Josh Balingit Code - Predicting Movie Ratings based on Reviews

Josh Balingit

2023-08-26

```
# SET WORKING DIRECTORY
setwd("C:/Users/Josh Balingit/OneDrive/Desktop/STA 141C Final Project")
```

```
# TRAIN SUMMARY
df = read.csv("df_train_raw.csv") # This is Training Data
df = df[,!(colnames(df) %in% "X")]
mean(df$rating == 1) # Percent of Positive Reviews
```

```
## [1] 0.488
```

```
mean(df$rating == 0) # Percent of Negative Reviews
```

```
## [1] 0.512
```

```
# GENERATE CORPUS
library("tm")        # Text Mining
```

```
## Warning: package 'tm' was built under R version 4.1.3
```

```
## Loading required package: NLP
```

```
library("SnowballC")  # Text Stemming (Text Lemmatizing has issues with adverbs ending with "ly")
generate_corpus = function(reviews){
  review_no_special = foreach(i = reviews, .combine = "rbind") %dopar% {
    no_apostophe = gsub("'", "", i)
      # Possessive  : Jack's to Jacks
      # Contraction : would've to wouldve
      # Title       : 'Harry Potter' to Harry Potter
    no_punctuation = gsub("[[:punct:]]", " ", no_apostophe)
  }
  corpus = Corpus(VectorSource(review_no_special))
  corpus = tm_map(corpus, content_transformer(tolower))
  corpus = tm_map(corpus, content_transformer(removeNumbers))
  corpus = tm_map(corpus, removeWords, c(stopwords("english"), "film", "films", "movie", "movies"))
    # Assume "film", "films", "movie", "movies" WON'T help distinguish
    # They appear frequently across POS and NEG
    # This is b/c data has reviews about film/movies in general
  corpus = tm_map(corpus, stemDocument)
  return(corpus)
}
```

```r
# DOCUMENT TERM MATRIX WITH TF-IDF WEIGHT ON TRAIN DATA
library("foreach")      # Parallel Computing
```

```
## Warning: package 'foreach' was built under R version 4.1.3
```

```r
library("doParallel")   # Parallel Computing
```

```
## Warning: package 'doParallel' was built under R version 4.1.3

## Loading required package: iterators

## Warning: package 'iterators' was built under R version 4.1.3

## Loading required package: parallel
```

```r
corpus = generate_corpus(df$review)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered

## Warning in tm_map.SimpleCorpus(corpus, content_transformer(tolower)):
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, content_transformer(removeNumbers)):
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, removeWords, c(stopwords("english"), :
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, stemDocument): transformation drops
## documents
```

```r
td = DocumentTermMatrix(corpus)
train_terms = Terms(td)
td =  td[,sort(train_terms)]
train_idf = log2(nDocs(td)/colSums(as.matrix(td>0)))
mat_td = t(t(as.matrix(td))*train_idf)
mat_td_std = scale(mat_td)
  # Standardizing columns with mean 0 and sd 1 is necessary for PCA
    # With mean 0, we can use SVD decomposition (A. Chandler MAT 167)
    # With sd 1, we prevent variance of each variable from being inflated/deflated (E. Furfaro STA 141A
    # Empirically shown to lead to faster convergence for gradient descent (K. Balasubramanian STA 142A
```

```r
# WORD CLOUD ON TRAIN DATA
library("wordcloud")    # Word Cloud Graph
```

```
## Warning: package 'wordcloud' was built under R version 4.1.3

## Loading required package: RColorBrewer

## Warning: package 'RColorBrewer' was built under R version 4.1.3
```

```r
library("RColorBrewer") # Word Cloud Color Palettes
word_cloud_visual = function(rating,max_words,color_gradient){
  mat_td_rating = mat_td_std[df$rating == rating,]
  stem = colnames(mat_td_rating)
  stem_size = colSums(mat_td_rating)
    # In this case, size does NOT refer to stem frequency across documents
    # We are taking column sums as a measure to compare stem given some rating
      # LARGE column sums => Stem is connected to rating
      # SMALL column sums => Stem is NOT connected to rating
  wc_plot = wordcloud(words = stem,
                      freq = stem_size,
                      max.words=max_words,
                      random.order=F,
                      colors = color_gradient,
                      scale = c(2,.1))
  print(wc_plot)
}
word_cloud_visual(1,100,c("lightgreen","green2","green4"))
```


```
## NULL
```

```r
word_cloud_visual(0,100,c("lightpink","red2","red4"))
```


```
## NULL
```

```r
# POWER METHOD FUNCTION TO FIND SINGULAR VALUES
singular_vals_power_method = function(num_pc,num_iter,q0,X){
  A = tcrossprod(X)
  u_vecs = c()
  eigenval = c()
  for(j in 1:num_pc){
    q = as.matrix(q0,nrow = u_length)
    for(i in 1:num_iter){
      z = A%*%q[,i]
      q = cbind(q,z/norm(z,type = "2"))
      if(min(norm(q[,i+1]-q[,i],type = "2"),
             norm(q[,i+1]+q[,i],type = "2")) < 1e-6){
        cat("Convergence at trial", i, "\n")
        break
      }
    }
    u_vecs = cbind(u_vecs,q[,ncol(q)])
    eigenval = append(eigenval,crossprod(u_vecs[,j],X)%*%crossprod(X,u_vecs[,j]))
      # (A%*%B)%*%x cost more flops than A%*%(B%*%x)
      # eigenval of tcrossprod(mat_td_std)
      # sqrt(eigenval) of tcrossprod(mat_td_std) = singval of mat_td_std
    A = A - eigenval[j]*tcrossprod(u_vecs[,j])
  }
```

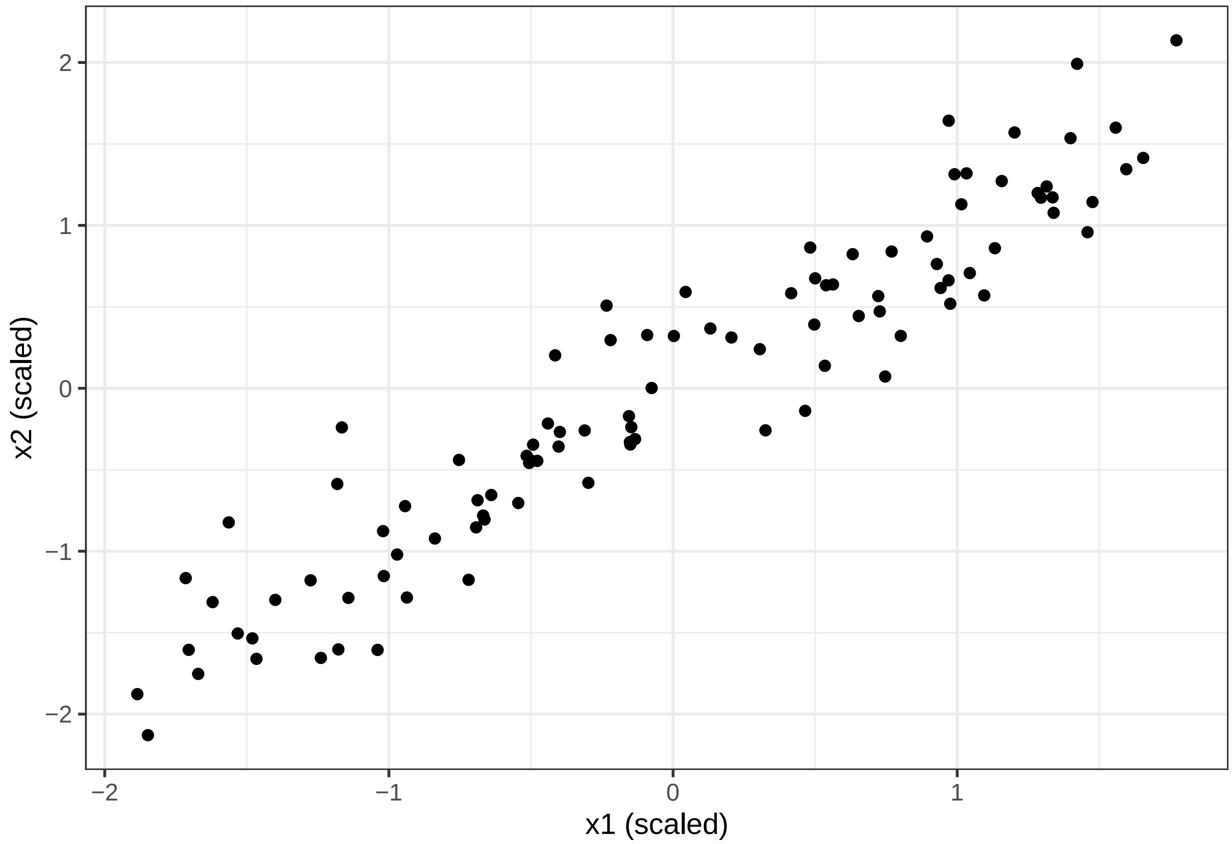
```
  return(sqrt(eigenval))
}
  # Power Method is used to find eigenvalues of A
  # If A is Xt_X, then square root of eigenvalues of A are singular values of X
```

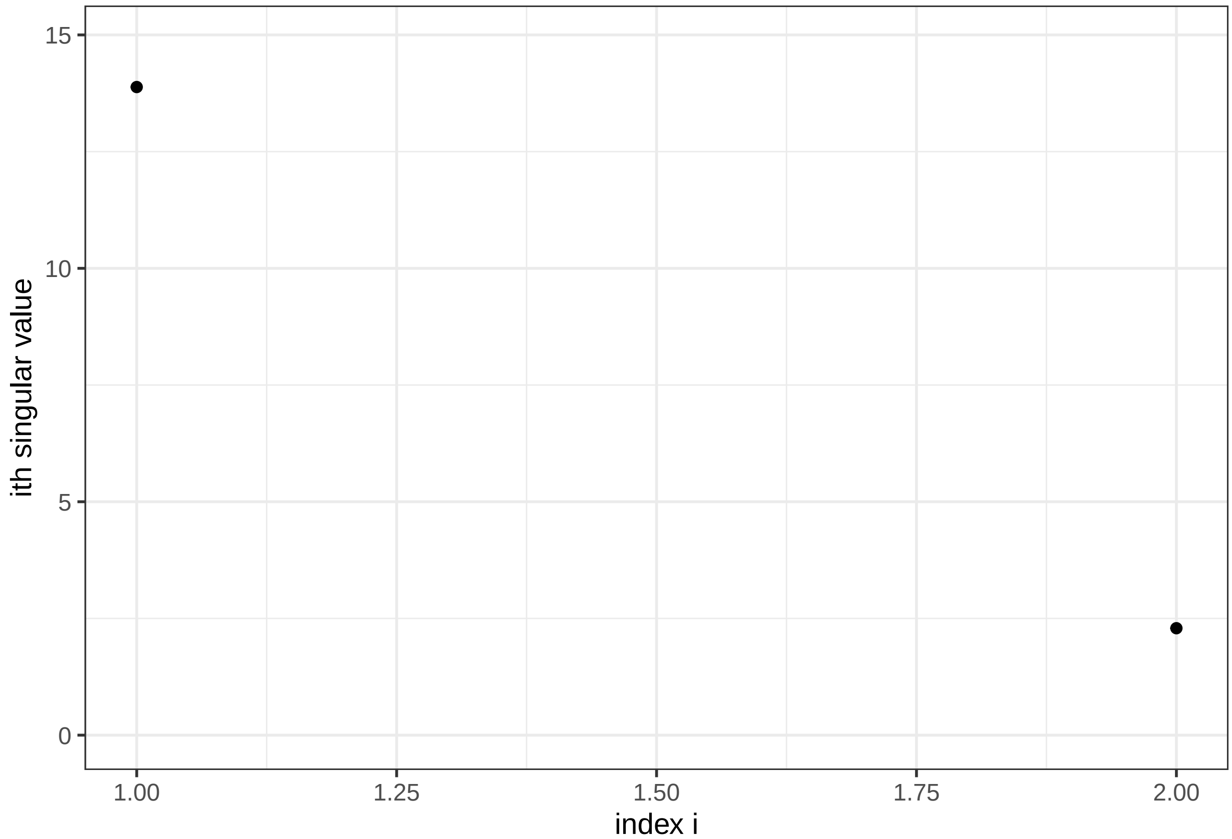
```
## GGPLOT FOR DATA VISUALIZATION
library("ggplot2")
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##     annotate
```

```
# SIMULATION FOR PCA
  # TO TEST IF POWER METHOD FUNCTION WORKS
  # TO ILLUSTRATE ISSUE OF PCA WITH TRAIN DATA
pca_sim = function(X_sim){
  # Plot of x1 vs x2
  plot_x1_x2 = ggplot(data=as.data.frame(X_sim))+
    geom_point(aes(x=x1_sim,y=x2_sim))+
    labs(x="x1 (scaled)",y="x2 (scaled)")+
    theme_bw()
  num_pc_sim = 2
  num_iter_sim = 10000
  u_length_sim = nrow(X_sim)
  z0_sim = rep(1,times=u_length_sim)
  q0_sim = z0_sim/norm(z0_sim,type = "2")
  singular_vals_sim = singular_vals_power_method(num_pc_sim,num_iter_sim,q0_sim,X_sim)
  # Plot of i vs ith singular value
  plot_i_sv = ggplot(data=data.frame("num_pc"=1:num_pc_sim,"singular_vals"=singular_vals_sim))+
    geom_point(aes(x=num_pc,y=singular_vals))+
    ylim(c(0,max(singular_vals_sim)+1))+
    labs(x="index i",y="ith singular value")+
    theme_bw()
  print(plot_x1_x2)
  print(plot_i_sv)
}
# CASE 1: HIGH CORRELATED COVARIATES
set.seed(1)
x1_sim = runif(100,min=-10,max=10)
x2_sim = 0.5*x1_sim + rnorm(100,mean=0,sd=1)
X_sim = scale(cbind(x1_sim,x2_sim))
pca_sim(X_sim)
```

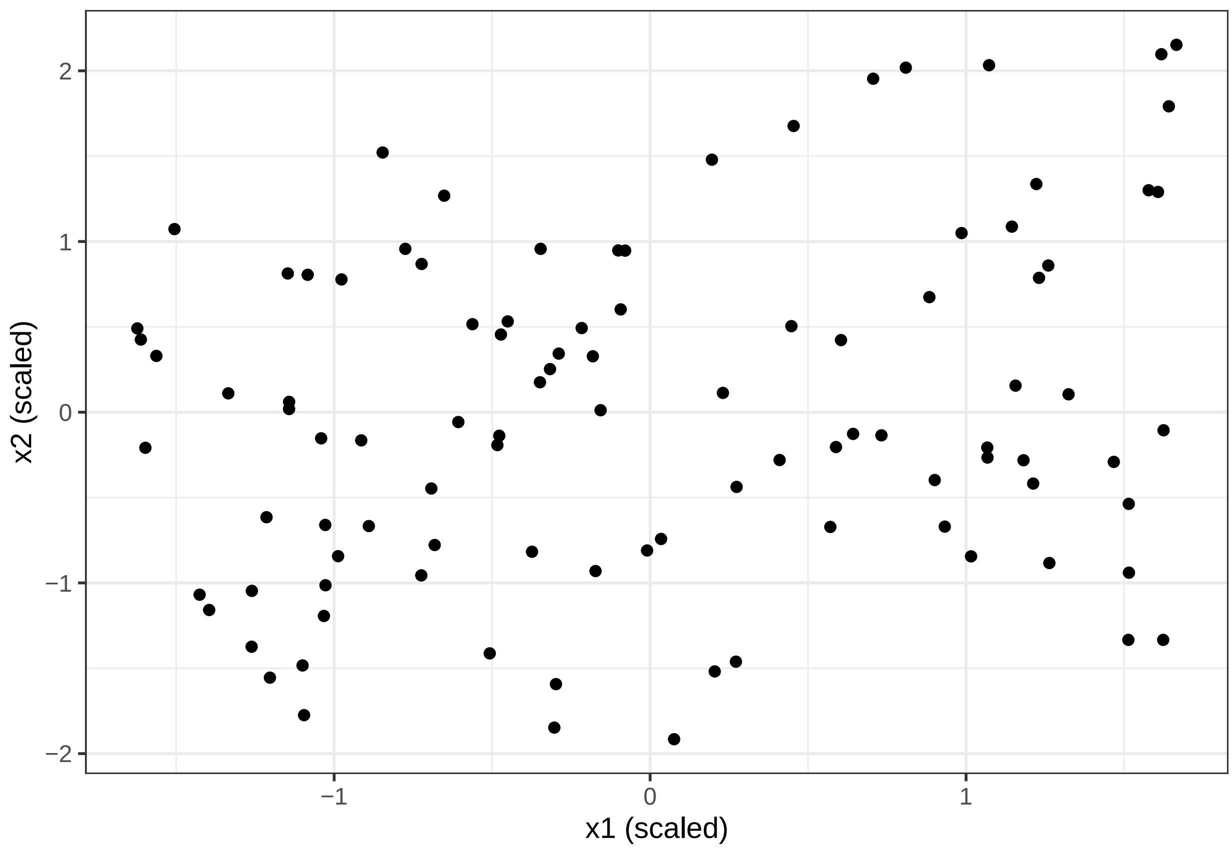
```
## Convergence at trial 5
## Convergence at trial 3
```

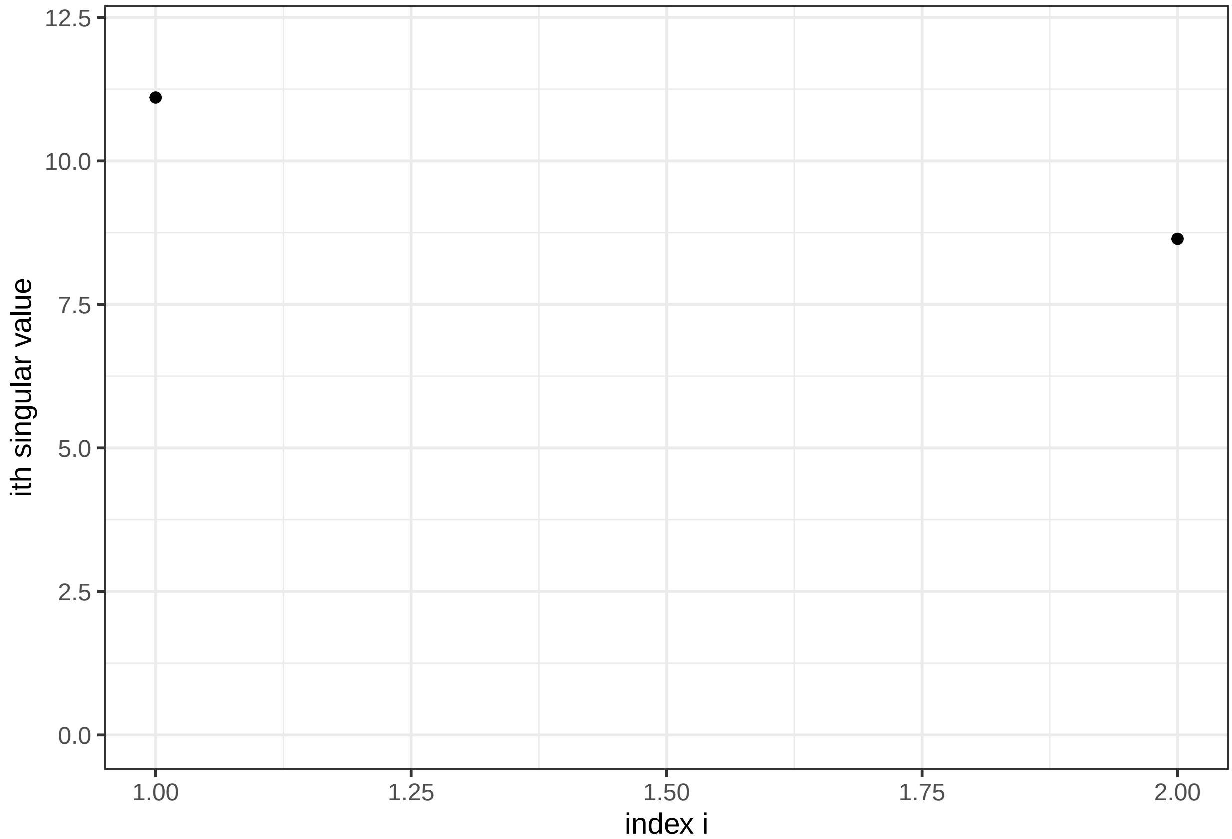

x2 (scaled)
2
1
0
−1
−2
−2
−1
0
1
x1 (scaled)




```
# CASE 2: WEAK CORRELATED COVARIATES
set.seed(2)
x1_sim = runif(100,min=-10,max=10)
x2_sim = 0.05*x1_sim + rnorm(100,mean=0,sd=1)
X_sim = scale(cbind(x1_sim,x2_sim))
pca_sim(X_sim)
```

```
## Convergence at trial 30
## Convergence at trial 3
```


x2 (scaled)
2
1
0
−1
−2
−1
0
1
x1 (scaled)

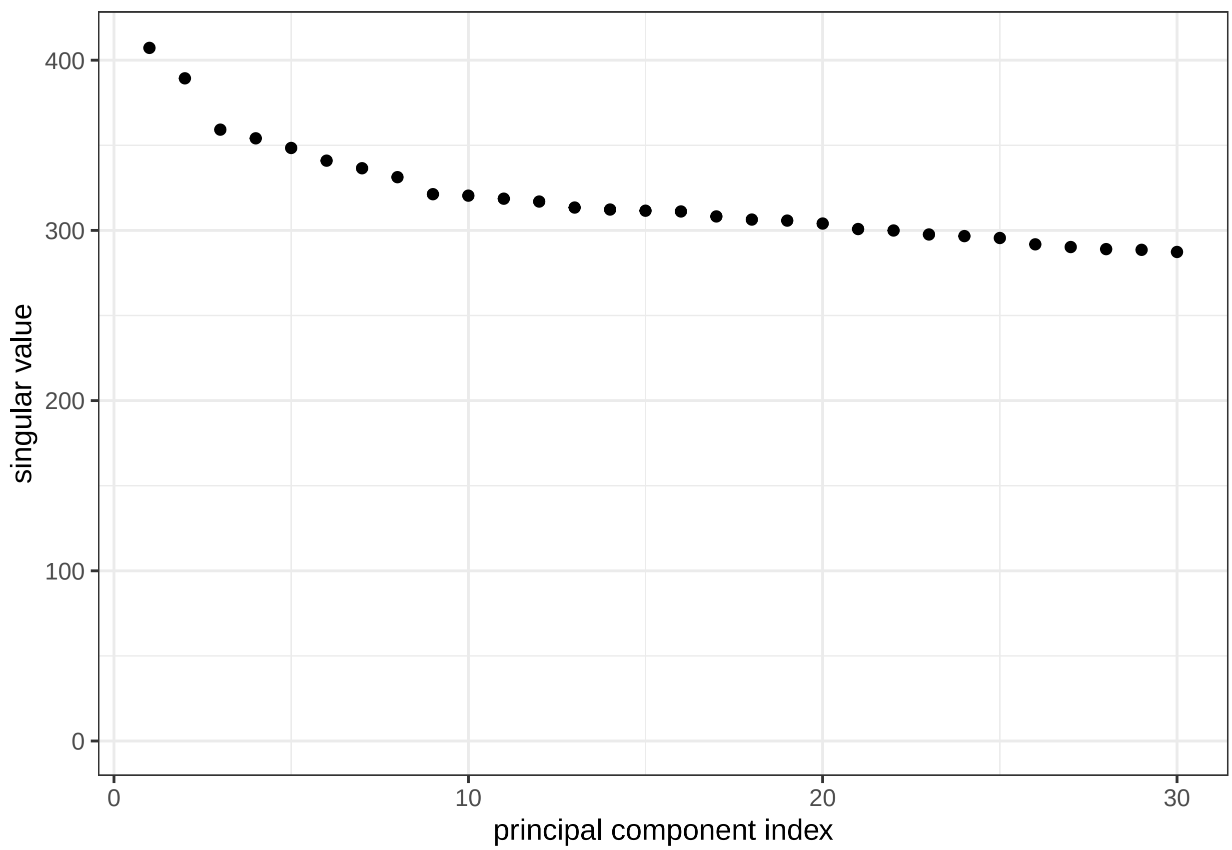



```r
# PCA (NOTE: TOO LONG TO RUN)
num_pc = 30
num_iter = 10000
u_length = nrow(mat_td_std)
z0 = rep(1, times = u_length)
q0 = z0/norm(z0,type = "2")
singular_vals = singular_vals_power_method(num_pc,num_iter,q0,mat_td_std)
```

```
## Convergence at trial 128
## Convergence at trial 57
## Convergence at trial 363
## Convergence at trial 305
## Convergence at trial 253
## Convergence at trial 394
## Convergence at trial 347
## Convergence at trial 183
## Convergence at trial 1282
## Convergence at trial 657
## Convergence at trial 1364
## Convergence at trial 457
## Convergence at trial 1077
## Convergence at trial 2018
## Convergence at trial 2960
## Convergence at trial 484
## Convergence at trial 908
```

```
## Convergence at trial 1971
## Convergence at trial 793
## Convergence at trial 498
## Convergence at trial 1417
## Convergence at trial 665
## Convergence at trial 1289
## Convergence at trial 1239
## Convergence at trial 332
## Convergence at trial 937
## Convergence at trial 981
## Convergence at trial 2694
## Convergence at trial 1164
## Convergence at trial 1224
```

```
ggplot(data=data.frame("num_pc"=1:num_pc,"singular_vals"=singular_vals))+
    geom_point(aes(x=num_pc,y=singular_vals))+
    ylim(c(0,max(singular_vals)+1))+
    labs(x="principal component index",y="singular value")+
    theme_bw()
```




```
  # Based on simulation for PCA,
    # Weak Correlated Variables in mat_td_std
    # Points of mat_td_std "look less correlated" given Dimension of mat_td_std is large
    # Correlation is likely "hidden" by addition "noise" stem terms
```

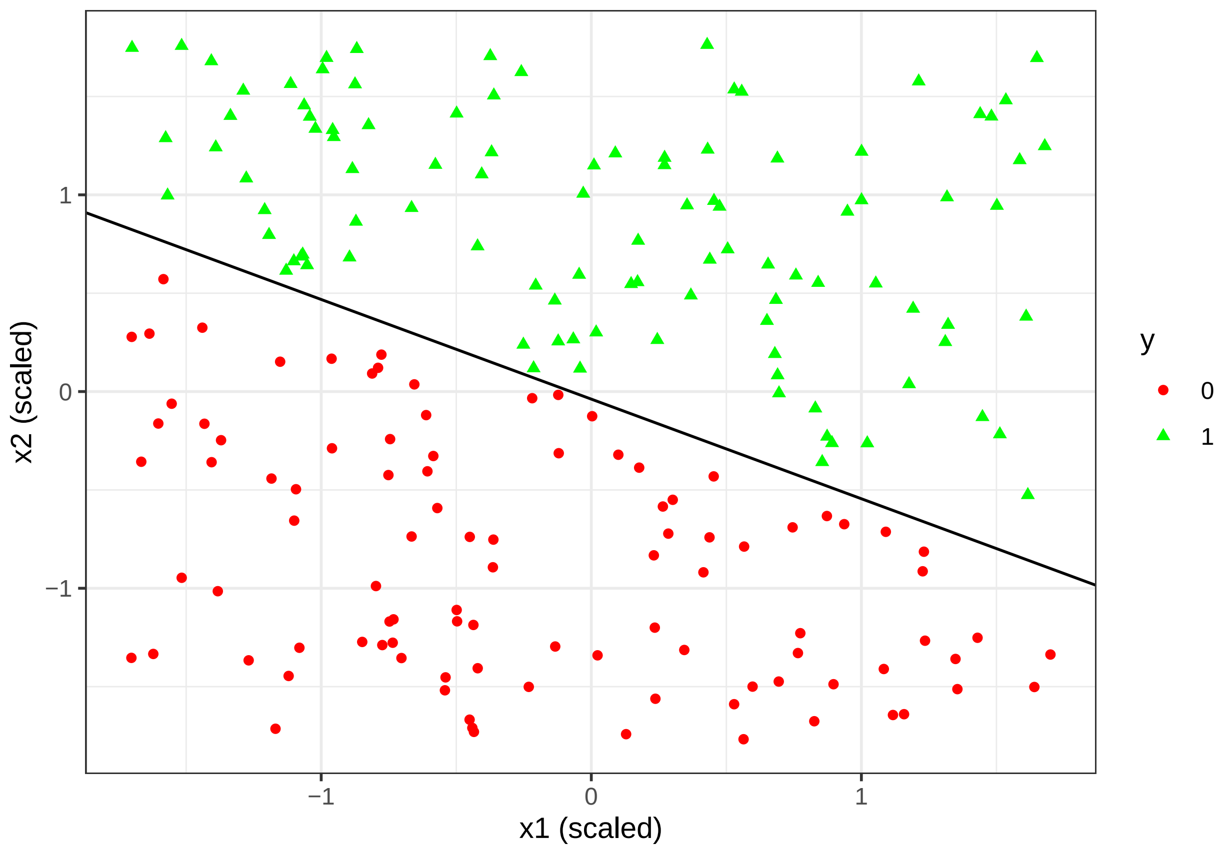
```
# GRADIENT DESCENT FUNCTION FOR LOGISTIC REGRESSION
exp_neg_X_beta = function(X,beta){
  exp = exp(-X%*%beta)

  # To prevent underflow
  smallest_double = .Machine$double.eps
  exp = ifelse(exp < smallest_double, smallest_double,exp)

  return(exp)
}
h_beta = function(X,beta){
  return(1/(1+exp_neg_X_beta(X,beta)))
}
loss = function(y,X,beta){
   return(-sum(y*log(h_beta(X,beta))+
              (1-y)*log(1-h_beta(X,beta))))
}
gradient_descent = function(X,y,num_iter,tol,alpha,beta_vecs){
  for(i in 1:num_iter){
    h_y = h_beta(X,beta_vecs[,i]) - y
    gradient = crossprod(X,h_y)
    beta_vec_new = beta_vecs[,i] - alpha*gradient

    # While loss increase, descrease learning rate (Assistance From ChatGPT)
    if(loss(y,X,beta_vec_new) > loss(y,X,beta_vecs[,i])) {
      alpha = alpha / 2
    }
    beta_vecs = cbind(beta_vecs,beta_vec_new)

    if(norm(beta_vecs[,i+1]-beta_vecs[,i],type = "2") < tol){
      converge_trial = i
      return(list("beta" = beta_vecs[,ncol(beta_vecs)],
                  "converge_trial" = converge_trial))
    }
  }
  return(list("beta" = beta_vecs[,ncol(beta_vecs)],
              "converge_trial" = "NO CONVERGE"))
}
```

```
# TEST RESULTS FOR LOGISTIC REGRESSION
test_results_logistic = function(X_test,y_test,beta_hat_train,n){
  log_odds_pred = X_test%*%beta_hat_train
  y_pred = ifelse(log_odds_pred >= 0, 1, 0)
  o_v_p = table("observed" = y_test,"predicted" = y_pred)
  error_rate = mean(y_test != y_pred)
  return(list("o_v_p" = o_v_p,
              "error_rate" = error_rate))
}
```

```
# SIMULATION FOR GRADIENT DESCENT FOR LOGISTIC REGRESSION
  # TO TEST IF GRADIENT DESCENT FUNCTION FOR LOGISTIC REGRESSION WORKS
gd_logisitic_sim = function(X_scale_sim,y_sim){
  n = nrow(X_scale_sim)
```

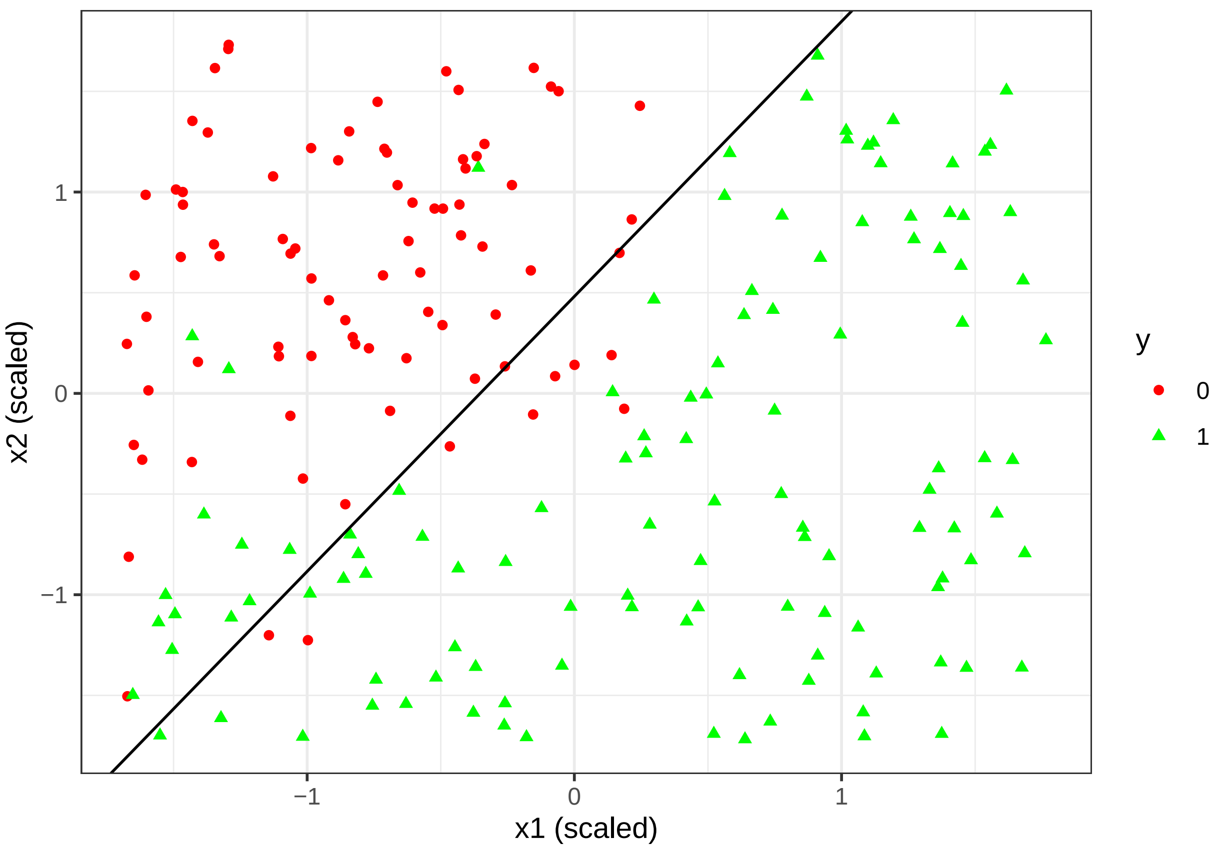
```r
  p = ncol(X_scale_sim)
  # CREATE 80% TRAIN AND 20% TEST SPLIT
  test_index_sim = sample(1:n,size=n/5)
  X_test_sim = X_scale_sim[test_index_sim,]
  y_test_sim = y_sim[test_index_sim]
  train_index_sim = -test_index_sim
  X_train_sim = X_scale_sim[train_index_sim,]
  y_train_sim = y_sim[train_index_sim]
  # FIT LOGISTIC MODEL USING GRADIENT DESCENT WITH TRAIN
  num_iter_sim = 100000
  tol_sim = 1e-6
  alpha_sim = 0.1
  beta_vec_0_sim = rep(0,times=p)
  beta_vecs_sim = as.matrix(beta_vec_0_sim)
  gd_results_sim = gradient_descent(X_train_sim,y_train_sim,num_iter_sim,tol_sim,alpha_sim,beta_vecs_si
  beta_hat_train_sim = gd_results_sim$beta
  # CALCULATE TEST ERROR RATE USING TRAIN LOGISTIC MODEL
  test_results_sim = test_results_logistic(X_test_sim,y_test_sim,beta_hat_train_sim,n=length(y_test_sim
  error_rate_sim = test_results_sim$error_rate
  # VISUAL OF TEST DATA POINTS WITH TRAIN DECISION BOUNDARY
  plot_sim = ggplot(data=data.frame("x1"=X_test_sim[,2],"x2"=X_test_sim[,3],"y"=as.factor(y_test_sim)))
    geom_point(aes(x=x1,y=x2,col=y,shape=y))+
    scale_color_manual(values = c("red","green"))+
    geom_abline(intercept = -beta_hat_train_sim[1]/beta_hat_train_sim[3],
                slope=-beta_hat_train_sim[2]/beta_hat_train_sim[3])+
    labs(x="x1 (scaled)",y="x2 (scaled)")+
    theme_bw()
  # SUMMARY
  cat("Train Beta is", beta_hat_train_sim, "\n")
  cat("Test Error Rate using Train Model is", error_rate_sim)
  print(plot_sim)
}
# CASE 1: GENERATE DATA BASED ON LINEAR DECISION BOUNDARY AND LOGISITIC PROBABILITIES
set.seed(3)
beta_sim = c(1,2,4)
xs_sim = sapply(1:2,function(i){
  return(runif(1000,min=-10,max=10))
})
X_sim = cbind(1,xs_sim[,1],xs_sim[,2])
prob_y_1_sim = h_beta(X_sim,beta_sim)
y_sim = sapply(prob_y_1_sim,function(i){
  return(rbinom(n=1,size=1,prob=i))
})
X_scale_sim = cbind(1,scale(X_sim[,2:ncol(X_sim)]))
gd_logisitic_sim(X_scale_sim,y_sim)
```

```
## Train Beta is 0.8379614 11.05075 21.81778
## Test Error Rate using Train Model is 0
```




```
  # TEST error is LOW
    # This makes sense because our data was generated based on logistic regression assumption
  # TRAIN beta does NOT equal TRUE beta
    # This makes sense because we scaled our data
    # This is fine because our goal is to NOT estimate TRUE beta
    # This is fine because our goal is to have LOW TEST error
# CASE 2: GENERATE DATA BASED ON EXPONENTIAL DECISION BOUNDARY AND ARCTAN PROBABILITES
set.seed(4)
xs_sim = sapply(1:2,function(i){
  return(runif(1000,min=-10,max=10))
})
model_sim = -4+2*exp(xs_sim[,1])-xs_sim[,2]
prob_y_1_sim = (atan(model_sim)+pi/2)/pi
y_sim = sapply(prob_y_1_sim,function(i){
  return(rbinom(n=1,size=1,prob=i))
})
X_scale_sim = cbind(1,scale(cbind(xs_sim[,1],xs_sim[,2])))
gd_logisitic_sim(X_scale_sim,y_sim)
```

```
## Train Beta is 1.042006 2.954612 -2.163468
## Test Error Rate using Train Model is 0.11
```




```
# TEST error is HIGHER
  # This makes sense because our data was NOT generated based on logistic regression assumption
```

```
# GENERATE STANDARDIZED DATA WITH RATING COLUMN (REDUCED TO TOP POS AND NEG STEM)
generate_df_std_red = function(stem_size_each,mat_td_std,rating){
  top_stem_each = sapply(0:1,function(i){
    mat_td_rating = mat_td_std[df$rating == i,]
    stem = colnames(mat_td_rating)
    stem_size = colSums(mat_td_rating)
    stem_top = head(stem[order(stem_size,decreasing = T)],stem_size_each)
    stem_top
  })
  stem1_top = top_stem_each[,2]
  stem0_top = top_stem_each[,1]
  top_stem = c(stem1_top[!(stem1_top %in% stem0_top)],
               stem0_top[!(stem0_top %in% stem1_top)])
    # Remove intersection between two ratings
  stem1_num = sum(stem1_top %in% top_stem)
  stem0_num = sum(stem0_top %in% top_stem)
  mat_td_std_red = mat_td_std[,top_stem]
    # Head stem_size_each columns are pos
    # Tail stem_size_each columns are neg
  df_std_red = as.data.frame(mat_td_std_red)
  df_std_red = cbind("beta0" = 1,df_std_red)
  df_std_red$rating = rating
  return(list("df_std_red" = df_std_red,
```

```
              "top_stem" = top_stem,
              "stem1_num" = stem1_num,
              "stem0_num" = stem0_num))
}
```

```
# TRAIN STANDARDIZED DATA WITH RATING COLUMN (REDUCED TO TOP POS AND NEG STEM)
start = proc.time()
test_error_rate_estimates = foreach(stem_size_each = c(400,600,800,1000),.combine = "rbind", .packages =
  df_std_red_results = generate_df_std_red(stem_size_each,mat_td_std,df$rating)
  df_std_red = df_std_red_results$df_std_red
  top_stem = df_std_red_results$top_stem
  stem1_num = df_std_red_results$stem1_num
  stem0_num = df_std_red_results$stem0_num
    # Extract stem size for each rating again b/c function removes intersection between two ratings
  num_betas = 1 + stem1_num + stem0_num
    # 1 b/c of intercept
  # K-FOLDS CROSS VALIDATION WITH GRADIENT DESCENT FOR LOGISTIC REGRESSION
  k_folds = 5
  k_sizes = nrow(df_std_red)/k_folds
  k_labels = rep(1:k_folds,each=k_sizes)
  k_df = split(df_std_red, k_labels)
    # By converting to mat_td_std_red to df_std_red, I preserve row and column names
  k_error_rate_valid = foreach(i=1:k_folds,.combine = "rbind")%dopar%{
    df_train_valid = k_df[[i]]
    X_train_valid = as.matrix(df_train_valid[,-(num_betas+1)])
    y_train_valid = df_train_valid[,(num_betas+1)]
    df_train_valid_index = as.integer(rownames(k_df[[i]]))
    df_train_fit = df_std_red[-df_train_valid_index,]
    X_train_fit = as.matrix(df_train_fit[,-(num_betas+1)])
    y_train_fit = df_train_fit[,(num_betas+1)]
    num_iter = 10000
    tol = 1e-3
    alpha = 0.01
    beta_vec_0 = c(0,rep(c(1,-1),times=c(stem1_num,stem0_num)))
      # log(p1/p0) = beta_0 + beta_1*x_1 + ... beta_(stem1_num+stem0_num)*x_(stem1_num+stem0_num)
      # log(p1/p0) > 0 <=> Pos Review <=> beta_i for i = 1, ... stem1_num are Pos
      # log(p1/p0) < 0 <=> Neg Review <=> beta_i for i = stem1_num+1, ... stem1_num+stem0_num are Neg
    beta_vecs = as.matrix(beta_vec_0)
    gd_results = gradient_descent(X_train_fit,y_train_fit,num_iter,tol,alpha,beta_vecs)
    beta_hat_fit = gd_results$beta
    test_results = test_results_logistic(X_train_valid,y_train_valid,beta_hat_fit,k_sizes)
    error_rate = test_results$error_rate
    error_rate
  }
  test_error_rate_estimate = mean(k_error_rate_valid)
  test_error_rate_estimate
}
end = proc.time()
end-start
```

```
##    user  system elapsed
##  918.16  134.41 2888.74
```

```
# FIT LOGISTIC REGRESSION WITH TRAIN SET WITH CERTAIN NUM OF STEM
stem_size_each = 1000
  # Yield lowest CV error
df_std_red_results = generate_df_std_red(stem_size_each,mat_td_std,df$rating)
df_std_red = df_std_red_results$df_std_red
top_stem = df_std_red_results$top_stem
stem1_num = df_std_red_results$stem1_num
stem0_num = df_std_red_results$stem0_num
  # Extract stem size for each rating again b/c function removes intersection between two ratings
num_betas = 1 + stem1_num + stem0_num
df_train = df_std_red
X_train = as.matrix(df_train[,-(num_betas+1)])
y_train = df_train[,(num_betas+1)]
num_iter = 10000
tol = 1e-3
alpha = 0.01
beta_vec_0 = c(0,rep(c(1,-1),times=c(stem1_num,stem0_num)))
  # log(p1/p0) = beta_1*x_1 + ... beta_(stem1_num+stem0_num)*x_(stem1_num+stem0_num)
  # log(p1/p0) > 0 <=> Pos Review <=> beta_i for i = 1, ... stem1_num are Pos
  # log(p1/p0) < 0 <=> Neg Review <=> beta_i for i = stem1_num+1, ... stem1_num+stem0_num are Neg
beta_vecs = as.matrix(beta_vec_0)
start = proc.time()
gd_results = gradient_descent(X_train,y_train,num_iter,tol,alpha,beta_vecs)
end = proc.time()
time_info = end - start
time_passed = time_info["elapsed"]
print(time_passed)
```

```
## elapsed
##    93.6
```

```
beta_hat_train = gd_results$beta
```

```
# TEST SUMMARY
df_test = read.csv("df_test_raw.csv") # This is Test Data
df_test = df_test[,!(colnames(df_test) %in% "X")]
mean(df_test$rating == 1)
```

```
## [1] 0.524
```

```
mean(df_test$rating == 0)
```

```
## [1] 0.476
```

```
# CONVERTING TEST SET IN TERMS OF TRAIN SET
# Code between was created with assitance from ChatGPT
corpus_test = generate_corpus(df_test$review)
```

```
## Warning in tm_map.SimpleCorpus(corpus, content_transformer(tolower)):
## transformation drops documents
```

```
## Warning in tm_map.SimpleCorpus(corpus, content_transformer(removeNumbers)):
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, removeWords, c(stopwords("english"), :
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, stemDocument): transformation drops
## documents
```

```
td_test = DocumentTermMatrix(corpus_test, control = list(dictionary = train_terms))
td_test = td_test[,sort(train_terms)]
mat_td_test = t(t(as.matrix(td_test))*train_idf)
mat_td_std_test = scale(mat_td_test, center = colMeans(mat_td), scale = apply(mat_td, 2, sd))
# Code between was created with assitance from ChatGPT
X_test = cbind("beta0" = 1,mat_td_std_test[,top_stem])
  # Head stem_size_each columns are pos
  # Tail stem_size_each columns are neg
y_test = df_test$rating
```

```
# LOGISTIC REGRESSION EVALUATION
test_results = test_results_logistic(X_test,y_test,beta_hat_train,test_size)
error_rate = test_results$error_rate
print(error_rate)
```

```
## [1] 0.2
```