DS5220 Homework 2

Alex Kramer

2024-02-24

Create Functions:

```
sim_nums = function(){
  # generate simulated data with random numbers
  x = runif(50, -2, 2)
  e = rnorm(50, 0, 2)
  y = 3 + 2 * x + e
  return (list('x'= x, 'y'=y))
}
```

```
analytic = function(x, y){
  # function to perform linear regression analysis
  model = lm(y~x)
  return (list('w0'=model$coefficients[1], 'w1'=model$coefficients[2]))
}
```

```
batch_gd = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5 # really small number
  grad_norm = 1
  while (grad_norm > e){
    y_hat = w0 + w1 * x
    grad_one = (-2 / 50) * sum(x*(y-y_hat))
    grad_zero = (-2 / 50) * sum((y-y_hat))
    w1 = w1-eta * grad_one
    w0 = w0-eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

```
stochastic_gd = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
  while (grad_norm >= e && iter < max_iter){
    i = sample(1:50, 1)
```

```r
    y_hat = (w0 + w1 * x[i])
    grad_one = (-2 / 50) * sum(x[i] * (y[i] - y_hat))
    grad_zero = (-2 / 50) * sum((y[i] - y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

```r
batch_gd_mae = function(x, y){
  w0 = 0.1
  w1 = 0.1
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
  while (grad_norm >= e && iter < max_iter){
    y_hat = w0 + w1 * x
    grad_one = (1 / 50) * sum(x * sign(y - y_hat))
    grad_zero = (1 / 50) * sum(sign(y - y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

```r
l2_huber = function(x, y, delta=1){
  w0 = 0
  w1 = 1
  eta = 0.01
  for (i in 1:1000){
    y_hat = w0 + w1 * x
    grad_one = ifelse(abs(y-y_hat) <= delta, 0.5 * (-2 / 50) * sum(x*(y-y_hat)), delta * (1 / 50) * sum
    grad_zero = ifelse(abs(y - y_hat) <= delta, 0.5 * (-2 / 50) * sum((y-y_hat)), delta * (1 / 50) * su
    w0 = w0 - eta * grad_zero
    w1 = w1 - eta * grad_one
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

```r
# correct version
huber_stochastic = function(x, y, delta = 1){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
```

```r
  while (grad_norm > e && iter < max_iter){
    i = sample(1:50, 1)
    y_hat = (w0 + w1 * x[i])
    grad_one = ifelse(abs(y[i]-y_hat) <= delta, 0.5 * (2 / 50) * sum(x[i]*(y[i]-y_hat)), delta * (-1 / 5
    grad_zero = ifelse(abs(y[i] - y_hat) <= delta, 0.5 * (2 / 50) * sum((y[i]-y_hat)), delta * (-1 / 50)
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

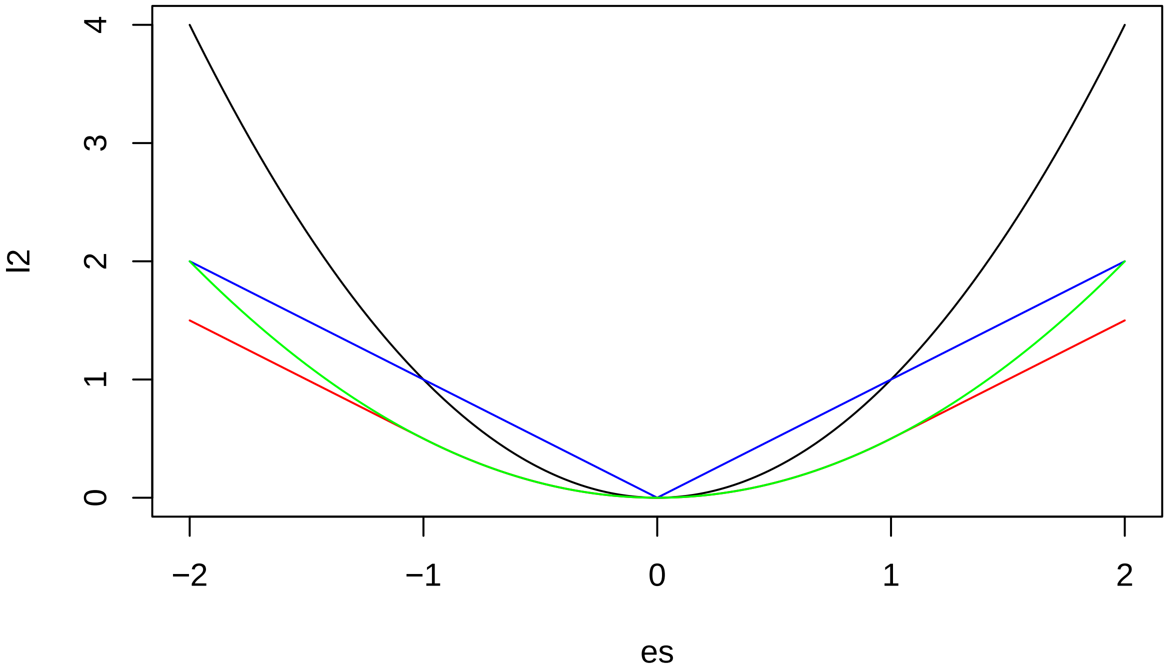
```r
stochastic_mae = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  while (grad_norm > e){
    i = sample(1:50, 1)
    y_hat = (w0 + w1 * x[i])
    grad_one = (1 / 50) * sum(x[i]*sign(y[i]- y_hat))
    grad_zero = (1 / 50) * sum(sign(y[i]-y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

```r
sim_nums_outliers = function(){
  x = runif(50, -2, 2)
  e = rnorm(50, 0, 2)
  y = 3 + 2 * x + e
  for (i in 1:50){
    p1 = runif(1)
    if (p1 < 0.1){
      p2 = runif(1)
      if (p2 < 0.5){
        y[i] = y[i] + (y[i] * 2)
      }
      else {
        y[i] = y[i] - (y[i] * 2)
      }
    }
  }
  return (list('x'= x, 'y'=y))
}
```

Question 3A: [Implmentation]

```r
# setting delta = 1 & 2
es = seq(-2, 2,length.out=1000) # range of e
```

```r
l2 = es^2 # l2 loss function
l1 = abs(es) # l1 loss function
huber_1 = ifelse(abs(es)<= 1, 0.5 * es^2, abs(es) - 0.5) #huber loss function for delta = 1
huber_2 = ifelse(abs(es)<= 2, 0.5 *es^2, 2 * abs(es) - 0.5 * 2^2) # huber loss function for delta = 2
plot(es, l2, col='black', type='l')
lines(es, l1, col='blue')
lines(es, huber_1, col='red')
lines(es,huber_2, col='green')
```



Use the graph to discuss the relative advantages and disadvantages of these loss functions for linear regression. The black line (l2), green line(huber_2), and the red line (huber_1): as they all get bigger they get closer to l2 loss it will descend faster but are also less likely to over shoot. Blue line (l1) is non differentiate.

Question 3B: [Implmentation]

Implement gradient descent for the loss functions above.

```r
batch_gd = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5 # really small number
  grad_norm = 1
  while (grad_norm > e){
    y_hat = w0 + w1 * x
    grad_one = (-2 / 50) * sum(x*(y-y_hat))
    grad_zero = (-2 / 50) * sum((y-y_hat))
    w1 = w1-eta * grad_one
    w0 = w0-eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

```
batch_gd_mae = function(x, y){
  w0 = 0.1
  w1 = 0.1
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
  while (grad_norm >= e && iter < max_iter){
    y_hat = w0 + w1 * x
    grad_one = (1 / 50) * sum(x * sign(y - y_hat))
    grad_zero = (1 / 50) * sum(sign(y - y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

```
l2_huber = function(x, y, delta=1){
  w0 = 0
  w1 = 1
  eta = 0.01
  for (i in 1:1000){
    y_hat = w0 + w1 * x
    grad_one = ifelse(abs(y-y_hat) <= delta, 0.5 * (-2 / 50) * sum(x*(y-y_hat)), delta * (1 / 50) * sum
    grad_zero = ifelse(abs(y - y_hat) <= delta, 0.5 * (-2 / 50) * sum((y-y_hat)), delta * (1 / 50) * su
    w0 = w0 - eta * grad_zero
    w1 = w1 - eta * grad_one
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

Question 3C: [Implmentation]

Implement stochastic gradient descent for the loss functions above

```
# correct version
stochastic_gd = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
  while (grad_norm >= e && iter < max_iter){
    i = sample(1:50, 1)
    y_hat = (w0 + w1 * x[i])
    grad_one = (-2 / 50) * sum(x[i] * (y[i] - y_hat))
    grad_zero = (-2 / 50) * sum((y[i] - y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
```

```
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

```
stochastic_mae = function(x, y){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  while (grad_norm > e){
    i = sample(1:50, 1)
    y_hat = (w0 + w1 * x[i])
    grad_one = (1 / 50) * sum(x[i]*sign(y[i]- y_hat))
    grad_zero = (1 / 50) * sum(sign(y[i]-y_hat))
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
  }
  return(list('w1'=w1, 'w0'=w0))
}
```

```
# correct version
huber_stochastic = function(x, y, delta = 1){
  w0 = 0
  w1 = 0
  eta = 0.01
  e = 1e-5
  grad_norm = 1
  max_iter = 1000  # Maximum number of iterations
  iter = 0
  while (grad_norm > e && iter < max_iter){
    i = sample(1:50, 1)
    y_hat = (w0 + w1 * x[i])
    grad_one = ifelse(abs(y[i]-y_hat) <= delta, 0.5 * (2 / 50) * sum(x[i]*(y[i]-y_hat)), delta * (-1 / 5
    grad_zero = ifelse(abs(y[i] - y_hat) <= delta, 0.5 * (2 / 50) * sum((y[i]-y_hat)), delta * (-1 / 50)
    w1 = w1 - eta * grad_one
    w0 = w0 - eta * grad_zero
    grad_norm = sqrt(grad_one^2 + grad_zero^2)
    iter = iter + 1
  }
  return(list('w1' = w1, 'w0' = w0))
}
```

Question 4: [Implmentation] In this question we will revisit JW Figure 3.3, and empirically evaluate various approaches to fitting linear regression.

Question 4A: [Implmentation]

```
nums = sim_nums()
x = nums$x
y = nums$y
```

```r
analytic_coef = analytic(x, y)
print('Analytic: ')
```

```
## [1] "Analytic: "
```

```r
analytic_coef$w1
```

```
##        x
## 1.917065
```

```r
batch_coef = batch_gd(x, y)
print('Batch Gradient Descent')
```

```
## [1] "Batch Gradient Descent"
```

```r
batch_coef$w1
```

```
## [1] 1.917067
```

```r
stochastic_coef = stochastic_gd(x, y)
print('Stochastic Gradient Descent')
```

```
## [1] "Stochastic Gradient Descent"
```

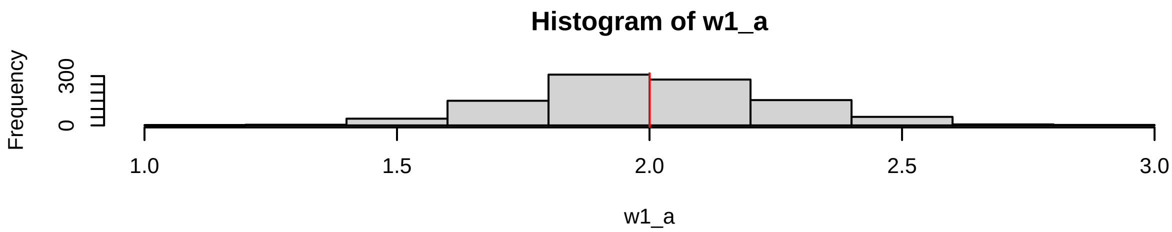
```r
stochastic_coef$w1
```

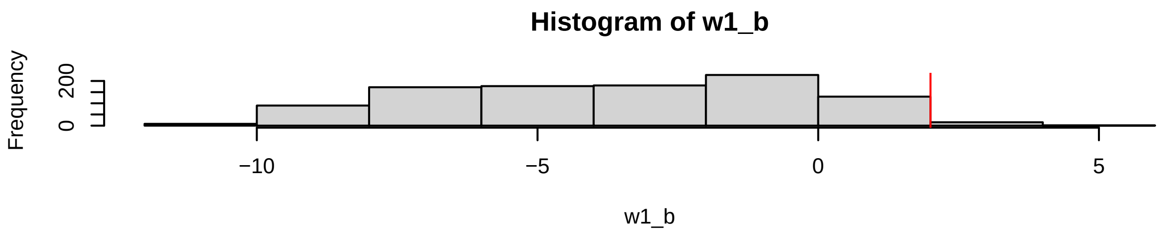
```
## [1] 0.8579095
```

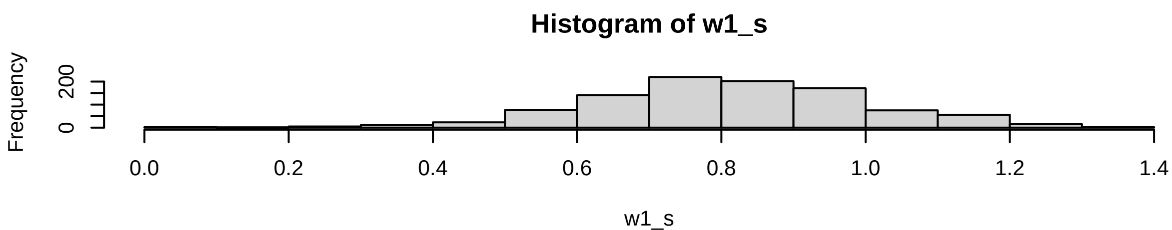
Question 4B: [Implmentation]

```r
# Set up vectors
w1_a = rep(0, 1000)
w1_b = rep(0, 1000)
w1_s = rep(0, 1000)

# Run each algorithm 1000 times
for (i in 1:1000){
    nums = sim_nums()
    x = nums$x
    y = nums$y
    analytic_coef = analytic(x, y)
    w1_a[i] = analytic_coef$w1

    batch_coef = batch_gd_mae(x, y)
    w1_b[i] = batch_coef$w1

    stochastic_coef = stochastic_gd(x, y)
    w1_s[i] = stochastic_coef$w1
}

# Make graph
par(mfrow = c(3,1)) # three stacked
hist(w1_a)
abline(v = 2, col = 'red')
hist(w1_b)
abline(v = 2, col = 'red')
```

```r
hist(w1_s)
abline(v = 2, col = 'red')
```







Question 4C: [Implmentation]

```r
nums = sim_nums()
x = nums$x
y = nums$y
analytic_coef = analytic(x, y)
analytic_coef$w1
```

```
##         x
## 1.691544
```

```r
batch_coef = batch_gd_mae(x, y)
batch_coef$w1
```

```
## [1] -3.34789
```

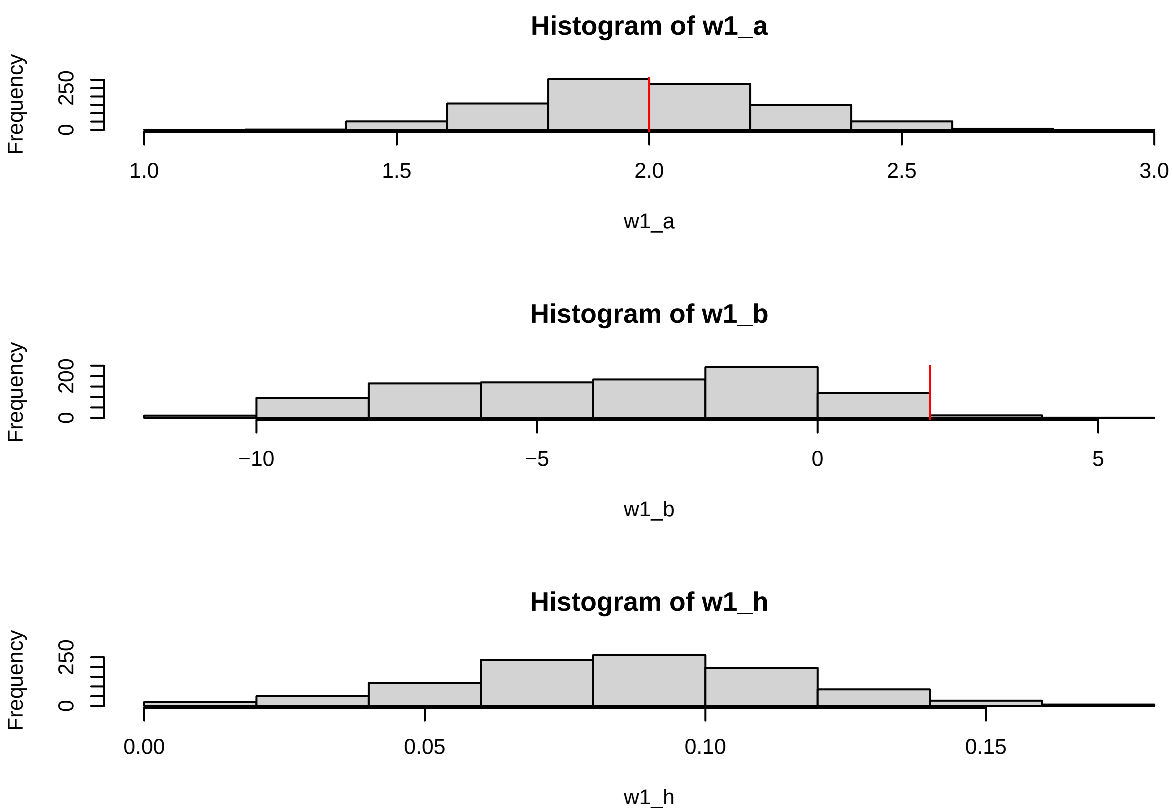
```r
huber_coef = huber_stochastic(x, y)
huber_coef$w1
```

```
## [1] 0.09898534
```

Question 4D: [Implmentation]

```r
# Set up vectors
w1_a = rep(0, 1000)
w1_b = rep(0, 1000)
```

```
w1_h = rep(0, 1000)

# Run each algorithm 1000 times
for (i in 1:1000){
    nums = sim_nums()
    x = nums$x
    y = nums$y
    analytic_coef = analytic(x, y)
    w1_a[i] = analytic_coef$w1

    batch_coef = batch_gd_mae(x, y)
    w1_b[i] = batch_coef$w1

    huber_coef = huber_stochastic(x, y)
    w1_h[i] = huber_coef$w1
}

# Make histograms
par(mfrow = c(3,1)) # three stacked
hist(w1_a)
abline(v = 2, col = 'red')
hist(w1_b)
abline(v = 2, col = 'red')
hist(w1_h)
abline(v = 2, col = 'red')
```



Question 4E: [Implmentation]

```r
nums = sim_nums_outliers()
x = nums$x
y = nums$y
#plot(x, y)

analytic = analytic(x, y)
analytic$w0
```

```
## (Intercept)
##    3.321656
```

```r
analytic$w1
```

```
##        x
## 1.949499
```

```r
l1 = batch_gd_mae(x, y)
l1$w0
```

```
## [1] -8.4592
```

```r
l1$w1
```

```
## [1] -2.576133
```

```r
huber = huber_stochastic(x, y, 1)

huber$w0
```

```
## [1] 0.1162877
```

```r
huber$w1
```

```
## [1] 0.09366245
```

Question 4F: [Implmentation]

```r
analytic = function(x, y){
  # function to perform linear regression analysis
  model = lm(y~x)
  return (list('w0'=model$coefficients[1], 'w1'=model$coefficients[2]))
}
# set up vectors
w1_a = rep(0, 1000)
w1_b = rep(0, 1000)
w1_h = rep(0, 1000)

for (i in 1:1000){
    nums = sim_nums_outliers()
    x = nums$x
    y = nums$y
    analytic_coef = analytic(x, y)
    w1_a[i] = analytic_coef$w1
    batch_coef = batch_gd_mae(x, y)
    w1_b[i] = batch_coef$w1
    huber_coef = huber_stochastic(x, y)
```

```
    w1_h[i] = huber_coef$w1
}
```

```
# Make histograms
par(mfrow = c(3,1)) # three stacked

hist(w1_a)
abline(v=2, col='red')
hist(w1_b)
abline(v=2, col='red')
hist(w1_h)
abline(v=2, col='red')
```