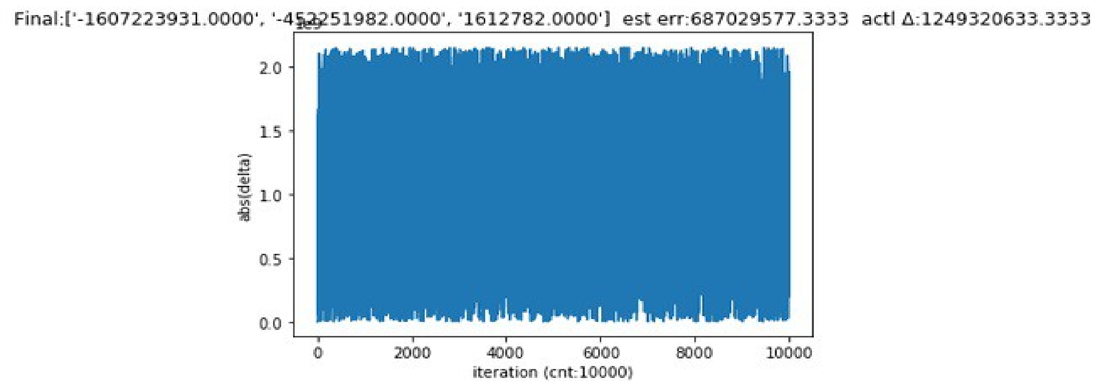
Tom Nguyen
Dr. Xiaobai Liu
CS 596 MW
February 25, 2018

Homework Assignment 2

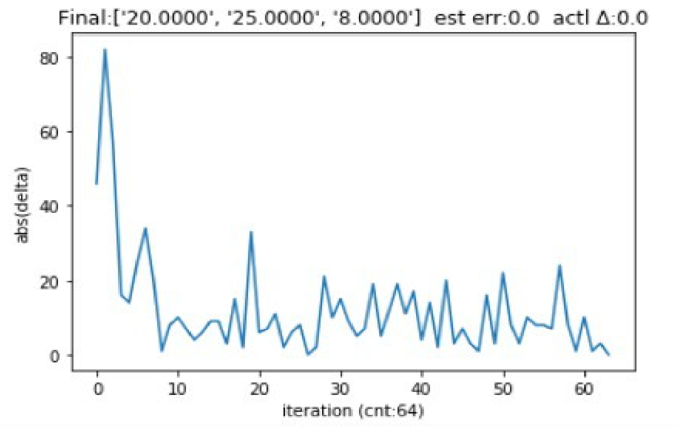
Alpha = 1
delta = abs(iterError)+np.mean(abs(estimatedUnitPrice – groundUnitPrice))
initial predictions = random

```
iteration:9998, delta:1123187954.6666665
estimated unite price  [-1607223931  -227335559   338987416]
ground unit price  [20 25  8]
estimatedUnitPrice updated:  [-1607223931  -452251982     1612782]
iteration:9999, delta:1249320633.3333335
estimation error:687029577.3333334
```



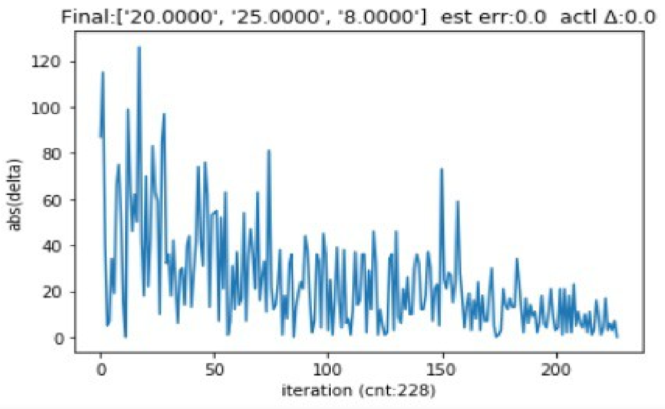
Alpha = .1
delta = abs(iterError)+np.mean(abs(estimatedUnitPrice - groundUnitPrice))
initial predictions = random

```
iteration:62, delta:3.0
estimated unite price  [20 25  8]
ground unit price  [20 25  8]
estimatedUnitPrice updated:  [20 25  8]
estimation error:0.0
```



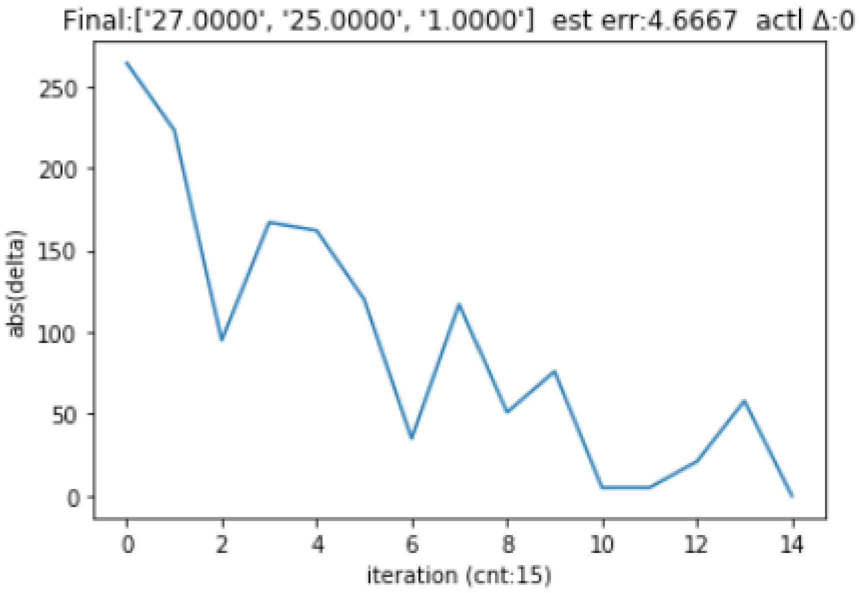
Alpha =.01
delta = abs(iterError)+np.mean(abs(estimatedUnitPrice – groundUnitPrice))
initial predictions =random

```
iteration:226, delta:7.0
estimated unite price  [20 25  8]
ground unit price  [20 25  8]
estimatedUnitPrice updated:  [20 25  8]
estimation error:0.0
```

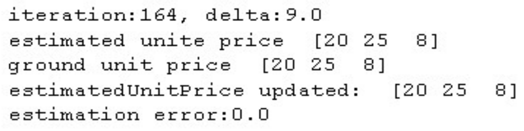


Alpha = .01
delta = iterError
Initial predictions= random

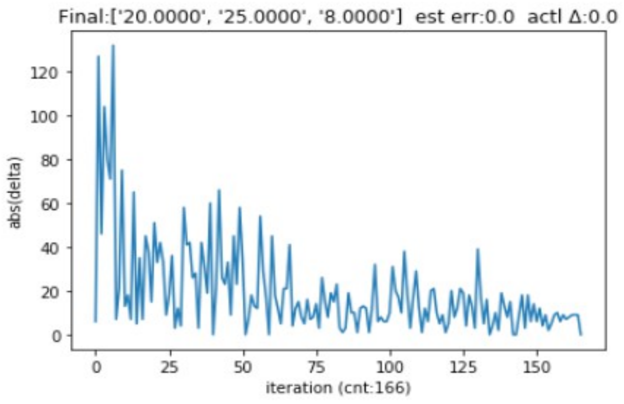
```
iteration:13, delta:58
estimated unite price  [27 25  1]
ground unit price  [20 25  8]
estimatedUnitPrice updated:  [27 25  1]
estimation error:4.666666666666667
```



Alpha = .01
delta = abs(iterError)+np.mean(abs(estimatedUnitPrice – groundUnitPrice))
Initial predictions= [10, 10, 10]



```
iteration:164, delta:9.0
estimated unite price  [20 25  8]
ground unit price  [20 25  8]
estimatedUnitPrice updated:  [20 25  8]
estimation error:0.0
```



Write-up

Importing math to use ceil and floor function:
For this assignment I imported math for the function math.ceil and math.floor. I noticed that without having the number round up or down sometimes the algorithm would return the same estimated unit price without changing it. This was because the unit price was in integers and would not round if we did not a high enough alpha value.

For delta:
I was not sure how to code delta but I wanted to have a perfect fit. I did this by making sure that both the iterError and the error was 0. Side note, it is not always a good thing in machine learning to have a perfect fit.

For updating estimatedUnitPrice:
I tried several equations for this but it didn't work out properly. From the slide it said to use :
delta_fish = delta_fish – (estimated price – true price) x_i.
In class I remember we discussed something about higher orders should have some weight in how we adjust our estimated unit price.
Equation:
estimatedUnitPrice[g]=estimatedUnitPrice[g]–ALPHA*(estimatedTotalPrice-cashierPrice)*(x_i/sum(x_i))
I did this because the alpha I wanted to use the zero if we encounter it and to incorporate the proportion to the meal placed order. For example if we order 9 fish, 0 chips, and 2 ketchup. I would not want ketchup and fish estimatedUnitPrice to increase at the same rate. The order that will affect the total price the most will be the fish.

Additional things:
Added some print statements for the estimated unit price, ground unit price, and the updated estimated unit price.
Added a flag for the special case of placing an order of [0, 0, 0]. Without this flag we will get an error for dividing by zero.

Code used in python:

```
import numpy as np
import matplotlib.pyplot as plt
import math
#from dataNormalization import rescaleNormalization

# Starting codes for the HA2 of CS596

# Fill in the codes between "%PLACEHOLDER#start" and "PLACEHOLDER#end"

# Ground-truth Cashier
groundUnitPrice = np.array([20, 25, 8]) # for fish, chip, and ketchup, respectively

# step 1: initialize your guess on the unit prices of fish, chip and ketchup.
estimatedUnitPrice = np.array(([10,10,10])) # initial unit prices.
MAX_POSSIBLE_UNIT_PRICE = 50
estimatedUnitPrice = np.random.randint(MAX_POSSIBLE_UNIT_PRICE,
size=len(groundUnitPrice))

# choose random initial guesses

#PLACEHOLDER_1#start: set your own stopping conditions and learning rate

#condition 1: maximal iterations, stop.
MAX_ITERATION = 10000
#condition 2: if the difference between your prediction and the cashier's price is smaller than a
threshold, stop.
MIN_DELTA = 0.001
# learning rate
ALPHA = .01#1e-3
#PLACEHOLDER_1#end
MAX_PLACE_ORDER = 10

# Y coordinates for plotting
deltaHistory = []
delta = 0

# step 2: iterative method
for i in range(0, MAX_ITERATION):
   yes_zero = False
   # order a meal (simulating training data)

   randomMealPortions = np.random.randint(MAX_PLACE_ORDER, size=3)

   # calculate the estimated price
   expectedTotalPrice = np.sum(estimatedUnitPrice * randomMealPortions )
```

```
    # calculate cashier/true price;
    cashierPrice = np.sum(groundUnitPrice * randomMealPortions)

    #%%%PLACEHOLDER_2#start
    print('estimated unit price ',estimatedUnitPrice)
    print('ground unit price ', groundUnitPrice)
#    print('random meal portions',randomMealPortions)
#    print('expected total price ',expectedTotalPrice)
#    print('cashier price ',cashierPrice)

    # calculate current error
    iterError = expectedTotalPrice - cashierPrice

    # append iterError to the history array
    deltaHistory.append(abs(iterError))
#    print('itererror: ',iterError)
    #update unit prices

#    print('eup[i]: ',estimatedUnitPrice[0])

   # print('random ',np.sum(randomMealPortions))
    for g in range(len(estimatedUnitPrice)):
        if g == 0 :#checks for randomMealPortions to see if it is all zero
            if randomMealPortions[g] ==0:
                if randomMealPortions[g+1] == 0:
                    if randomMealPortions[g+2]==0:
                        yes_zero=True;
                        break;
        #change = ALPHA * (iterError/len(estimatedUnitPrice))
*(randomMealPortions[g]/np.sum(randomMealPortions))
        change =ALPHA * iterError *(randomMealPortions[g]/np.sum(randomMealPortions))
        if change < 0:
            change=math.floor(change)
        else:
            change=math.ceil(change)
        estimatedUnitPrice[g] = estimatedUnitPrice[g] -change
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA*(estimatedUnitPrice[g]-
(cashierPrice*(randomMealPortions[g]/np.sum(randomMealPortions))))
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] -
(randomMealPortions[g]/np.sum(randomMealPortions))*(expectedTotalPrice-cashierPrice)
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA*iterError
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA* (estimatedUnitPrice[g]-
groundUnitPrice[g])
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA*(1/
(2*np.sum(randomMealPortions)))*(iterError)
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] -
ALPHA*(iterError)*(randomMealPortions[g]/np.sum(randomMealPortions))
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA*(estimatedUnitPrice[g]-
groundUnitPrice[g])*randomMealPortions[g]
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] - ALPHA * (expectedTotalPrice-cashierPrice)
        #d_of_f = ALPHA *(estimatedUnitPrice[g]-groundUnitPrice[g])*randomMealPortions[g]
        #estimatedUnitPrice[g] = estimatedUnitPrice[g] -
ALPHA*(randomMealPortions[g]/np.sum(randomMealPortions))*iterError
    if yes_zero == True:
        continue
    print('estimatedUnitPrice updated: ',estimatedUnitPrice)

    #%%%%PLACEHOLDER_2#end

    #delta = iterError
    delta=abs(iterError)+np.mean(abs(estimatedUnitPrice - groundUnitPrice))
    #print(delta)
    #check stop conditions
    if abs(delta) < MIN_DELTA:
        break

    print('iteration:{}, delta:{}'.format(i, abs(delta)))


# step 3: evaluation
error = np.mean(abs(estimatedUnitPrice - groundUnitPrice))
print('estimation error:{}'.format(error))

# visualize convergence curve: error v.s. iterations

plt.plot(range(0, len(deltaHistory)), deltaHistory)
plt.xlabel('iteration (cnt:{})'.format(len(deltaHistory)))
plt.ylabel('abs(delta)')
plt.title('Final:{}  est err:{}  actl Δ:{}'.format([ '%.4f' % elem for elem in estimatedUnitPrice ],
round(error, 4), round(delta, 4)))
plt.show()
```