Home Work 3

Johnpaul Ogah

October 16, 2016

Question 1

Logistic regression is given by :

$p(y \mid x) = \frac{1}{1+e^{(y<\theta x>)}}$

$p(y=+1 \mid x) = \frac{1}{1+e^{<\theta x>}}$

$p(y=-1 \mid \ x) = 1 - p(y=+1 \mid x) = \frac{e^{<\theta x>}}{1+e^{<\theta x>}}$

where $<\theta, x>$ is the dot product of the transpose of the vector of weight parameter and the feature vector

$\theta_{MLE} = \prod_{i=1}^{N} p(y=y_i \mid x=x_i, \theta) = \sum_{i=1}^{N} \ln p(y=y_i \mid x=x_i, \theta)$

Now I want to introduce an indicator variable $\mathbb{I}(y=+1)$, this indicated that output label(y) is +1 and $\mathbb{I}(y=-1)$ indicate that the output label(y) is -1, using this knowledge and subsititing this values into maximum likelihood equation we have:

$\mathcal{LL}(\theta) = \sum_{i=1}^{N} \mathbb{I}(y=+1) \ln p(y=+1 \mid x=x_i, \theta) + \mathbb{I}(y=-1) \ln p(y=-1 \mid x=x_i, \theta)$

It is also true that: $\mathbb{I}(y=-1) = 1 - \mathbb{I}(y=+1)$

$\mathcal{LL}(\theta) = \sum_{i=1}^{N} \mathbb{I}(y=+1) \ln p(y=+1 \mid x_i, \theta) + (1-\mathbb{I}(y=+1)) \ln p(y=-1 \mid x_i, \theta)$

substituting the value for logistic regression and simplying for a single datapoint we have,

$\mathcal{LL}(\theta) = \mathbb{I}(y_i=+1) \ln \frac{1}{1+e^{<\theta x_i>}} + (1-\mathbb{I}(y_i=+1)) \ln \frac{e^{<\theta x_i>}}{1+e^{<\theta x_i>}}$

$= -\mathbb{I}(y_i=+1) \ln(1+e^{<\theta x_i>}) + (1-\mathbb{I}(y_i=+1))(<\theta x_i> - \ln(1+e^{<\theta x_i>}))$

$\mathcal{LL}(\theta) = -\mathbb{I}(y_i=+1) \ln(1+e^{<\theta x_i>}) + <\theta x_i> - \ln(1+e^{<\theta x_i>}) - \mathbb{I}(y_i=+1) <\theta x_i> + \mathbb{I}(y_i=+1) \ln(1+e^{<\theta x_i>})$

$= <\theta x_i> - \mathbb{I}(y_i=+1) <\theta x_i> - \ln(1+e^{<\theta x_i>})$

$= (1-\mathbb{I}(y_i=+1)) <\theta x_i> - \ln(1+e^{<\theta x_i>})$

$\frac{\partial \mathcal{LL}}{\partial \theta_j} = (1-\mathbb{I}(y_i=+1)) \frac{\partial(<\theta x_i>)}{\partial \theta_j} - \frac{\partial(\ln(1+e^{<\theta x_i>}))}{\partial \theta_j}$

$= (1-\mathbb{I}(y_i=+1))_j <x_i> - \frac{_j<x_i>e^{<\theta x_i>}}{(1+e^{<\theta x_i>})}$

$= (1-\mathbb{I}(y_i=+1))_j <x_i> -_j <x_i> p(y=-1 \mid x_i, \theta)$

$=_j <x_i> (p(y=+1 \mid x_i, \theta) - \mathbb{I}(y_i=+1))$

The derivative/gradient for one data point is given as:

$_j <x_i> (p(y=+1 \mid x_i, \theta) - \mathbb{I}(y_i=+1))$

adding the derivative for all the data point we have :

$\frac{\partial \mathcal{LL}}{\partial \theta_j} = \sum_{i=1}^{N} {}_j <x_i> (p(y=+1 \mid x_i, \theta) - \mathbb{I}(y_i=+1))$

where $_j <x_i>$ is the feature vector associated with j^{th} parameter .

The puedocode for training a logistic classifiers is:

```
- Initialize the weight vectors $\theta$ to random values, v = 1( v is the number of iteration)
- While $||\,\Delta\mathcal{L}(\theta)\,|| > \varepsilon$ , where $\varepsilon$ is some tolerance level or threshold
    - for j = 0 . . . . . . D
    - partial_derivative[j] = $\sum_{i=1\,\mathbf{j}}^{N} < \mathbf{x_i} > (\mathbf{p}(\mathbf{y} = +\mathbf{1} \mid \mathbf{x_i}, \theta) - \mathbb{I}(\mathbf{y_i} = +\mathbf{1}))$
    - $\theta^{(\mathbf{v}+\mathbf{1})} = \theta^{(\mathbf{v})} + \eta$ partial_derivative[j]
- v = v + 1
```

Number of operation for each gradient descent iteration

- 1 * d(n(2) + 2) = 2d(n+1)
- each iteration will perform n addition and n substraction to calculate the partial derivative for that iteration
- each iteration will perform one addition and one multiplication to calculate the new weight for the next iteration
- each iteration will perform the above mentioned step for d times , where d is the dimensionality

Question 2

The implementation of the gradient descent algorithm I derived in question1 is shown in listing 2 code.The function logistic_regression takes as input parameter the feature matrix, the target/response vector , the step size and the maximum number of iteration.

```r
#Listing 2

# this function takes a data_frame and return the feauture matrix

get.feature.matrix = function(data, features){
  data[,"constant"] = 1
  new_features = c("constant",features)
  temp_s_array= data[,new_features]
  result = data.matrix(temp_s_array)
  return(result)
}

# this function return the label array

label_array = function(data, label){
  temp_data = data[,label]
  result = data.matrix(temp_data)
  return(result)
}

# below function compute the estimated probalility given by the logistic regression formular

estimate_probality = function(feature_matrix, coefficients){
  scores = feature_matrix %*% coefficients
  result = 1 / ( 1 + exp(scores))
  return(result)
}
```

```r
# Below function compute the derivative

feature_derivative = function(errors, feature){
  result = errors %*% feature
  return(result)
}

# Below function implements gradient descent for estimating coeficients of logistic regression model

logistic_regression  = function(feature_matrix, target,init_coefficients, step_size, max_iter){
  coefficients = init_coefficients
  for ( itr in seq(1:max_iter)){
    predictions = estimate_probality(feature_matrix,coefficients)
    indicator = (target == 1)
    errors = predictions - indicator
    total_derivative = feature_derivative(as.vector(errors), feature_matrix)
    for( j in seq(1:length(coefficients))){
    derivative = feature_derivative(as.vector(errors),feature_matrix[,j])

    coefficients[j] = coefficients[j] + step_size * derivative
    }


  }
  return(coefficients)
}

# below function compute the prediction

class_prediction = function(feature_matrix, coefficients){
  scores = feature_matrix %*% coefficients
  result = sapply(scores , function(x){if(x < 0) return(1) else return(-1)})
  return(result)
}


# below function calculate the accuracy of my model
compute_accuracy = function(target, result){
  num = sum(target == result)
  temp = num/ length(target)
  return(temp)
}
```

Question 3 & Question 4

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

As seen in listing 3.1 code, Initializing the coefficeints to zeros i.e setting $< \theta >$ to a vector of length 10 and all zeros , a maximum iteration of 300 and step size of 0.001. The accuracy on the train data was ~97% and the accuracy on test data was ~96%. With the glm package with the same initial coefficients and maximum number of iteration the accuracy on train data was ~98 and the accuracy on test data is ~97% . one percent difference in accuracy with glm doing slighly better with same initial parameters.

```
#Listing 3.1

asNumeric = function(x) as.numeric(as.character(x))
factorsNumeric = function(d) modifyList(d, lapply(d[, sapply(d, is.factor)],
                                                   asNumeric))

breast_cancer_data = na.omit(BreastCancer)

breast_cancer_data$Class_1 = sapply(breast_cancer_data[,"Class"],
                                    function(x){if(x=="benign")
                                      return(1) else
                                      return(-1)})

breast_cancer_data$Class = sapply(breast_cancer_data[,"Class"],
                                  function(x){if(x=="benign")
                                    return(1) else return(0)})

breast_cancer_data = factorsNumeric(breast_cancer_data)

n_row = nrow(breast_cancer_data)

random_perm = sample(n_row,n_row)
first_index = random_perm[1:floor(n_row*0.7)]
second_index = random_perm[(floor(n_row * 0.7)+1):n_row]
train_data = breast_cancer_data[first_index,]
test_data = breast_cancer_data[second_index,]

train_data = factorsNumeric(train_data)
test_data = factorsNumeric(test_data)


features = setdiff(names(breast_cancer_data),c("Id","Class","Class_1"))
feature_matrix = get.feature.matrix(train_data,features)
target_1 = label_array(train_data,"Class_1")

initial_weight = rep(0,10)
step_size =  0.001
max_iter = 300
tolerance = 1e-8

weight = logistic_regression(feature_matrix,target_1,
                              initial_weight,step_size,max_iter)


my_prediction = class_prediction(feature_matrix,weight)

 accuracy = compute_accuracy(as.vector(target_1),my_prediction)
 cat("Train Accuracy:",accuracy)
```

```
## Train Accuracy: 0.9644351
```

```
 test_feature_matrix = get.feature.matrix(test_data,features)
 test_prediction = class_prediction(test_feature_matrix,weight)
 test_target_label = label_array(test_data,"Class_1")
 test_accuracy = compute_accuracy(as.vector(test_target_label)
                                  ,test_prediction)

 cat("Test Accuracy:", test_accuracy)
```

```
## Test Accuracy: 0.9658537
```

```
 model = glm(Class~Cl.thickness+Cell.size+Cell.shape+Marg.adhesion
             +Epith.c.size+Bare.nuclei+Bl.cromatin+Normal.nucleoli
             +Mitoses,data=train_data,family=binomial(logit)
             ,start=initial_weight, control=list(maxit=300))

train_accuracy_glm = predict(model, newdata = train_data)
train_accuracy_glm = ifelse(train_accuracy_glm > 0.5,1,0)
misClasificError = mean( train_accuracy_glm != train_data$Class)
cat("train_accuracy_glm:", 1-misClasificError)
```

```
## train_accuracy_glm: 0.9748954
```

```
test_accuracy_glm = predict(model, newdata = test_data)
test_accuracy_glm = ifelse(test_accuracy_glm > 0.5,1,0)
misClasificError = mean( test_accuracy_glm != test_data$Class)
cat("test_accuracy_glm:", 1-misClasificError)
```

```
## test_accuracy_glm: 0.9658537
```

Repeat the random partition of 70% and 30% 10 times and average the test accuracy results over the 10 repetitions, also compare your implementation with the glm pacakage result.

The code for this section is shown in listing 3.2, The average accuracy on the train data was ~97% and the accuracy on test data was ~96% with my implementation. With the glm package the average accuracy on train data was ~98 and the average accuracy on test data was ~97% . one percent difference in accuracy with glm still doing slighly better with same initial parameters.

```
# Listing 3.2

 test_accuracy = rep(0,10)
 train_accuracy = rep(0,10)
 test_accuracy_glm=rep(0,10)
 train_accuracy_glm = rep(0,10)

 for(num in seq(1:10)){

   random_perm = sample(n_row,n_row)
  first_index = random_perm[1:floor(n_row*0.7)]
  second_index = random_perm[(floor(n_row * 0.7)+1):n_row]
  train_data = factorsNumeric(breast_cancer_data[first_index,])
  test_data = factorsNumeric(breast_cancer_data[second_index,])
  features = setdiff(names(breast_cancer_data),c("Id","Class","Class_1"))
 feature_matrix = get.feature.matrix(train_data,features)
 target_1 = label_array(train_data,"Class_1")
 weight = logistic_regression(feature_matrix,target_1,initial_weight
                              ,step_size,max_iter)

 my_prediction = class_prediction(feature_matrix,weight)

 train_accuracy[num] = compute_accuracy(as.vector(target_1),my_prediction)
 test_feature_matrix = get.feature.matrix(test_data,features)
 test_prediction = class_prediction(test_feature_matrix,weight)
 test_target_label = label_array(test_data,"Class_1")
 test_accuracy[num] = compute_accuracy(as.vector(test_target_label),
                                       test_prediction)
 model2 = glm(Class~Cl.thickness+Cell.size+Cell.shape+Marg.adhesion
              +Epith.c.size+Bare.nuclei+Bl.cromatin
              +Normal.nucleoli+Mitoses,data=train_data,start = initial_weight,
              family=binomial(logit),control = list(maxit = max_iter))
 train_glm = predict(model2, newdata = train_data)
train_glm = ifelse(train_glm > 0.5,1,0)
misClasificError = mean( train_glm != train_data$Class)
train_accuracy_glm[num] = 1-misClasificError

test_glm = predict(model2, newdata = test_data)
test_glm = ifelse(test_glm > 0.5,1,0)
misClasificError = mean( test_glm != test_data$Class)
test_accuracy_glm[num]= 1-misClasificError

 }
 cat("Average accuracy on test data:", mean(test_accuracy))
```

```
## Average accuracy on test data: 0.9639024
```

```
 cat("Average accuracy on train data:", mean(train_accuracy))
```

```
## Average accuracy on train data: 0.9654812
```

```r
cat("Average accuracy on test data using glm:", mean(test_accuracy_glm))
```

```
## Average accuracy on test data using glm: 0.9702439
```

```r
cat("Average accuracy on train data using glm :", mean(train_accuracy_glm))
```

```
## Average accuracy on train data using glm : 0.9736402
```

```r
#Listing 3.3

  random_perm = sample(n_row,n_row)
  first_index = random_perm[1:floor(n_row*0.7)]
  second_index = random_perm[(floor(n_row * 0.7)+1):n_row]
  train_data = factorsNumeric(breast_cancer_data[first_index,])
  test_data = factorsNumeric(breast_cancer_data[second_index,])

  df <- data.frame(matrix(ncol = 1, nrow = 10))

 for(num in seq(1:10)){
   initial_weight = runif(10)
   random_perm = sample(n_row,n_row)
  features = setdiff(names(breast_cancer_data),c("Id","Class","Class_1"))
 feature_matrix = get.feature.matrix(train_data,features)
 target_1 = label_array(train_data,"Class_1")
 weight = logistic_regression(feature_matrix,target_1,
                              initial_weight,step_size,max_iter)
 colname = paste("iter",num)
 df[,colname] = as.vector(weight)
 my_prediction = class_prediction(feature_matrix,weight)

 train_accuracy[num] = compute_accuracy(as.vector(target_1),my_prediction)
 test_feature_matrix = get.feature.matrix(test_data,features)
 test_prediction = class_prediction(test_feature_matrix,weight)
 test_target_label = label_array(test_data,"Class_1")
 test_accuracy[num] = compute_accuracy(as.vector(test_target_label),
                                       test_prediction)

model3 = glm(Class~Cl.thickness+Cell.size+Cell.shape
             +Marg.adhesion+Epith.c.size+Bare.nuclei
             +Bl.cromatin+Normal.nucleoli+Mitoses
            ,data=train_data,start = initial_weight,
            family=binomial(link="logit"),control = list(maxit = max_iter))
 train_glm = predict(model3, newdata = train_data)
train_glm = ifelse(train_glm > 0.5,1,0)
misClasificError = mean( train_glm != train_data$Class)
train_accuracy_glm[num] = 1-misClasificError

test_glm = predict(model3, newdata = test_data)
test_glm = ifelse(test_glm > 0.5,1,0)
misClasificError = mean( test_glm != test_data$Class)
test_accuracy_glm[num]= 1-misClasificError
 }

 cat("Average accuracy on test data:", mean(test_accuracy))
```

```
## Average accuracy on test data: 0.9785366
```

```r
cat("Average accuracy on train data:", mean(train_accuracy))
```

```
## Average accuracy on train data: 0.9665272
```

```r
cat("Average accuracy on test data using glm:", mean(test_accuracy_glm))
```

```
## Average accuracy on test data using glm: 0.9785366
```

```r
cat("Average accuracy on train data using glm :", mean(train_accuracy_glm))
```

```
## Average accuracy on train data using glm : 0.9682008
```

```r
df$matrix.ncol...1..nrow...10.= NULL
```

Try several different selections of starting positions - did this change the parameter value that the model learned?

As seen from the table below, we see the parameter value changing for 10 different initial stating position.The average accuracy with my model on train data was approximately ~97% and ~96% on the test data. For the glm R package the average accuracy on train data was ~98% and ~96% on test data.

```r
print(df)
```

```
##         iter 1      iter 2      iter 3      iter 4      iter 5      iter 6
## 1  -5.67367272 -5.72681624 -5.62966374 -5.65399329 -5.70025856 -5.70078385
## 2   0.20573764  0.17783691  0.20284594  0.20430821  0.17339022  0.20747553
## 3   0.28866402  0.25735779  0.29919098  0.29227428  0.26159284  0.28267270
## 4   0.21783591  0.19583600  0.21864159  0.21950750  0.19481313  0.21686351
## 5   0.23220064  0.21678962  0.23199285  0.23228967  0.21522919  0.23230215
## 6  -0.14512647 -0.16411107 -0.14885765 -0.14682776 -0.16831971 -0.14290530
## 7   0.42448142  0.39672909  0.42884610  0.42640430  0.39722917  0.42186514
## 8   0.06529661  0.04683378  0.06101798  0.06349723  0.04231266  0.06781162
## 9   0.24655893  0.22162822  0.25040561  0.24833706  0.22211259  0.24419851
## 10  0.21567717  0.21221142  0.21106392  0.21391148  0.20865347  0.21842378
##         iter 7      iter 8     iter 9     iter 10
## 1  -5.60547130 -5.61657191 -5.6074078 -5.67357136
## 2   0.20106133  0.20199941  0.2012738  0.20570882
## 3   0.30428662  0.30268670  0.3043442  0.28853475
## 4   0.22001121  0.21848134  0.2193393  0.21802537
## 5   0.23206011  0.23189083  0.2319652  0.23222791
## 6  -0.15101998 -0.14998628 -0.1508271 -0.14514275
## 7   0.43128577  0.43017238  0.4310957  0.42448952
## 8   0.05865675  0.05968101  0.0588092  0.06529975
## 9   0.25258936  0.25154225  0.2523866  0.24657729
## 10  0.20872368  0.20959897  0.2088014  0.21570827
```

Try to play with different convergence criteria to get better accuracy.

- The code for this question is shown in listing 3.3 . After trying different combination of number of iteration and step size , the best accuracy for my model on the test data was ~98% with the step size of 0.01 and maximum number of iteration of 200. The glm package recorded highest accuracy on test data of ~99% slighly below the performance of my model.

```
#Listing 3.3

max_iteration = c(25,50,100,200,300,400,500,600,700,800,900,1000)
train_accuracy = rep(0,10)
test_accuracy = rep(0,10)
initial_weight = rep(0,10)
train_accuracy_glm = rep(0,10)
test_accuracy_glm = rep(0,10)
step_size = 0.01
max_test_accuracy = -99999
max_test_accuracy_glm = -99999
best_iteration = 0
best_iteration_glm = 0


 for (i in seq(1:length(max_iteration))) {
weight = logistic_regression(feature_matrix,target_1,initial_weight,
                             step_size,max_iteration[i])

 my_prediction = class_prediction(feature_matrix,weight)

 train_accuracy[i] = compute_accuracy(as.vector(target_1),my_prediction)
 test_feature_matrix = get.feature.matrix(test_data,features)
 test_prediction = class_prediction(test_feature_matrix,weight)
 test_target_label = label_array(test_data,"Class_1")
 test_accuracy[i] = compute_accuracy(as.vector(test_target_label),
                                     test_prediction)
 if( test_accuracy[i] > max_test_accuracy){
   max_test_accuracy = test_accuracy[i]
   best_iteration = max_iteration[i]
 }
 model4 = glm(Class~Cl.thickness+Cell.size+Cell.shape+
                Marg.adhesion+Epith.c.size+Bare.nuclei+
                Bl.cromatin+Normal.nucleoli+
                Mitoses,data=train_data,family=binomial(link="logit")
              ,control = list(maxit = max_iteration[i]))
 train_glm = predict(model4, newdata = train_data)
train_glm = ifelse(train_glm > 0.5,1,0)
misClasificError = mean( train_glm != train_data$Class)
train_accuracy_glm[i] = 1-misClasificError

test_glm = predict(model4, newdata = test_data)
test_glm = ifelse(test_glm > 0.5,1,0)
misClasificError = mean( test_glm != test_data$Class)
test_accuracy_glm[i]= 1-misClasificError

if (test_accuracy_glm[i] > max_test_accuracy){
  max_test_accuracy_glm = test_accuracy_glm[i]
  best_iteration_glm = max_iteration[i]

}
 }


cat("Average accuracy on test data:", mean(test_accuracy))
```

```
## Average accuracy on test data: 0.9593496
```

```
cat("Average accuracy on train data:", mean(train_accuracy))
```

```
## Average accuracy on train data: 0.9522315
```

```
cat("Average accuracy on test data using glm:", mean(test_accuracy_glm))
```

```
## Average accuracy on test data using glm: 0.9853659
```

```
cat("Average accuracy on train data using glm :", mean(train_accuracy_glm))
```

```
## Average accuracy on train data using glm : 0.9686192
```

```
cat("maximum test accuracy",max_test_accuracy)
```

```
## maximum test accuracy 0.9756098
```

```
cat("best number of iteration",best_iteration)
```

```
## best number of iteration 200
```

```
cat("maximum test accuracy for glm",max_test_accuracy_glm)
```

```
## maximum test accuracy for glm 0.9853659
```

```
cat("best number of iteration glm",best_iteration_glm)
```

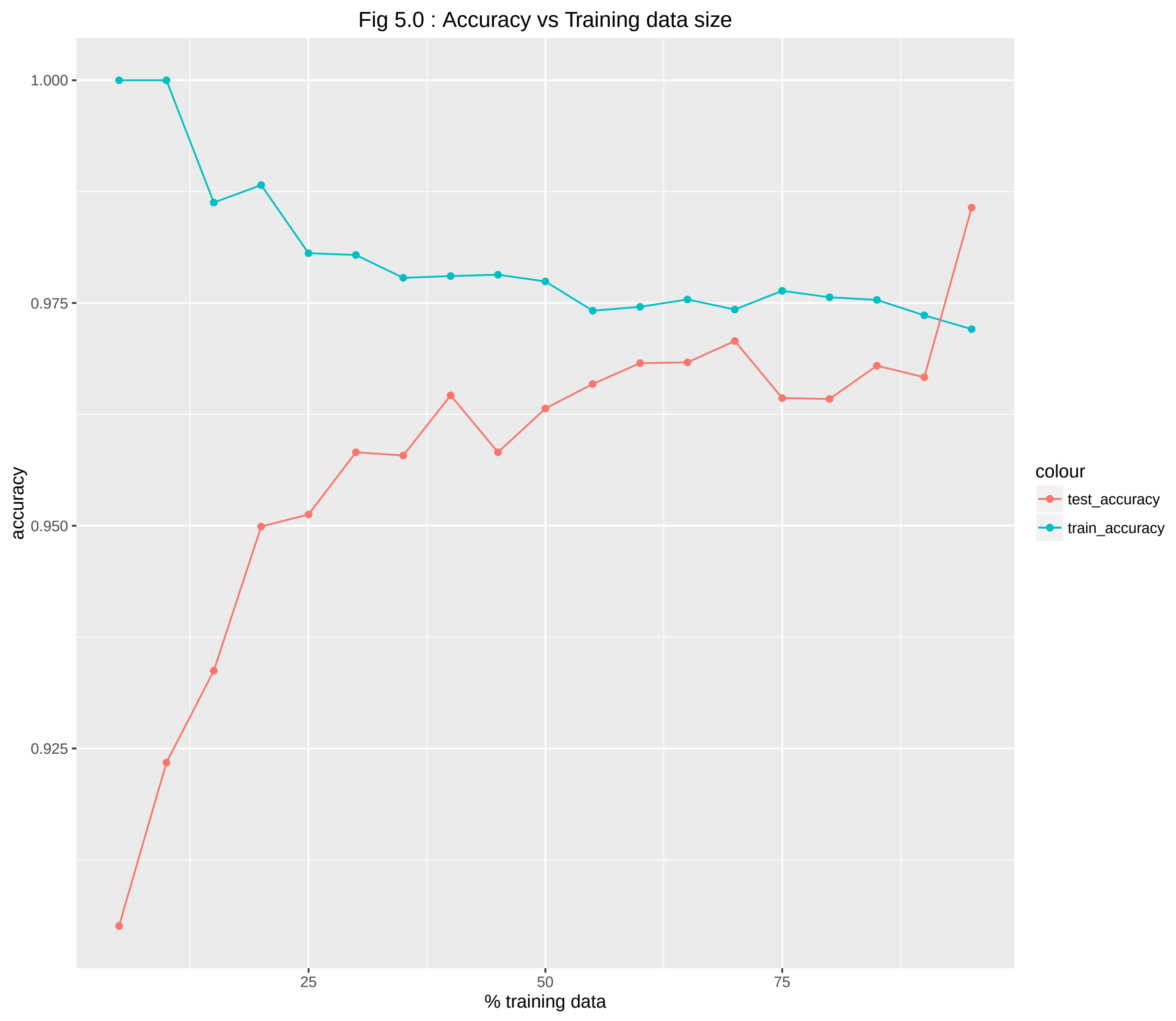
```
## best number of iteration glm 1000
```

Question 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

- The code for this question is shown in listing 5.1,we see from Figure 5.0 the train accuracy decreasing with increasing data size if we ignore the noise and also the test accuracy increasing with increasing data up to using 95% of the data for training(ignoring the noise) where the accuracy is ~99%.

```r
# Listing 5.1
train_set = seq(5,95, by=5)

accuracy.test.data = rep(0,19)
accuracy.on.train.data = rep(0,19)
train_accuracy = rep(0,10)
test_accuracy= rep(0,10)
max_iter = 500
step_size = 0.001

for (num in seq(1:19)){
  for(i in seq(1:10)){
    local_var = train_set[num]
    local_var = local_var/100
  random_perm = sample(n_row,n_row)
  first_index = random_perm[1:floor(n_row*local_var)]
  second_index = random_perm[(floor(n_row * local_var)+1):n_row]
  train_data = breast_cancer_data[first_index,]
  test_data = breast_cancer_data[second_index,]
  model5 = glm(Class~Cl.thickness+Cell.size+Cell.shape+
                Marg.adhesion+Epith.c.size+Bare.nuclei+
                Bl.cromatin+Normal.nucleoli+
                Mitoses,data=train_data,family=binomial(link="logit")
              ,control = list(maxit = max_iter))
 train_glm = predict(model5, newdata = train_data)
 train_glm = ifelse(train_glm > 0.5,1,0)
train_misClasificError = mean( train_glm != train_data$Class)
train_accuracy[i] = 1-train_misClasificError

test_glm = predict(model5, newdata = test_data)
test_glm = ifelse(test_glm > 0.5,1,0)
test_misClasificError = mean( test_glm != test_data$Class)
test_accuracy[i]= 1-test_misClasificError

  }

  accuracy.on.train.data[num] = mean(train_accuracy)
  accuracy.test.data[num] = mean(test_accuracy)
}

result = data.frame(training_data=train_set,
                    accuracy_train_data=accuracy.on.train.data,
                    accuracy_test_data=accuracy.test.data)

ggplot(result,aes(x=training_data)) + geom_line(aes(y=accuracy_train_data,color="train_accuracy")) + ge
geom_point(aes(y=accuracy_train_data,color="train_accuracy")) + geom_point(aes(y=accuracy_test_data,col
                   ylab("accuracy") +xlab("% training data") + ggtitle("Fig 5.0 : Accuracy vs Training
```



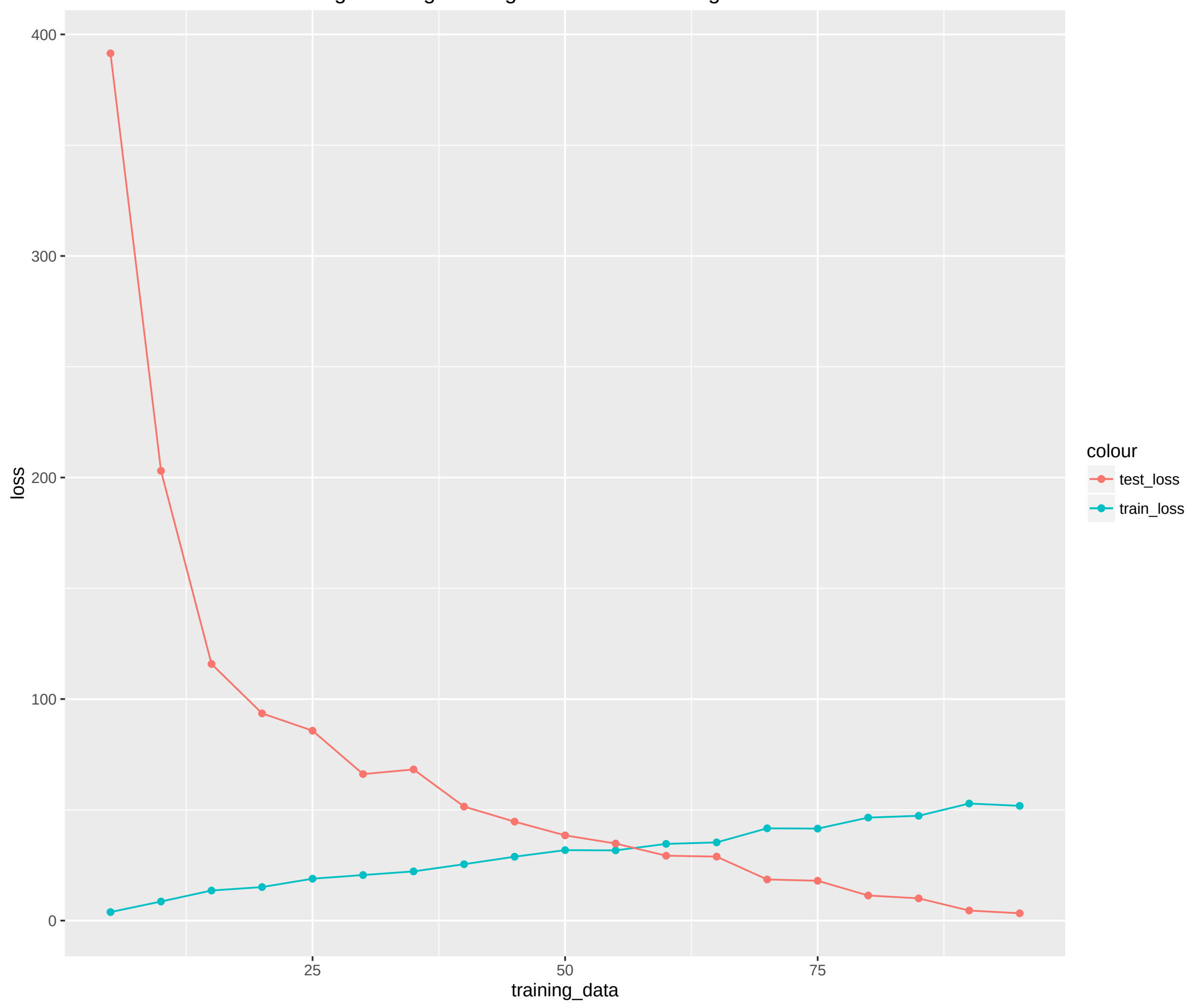
Question6

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.

- The code is shown in listing 6.1 , we can see in Figure 6 the negative log likelihood is increasing for the training data set and decreasing for the testing data set as the training size increases. This is expected because the more training data available the better the model and hence the better the model minimize the loglikelihood on the test data.Also the fewer the data available for training the less likely that our model will make mistake on the training data, so with less data the lower the negative log likelihood on the training set. In other words , the negative loglikehood will increase on the training set with increase training data.

```
#Listing 6.1

compute_loss_function =  function(data, weight, output_label){
  indicator = (output_label == 1)
  score = data %*% weight
  ds = -((1-indicator)*score-log(1 + exp(score)))
  return(sum(ds))

}


train_set = seq(5,95, by=5)

loss.test.data = rep(0,19)
loss.on.train.data = rep(0,19)
train_loss_glm = rep(0,10)
test_loss_glm = rep(0,10)
initial_weight = rep(0,10)

for (num in seq(1:19)){
  for(i in seq(1:10)){
    local_var = train_set[num]
    local_var = local_var/100
  random_perm = sample(n_row,n_row)
  first_index = random_perm[1:floor(n_row*local_var)]
  second_index = random_perm[(floor(n_row * local_var)+1):n_row]
  train_data = breast_cancer_data[first_index,]
  test_data = breast_cancer_data[second_index,]
train_target = label_array(train_data,"Class")
train_data_matrix = get.feature.matrix(train_data,features)
weight = logistic_regression(train_data_matrix,train_target,
                             initial_weight,0.001,500)

train_loss_glm[i] = compute_loss_function(train_data_matrix,
                                          weight,train_target)
test_data_matrix = get.feature.matrix(test_data,features)
test_target = label_array(test_data,"Class")
test_loss_glm[i] = compute_loss_function(test_data_matrix,
                                         weight,test_target)

  }

  loss.on.train.data[num] = mean(train_loss_glm)
  loss.test.data[num] = mean(test_loss_glm)
}

loss_function_result = data.frame(training_data=train_set,
                                  loss_train_data=loss.on.train.data,
                                  loss_test_data=loss.test.data)


ggplot(loss_function_result,aes(x=training_data)) + geom_point(aes(y=loss_train_data,color="train_loss")
```

Fig 6.0: Negative log likelihood vs training data size


loss
training_data
colour
test_loss
train_loss