# Assignment 1

Brian Choi

20/03/2023

## Question 1

### Part A

$$\nabla C_1(w; Data) = -\frac{1}{m}\Sigma_{i=1}^{m}\nabla(y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)}))$$

$$= -\frac{1}{m}\Sigma_{i=1}^{m}\nabla y^{(i)}log(\sigma(w^T x)) + (1 - y^{(i)})log(1 - \sigma(w^T x))$$

$$= \frac{1}{m}(\sigma(w^T x^{(i)}) - y^{(i)})x^{(i)}$$

### Part B

$$\nabla C_2(w; Data) = -\frac{1}{m}\Sigma_{i=1}^{m}\nabla(\sigma(w^T x^{(i)} - y^{(i)})^2$$

$$= \frac{1}{m}(\sigma(W^T x^{(i)}) - y^{(i)})\sigma'(w^T x^{(i)})x^{(i)}$$

### Part C

$$H_1 = \nabla^2(C_1(w; Data)) = \frac{1}{m}\sigma(w^T x^{(i)})(1 - \sigma(w^T x^{(i)}))x^{(i)}(x^{(i)})^T$$

### Part D

$$H_2 = \nabla^2(C_1(w; Data)) = \frac{1}{m}\sigma(w^T x^{(i)})(1 - \sigma(w^T x^{(i)}))(x^{(i)})^T(x^{(i)})^T$$

### Part E]

To show that the function is convex, the following equation needs to hold true:

$$v^T H_1 = \frac{1}{m}\sigma(w^T x^{(i)})(1 - \sigma(w^T x^{(i)}))(v^T x^{(i)})^2 \geq 0$$

Since the sigmoid function always has a value between 0 and 1 the equation is always positive semidefinite. Therefore, $C_1(w; Data)$ is convex.

## Part F

$$H_2 = \nabla^2(C_1(w; Data)) = \frac{1}{m}\sigma(w^T x^{(i)})(1-\sigma(w^T x^{(i)}))(x^{(i)})^T(x^{(i)})^T$$

Let $w = (0,0)^T$ then,

$$H_2 = \frac{1}{m}(x^{(i)})^T(x^{(i)})^T$$

Considering the case where $w = (0,0)^T$, the matrix si not positive definite since the eigenvalues are approximately -0.114 and 24.114 according to the data given.

## Part G

Since $C_1(w;Data) is convex it should be used for training the model as it can be used for gradient descent optimisation techniques.

## Part H

```
library(plot3D)
```

```
## Warning: package 'plot3D' was built under R version 4.1.3
```

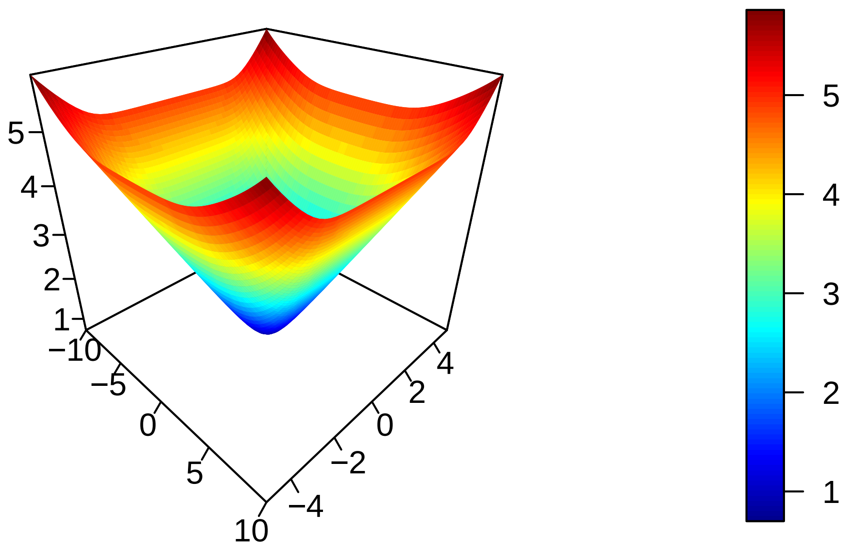
```
x = cbind(c(1,1,1,1,1,1),c(-3,-2,-1,1,2,3))
y = c(1,0,0,1,1,0)
sigmoid <- function(x){
  sigmoid <- 1 / (1 + exp(-x))
}

w1_vals = seq(-10, 10, length.out = 100)
w2_vals = seq(-5, 5, length.out = 50)
w1_grid = outer(w1_vals, w2_vals, FUN = "*")
w2_grid = outer(w1_vals, w2_vals, FUN = "*")

loss_grid <- matrix(0, nrow = length(w1_vals), ncol = length(w2_vals))

for (i in 1:length(w1_vals)) {
  for (j in 1:length(w2_vals)) {
    w <- c(w1_vals[i], w2_vals[j])
    loss_grid[i, j] <- mean(-y*log(sigmoid(x%*%w)) - (1-y) * log(1-sigmoid(x%*%w)))
  }
}

p <- persp3D(w1_vals,w2_vals,loss_grid,theta=45, phi=30, axes=TRUE,scale=2, box=TRUE, nticks=5,
        ticktype="detailed", xlab="", ylab="", zlab="")
```



```
p
```

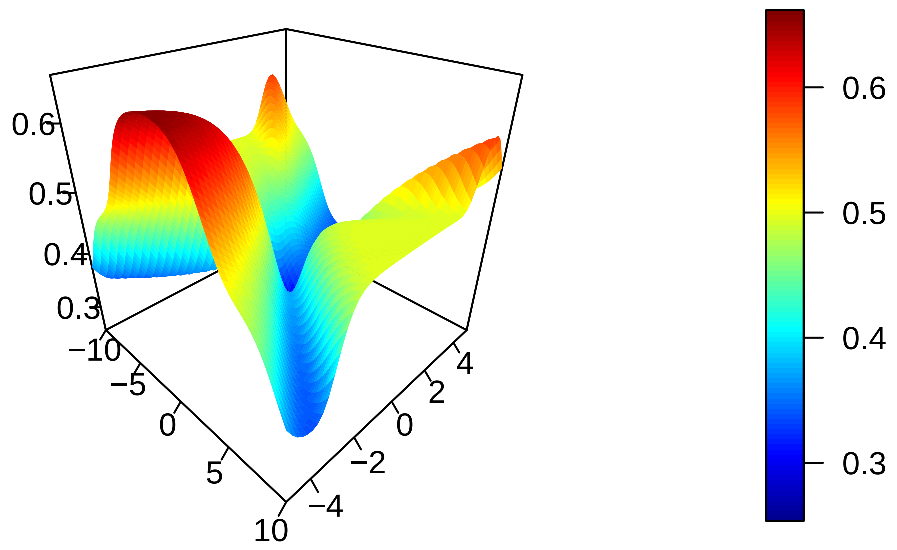
```
##                  [,1]        [,2]        [,3]        [,4]
## [1,]  7.071068e-02 -0.03535534  0.06123724 -0.06123724
## [2,]  1.414214e-01  0.07071068 -0.12247449  0.12247449
## [3,] -1.649186e-17  0.32987460  0.19045319 -0.19045319
## [4,]  5.484830e-17 -1.09709011 -3.36545608  4.36545608
```

## Part I

```
loss_grid <- matrix(0, nrow = length(w1_vals), ncol = length(w2_vals))

for (i in 1:length(w1_vals)) {
  for (j in 1:length(w2_vals)) {
    w <- c(w1_vals[i], w2_vals[j])
    loss_grid[i, j] <- mean(((sigmoid(x%*%w)) - y)^2)
  }
}

p <- persp3D(w1_vals,w2_vals,loss_grid,theta=45, phi=30, axes=TRUE,scale=2, box=TRUE, nticks=5,
        ticktype="detailed", xlab="", ylab="", zlab="")
```



```
p
```

```
##                [,1]        [,2]        [,3]        [,4]
## [1,]  7.071068e-02 -0.03535534  0.06123724 -0.06123724
## [2,]  1.414214e-01  0.07071068 -0.12247449  0.12247449
## [3,] -2.118787e-16  4.23805354  2.44684135 -2.44684135
## [4,]  9.700277e-17 -1.94027487 -3.85226903  4.85226903
```

### Part J

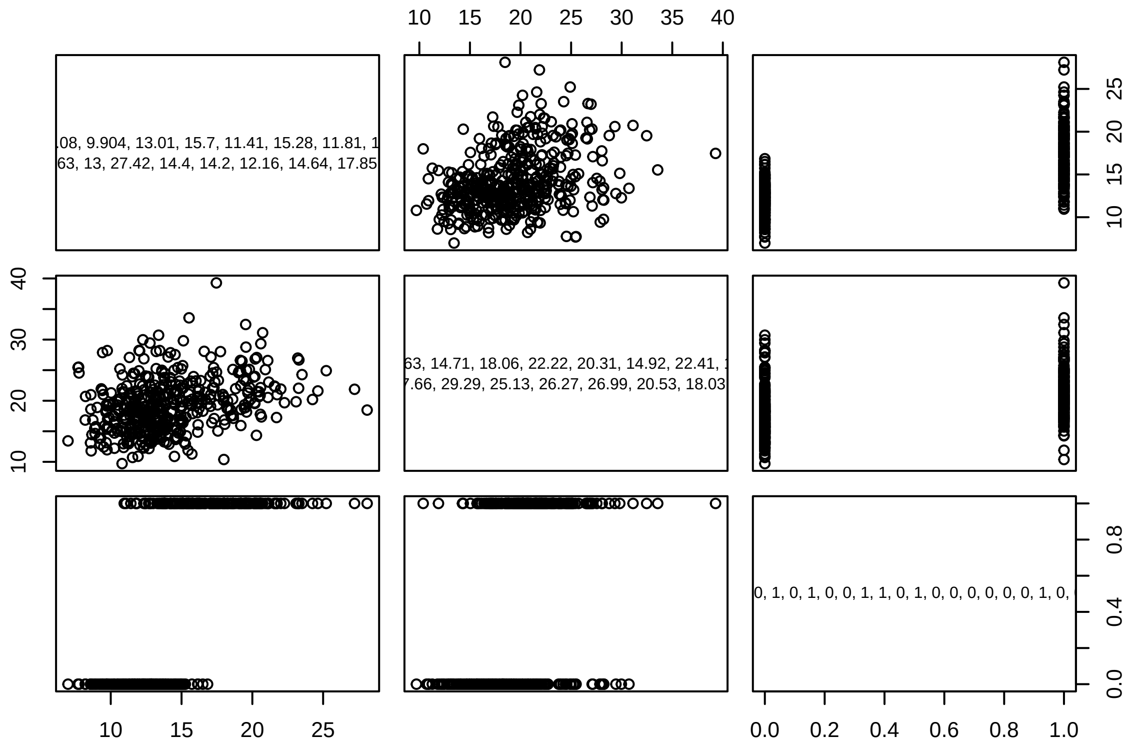
The first plot shows a bowl shaped graph and has a single minimum whereas the second plot shows multiple global minima and therefore is not convex. Therefore, it is better to use the first loss function as there is less complexity and easier and more effective to optimise.

## Question 2

### Part A

```
df <- read.csv("C:/Users/brian/Google Drive/uni/STAT8178/ass1/BinaryClassifier.csv", header=FALSE)
colnames(df) <- c("x1","x2","y")
train <- df[1:456,]
test <- df[457:569,]
```

```
plot(train, test)
```



The scatter plots of the train and test sets show that they are both randomly chosen.

## Part B

```
model <- glm(formula = y ~ x1 + x2, family = "binomial", data = train)
```

## Part C

```
library(dplyr)
```

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

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
## Loading required package: lattice
```

```
library(ggplot2)
test$prob <- predict(model, test, type = "response")
test <- test %>% mutate(pred = ifelse(prob>.5, "1", "0"))
confusionMatrix(factor(test$pred),factor(test$y), mode="everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 67  7
##          1  6 33
##
##                Accuracy : 0.885
##                  95% CI : (0.8113, 0.9373)
##     No Information Rate : 0.646
##     P-Value [Acc > NIR] : 7.219e-09
##
##                   Kappa : 0.747
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9178
##             Specificity : 0.8250
##          Pos Pred Value : 0.9054
##          Neg Pred Value : 0.8462
##               Precision : 0.9054
##                  Recall : 0.9178
##                      F1 : 0.9116
##              Prevalence : 0.6460
##          Detection Rate : 0.5929
##    Detection Prevalence : 0.6549
##       Balanced Accuracy : 0.8714
##
##        'Positive' Class : 0
##
```

The confusion matrix is shown above with the a good F1 score of 0.9116.

## Part D

```r
sigmoid<-function(a){
  return(1/(1+exp(-a)))
}
make_prediction <- function(w, x, classify = FALSE) {
  z <- sigmoid(sum(w * x))
  if (classify) {
    return (as.integer(z > 0.5))
  } else {
    return (z)
  }
}

cross_entropy <- function(y_true, y_pred) {
  data_num <- length(y_true)
  y_true <- as.numeric(y_true)
  y_pred <- as.numeric(y_pred)
  total <- -sum(y_true * log(y_pred) + (1 - y_true) * log(1 - y_pred))
  return (total / data_num)
}

gradient_descent <- function(alpha, epoch, weight, X, y, threshold, print_option = TRUE, get_cost = FAL
  y_pred <- sapply(X, function(x) make_prediction(weight, x))
  data_num <- length(y)
  cost <- c()
  for (i in 1:epoch) {
    dw <- sum((y_pred - y) * X) / data_num
    weight <- weight - alpha * dw
    y_pred <- sapply(X, function(x) make_prediction(weight, x))
    new_cost <- cross_entropy(y, y_pred)
    cost <- c(cost, new_cost)
    if (print_option && i %% 50 == 0) {
      cat(sprintf("Iteration %d, Cost: %f\n", i, new_cost))
    }
    if ((i > 3 && cost[length(cost) - 1] - cost[length(cost)]) < threshold) {
      break
    }
  }
  if (get_cost) {
    return (cost)
  } else {
    return (weight)
  }
}

logistic_regression <- function(training_set, label, test_set, alpha, epoch, threshold = 0.0001, print_
  weight <- runif(length(training_set[1]))
  if (get_cost) {
    cost <- gradient_descent(alpha, epoch, weight, training_set, label, threshold, print_option, get_co
    return (cost)
  } else {
    new_weight <- gradient_descent(alpha, epoch, weight, training_set, label, threshold, print_option)
    prediction <- sapply(test_set, function(instance) make_prediction(new_weight, instance, classify = T
    return (as.integer(prediction))
  }
}
```

## Part E

```
X_train <- train[,1:2]
X_test <- test[,1:2]
y_train <- train[,3]
y_test <- test[,3]
```

```
y_pred <- logistic_regression(X_train, y_train, X_test, 0.1, 1000, threshold=0.00001, print=TRUE)
```

I could not finish completing part E due to some errors and continued the assignment assuming that the code in part E works.

## Part F

```
accuracy_score <- function(y_true, y_pred) {
  count <- 0
  for (i in 1:length(y_true)) {
    if (y_true[i] == y_pred[i]) {
      count <- count + 1
    }
  }
  return (count / length(y_true))
}

plot_accuracy <- function(alpha, epoch, X_train, y_train, X_test, y_test) {
  accuracy <- c()
  iter_range <- seq(1, epoch)
  for (iter_num in iter_range) {
    y_hat <- logistic_regression(X_train, y_train, X_test, alpha, iter_num)
    accuracy <- c(accuracy, accuracy_score(y_hat, y_test))
  }
  plot(iter_range, accuracy, type = "l", col = "skyblue", xlab = "Epochs", ylab = "Accuracy")
}
```

The plot of convergence of the model can be shown by the following code:

plot_accuracy(0.1, 200)

The confusion matrix is provided by the following code:

```
confusionMatrix(factor(test$pred),factor(test$y), mode="everything")
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 67  7
##          1  6 33
##
##                Accuracy : 0.885
##                  95% CI : (0.8113, 0.9373)
##     No Information Rate : 0.646
##     P-Value [Acc > NIR] : 7.219e-09
##
##                   Kappa : 0.747
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9178
##             Specificity : 0.8250
##          Pos Pred Value : 0.9054
##          Neg Pred Value : 0.8462
##               Precision : 0.9054
##                  Recall : 0.9178
##                      F1 : 0.9116
##              Prevalence : 0.6460
##          Detection Rate : 0.5929
##    Detection Prevalence : 0.6549
##       Balanced Accuracy : 0.8714
##
##        'Positive' Class : 0
##
```

## Part G

The accuracy is computed by $(truepositives + truenegatives)/(allpredictions)$