homework 3

Weifeng She

10/12/2016

Note: question 1 is writen in latex and is attached at the last of the document. All other question is compiled by R markdown.

2. Implement in R logistic regression based on gradient descent. To avoid unnecessary slowdown use vectorized code when computing the gradient (avoid loops).

```
# import all the packages need for this homework
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
##
## The following objects are masked _by_ '.GlobalEnv':
##
##     diamonds, mpg
```

```
library(reshape)
library(mlbench)
library(glm2)
library(ROCR)
```

```
## Loading required package: gplots
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(Amelia)
```

```
## Loading required package: Rcpp
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.4, built: 2015-12-05)
## ## Copyright (C) 2005-2016 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
gradient_descent <- function(X, Y, eta = 0.01, max_iteration = 20000, epsilon = 0.0001) {

        theta <- replicate(ncol(X), 0.0)

        for(i in 1:  max_iteration)
```

```
        {
                z <- as.matrix(X_train) %*% theta
                H <- Y_train / (1 + exp(-Y_train * z))
                # gradient in-sample error
                er_in <- t(X_train) %*% H / nrow(X_train)
                theta_new <- theta - eta * er_in

                if(sum(abs(theta_new - theta)) <= epsilon) {
                                print(i)
                               break;}

                theta <- theta_new

                        }
        theta
  }
```

3. Train and evaluate your code on the BreastCancer data from the mlbench R package.Specifically, randomly divide the dataset into 70% for training and 30% for testing and train on the training set and report your accuracy (fraction of times the model made a mistake) on the train set and on the test set. Repeat the random partition of 70% and 30% 10 times and average the test accuracy results over the 10 repetitions. Try several different selections of starting positions - did this change the parameter value that the model learned? Try to play with different convergence criteria to get better accuracy.

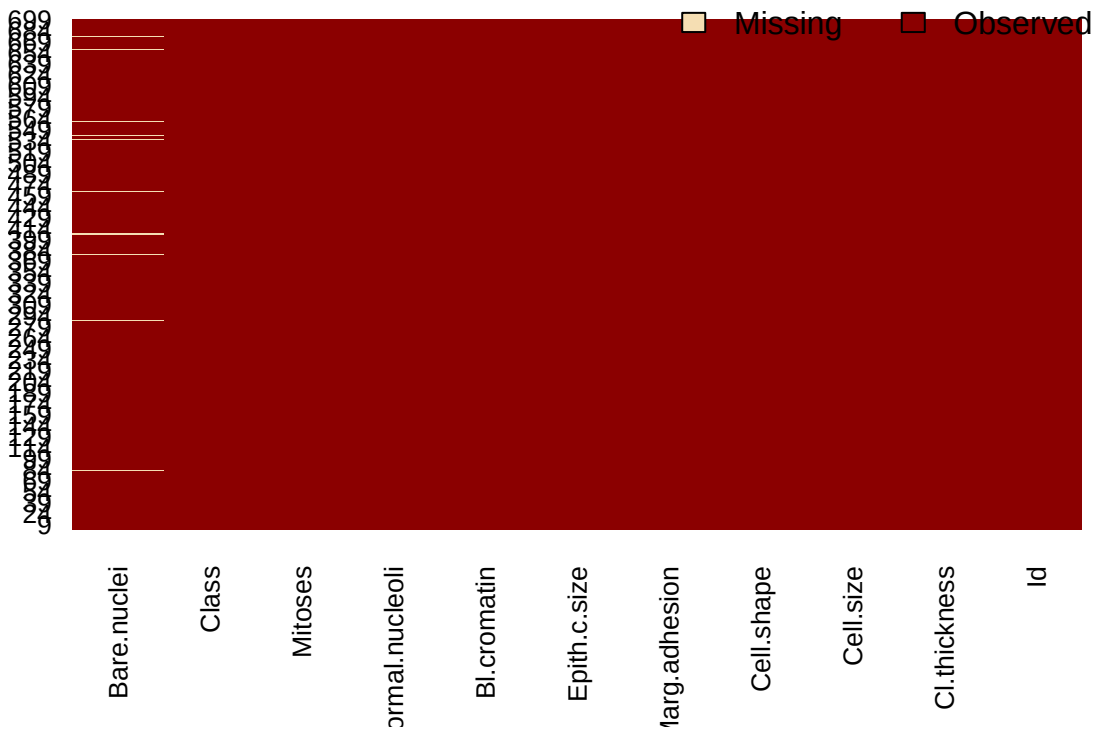
```
options(warn = -1) # used to suppress warning from glm2 not converaging

data(BreastCancer)
#dim(BreastCancer)
#head(BreastCancer)
# check the missing values
sapply(BreastCancer, function(x) sum(is.na(x)))
```

```
##              Id    Cl.thickness       Cell.size      Cell.shape
##               0               0               0               0
##   Marg.adhesion    Epith.c.size     Bare.nuclei     Bl.cromatin
##               0               0              16               0
## Normal.nucleoli         Mitoses           Class
##               0               0               0
```

```
# check the structure of the data
sapply(BreastCancer, function(x) length(unique(x)))
```

```
##              Id    Cl.thickness       Cell.size      Cell.shape
##             645              10              10              10
##   Marg.adhesion    Epith.c.size     Bare.nuclei     Bl.cromatin
##              10              10              11              10
## Normal.nucleoli         Mitoses           Class
##              10               9               2
```

```
# graph the missing value
missmap(BreastCancer, main = "Missing values vs observed")
```




```
# check the encoding for each column
sapply(BreastCancer, function(x) is.factor(x))
```

```
##              Id    Cl.thickness       Cell.size      Cell.shape
##           FALSE            TRUE            TRUE            TRUE
##   Marg.adhesion    Epith.c.size     Bare.nuclei     Bl.cromatin
##            TRUE            TRUE            TRUE            TRUE
## Normal.nucleoli         Mitoses           Class
##            TRUE            TRUE            TRUE
```

```
# we can see all the columns are factor variable, so we need to convert all of them except for last to

# convert the factor to numeric
for(i in 2:10){
  BreastCancer[, i] <- as.numeric(as.character(BreastCancer[, i]))
}

# convert missing value to the meaning
BreastCancer$Bare.nuclei[is.na(BreastCancer$Bare.nuclei)] <-
  mean(BreastCancer$Bare.nuclei, na.rm = T)

# make sure all NA is converted to numeric
sum(is.na(BreastCancer$Bare.nuclei))
```

```
## [1] 0
```

```r
# remove labelling column
BreastCancer$Id <- NULL
BreastCancer$Class <- ifelse(BreastCancer$Class == "malignant", 1, 0)

sample_size <- floor(0.7 * nrow(BreastCancer))
train_index <- sample(seq_len(nrow(BreastCancer)), size = sample_size)

# extract the feature columns
# set the intercept column
intercept <- replicate(nrow(BreastCancer), 1)
# estract all the feature columns and combine with intercept column
extracted_column <- ncol(BreastCancer) - 1
X <- data.matrix(cbind(intercept, BreastCancer[, 1:extracted_column]))
#head(X)
Y <- BreastCancer[, ncol(BreastCancer)]
#head(Y)
# convert Y to 1, -1 instead of 1 and 0
Y <- ifelse(Y == 1, 1, -1)
X_train <- data.matrix(X[train_index, ])
Y_train <- data.matrix(Y[train_index])
X_test <- data.matrix(X[-train_index, ])
Y_test <- data.matrix(Y[-train_index])

# train the model on
theta <- gradient_descent(X_train, Y_train)

train_predict <- 1 / (1 + exp(as.matrix(X_train) %*% theta) )
train_predict <- ifelse(train_predict >= 0.5, 1, -1)
# calculate the prediction accuracy on training set
print(paste("On single run, the accuracy on train data is", mean(train_predict == Y_train)), sep = " ")
```

```
## [1] "On single run, the accuracy on train data is 0.955010224948875"
```

```r
# predict the test data set
test_predict <- 1 / (1 + exp(as.matrix(X_test) %*% theta))
test_predict <- ifelse(test_predict >= 0.5, 1, -1)
# calculate the prediction accuracy on training set
print(paste("On single run, the accuracy on test data is", mean(test_predict == Y_test)), sep = " ")
```

```
## [1] "On single run, the accuracy on test data is 0.985714285714286"
```

```r
#Repeat the random partition of 70% and 30% 10 times and average the test accuracy results over the 10
train_accur = 0
test_accur = 0
for(j in 1:10) {
  train_index <- sample(seq_len(nrow(BreastCancer)), size = sample_size)
  X_train <- data.matrix(X[train_index, ])
  Y_train <- data.matrix(Y[train_index])
  X_test <- data.matrix(X[-train_index, ])
  Y_test <- data.matrix(Y[-train_index])
  #training model
  theta <- gradient_descent(X_train, Y_train)
```

```
  #print(theta)
  train_predict <- 1 / (1 + exp(as.matrix(X_train) %*% theta) )
  train_predict <- ifelse(train_predict >= 0.5, 1, -1)
  # calculate the prediction accuracy on training set
  train_accur = train_accur + mean(train_predict == Y_train)

  # predict the test data set
  test_predict <- 1 / (1 + exp(as.matrix(X_test) %*% theta))
  test_predict <- ifelse(test_predict >= 0.5, 1, -1)
  # calculate the prediction accuracy on training set
  test_accur =test_accur + mean(test_predict == Y_test)
    }

  mean_train_accur <- train_accur / 10
  mean_test_accur <- test_accur / 10

print(paste("after 10 runs, the average accuracy on train data is", mean_train_accur, sep = " "))
```

```
## [1] "after 10 runs, the average accuracy on train data is 0.965439672801636"
```

```
print(paste("after 10 runs, the average accuracy on test data is", mean_test_accur, sep = " "))
```

```
## [1] "after 10 runs, the average accuracy on test data is 0.964761904761905"
```

```
#try several different selections of starting positions and different convergence criteria

gradient_descent_theta <- function(X, Y, theta = theta, eta = 0.01, max_iteration = 30000, epsilon = 0.0



       for(i in 1:  max_iteration)
       {
               z <- as.matrix(X_train) %*% theta
               H <- Y_train / (1 + exp(-Y_train * z))
               # gradient in-sample error
               er_in <- t(X_train) %*% H / nrow(X_train)
               theta_new <- theta - eta * er_in

               if((any(abs(theta_new - theta))) <= epsilon) {
                              print(i)
                             break;}

               theta <- theta_new

                       }
       theta
  }

theta_list <- c(-10, 0, 10)
epsilon_list <- c(0.01, 0.0001,  0.000001)

result <- vector(, 4)
```

```
for(i in seq_along(theta_list)){

    theta <- replicate(ncol(X), theta_list[i])
    for(j in seq_along(epsilon_list)){

      theta <- gradient_descent_theta(X_train, Y_train, theta = theta, epsilon = epsilon_list[j])

      train_predict <- 1 / (1 + exp(as.matrix(X_train) %*% theta) )
      train_predict <- ifelse(train_predict >= 0.5, 1, -1)
      # calculate the prediction accuracy on training set
      train_accur = mean(train_predict == Y_train)

      # predict the test data set
      test_predict <- 1 / (1 + exp(as.matrix(X_test) %*% theta))
      test_predict <- ifelse(test_predict >= 0.5, 1, -1)
      # calculate the prediction accuracy on training set
      test_accur <-  mean(test_predict == Y_test)
      this_sample <- c(theta_list[i], epsilon_list[j], train_accur, test_accur)
      result <- rbind(result, this_sample)

  }
}
#drop the rowname for result
rownames(result) <- NULL
#drop the first row of placeholder
result <- result[-1, ]
colnames(result) <- c("initial_theta", "epsilon", "train_accur", "test_accur")
print(result)
```

```
##       initial_theta epsilon train_accur test_accur
##  [1,]           -10   1e-02   0.9672802  0.9666667
##  [2,]           -10   1e-04   0.9652352  0.9619048
##  [3,]           -10   1e-06   0.9672802  0.9619048
##  [4,]             0   1e-02   0.9652352  0.9619048
##  [5,]             0   1e-04   0.9652352  0.9619048
##  [6,]             0   1e-06   0.9672802  0.9619048
##  [7,]            10   1e-02   0.9693252  0.9619048
##  [8,]            10   1e-04   0.9693252  0.9619048
##  [9,]            10   1e-06   0.9693252  0.9619048
```

for the table we can see different initial theta and convergence criteria do not affect accuracy much.

4. Repeat (3) but this time using logistic regression training code from an R package such as glm2. How did the accuracy in (4) compare to the accuracy in (3).

```
sample_size <- floor(0.7 * nrow(BreastCancer))
train_index <- sample(seq_len(nrow(BreastCancer)), size = sample_size)

train <- BreastCancer[train_index,]
test <- BreastCancer[-train_index,]

glm_model <- glm2(Class~., family = binomial(link = "logit"), data = train)
```

```r
#summary(glm_model)
#anova(glm_model, test = "Chisq")
fitted_results <- predict(glm_model, newdata = test, type = "response")
fitted_results <- ifelse(fitted_results > 0.5, 1, 0)
miss_classific_error <- mean(fitted_results != test$Class)
print(paste('test data accuracy of glm2 package is', 1- miss_classific_error))
```

```
## [1] "test data accuracy of glm2 package is 0.961904761904762"
```

```r
#p <- predict(glm_model, newdata = test, type = "response")
#pr <- prediction(p, test$Class)
#prf <- performance(pr, measure = 'tpr', x.measure = 'fpr')

#plot(prf)
# ROC is a curve generated by plotting the true positive rate(TPR) against the
# false positive rate(FPR) at various threshold setting while the AUC area
# under the ROC curve.
#auc <- performance(pr, measure = "auc")
#auc <- auc@y.values[[1]]
#auc
```

We can see my model accuracy is very close to glm2 package.

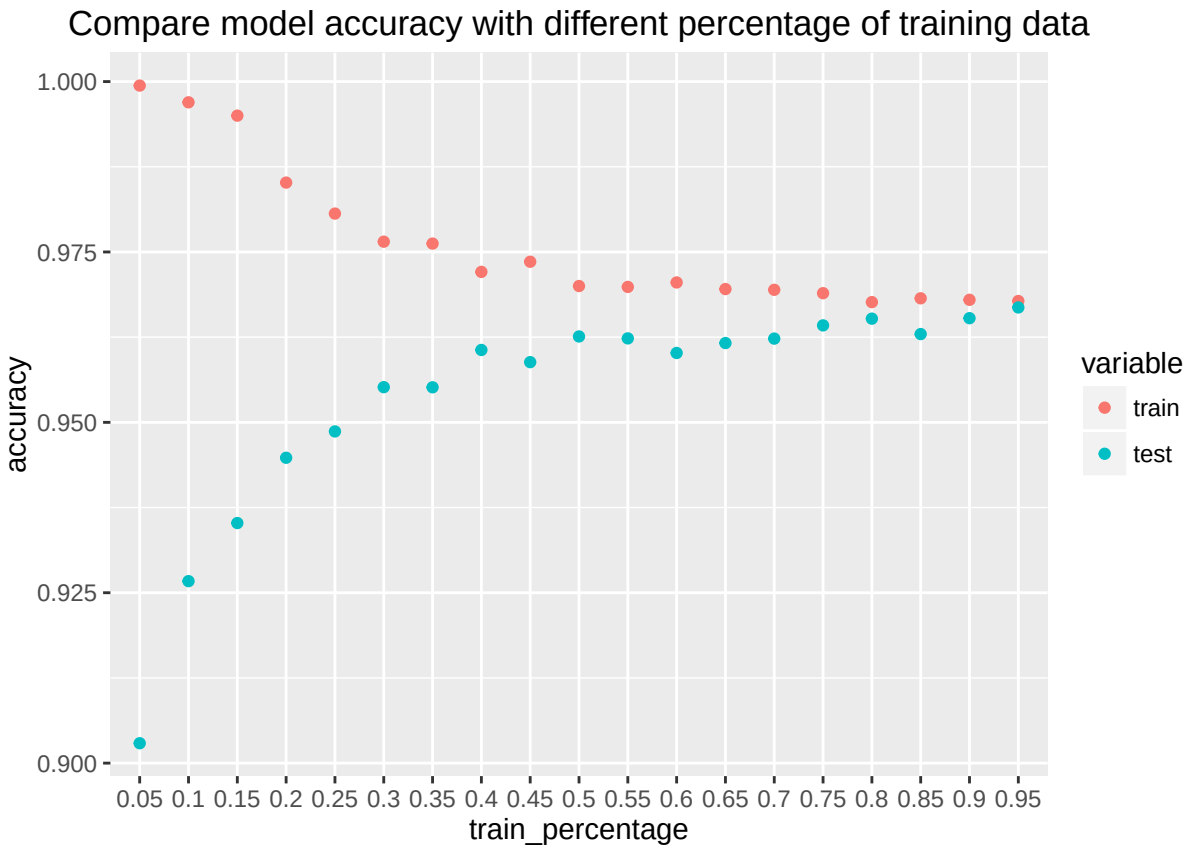
5. Repeat (4), but replace the 70%-30% train-test split with each of the following splits: 5%-95%, 10%-90%, . . . , 95%-5%. Graph the accuracy over the training set and over the testing set as a function of the size of the train set. Remember to average the accuracy over 10 random divisions of the data into train and test sets of the above sizes so the graphs will be less noisy.

```r
calculate_accuracy <- function(percent){

  # splict data to train and test
  sample_size <- floor(percent * nrow(BreastCancer))
  train_index <- sample(seq_len(nrow(BreastCancer)), size = sample_size)
  train <- BreastCancer[train_index,]
  test <- BreastCancer[-train_index,]

  # train model
  glm_model <- glm2(Class~., family = binomial(link = "logit"), data = train)

  # predict train accuracy
  train_pred <- predict(glm_model, newdata = train, type = "response")
  train_pred <- ifelse(train_pred > 0.5, 1, 0)
  miss_classific_error <- mean(train_pred != train$Class)
  train_accur <- 1- miss_classific_error

  # predict test accuracy
  test_pred <- predict(glm_model, newdata = test, type = "response")
  test_pred <- ifelse(test_pred > 0.5, 1, 0)
  miss_classific_error <- mean(test_pred != test$Class)
  test_accur <- 1- miss_classific_error

  # combine train_accur and test_accur
```

```
  accur <- c(train_accur,test_accur)

  accur

}


final_accuracy <- vector(, 3)
for(i in 0:18){

  percent <- 0.05 + i * 0.05

  each_accur <- c(0, 0)
    for(j in 1:100) {
    accur <- calculate_accuracy(percent)
    each_accur <- each_accur + accur
    }
    each_accur <- each_accur / 100
    each_accur <- c(percent, each_accur)

  final_accuracy <- rbind(final_accuracy, each_accur)

}
# remove the first placeholder row
final_accuracy <- final_accuracy[-1, ]
colnames(final_accuracy) <- c("train_percent", "train", "test")
#head(final_accuracy)
rownames(final_accuracy) <- NULL


final_accuracy <- as.data.frame(final_accuracy)
final_accuracy$train_percent <- factor(as.character(final_accuracy$train_percent))
head(final_accuracy)
```

```
##   train_percent     train      test
## 1          0.05 0.9994118 0.9029173
## 2           0.1 0.9969565 0.9266984
## 3          0.15 0.9950000 0.9352269
## 4           0.2 0.9851799 0.9448036
## 5          0.25 0.9806322 0.9486667
## 6           0.3 0.9765072 0.9551633
```

```
melt_accuracy <- melt(final_accuracy, id = "train_percent")

colnames(melt_accuracy) <- c("train_percentage", "variable", "accuracy")

ggplot(melt_accuracy,
       aes(x = train_percentage, y=accuracy, color = variable)) +
  geom_point() +

  ggtitle("Compare model accuracy with different percentage of training data")
```



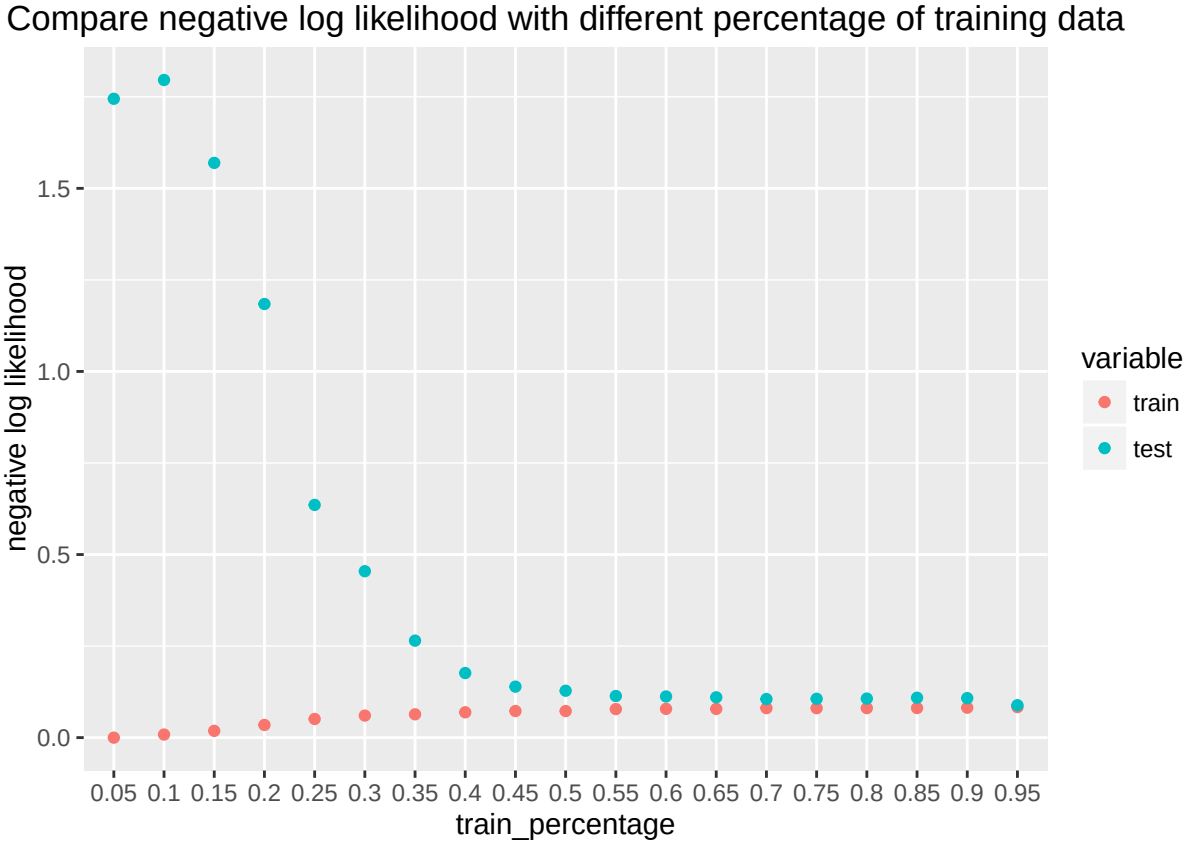
From graph we can see, with lower percentage of training data, model has relative low accuracy on test dataset and that means our model is under trained. Until we used at least 50% of data to train the model, the model gives a good prediction on test dataset.

6. Repeat (5) but instead of graphing the train and test accuracy, graph the logistic regression loss function (negative log likelihood) over the train set and over the test set as a function of the train set size.

```
calculate_nll <- function(percent){

  # splict data to train and test
  #print("head of Breastcancer")
  #sapply(BreastCancer, function(x) is.factor(x))
  #head(BreastCancer)
  sample_size <- floor(percent * nrow(BreastCancer))

  train_index <- sample(seq_len(nrow(BreastCancer)), size = sample_size)
  train <- BreastCancer[train_index,]
  #print(head(train))
  test <- BreastCancer[-train_index,]
  #print(head(train))

  # train model
  glm_model <- glm2(Class~., family = binomial(link = "logit"), data = train)


  # predict train probability
  train_pred <- predict(glm_model, newdata = train, type = "response")
```

```
  # calculate the average negative logigstic likelihood
  train_nll <- -1 * mean(train$Class * log(train_pred) + (1 - train$Class) * log(1 - train_pred))

  # predict test probability
  test_pred <- predict(glm_model, newdata = test, type = "response")
  test_nll <- -1 * mean(test$Class * log(test_pred) + (1 - test$Class) * log( 1 - test_pred))

  nll <- c(train_nll, test_nll)
  #print(nll)
  nll

}

#calculate the final nll for different percent of train and test

# create a matrix to collect nll for each percent of train and test
final_nll <- vector(, 3)
for(i in 0:18){

  percent <- 0.05 + i * 0.05

  each_nll <- c(0, 0)
    for(j in 1:100) {
    nll <- calculate_nll(percent)
    each_nll <- each_nll + nll
    }

  each_nll <- each_nll / 100
  #print(i)
  each_nll <- c(percent, each_nll)
  final_nll <- rbind(final_nll, each_nll)

}
# elimilate first row which is placeholder
final_nll <- final_nll[-1, ]
#colnames(final_nll) <- c("train", "test")
#head(final_nll)
# remove row index
rownames(final_nll) <- NULL

# create the row index

colnames(final_nll) <-  c("train_percent", "train", "test")
#final_nll
final_nll <- as.data.frame(final_nll)
final_nll$train_percent <- factor(as.character(final_nll$train_percent))
#head(final_nll)
melt_nll <- melt(final_nll, id = "train_percent")
#colnames(melt_nll)

colnames(melt_nll) <- c("train_percentage", "variable", "nll")

ggplot(melt_nll,
```

```
      aes(x = train_percentage, y=nll, color = variable)) +
  geom_point() +
  ylab("negative log likelihood") +
  ggtitle("Compare negative log likelihood with different percentage of training data")
```



It showed very similar trend as last question. We can see with the increase of traning data, the negative log likelihood value of test dataset is decresing. Until it equals to the value of train dataset at around 55%.