Logistic regression algorithm implementation

For the below code:
* the `feature_matrix` is the matrix of features
* the `labels` is the vector of “true” classes
* the `learn_rate` is the learning rate for the gradient descent procedure
* the `iters` is the number of iterations to run gradient descent
* the `betas` is a vector of the coefficients
* the `n` is the number of predictors
* the `m` is the number of training examples (e.g., number of subjects)

The `sigmoid` logistic function:

```
sigmoid <- function(x){
  1/(1+exp(-x))
}
```

The `initialize_betas` function:

```
initialize_betas <- function(feature_matrix){
  betas <- as.matrix(rep(0, ncol(feature_matrix)))
}
```

The `cost` function to calculate logistic loss:

```
cost <- function(betas, feature_matrix, labels){
  m <- nrow(feature_matrix)
  probs <- sigmoid(feature_matrix%*%betas)
  L <- (1/m)*sum( labels * log(probs) + (1-labels) * log( 1-probs ) )
}
```

The `gradient` descent function to calculate the average gradient:

```
gradient <- function(betas, feature_matrix, labels){
  n <- ncol(feature_matrix)
  probs <- sigmoid(feature_matrix%*%betas)
  dcost_dbeta <- t(feature_matrix) %*% (probs-labels)
  avg_gradient <- (1/n)*dcost_dbeta
}
```

The `update_betas` function to use the gradient calculated above and learning rate to update the betas:

```
update_betas <- function(betas, feature_matrix, labels, learn_rate){
  gradient <- gradient(betas, feature_matrix, labels)
  betas <- betas - learn_rate*gradient
}
```

The `logistic_regression` function combining all of the above:

```
logistic_regression <- function(feature_matrix, labels, learn_rate, iters, print_every){

  # add a column of ones at the beginning of the feature matrix to calculate the
  # intercept
  feature_matrix <- cbind(as.matrix(rep(1,nrow(feature_matrix))),feature_matrix)

  # save the cost to print
  cost_print <- c()

  # get the initial beta values
  betas <- initialize_betas(feature_matrix)

  for(i in 1:iters){
    betas <- update_betas(betas, feature_matrix, labels, learn_rate)
    cost <- cost(betas, feature_matrix, labels)
    cost_print[i] <- cost

    if(i %% print_every == 0){
      print(paste("Iteration#:",i))
      print(paste("Cost:",cost_print[i]))
    }
  }

  return(list(betas = betas, final_cost = cost, all_costs = cost_print))

}
```

The predictor function to calculate predictions from betas obtained above:

```
predictor <- function(feature_matrix, betas){
  preds <- sigmoid(feature_matrix%*%betas)
  preds
}
```

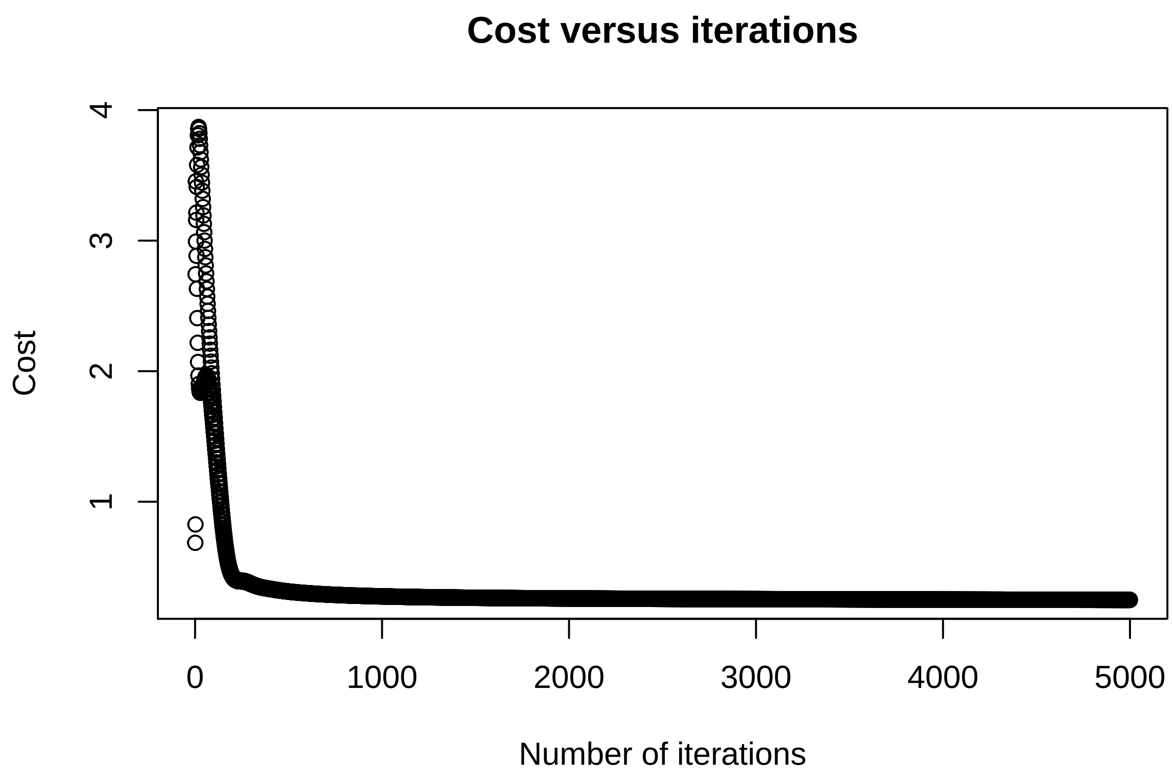
Train the model:

```
log_model <- logistic_regression(train_data, true_y, learn_rate = 0.1, iters = 5000,
                                 print_every = 1000)
```

```
## [1] "Iteration#: 1000"
## [1] "Cost: -0.274708347977415"
## [1] "Iteration#: 2000"
## [1] "Cost: -0.259049322661902"
## [1] "Iteration#: 3000"
## [1] "Cost: -0.253871873882793"
## [1] "Iteration#: 4000"
## [1] "Cost: -0.250626029160736"
## [1] "Iteration#: 5000"
## [1] "Cost: -0.248113438653087"
```

Plot cost by iterations:

```
plot(1:length(log_model$all_costs), -1*log_model$all_costs, main = "Cost versus iterations",
     xlab = "Number of iterations", ylab = "Cost")
```

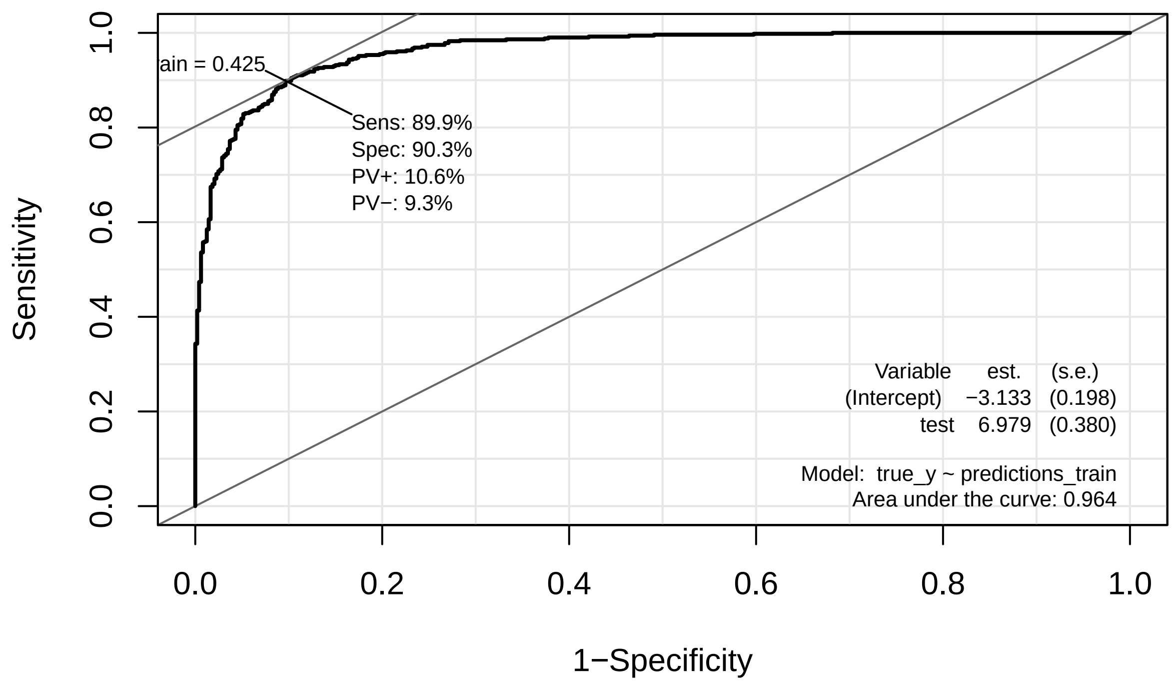


Test the model:

```
# add a row of 1s to the train_data to conform to betas having an intercept term calculated
train_data_1 <- cbind(1, train_data)

# assess the predictions on the training data
predictions_train <- predictor(train_data_1, log_model$betas)

# find the optimal cutoff with an ROC curve
ROC(test = predictions_train, stat = true_y, plot = "ROC")
```



The best cutoff based on the ROC curve is 0.48 to maximize both sensitivity and specificity, 91.4% and 89.7%, respectively.

Make predictions using this cutoff value:

```
y_preds <- ifelse(predictions_train<0.425, 0, 1)

# print confusion matrix
table(true_y, y_preds)
```

```
##         y_preds
## true_y    0   1
##       0 440  47
##       1  52 461
```

Make predictions on the test data:

```
# add a row of 1s to the test_data to conform to betas having an intercept term calculated
test_data_1 <- cbind(1, test_data)

predictions_test <- predictor(test_data_1, log_model$betas)

# using the same cutoff value that was found above on the train data
y_test <- ifelse(predictions_test<0.425, 0, 1)
table(y_test)
```

```
## y_test
##   0   1
## 479 521
```