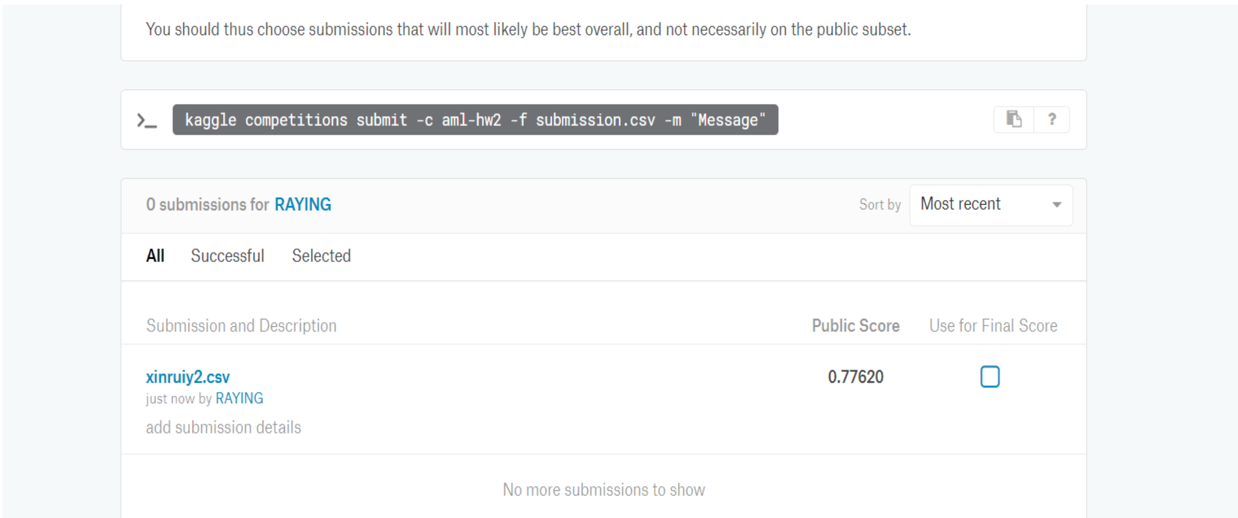
The best accuracy is 0.77620

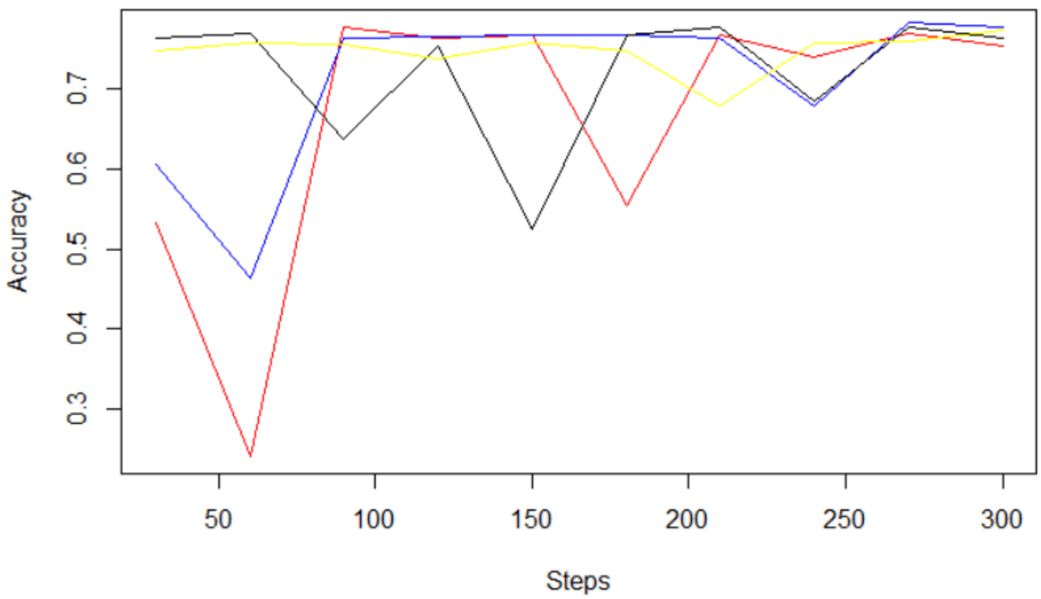


Red line: lambda = 0.001

Blue line: lambda = 0.01

Black line: lambda = 0.1

Yellow line: lambda = 1

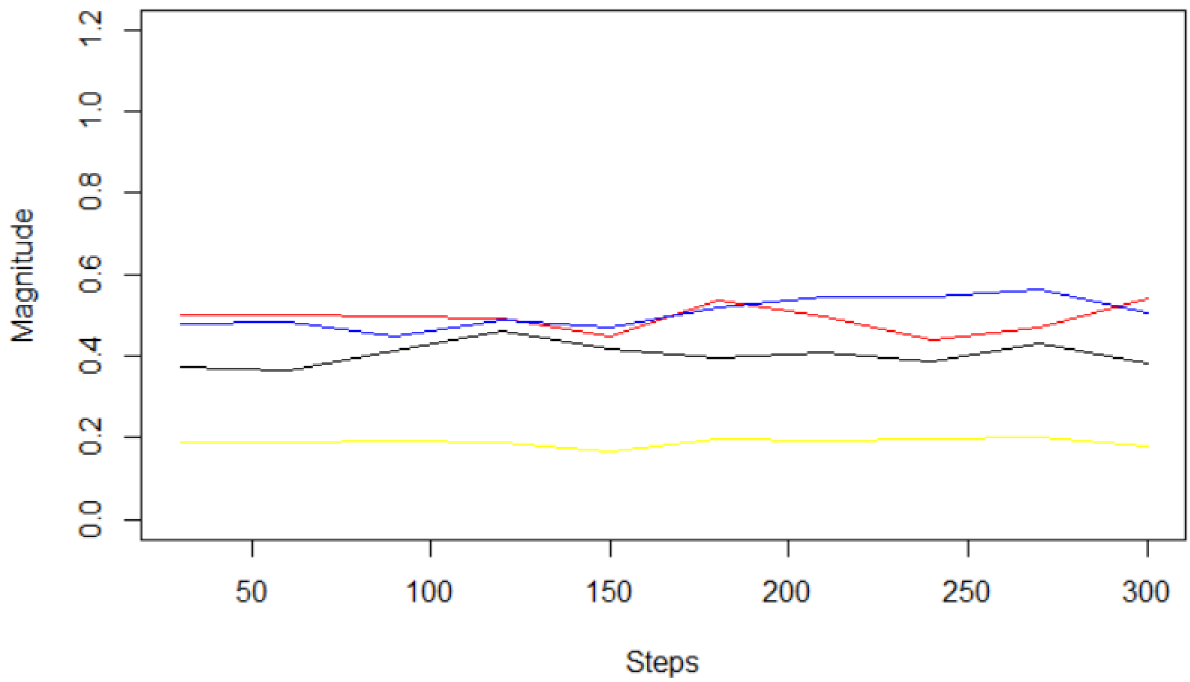


Red line: lambda = 0.001

Blue line: lambda = 0.01

Black line: lambda = 0.1

Yellow line: lambda = 1



I choose lambda = 0.01, and the learning rate I choose for is the after 300th step using lambda = 0.01. It seems that I have better accuracy over lambda = 0.01 in general. When I test on the validation set, the 300th step for lambda = 0.01 have the best accuracy. The truth is after all 300th steps, the accuracy is all pretty convincing to me.

```r
#normalize train
train$V1 = (as.double(train$V1) - col_1_mean)/as.double(sqrt(col_1_var))
var(as.double(train$V1))
col_1_mean = mean(as.double(train$V1))
train$V1 =as.double(train$V1)- col_1_mean


train$V3 = (as.double(train$V3) - col_3_mean)/as.double(sqrt(col_3_var))
var(as.double(train$V3))
col_3_mean = mean(as.double(train$V3))
train$V3 =as.double(train$V3)- col_3_mean

train$V5 = (as.double(train$V5) - col_5_mean)/as.double(sqrt(col_5_var))
var(as.double(train$V5))
col_5_mean = mean(as.double(train$V5))
train$V5 =as.double(train$V5)- col_5_mean

train$V11 = (as.double(train$V11) - col_11_mean)/as.double(sqrt(col_11_var))
var(as.double(train$V11))
col_11_mean = mean(as.double(train$V11))
train$V11 =as.double(train$V11)- col_11_mean

train$V12 = (as.double(train$V12) - col_12_mean)/as.double(sqrt(col_12_var))
var(as.double(train$V12))
col_12_mean = mean(as.double(train$V12))
train$V12 =as.double(train$V12)- col_12_mean

train$V13 = (as.double(train$V13) - col_13_mean)/as.double(sqrt(col_13_var))
var(as.double(train$V13))
col_13_mean = mean(as.double(train$V13))
train$V13 =as.double(train$V13)- col_13_mean
```

```r
vector_b[1] = -0.52242
vector_lambda = vector(mode = "double", 4)
vector_lambda = c(1e-3, 1e-2, 1e-1, 1)
vector_a = vector(mode = "double", 6)
vector_a = c(0.15835, -0.01414, -0.12151, 0.19379, 0.11909, 0.14248)
```

#magnitutde function

```{R}
vector_mag = vector(mode = "double",40)
for(i in 1:40){
  result = sqrt(as.numeric(df1[i,1])*as.numeric(df1[i,1])+as.numeric(df1[i,2])*as.numeric(df1[i,2])+as.numeric(df1[i,3])*as.numeric(df1[i,3])+as.numeric(df1[i,4])*as.numeric(df1[i,4])+as.numeric(df1[i,5])*as.numeric(df1[i,5])+as.numeric(df1[i,6])*as.numeric(df1[i,6]))
  vector_mag[i] = result
}
vector_mag
```

#test for 30 rounds per time

```{r}
library("dplyr")
  for(i in 1:30){
    mat_s = sample_n(train_fvector, size = 50)
    eta = m/(i+n)
    p_a = vector(mode = "double", 6)
    p_b = 0
    for(k in 1:50){
      p_a_1 = (gradient_a(vector_lambda[4], vector_a, mat_s[k,1:6], mat_s[k,8], vector_b[1]))
      p_b = p_b + gradient_b(vector_lambda[4], vector_a, mat_s[k,1:6], mat_s[k,8], vector_b[1])
      p_a = p_a + p_a_1
    }
    p_a = -p_a/50 - vector_lambda[4]*vector_a
    p_b = -p_b/50 - vector_lambda[4]*vector_b[1]
    vector_a = (vector_a + p_a*eta)
    vector_b[1] = vector_b[1] + eta*p_b
  }
multiply = function(a){
  count = 0
  for(i in 1:4389){
    val_needue_unique = as.numeric(df1[a,1]) * as.numeric(val_need[i,1]) + as.numeric(df1[a,2]) * as.numeric(val_need[i,2]) + as.numeric(df1[a,3]) * as.numeric(val_need[i,3]) + as.numeric(df1[a,4]) * as.numeric(val_need[i,4]) + as.numeric(df1[a,5]) * as.numeric(val_need[i,5]) + as.numeric(df1[a,6]) * as.numeric(val_need[i,6])  + as.numeric(df2[a,2])
    if(val_needue_unique>=0 && val_need[i,8] == 1){count = count + 1}
    else if(val_needue_unique<0 && val_need[i,8] == -1){count = count+1}
  }
  return(count/4389)
}
```