Machine Learning

Logistic Regression

Ayşe Ceren Çiçek

Logistic Regression is a classification model which is used to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.

- The dependent variable should be binary like yes or no.
- It can help you predict the likelihood of an event happening or a choice being made.

Linear Regression outputs continuous value, and it has a straight line to map the input variables to the dependent variables. The output can be any of an infinite number of possibilities. On the other hand, Logistic Regression uses a logistic function to map the input variables to categorical dependent variables. In contrast to Linear Regression, Logistic Regression outputs a probability between 0 and 1.

Dataset: https://www.kaggle.com/rakeshrau/social-network-ads

This dataset shows which of the users purchased/not purchased a particular product.

The columns are:

- User ID
- Gender
- Age
- EstimatedSalary
- Purchased

Importing libraries

```r
library(caTools)
library(caret)
```

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

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

```r
library(dplyr)
```

```
##
## 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
```

Loading dataset

```
dataset = read.csv('dataset.csv')
dataset = dataset[, 3:5]
head(dataset, n=5)
```

```
##   Age EstimatedSalary Purchased
## 1  19           19000         0
## 2  35           20000         0
## 3  26           43000         0
## 4  27           57000         0
## 5  19           76000         0
```

We will only use Age, Salary and Purchased columns.

Split data into Train and Test set

Purchased column is our dependent variable.

```
set.seed(123)
splitted = sample.split(dataset$Purchased, SplitRatio = 0.75)
trainSet = subset(dataset, splitted == TRUE)
testSet = subset(dataset, splitted == FALSE)
```

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data.

We will scale all the features except our dependent variable, Purchased.

```
trainSet[, 1:2] = scale(trainSet[, 1:2])
testSet[, 1:2] = scale(testSet[,1:2])
```

Apply Logistic Regression

```
model = glm(formula = Purchased ~ ., family = binomial, data = trainSet)
```

Prediction

Probability prediction show us predicted probabilities that the user will buy the product.

```
probability.prediction = predict(model, type = 'response', newdata = testSet[-3])
probability.prediction
```

```
##            2            4            5            9           12           18
## 0.0162395375 0.0117148379 0.0037846461 0.0024527456 0.0073339436 0.2061576580
##           19           20           22           29           32           34
## 0.2669935073 0.3851475689 0.5448578778 0.0103005636 0.2994922143 0.0084168787
##           35           38           45           46           48           52
## 0.0494471952 0.0171641479 0.0485051303 0.0008343060 0.0102561619 0.0007055347
##           66           69           74           75           82           84
## 0.0058448457 0.0044534947 0.3933468488 0.0071065671 0.1068589185 0.2580084947
##           85           86           87           89          103          104
## 0.0303248927 0.3303649169 0.0051132916 0.0263861849 0.1310148056 0.7649772313
##          107          108          109          117          124          126
## 0.0034367786 0.0473827096 0.0327965105 0.1626049288 0.0675494054 0.2189658514
##          127          131          134          139          148          154
## 0.4142562486 0.0324337750 0.0043457839 0.0163538708 0.1030590600 0.0751093248
##          156          159          162          163          170          175
## 0.0048556976 0.0027487256 0.0306647902 0.0463555716 0.0122981409 0.1169016711
##          176          193          199          200          208          213
## 0.0011936610 0.0103005636 0.0252589417 0.0177353905 0.9870859806 0.9453359968
##          224          226          228          229          230          234
## 0.9969454446 0.1064430571 0.9979393884 0.3705093415 0.5807527959 0.9117762840
##          236          237          239          241          255          264
## 0.7817273411 0.2310672929 0.8037996043 0.9682706714 0.6686007827 0.1451169281
##          265          266          273          274          281          286
## 0.9060311409 0.8293112410 0.9568520348 0.6781064291 0.9926955397 0.4170486388
##          292          299          302          305          307          310
## 0.9220096987 0.7363498859 0.8247736816 0.2558136823 0.9932007105 0.1178058928
##          316          324          326          332          339          341
## 0.3442845494 0.3958138650 0.3059412440 0.9725035550 0.1431602303 0.9842795480
##          343          347          353          363          364          367
## 0.2073273008 0.9371909698 0.6843940060 0.5559479117 0.5698028861 0.9440512240
##          368          369          372          373          380          383
## 0.8427877409 0.2549836305 0.9928717092 0.3243409327 0.8519685008 0.9697473704
##          389          392          395          400
## 0.3793408625 0.2718336775 0.2040229226 0.5236436275
```

But we need binary variable like 0 and 1. If the probability is greater than 0.5 turn 1, otherwise return 0.

```
prediction = ifelse(probability.prediction > 0.5, 1, 0)
prediction
```

```
##   2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48  52  66  69
##   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0
##  74  75  82  84  85  86  87  89 103 104 107 108 109 117 124 126 127 131 134 139
##   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
## 148 154 156 159 162 163 170 175 176 193 199 200 208 213 224 226 228 229 230 234
##   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   0   1   0   1   1
## 236 237 239 241 255 264 265 266 273 274 281 286 292 299 302 305 307 310 316 324
##   1   0   1   1   1   0   1   1   1   1   1   0   1   1   1   0   1   0   0   0
## 326 332 339 341 343 347 353 363 364 367 368 369 372 373 380 383 389 392 395 400
##   0   1   0   1   0   1   1   1   1   1   1   0   1   0   1   1   0   0   0   1
```

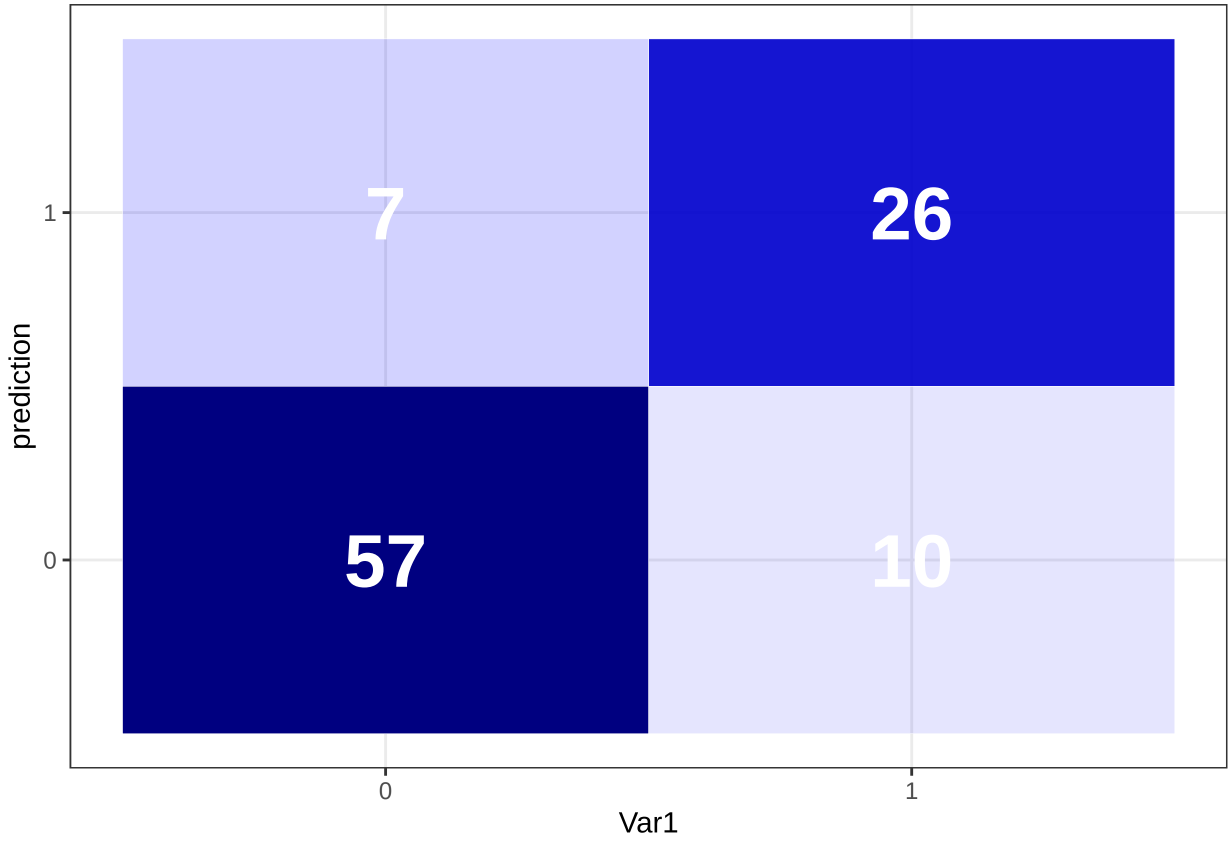
Confusion Matrix

```
# Generate confusion matrix
conf.matrix <- confusionMatrix(table(testSet[, 3], prediction))
conf.matrix
```

```
## Confusion Matrix and Statistics
##
##    prediction
##      0  1
##   0 57  7
##   1 10 26
##
##                Accuracy : 0.83
##                  95% CI : (0.7418, 0.8977)
##     No Information Rate : 0.67
##     P-Value [Acc > NIR] : 0.0002624
##
##                   Kappa : 0.6242
##
##  Mcnemar’s Test P-Value : 0.6276258
##
##             Sensitivity : 0.8507
##             Specificity : 0.7879
##          Pos Pred Value : 0.8906
##          Neg Pred Value : 0.7222
##              Prevalence : 0.6700
##          Detection Rate : 0.5700
##    Detection Prevalence : 0.6400
##       Balanced Accuracy : 0.8193
##
##        ’Positive’ Class : 0
##
```

The accurary is 83%. We have 10 + 7 incorrect classifications.

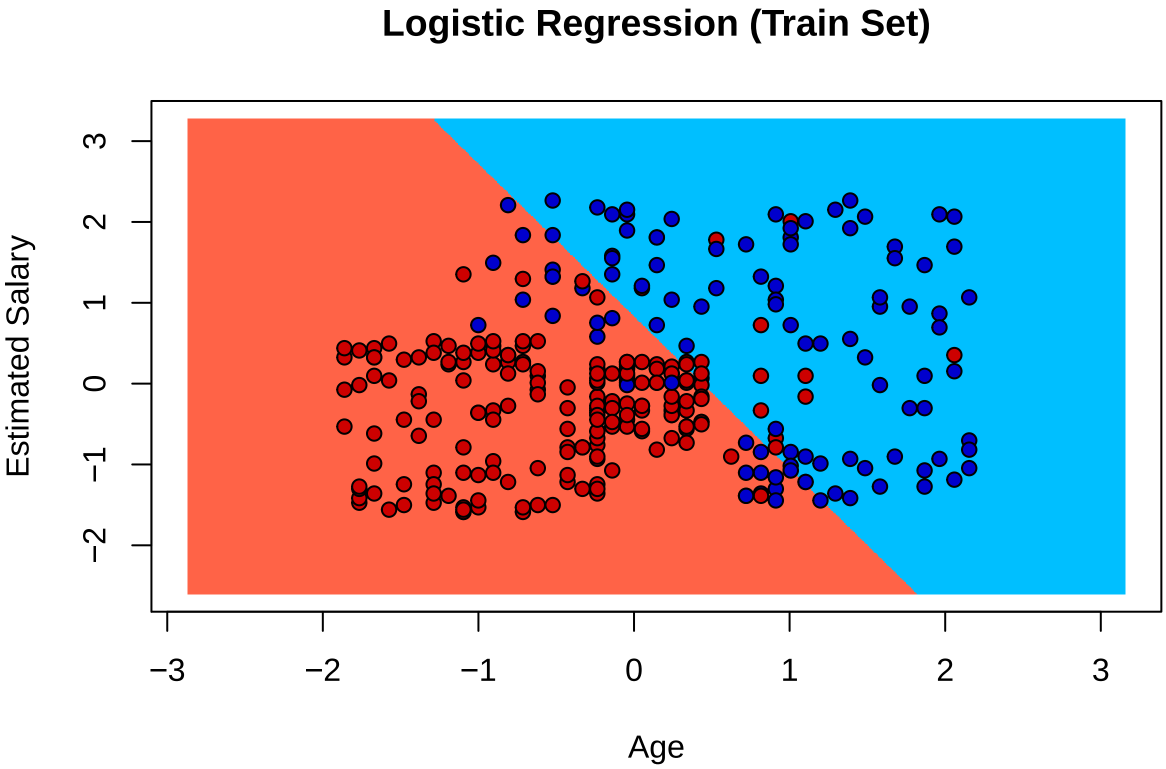
```
# Heatmap visualization of confusion matrix
table <- data.frame(conf.matrix$table)


plotTable <- table %>%
  group_by(prediction) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data =  plotTable, mapping = aes(x = Var1, y = prediction, alpha = prop)) +
  geom_tile(aes(fill = Freq), colour = "white") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1, color="white", size=10) +
  scale_fill_gradient(low = "blue", high = "navyblue") +
  theme_bw() + theme(legend.position = "none")
```



Visualize Train Set Results

```
set = trainSet
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)

colnames(grid_set) = c('Age', 'EstimatedSalary')

prob_set = predict(model, type = 'response', newdata = grid_set)

y_grid = ifelse(prob_set > 0.5, 1, 0)

plot(set[, -3], main = 'Logistic Regression (Train Set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
```

```
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'deepskyblue', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'blue3', 'red3'))
```



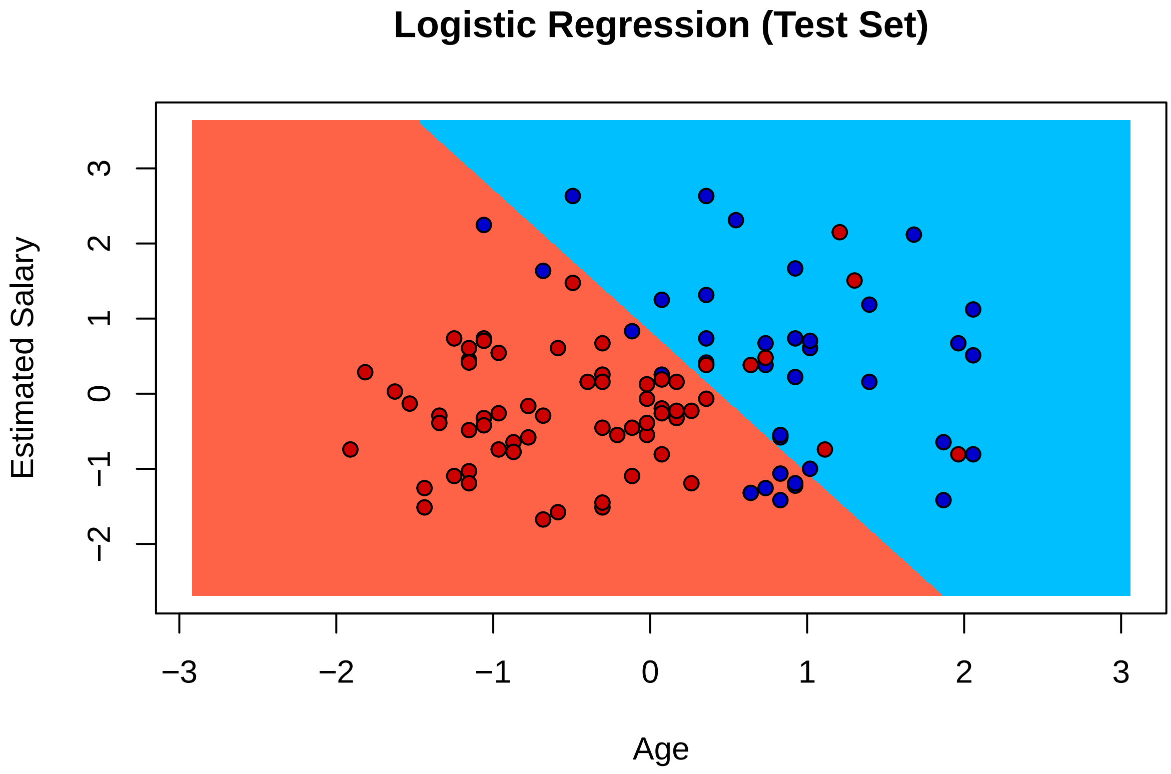
Young people who did not buy the product are on the red are with red points. On the other hand, older people who bought the product are on the blue part with blue points. The blue points which are on the red side are the ones who are young but bought the product. And those red points which are on the blue part, they are older people who did not buy the product.

The straight line seen on the plot is called the Prediction Boundary.

Visualize Test Set Results

```
set = testSet
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)

colnames(grid_set) = c('Age', 'EstimatedSalary')

prob_set = predict(model, type = 'response', newdata = grid_set)

y_grid = ifelse(prob_set > 0.5, 1, 0)
```

```
plot(set[, -3], main = 'Logistic Regression (Test Set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'deepskyblue', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'blue3', 'red3'))
```



Majority of the blue and red points are on the right parts of the plot. We have 17 incorrect predictions.