Pokemon Project

Data

— add description here

Feature extraction

Import the df and visualize the columns' name.

```
df <- read.csv("data.csv")
names(df)
```

```
##  [1] "X."         "Name"       "Type.1"     "Type.2"     "Total"
##  [6] "HP"         "Attack"     "Defense"    "Sp..Atk"    "Sp..Def"
## [11] "Speed"      "Generation" "Legendary"
```

Drop useless columns X., Name.

```
df <- df |>
    mutate(Legendary = as.integer(as.logical(Legendary))) |>
    select(-X., -Name)
```

Check that Tot is just a linear combination of other columns. If yes, drop it.

```
if (all((df$HP + df$Attack + df$Defense + df$SP..Attack + df$SP..Defense + df$Speed) == df$Total)) {
    df <- df |> select(-Total)
}
```

Hot-encoding from Type.1 and Type.2.

```
unique_types <- c(df$Type.1, df$Type.2) |> unique()

for (typ in unique_types[-1]) {
    if (typ == "") next
    df[typ] <- 0
    for (row in 1:nrow(df)) {
        has_type <- typ %in% df[row, c("Type.1", "Type.2")]
        if (has_type) {
            df[row, typ] <- 1
        }
    }
}
```

Encode Generation.

```
for (i in unique(df$Generation)) {
    if (i == 1) next
    col_name <- paste0("gen_", i)
    df[col_name] <- 0
}

for (row in 1:nrow(df)) {
```

```r
    gen <- df[row, "Generation"]
    if (gen == 1) next
    col_name <- paste0("gen_", gen)
    df[row, col_name] <- 1
}
```

Drop the categorical columns that we don't need anymore and set Legendary to factor.

```r
df <- df |>
    select(-Type.1, -Type.2, -Generation) |>
    mutate(Legendary = as.factor(ifelse(Legendary == 0, "No", "Yes")))
```

```r
colnames(df)[c(1, 4, 5)] <- c("Hit_Point", "Special_Attack", "Special_Defense")
```

```r
set.seed(123)
train_test_split <- function(df, perc_train = 0.7) {
    i_train <- sample(1:nrow(df), floor(0.7 * nrow(df)), F)
    list_out <- list(train = df[i_train, ], test = df[-i_train, ])
    return(list_out)
}
df_split <- train_test_split(df)
```

```r
train <- df_split$train
test <- df_split$test
```

```r
numerical_vars <- names(train)[1:6]
```

Verify whether the prob of having Legendary is equal accross Generation to decide whether to keep the variable

```r
# train %>%
#    group_by(Generation) %>%
#    summarise(mean(Legendary))
```

Categorical data

```r
train %>%
    select(-numerical_vars, -Legendary) %>%
    colMeans() * 100 # Percentage values
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(numerical_vars)
##
##   # Now:
##   data %>% select(all_of(numerical_vars))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
##      Fire     Water       Bug    Normal    Poison  Electric    Ground     Fairy
##  8.214286 17.678571  9.107143 11.964286  7.142857  6.428571  9.107143  5.178571
##  Fighting   Psychic      Rock     Ghost       Ice    Dragon      Dark     Steel
##  6.607143 11.071429  7.142857  5.535714  4.642857  5.714286  5.892857  6.250000
```

```
##    Flying     gen_2     gen_3     gen_4     gen_5     gen_6
## 12.678571 13.750000 20.535714 14.464286 21.250000  9.642857
```

```
# Numerical Variables
train %>%
    select(numerical_vars) %>%
    summary()
```

```
##    Hit_Point          Attack          Defense       Special_Attack
##  Min.   :  1.00   Min.   :  5.00   Min.   :  5.00   Min.   : 10.00
##  1st Qu.: 54.00   1st Qu.: 55.00   1st Qu.: 50.00   1st Qu.: 50.00
##  Median : 65.50   Median : 75.00   Median : 70.00   Median : 65.00
##  Mean   : 70.39   Mean   : 79.29   Mean   : 73.81   Mean   : 72.60
##  3rd Qu.: 84.00   3rd Qu.:100.00   3rd Qu.: 90.00   3rd Qu.: 94.25
##  Max.   :255.00   Max.   :190.00   Max.   :230.00   Max.   :194.00
##  Special_Defense      Speed
##  Min.   : 20.00   Min.   : 10.00
##  1st Qu.: 53.00   1st Qu.: 45.75
##  Median : 70.00   Median : 65.00
##  Mean   : 71.79   Mean   : 68.59
##  3rd Qu.: 87.00   3rd Qu.: 90.00
##  Max.   :200.00   Max.   :160.00
```

```
par(mfrow = c(2, 4), mar = c(3, 3, 3, 3))
lapply(numerical_vars, function(col_name) {
    hist(train[[col_name]], main = paste("Histogram of", col_name), xlab = "variable")
})
```

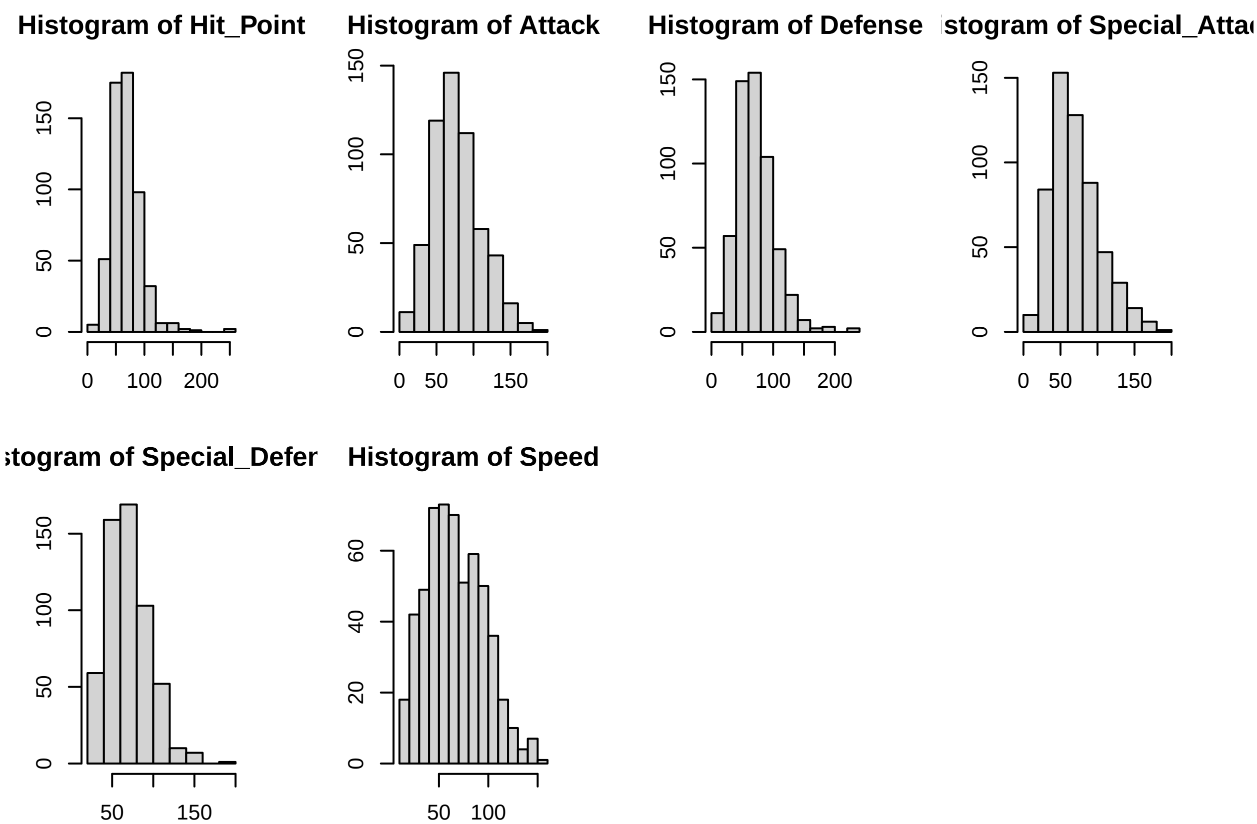
```
## [[1]]
## $breaks
##  [1]   0  20  40  60  80 100 120 140 160 180 200 220 240 260
##
## $counts
##  [1]   5  51 175 182  98  32   6   6   2   1   0   0   2
##
## $density
##  [1] 4.464286e-04 4.553571e-03 1.562500e-02 1.625000e-02 8.750000e-03
##  [6] 2.857143e-03 5.357143e-04 5.357143e-04 1.785714e-04 8.928571e-05
## [11] 0.000000e+00 0.000000e+00 1.785714e-04
##
## $mids
##  [1]  10  30  50  70  90 110 130 150 170 190 210 230 250
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[2]]
## $breaks
##  [1]   0  20  40  60  80 100 120 140 160 180 200
##
```

```
## $counts
##  [1]  11  49 119 146 112  58  43  16   5   1
##
## $density
##  [1] 9.821429e-04 4.375000e-03 1.062500e-02 1.303571e-02 1.000000e-02
##  [6] 5.178571e-03 3.839286e-03 1.428571e-03 4.464286e-04 8.928571e-05
##
## $mids
##  [1]  10  30  50  70  90 110 130 150 170 190
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[3]]
## $breaks
##  [1]   0  20  40  60  80 100 120 140 160 180 200 220 240
##
## $counts
##  [1]  11  57 149 154 104  49  22   7   2   3   0   2
##
## $density
##  [1] 0.0009821429 0.0050892857 0.0133035714 0.0137500000 0.0092857143
##  [6] 0.0043750000 0.0019642857 0.0006250000 0.0001785714 0.0002678571
## [11] 0.0000000000 0.0001785714
##
## $mids
##  [1]  10  30  50  70  90 110 130 150 170 190 210 230
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[4]]
## $breaks
##  [1]   0  20  40  60  80 100 120 140 160 180 200
##
## $counts
##  [1]  10  84 153 128  88  47  29  14   6   1
##
## $density
##  [1] 8.928571e-04 7.500000e-03 1.366071e-02 1.142857e-02 7.857143e-03
##  [6] 4.196429e-03 2.589286e-03 1.250000e-03 5.357143e-04 8.928571e-05
##
```

```
## $mids
##  [1]  10  30  50  70  90 110 130 150 170 190
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[5]]
## $breaks
##  [1]  20  40  60  80 100 120 140 160 180 200
##
## $counts
## [1]  59 159 169 103  52  10   7   0   1
##
## $density
## [1] 5.267857e-03 1.419643e-02 1.508929e-02 9.196429e-03 4.642857e-03
## [6] 8.928571e-04 6.250000e-04 0.000000e+00 8.928571e-05
##
## $mids
## [1]  30  50  70  90 110 130 150 170 190
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[6]]
## $breaks
##  [1]  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
##
## $counts
##  [1] 18 42 49 72 73 70 51 59 50 36 18 10  4  7  1
##
## $density
##  [1] 0.0032142857 0.0075000000 0.0087500000 0.0128571429 0.0130357143
##  [6] 0.0125000000 0.0091071429 0.0105357143 0.0089285714 0.0064285714
## [11] 0.0032142857 0.0017857143 0.0007142857 0.0012500000 0.0001785714
##
## $mids
##  [1]  15  25  35  45  55  65  75  85  95 105 115 125 135 145 155
##
## $xname
## [1] "train[[col_name]]"
##
## $equidist
```

```
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```r
par(mfrow = c(1, 1))
```



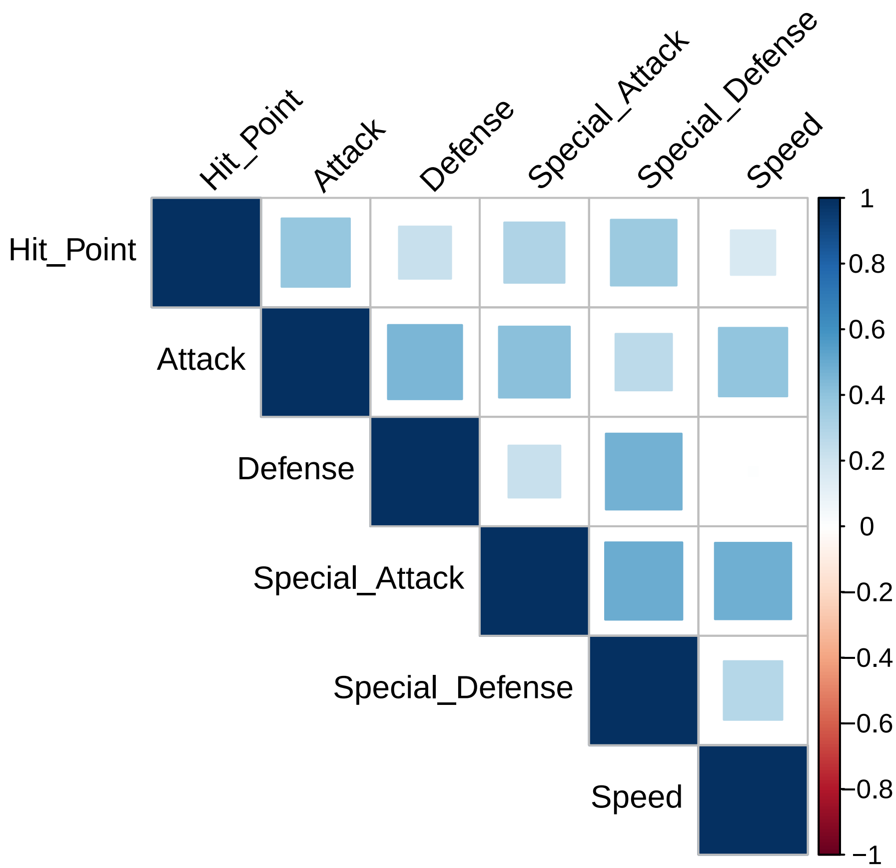

```r
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```r
cor_mat <- cor(train[numerical_vars])
print(cor_mat)
```

```
##                 Hit_Point    Attack    Defense Special_Attack Special_Defense
## Hit_Point       1.0000000 0.3890117 0.22626454      0.3025508       0.3600523
## Attack          0.3890117 1.0000000 0.45833086      0.4189784       0.2655658
## Defense         0.2262645 0.4583309 1.00000000      0.2245206       0.4795059
## Special_Attack  0.3025508 0.4189784 0.22452059      1.0000000       0.4970868
## Special_Defense 0.3600523 0.2655658 0.47950586      0.4970868       1.0000000
## Speed           0.1647429 0.3902713 0.00647584      0.4858104       0.2862043
##                      Speed
## Hit_Point       0.16474292
## Attack          0.39027129
## Defense         0.00647584
## Special_Attack  0.48581040
## Special_Defense 0.28620434
## Speed           1.00000000
```

```r
corrplot(cor_mat, method = "square", type = "upper", tl.col = "black", tl.srt = 45)
```




```r
train_std <- train
train_std[numerical_vars] <- train |>
    select(numerical_vars) |>
    mutate(across(where(is.numeric), scale))

# Apply the same scaling to test data
test_std <- test
test_std[numerical_vars] <- test |>
    select(numerical_vars) |>
    mutate(across(where(is.numeric), scale))
```

Checks

```r
sapply(list(mean = mean, sd = sd), mapply, train_std |> select(numerical_vars))
```

```
##                          mean sd
## Hit_Point        1.462889e-16  1
## Attack          -4.257643e-17  1
## Defense         -2.249584e-16  1
## Special_Attack   2.896194e-17  1
## Special_Defense  2.191568e-16  1
## Speed            4.712891e-17  1
```

```r
sapply(list(mean = mean, sd = sd), mapply, test_std |> select(numerical_vars))
```

```
##                          mean sd
```

```
## Hit_Point        1.084229e-16  1
## Attack          -1.987194e-16  1
## Defense          8.196794e-17  1
## Special_Attack  -8.627719e-17  1
## Special_Defense -2.920344e-17  1
## Speed           -1.651078e-16  1
```

```
extract_best_roc <- function(fit) fit$results[which.max(fit$results$ROC), ]
```

Fitting a logistic elastic net. Grid search of optimal alpha and lambda. CV. Maximize ROC.

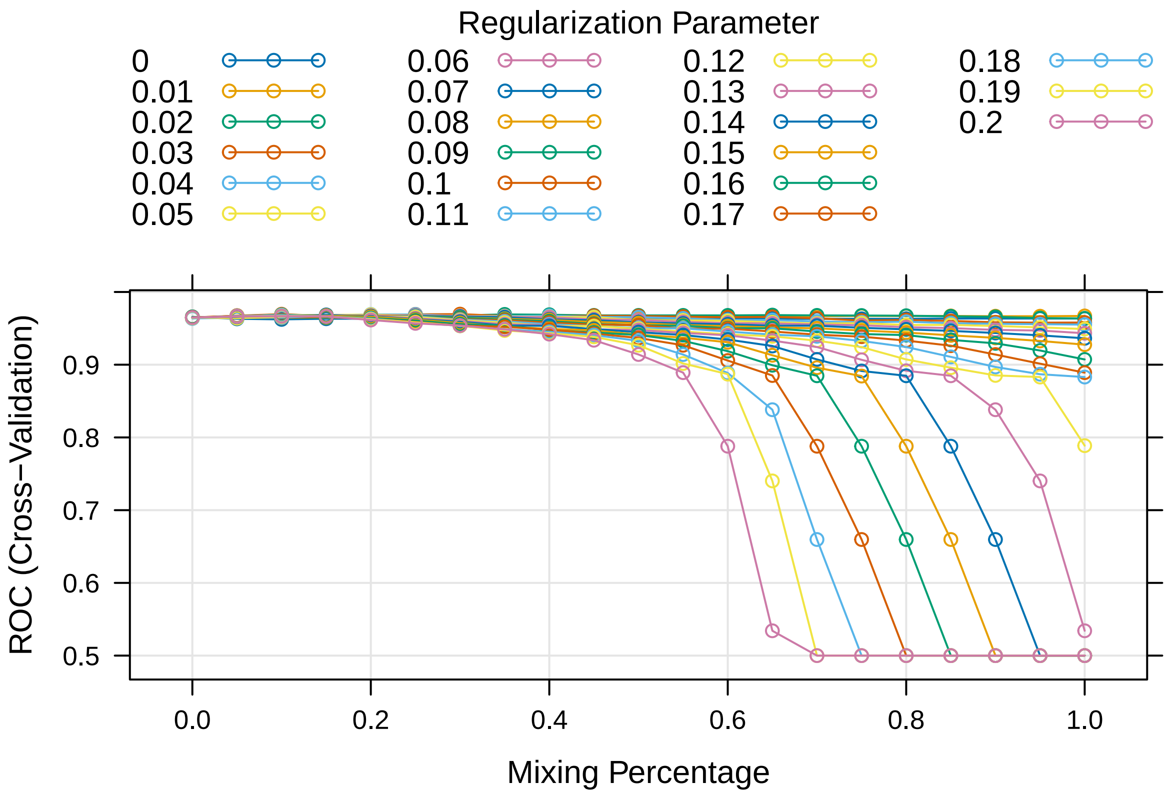
```
grid_ <- expand.grid(
    .alpha = seq(0, 1, by = 0.05),
    .lambda = seq(0, 0.2, by = 0.01)
)

train_control_1 <- trainControl(
    method = "cv",
    number = 10,
    classProbs = TRUE,
    summaryFunction = twoClassSummary
)

logistic_elnet_1 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "ROC",
    tuneGrid = grid_,
    trControl = train_control_1
)

print(logistic_elnet_1$bestTune)
```

```
##     alpha lambda
## 130   0.3   0.03
```

```
print(extract_best_roc(logistic_elnet_1))
```

```
##     alpha lambda       ROC      Sens Spec     ROCSD      SensSD    SpecSD
## 130   0.3   0.03 0.9695305 0.9980392 0.41 0.0248259 0.006200544 0.2875181
```

```
plot(logistic_elnet_1)
```



Fitting a logistic elastic net. Grid search of optimal alpha and lambda. SMOTE CV. Maximize ROC.

```
train_control_2 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "smote",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
)

logistic_elnet_2 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "ROC",
    tuneGrid = grid_,
    trControl = train_control_2
)
```

```
## Loading required package: recipes

##
## Attaching package: 'recipes'

## The following object is masked from 'package:stringr':
##
##     fixed
```

```
## The following object is masked from 'package:stats':
##
##     step
```

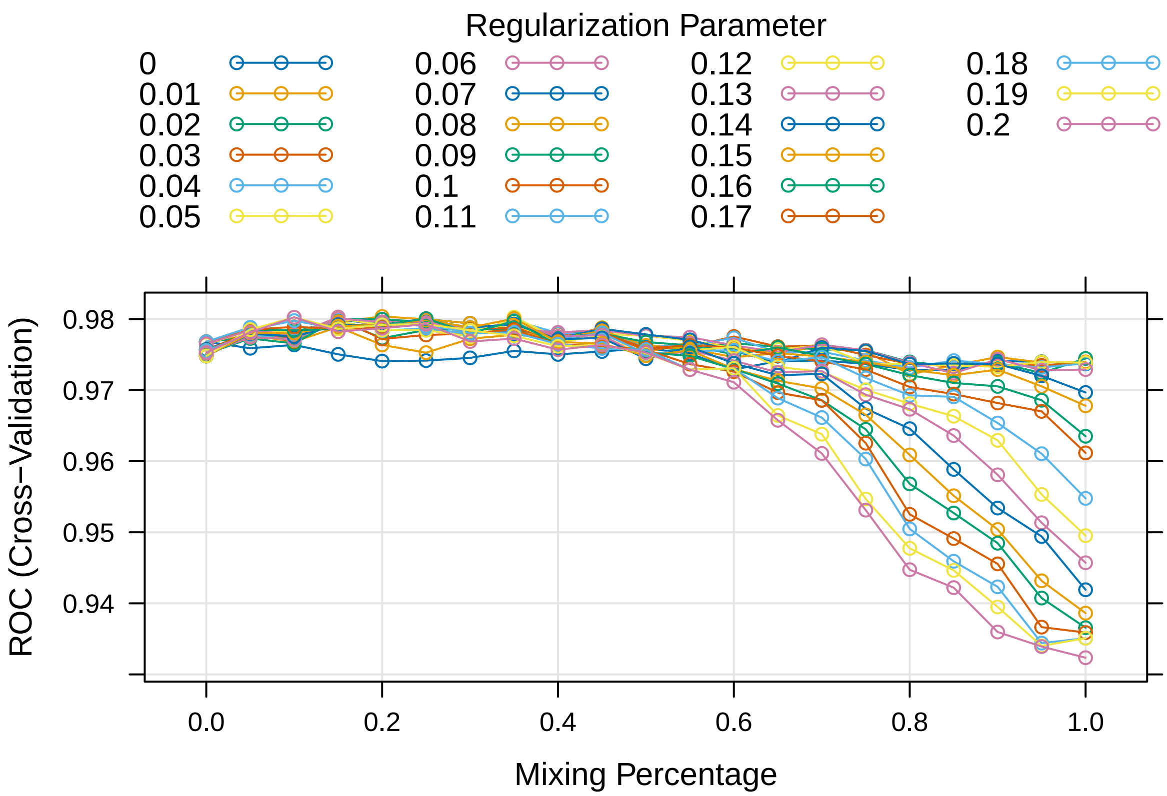
```r
print(logistic_elnet_2$bestTune)
```

```
##    alpha lambda
## 93   0.2   0.08
```

```r
print(extract_best_roc(logistic_elnet_2))
```

```
##    alpha lambda       ROC      Sens Spec      ROCSD     SensSD     SpecSD
## 93   0.2   0.08 0.9803601 0.9143288 0.98 0.01310457 0.0295435 0.06324555
```

```r
plot(logistic_elnet_2)
```



Fitting a logistic elastic net. Grid search of optimal alpha and lambda. Upsampling CV. Maximize ROC.

```r
train_control_3 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "up",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
)

logistic_elnet_3 <- train(
    Legendary ~ .,
    data = train_std,
```

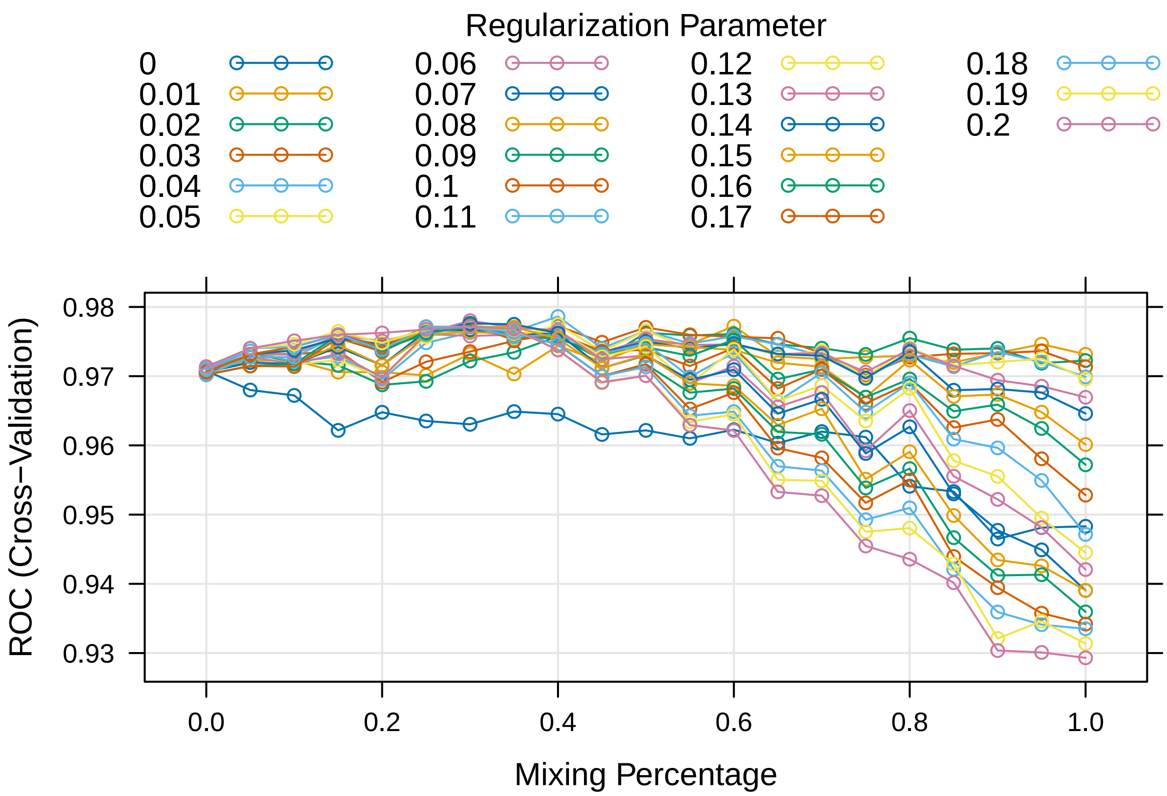
```r
    method = "glmnet",
    family = "binomial",
    metric = "ROC",
    tuneGrid = grid_,
    trControl = train_control_3
)
print(logistic_elnet_3$bestTune)
```

```
##     alpha lambda
## 173   0.4   0.04
```

```r
print(extract_best_roc(logistic_elnet_3))
```

```
##     alpha lambda       ROC      Sens  Spec      ROCSD     SensSD     SpecSD
## 173   0.4   0.04 0.9786086 0.9105204 0.975 0.02179183 0.06424069 0.07905694
```

```r
plot(logistic_elnet_3)
```



Fitting a logistic elastic net. Grid search of optimal alpha and lambda. Downsampling CV. Maximize ROC.

```r
train_control_4 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "down",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
)
```

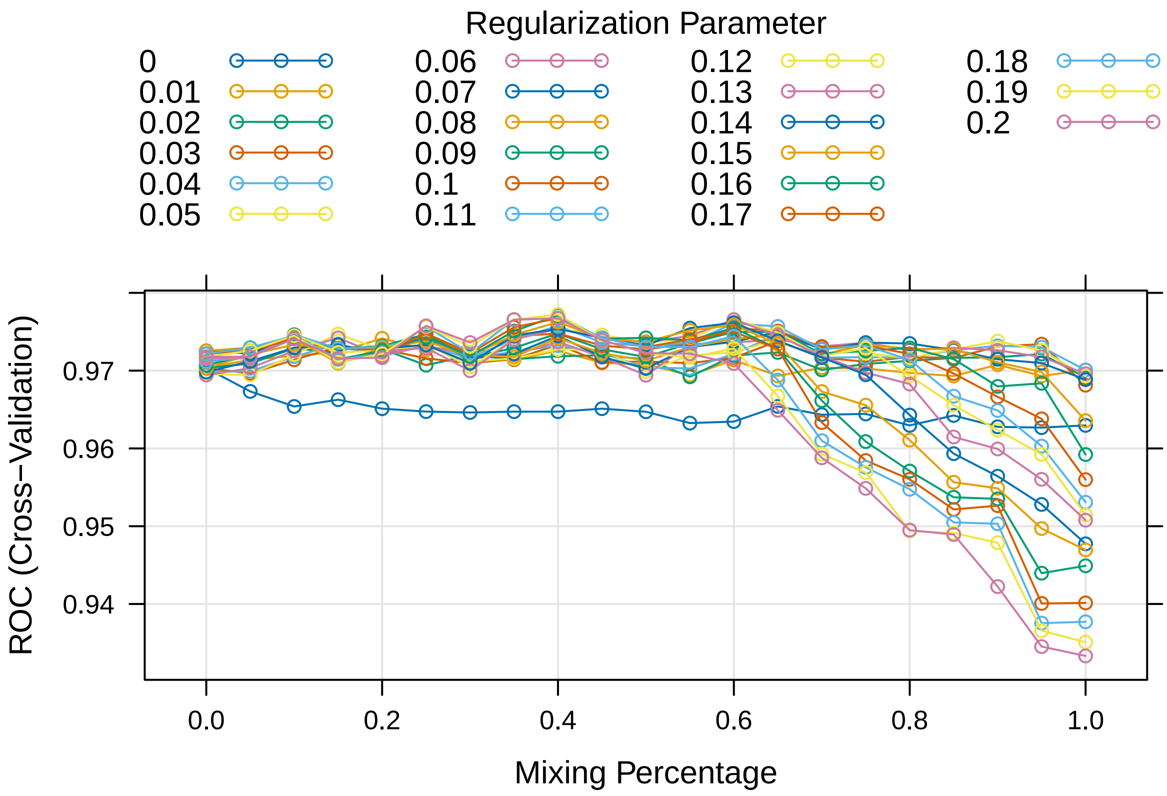
```r
logistic_elnet_4 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "ROC",
    tuneGrid = grid_,
    trControl = train_control_3
)
print(logistic_elnet_4$bestTune)
```

```
##     alpha lambda
## 188   0.4   0.19
```

```r
print(extract_best_roc(logistic_elnet_4))
```

```
##     alpha lambda       ROC      Sens Spec      ROCSD     SensSD     SpecSD
## 188   0.4   0.19 0.9772247 0.8791855 0.98 0.02090198 0.05468834 0.06324555
```

```r
plot(logistic_elnet_4)
```



Prediction

```r
get_prediction <- function(fit, type = "raw") {
    predict(fit, type = type, newdata = test_std |> select(-Legendary))
}
```

```r
get_accuracy <- function(fit) {
    y <- test_std$Legendary
    y_hat <- predict(fit, type = "raw", newdata = test_std |>
        select(-Legendary))
    return(mean(y == y_hat))
}
```

```r
lapply(list(logistic_elnet_1, logistic_elnet_2, logistic_elnet_3, logistic_elnet_4), get_accuracy)
```

```
## [[1]]
## [1] 0.925
##
## [[2]]
## [1] 0.925
##
## [[3]]
## [1] 0.9125
##
## [[4]]
## [1] 0.8833333
```

```r
# test_std <- test_std %>% select(-Legendary)
# best_elnet_pred <- predict(logistic_elnet, newx = test_std, type = "raw")
# print(best_elnet_pred)
```

Now we check:

```r
# test$Legendary <- as.factor(test$Legendary)
# levels(test$Legendary) <- c("No", "Yes")
# confusionMatrix(best_elnet_pred, test$Legendary)

# nrow(train_std)
# nrow(test_std)
# length(best_elnet_pred)
```