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)
}
```

One-hot encoding from Type.1 and Type.2.

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

for (typ in unique_types) {
    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)) {
```

```
    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.

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

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

```
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)
```

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

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

Categorical data

We start by checking the frequency of Characteristics across our Pokemon population in the training sample:

```
train %>%
    select(-numerical_vars, -Legendary, -gen_1, -gen_2, -gen_3, -gen_4, -gen_5, -gen_6) %>%
    summarise(across(everything(), sum, na.rm = TRUE)) %>%
    pivot_longer(cols = everything(), names_to = "Feature", values_to = "Total_Sum") %>%
    mutate(Percentage = (Total_Sum / sum(Total_Sum)) * 100) %>%
    select(Feature, Percentage) %>%
    kable()
```

Feature	Percentage
Grass	7.420495
Fire	5.418139
Water	11.660777
Bug	6.007067
Normal	7.891637
Poison	4.711425
Electric	4.240283
Ground	6.007067
Fairy	3.415783
Fighting	4.358068
Psychic	7.302709
Rock	4.711425
Ghost	3.651355
Ice	3.062426
Dragon	3.769140

Feature	Percentage
Dark	3.886926
Steel	4.122497
Flying	8.362780

```r
train %>%
    select(-numerical_vars, -gen_1, -gen_2, -gen_3, -gen_4, -gen_5, -gen_6) |>
    pivot_longer(cols = !Legendary, names_to = "Feature", values_to = "value") |>
    filter(value != 0) |>
    group_by(Feature) |>
    summarise("% Legendary" = 100*mean(as.numeric(Legendary) - 1)) |>
    kable()
```

Feature	% Legendary
Bug	0.000000
Dark	6.060606
Dragon	34.375000
Electric	8.333333
Fairy	10.344828
Fighting	8.108108
Fire	15.217391
Flying	12.676056
Ghost	6.451613
Grass	1.587302
Ground	3.921569
Ice	7.692308
Normal	2.985075
Poison	0.000000
Psychic	22.580645
Rock	7.500000
Steel	14.285714
Water	4.040404

Check if the probability of having a Legendary is equal accross generation to decide whether to keep the variable.

```r
train %>%
    select(gen_1, gen_2, gen_3, gen_4, gen_5, gen_6) %>%
    summarise(across(everything(), sum, na.rm = TRUE)) %>%
    pivot_longer(cols = everything(), names_to = "Feature", values_to = "Sum") %>%
    mutate(Percentage = (Sum / nrow(train)) * 100) %>% # Use nrow(train) here
    select(Feature, Percentage) %>%
    kable()
```

Feature	Percentage
gen_1	20.357143
gen_2	13.750000
gen_3	20.535714
gen_4	14.464286
gen_5	21.250000
gen_6	9.642857

Feature	Percentage

```r
train %>%
    select(gen_1, gen_2, gen_3, gen_4, gen_5, gen_6, Legendary) |>
    pivot_longer(cols = !Legendary, names_to = "Feature", values_to = "value") |>
    filter(value != 0) |>
    group_by(Feature) |>
    summarise("% Legendary" = 100*mean(as.numeric(Legendary) - 1)) |>
    kable()
```

Feature	% Legendary
gen_1	4.385965
gen_2	6.493506
gen_3	11.304348
gen_4	11.111111
gen_5	6.722689
gen_6	11.111111

Now we can drop Generation 1, so that it is the baseline:

```r
train <- train %>%
    select(-gen_1)
```

Since it's not equally likely to find a legendary Pokemon in each generation, with odd generations presenting more datapoints, it is important to keep it.

Numerical data

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

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

```r
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")
})
```

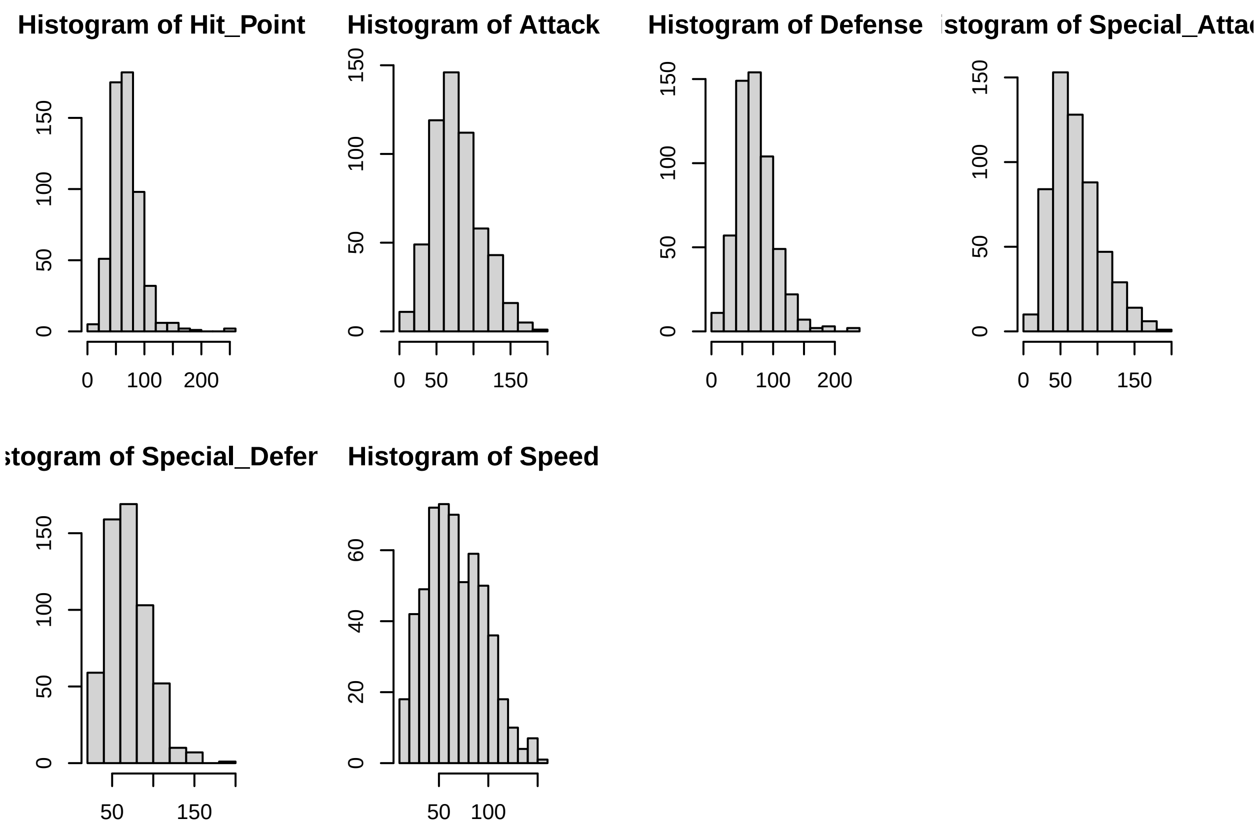
```
## [[1]]
## $breaks
```

```
##  [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"
```

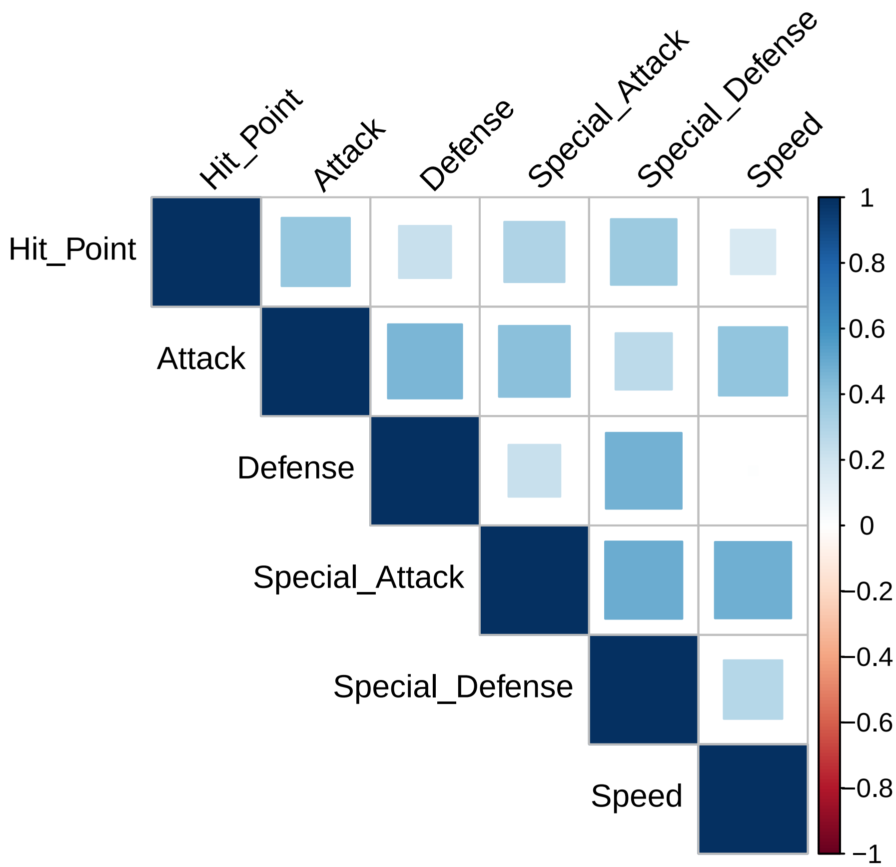
```r
par(mfrow = c(1, 1))
```




```r
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
scaling_params <- sapply(train[numerical_vars], mean)
scaling_params_sd <- sapply(train[numerical_vars], sd)

train_std <- train
train_std[numerical_vars] <- sweep(train_std[numerical_vars], 2, scaling_params, "-")
train_std[numerical_vars] <- sweep(train_std[numerical_vars], 2, scaling_params_sd, "/")

# Apply the same scaling to test data
test_std <- test
test_std[numerical_vars] <- sweep(test_std[numerical_vars], 2, scaling_params, "-")
test_std[numerical_vars] <- sweep(test_std[numerical_vars], 2, scaling_params_sd, "/")
```

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            4.330444e-17  1
```

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

```
##                        mean        sd
## Hit_Point       -0.142726299 0.8682415
```

```
## Attack          -0.029657333 0.9690458
## Defense          0.003810264 1.0290146
## Special_Attack   0.022203552 1.0254181
## Special_Defense  0.014580900 1.1861049
## Speed           -0.036205070 1.0046306
```

```
cv_seed <- list(
    c(7, 18, 21, 34, 50, 67, 71, 85, 90, 44, 62, 37, 12, 55, 28, 99, 46, 19, 38, 68, 23),
    c(9, 16, 11, 40, 51, 53, 45, 64, 39, 57, 69, 27, 72, 61, 36, 56, 75, 80, 66, 26, 32),
    c(18, 25, 48, 31, 42, 35, 63, 22, 20, 77, 24, 74, 49, 10, 16, 82, 33, 13, 14, 58, 60),
    c(39, 41, 17, 55, 59, 26, 65, 30, 79, 19, 73, 12, 27, 70, 50, 84, 76, 28, 20, 35, 37),
    c(61, 43, 33, 44, 52, 72, 31, 78, 57, 49, 22, 76, 56, 47, 35, 69, 66, 21, 62, 9, 36),
    c(24, 83, 75, 59, 32, 64, 60, 52, 25, 58, 48, 71, 40, 50, 54, 39, 53, 23, 15, 12, 20),
    c(45, 14, 37, 19, 80, 53, 28, 55, 41, 23, 51, 29, 64, 47, 67, 60, 22, 32, 49, 66, 68),
    c(63, 15, 48, 40, 26, 34, 77, 39, 61, 29, 52, 46, 69, 73, 16, 59, 79, 41, 17, 10, 54),
    c(62, 55, 77, 56, 24, 38, 81, 22, 18, 71, 48, 63, 60, 35, 45, 73, 49, 68, 32, 50, 28),
    c(84, 36, 29, 68, 16, 59, 14, 79, 25, 57, 71, 34, 53, 67, 40, 51, 15, 46, 69, 76, 33),
    99 # Last element with a single integer
)
```

```
extract_best_f <- function(fit) fit$results[which.max(fit$results$F), ]
```

```
grid_search_threshold <- function(fit) {
    best_given_threshold <- data.frame(matrix(ncol = 4, nrow = 0))
    colnames(best_given_threshold) <- c("alpha", "lambda", "prob_threshold", "F1")
    all_threhsolds <- seq(0.3, 0.5, 0.1)
    for (tr in all_threhsolds) {
        res <- thresholder(fit, tr, F, "all")
        best <- res[which.max(res$F1), ]
        best_given_threshold <- rbind(best_given_threshold, best)
    }
    return(best_given_threshold)
}
extract_best_threshold_f <- function(grid_df) {
    grid_df[which.max(grid_df$F1), ]
}
```

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

```
grid_ <- expand.grid(
    .alpha = seq(0, 1, by = 0.1),
    .lambda = 10^(-c(0, 1, 10, 100, 1000))
)

train_control_1 <- trainControl(
    method = "cv",
    number = 10,
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seeds = cv_seed
)

logistic_elnet_1 <- train(
    Legendary ~ .,
    data = train_std,
```

```
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_1,
    intercept = FALSE
)

print(logistic_elnet_1$bestTune)
```

```
##    alpha lambda
## 13   0.2  1e-10
```

```
print(extract_best_f(logistic_elnet_1))
```

```
##    alpha lambda       AUC Precision    Recall         F      AUCSD PrecisionSD
## 11   0.2      0 0.9738336 0.9581359 0.9651207 0.9613257 0.00643858  0.02575424
##      RecallSD        FSD
## 11 0.02177563 0.01621389
```

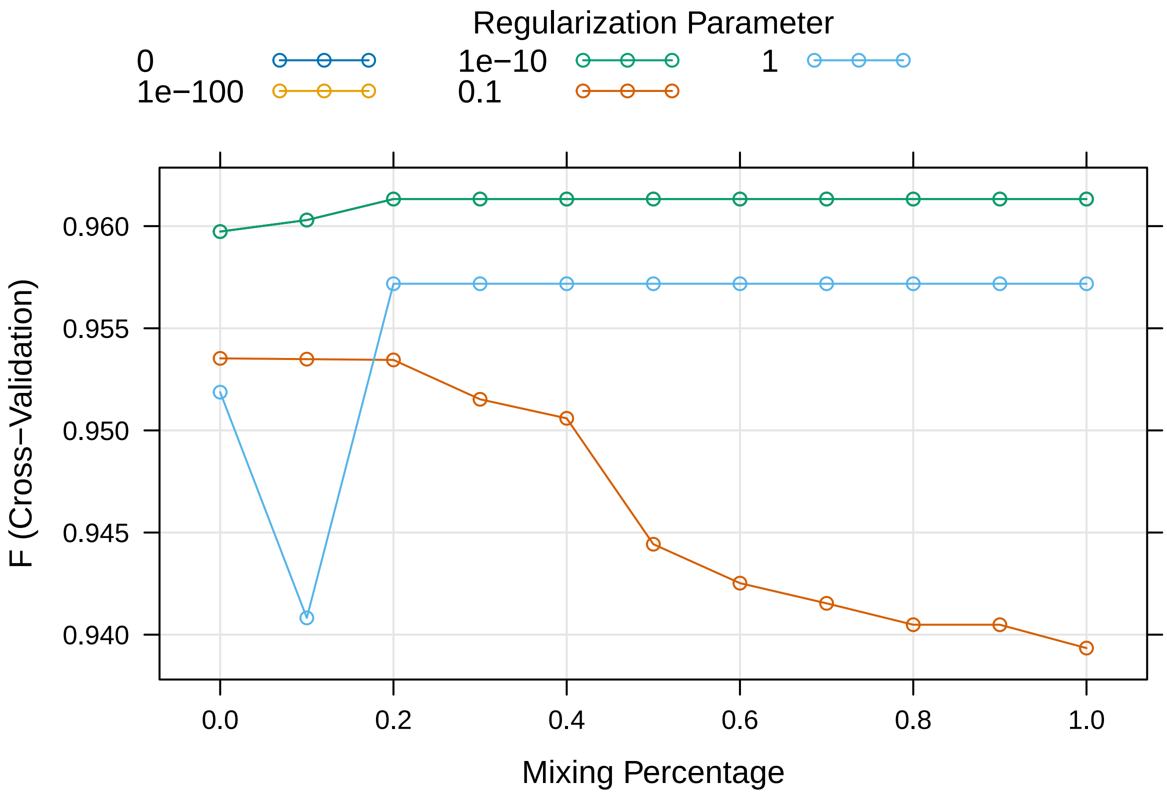
```
best_f_1 <- grid_search_threshold(logistic_elnet_1)
print(best_f_1)
```

```
##    alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 6    0.1      0            0.3   0.9748115       0.415      0.9495121
## 61   0.1      0            0.4   0.9670814       0.485      0.9545761
## 11   0.2      0            0.5   0.9651207       0.525      0.9581359
##    Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 6       0.5614286 0.9495121 0.9748115 0.9617127  0.9179107      0.8947254
## 61      0.5714286 0.9545761 0.9670814 0.9604477  0.9179107      0.8875815
## 11      0.5680952 0.9581359 0.9651207 0.9613257  0.9179107      0.8857957
##    Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 6             0.9429454         0.6949057 0.9285948 0.4221007 0.3898115
## 61            0.9304432         0.7260407 0.9268091 0.4612282 0.4520814
## 11            0.9250860         0.7450603 0.9285948 0.4877420 0.4901207
##         Dist
## 6  0.5895266
## 61 0.5204559
## 11 0.4805918
```

```
print(extract_best_threshold_f(best_f_1))
```

```
##   alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 6   0.1      0            0.3   0.9748115       0.415      0.9495121
##   Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 6      0.5614286 0.9495121 0.9748115 0.9617127  0.9179107      0.8947254
##   Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 6            0.9429454         0.6949057 0.9285948 0.4221007 0.3898115
##        Dist
## 6 0.5895266
```

```
plot(logistic_elnet_1)
```



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

```
train_control_2 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "smote",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_2 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_2,
    intercept = FALSE
)
print(logistic_elnet_2$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_2))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9179107      1 0.9571795     0 0.008648344        0
##            FSD
## 25 0.004694285
```

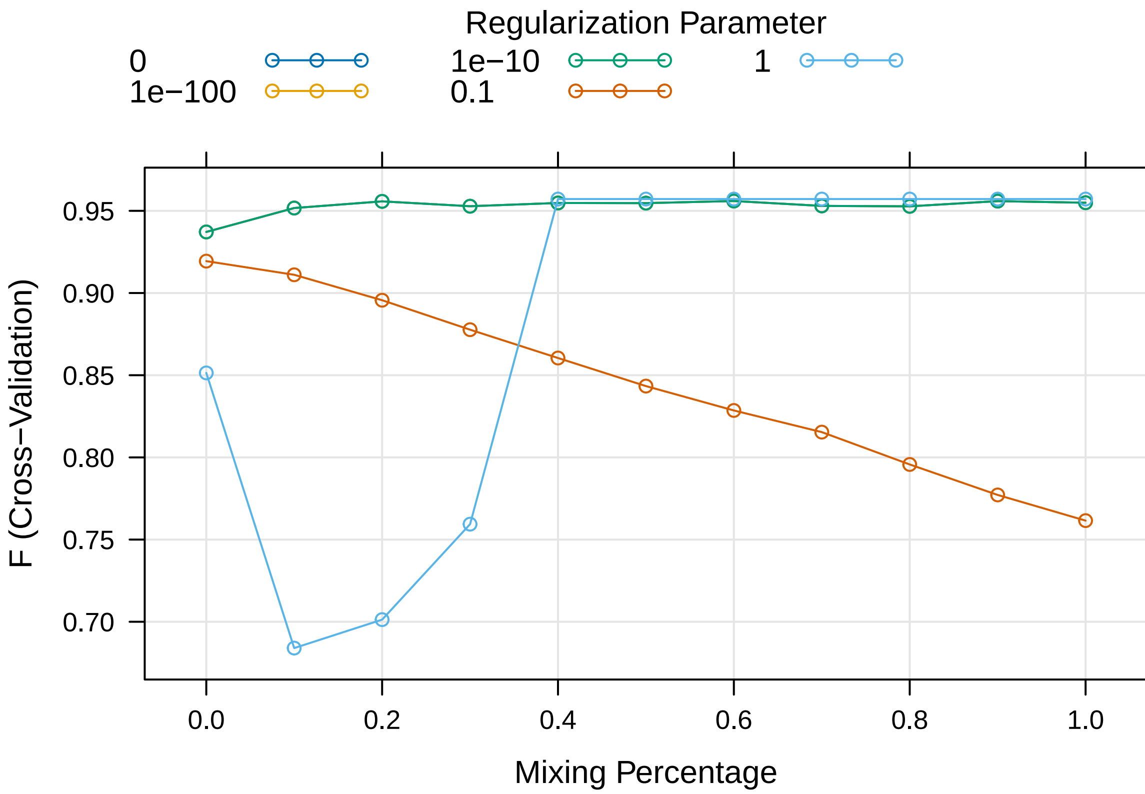
```r
best_f_2 <- grid_search_threshold(logistic_elnet_2)
print(best_f_2)
```

```
##    alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 4    0.0    0.1            0.3   0.9494344       0.820      0.9841490
## 15   0.2    1.0            0.4   1.0000000       0.200      0.9332614
## 31   0.6    0.0            0.5   0.9338235       0.775      0.9797338
##    Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 4       0.6049206 0.9841490 0.9494344 0.9662289  0.9179107      0.8714764
## 15      1.0000000 0.9332614 1.0000000 0.9653466  0.9179107      0.9179107
## 31      0.5292063 0.9797338 0.9338235 0.9559835  0.9179107      0.8571873
##    Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 4             0.8858271         0.8847172 0.9392151 0.6538622 0.7694344
## 15            0.9839912         0.6000000 0.9339195 0.2580870 0.2000000
## 31            0.8750792         0.8544118 0.9213847 0.5794151 0.7088235
##         Dist
## 4  0.2090055
## 15 0.8000000
## 31 0.2511665
```

```r
print(extract_best_threshold_f(best_f_2))
```

```
##   alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 4     0    0.1            0.3   0.9494344        0.82       0.984149
##   Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 4      0.6049206  0.984149 0.9494344 0.9662289  0.9179107      0.8714764
##   Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 4            0.8858271         0.8847172 0.9392151 0.6538622 0.7694344
##        Dist
## 4 0.2090055
```

```r
plot(logistic_elnet_2)
```



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

```
train_control_3 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "up",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_3 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_3,
    intercept = FALSE
)
print(logistic_elnet_3$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_3))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9179375      1 0.9571954     0 0.008346938        0
##           FSD
## 25 0.00453091
```

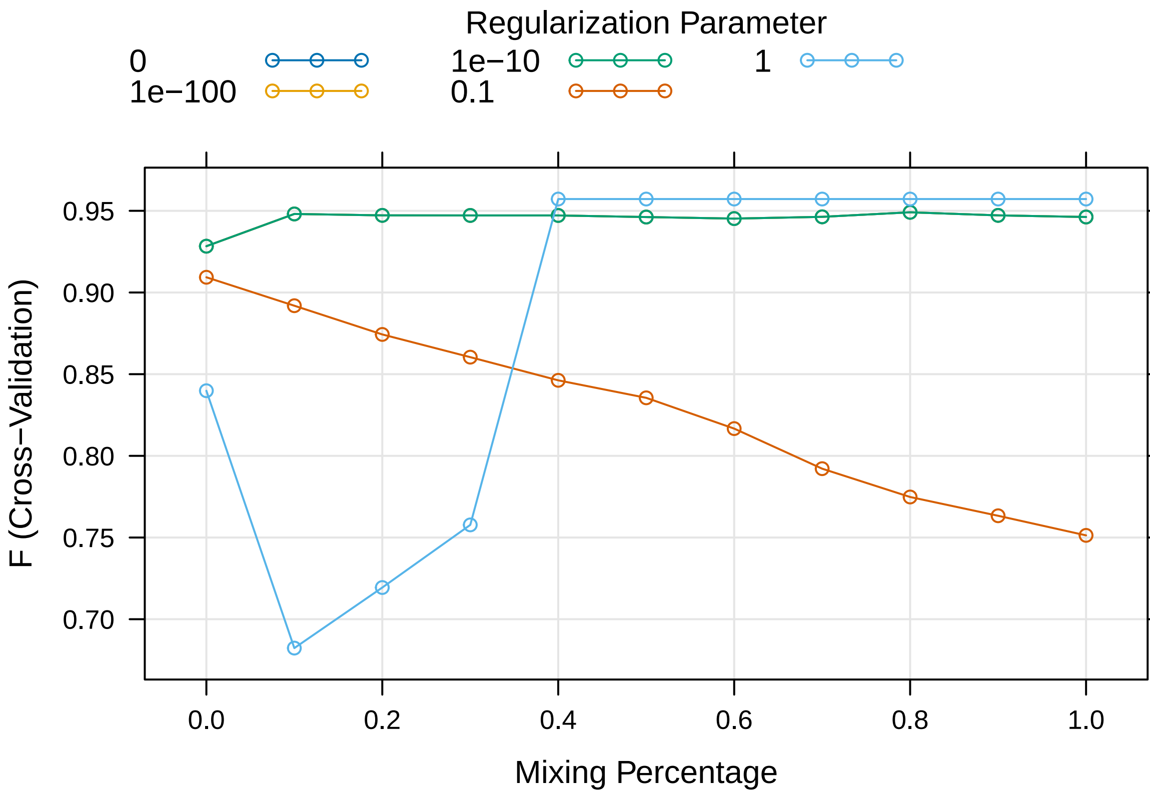
```r
best_f_3 <- grid_search_threshold(logistic_elnet_3)
print(best_f_3)
```

```
##    alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 5    0.0      1            0.3   0.9941931       0.345      0.9447972
## 10   0.1      1            0.4   0.9495098       0.735      0.9765073
## 41   0.8      0            0.5   0.9182881       0.840      0.9837622
##    Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 5       0.8981481 0.9447972 0.9941931 0.9687647  0.9179375      0.9126116
## 10      0.6153571 0.9765073 0.9495098 0.9623554  0.9179375      0.8715636
## 41      0.5237202 0.9837622 0.9182881 0.9490746  0.9179375      0.8427928
##    Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 5             0.9661620         0.6695965 0.9411238 0.4401238 0.3391931
## 10            0.8929967         0.8422549 0.9321930 0.6056858 0.6845098
## 41            0.8569845         0.8791440 0.9106636 0.5779141 0.7582881
##         Dist
## 5  0.6551158
## 10 0.2819127
## 41 0.2087669
```

```r
print(extract_best_threshold_f(best_f_3))
```

```
##   alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 5     0      1            0.3   0.9941931       0.345      0.9447972
##   Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 5      0.8981481 0.9447972 0.9941931 0.9687647  0.9179375      0.9126116
##   Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 5             0.966162         0.6695965 0.9411238 0.4401238 0.3391931
##        Dist
## 5 0.6551158
```

```r
plot(logistic_elnet_3)
```



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

```
train_control_4 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "down",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_4 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_4,
    intercept = FALSE
)
print(logistic_elnet_4$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_4))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9178839      1 0.9571637     0 0.008939504        0
##            FSD
## 25 0.004852104
```

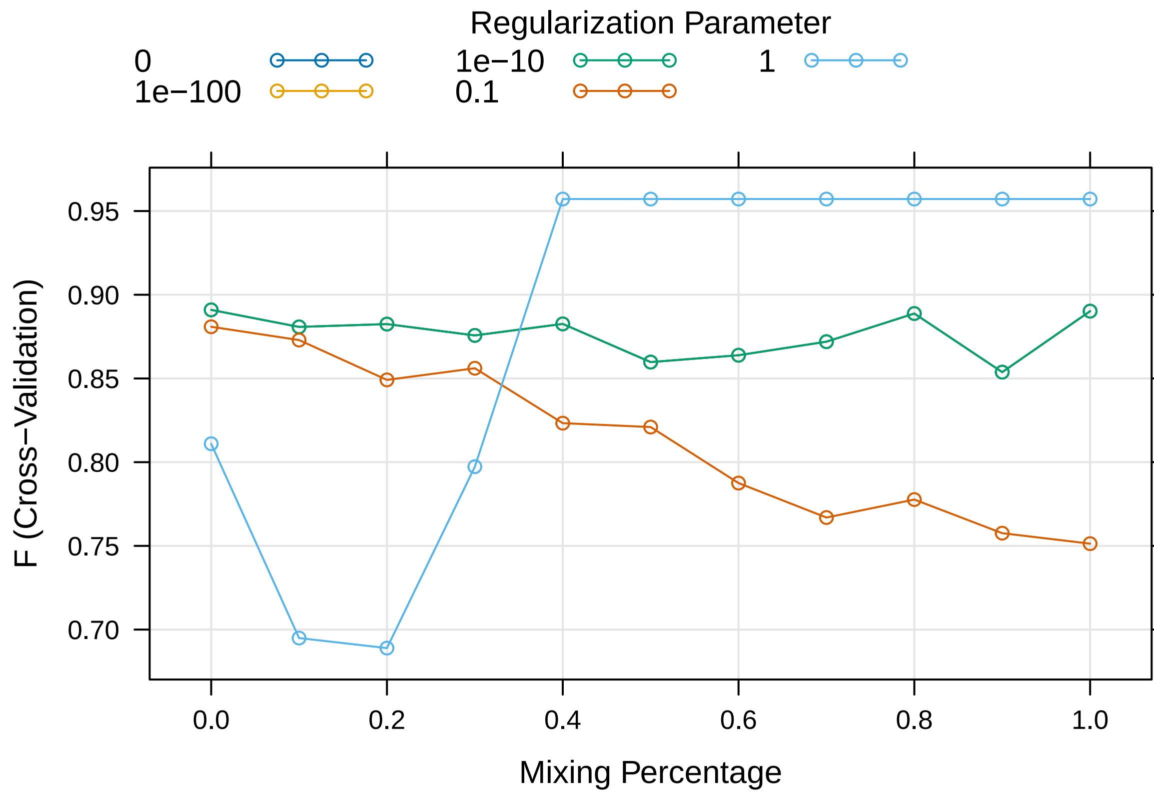
```r
best_f_4 <- grid_search_threshold(logistic_elnet_4)
print(best_f_4)
```

```
##    alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 5    0.0      1            0.3   0.9824284       0.430      0.9514584
## 10   0.1      1            0.4   0.9454374       0.855      0.9860753
## 1    0.0      0            0.5   0.8111991       0.930      0.9932806
##    Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 5       0.7533333 0.9514584 0.9824284 0.9665062  0.9178839      0.9017464
## 10      0.5978571 0.9860753 0.9454374 0.9650965  0.9178839      0.8678480
## 1       0.3267460 0.9932806 0.8111991 0.8909693  0.9178839      0.7446850
##    Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 5             0.9481146         0.7062142 0.9374943 0.4785062 0.4124284
## 10            0.8803167         0.9002187 0.9374954 0.6610394 0.8004374
## 1             0.7500422         0.8705995 0.8214440 0.4035186 0.7411991
##         Dist
## 5  0.5706289
## 10 0.1785268
## 1  0.2345798
```

```r
print(extract_best_threshold_f(best_f_4))
```

```
##   alpha lambda prob_threshold Sensitivity Specificity Pos Pred Value
## 5     0      1            0.3   0.9824284        0.43      0.9514584
##   Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate
## 5      0.7533333 0.9514584 0.9824284 0.9665062  0.9178839      0.9017464
##   Detection Prevalence Balanced Accuracy  Accuracy     Kappa         J
## 5            0.9481146         0.7062142 0.9374943 0.4785062 0.4124284
##        Dist
## 5 0.5706289
```

```r
plot(logistic_elnet_4)
```



Prediction

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

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

```
y_hat <- get_prediction(logistic_elnet_1)
confusionMatrix(y_hat, test$Legendary)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  215   9
##        Yes   6  10
##
##                Accuracy : 0.9375
##                  95% CI : (0.899, 0.9646)
##     No Information Rate : 0.9208
```

```
##     P-Value [Acc > NIR] : 0.2041
##
##                   Kappa : 0.538
##
##  Mcnemar's Test P-Value : 0.6056
##
##             Sensitivity : 0.9729
##             Specificity : 0.5263
##          Pos Pred Value : 0.9598
##          Neg Pred Value : 0.6250
##              Prevalence : 0.9208
##          Detection Rate : 0.8958
##    Detection Prevalence : 0.9333
##       Balanced Accuracy : 0.7496
##
##        'Positive' Class : No
##
```

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

```
## [[1]]
## [1] 0.9375
##
## [[2]]
## [1] 0.9208333
##
## [[3]]
## [1] 0.9208333
##
## [[4]]
## [1] 0.9208333
```

get_prediction(logistic_elnet_1)

```r
get_prediction_thresh <- function(best_f, trainControl) {
    best <- extract_best_threshold_f(best_f)
    alpha <- best$alpha
    lambda <- best$lambda
    tr <- best$prob_threshold
    grid_ <- data.frame(.alpha = alpha, .lambda = lambda)
    fit <- train(
        Legendary ~ .,
        data = train_std,
        method = "glmnet",
        family = "binomial",
        metric = "F",
        tuneGrid = grid_,
        trControl = trainControl,
        intercept = FALSE
    )
    probs <- get_prediction(fit, "prob")[, 2]
    y_hat <- as.factor(ifelse(probs > tr, "Yes", "No"))
    attr(y_hat, "threshold") <- tr
    attr(y_hat, "alpha") <- alpha
    attr(y_hat, "lambda") <- lambda
    return(y_hat)
```

```r
}
get_accuracy_thresh <- function(y_hat) {
    return(mean(test_std$Legendary == y_hat))
}
```

```r
get_prediction_thresh(best_f_3, train_control_3)
```

```
##   [1] No  Yes Yes Yes Yes No  No  No  Yes No  No  Yes Yes No  Yes Yes Yes Yes
##  [19] Yes No  No  Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
##  [37] Yes Yes Yes Yes Yes Yes No  Yes No  Yes Yes Yes Yes Yes Yes Yes Yes Yes
##  [55] Yes No  No  Yes Yes Yes Yes Yes Yes Yes Yes No  Yes Yes Yes Yes Yes Yes
##  [73] Yes Yes No  Yes Yes No  Yes Yes Yes Yes Yes Yes No  Yes Yes No  Yes No
##  [91] Yes No  Yes No  Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes No
## [109] Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
## [127] Yes Yes No  Yes Yes Yes Yes No  Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
## [145] Yes Yes No  No  Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
## [163] Yes Yes Yes Yes Yes No  Yes No  Yes Yes No  No  No  No  Yes No  Yes Yes
## [181] Yes No  Yes Yes Yes Yes Yes Yes Yes No  Yes Yes Yes Yes Yes Yes Yes Yes
## [199] Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes No  No  Yes
## [217] No  Yes Yes Yes Yes Yes Yes Yes Yes Yes No  Yes Yes Yes Yes Yes Yes
## [235] No  Yes Yes Yes Yes Yes
## attr(,"threshold")
## [1] 0.3
## attr(,"alpha")
## [1] 0
## attr(,"lambda")
## [1] 1
## Levels: No Yes
```

```r
get_prediction_thresh(best_f_1, train_control_1) |> get_accuracy_thresh()
```

```
## [1] 0.9
```

```r
get_prediction_thresh(best_f_2, train_control_2) |> get_accuracy_thresh()
```

```
## [1] 0.6875
```

```r
get_prediction_thresh(best_f_3, train_control_3) |> get_accuracy_thresh()
```

```
## [1] 0.2458333
```

```r
get_prediction_thresh(best_f_4, train_control_4) |> get_accuracy_thresh()
```

```
## [1] 0.2
```

```r
get_confusion_matrix <- function(y_hat) {
    confusionMatrix(y_hat, as.factor(test_std$Legendary))
}
get_prediction_thresh(best_f_1, train_control_1) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  205   8
##        Yes  16  11
##
##                Accuracy : 0.9
```

```
##                  95% CI : (0.8549, 0.9349)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 0.9024
##
##                   Kappa : 0.4248
##
##  Mcnemar's Test P-Value : 0.1530
##
##             Sensitivity : 0.9276
##             Specificity : 0.5789
##          Pos Pred Value : 0.9624
##          Neg Pred Value : 0.4074
##              Prevalence : 0.9208
##          Detection Rate : 0.8542
##    Detection Prevalence : 0.8875
##       Balanced Accuracy : 0.7533
##
##        'Positive' Class : No
##
```

```r
get_prediction_thresh(best_f_2, train_control_2) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  146   0
##        Yes  75  19
##
##                Accuracy : 0.6875
##                  95% CI : (0.6247, 0.7456)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2356
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6606
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.2021
##              Prevalence : 0.9208
##          Detection Rate : 0.6083
##    Detection Prevalence : 0.6083
##       Balanced Accuracy : 0.8303
##
##        'Positive' Class : No
##
```

```r
get_prediction_thresh(best_f_3, train_control_3) |> get_confusion_matrix()
```

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

```
## Prediction  No Yes
##        No   40   0
##        Yes 181  19
##
##                Accuracy : 0.2458
##                  95% CI : (0.1927, 0.3053)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0338
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.1810
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.0950
##              Prevalence : 0.9208
##          Detection Rate : 0.1667
##    Detection Prevalence : 0.1667
##       Balanced Accuracy : 0.5905
##
##        'Positive' Class : No
##
```

```r
get_prediction_thresh(best_f_4, train_control_4) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No   29   0
##        Yes 192  19
##
##                Accuracy : 0.2
##                  95% CI : (0.1513, 0.2563)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0234
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.13122
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.09005
##              Prevalence : 0.92083
##          Detection Rate : 0.12083
##    Detection Prevalence : 0.12083
##       Balanced Accuracy : 0.56561
##
##        'Positive' Class : No
##
```

```r
f1_elnet <- data.frame()
for (fit in list(logistic_elnet_1, logistic_elnet_2, logistic_elnet_3, logistic_elnet_4)) {
    if (nrow(f1_elnet) == 0) {
        f1_elnet <- extract_best_f(fit)
    } else {
        f1_elnet <- rbind(f1_elnet, extract_best_f(fit))
    }
}
rownames(f1_elnet) <- c("Default-Case", "SMOTE", "Up-sampling", "Down-sampling")
f1_elnet[1:6] |>
    select(-AUC) |>
    kable()
```

	alpha	lambda	Precision	Recall	F
Default-Case	0.2	0	0.9581359	0.9651207	0.9613257
SMOTE	0.4	1	0.9179107	1.0000000	0.9571795
Up-sampling	0.4	1	0.9179375	1.0000000	0.9571954
Down-sampling	0.4	1	0.9178839	1.0000000	0.9571637

```r
f1_elnet_thresh <- data.frame()
for (best in list(best_f_1, best_f_2, best_f_3, best_f_4)) {
    ext <- extract_best_threshold_f(best)
    if (nrow(f1_elnet_thresh) == 0) {
        f1_elnet_thresh <- ext
    } else {
        f1_elnet_thresh <- rbind(f1_elnet_thresh, ext)
    }
}
rownames(f1_elnet_thresh) <- c("Default-Case", "SMOTE", "Up-sampling", "Down-sampling")
f1_elnet_thresh |>
    mutate(F = F1) |>
    select(alpha, lambda, prob_threshold, Precision, Recall, F) |>
    kable()
```

	alpha	lambda	prob_threshold	Precision	Recall	F
Default-Case	0.1	0.0	0.3	0.9495121	0.9748115	0.9617127
SMOTE	0.0	0.1	0.3	0.9841490	0.9494344	0.9662289
Up-sampling	0.0	1.0	0.3	0.9447972	0.9941931	0.9687647
Down-sampling	0.0	1.0	0.3	0.9514584	0.9824284	0.9665062

— add description here

Feature extraction

Import the df and visualize the columns' name.

```r
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)
}
```

One-hot encoding from Type.1 and Type.2.

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

for (typ in unique_types) {
    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)) {
    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.

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

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

```
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]
```

Categorical data

We start by checking the frequency of Characteristics across our Pokemon population in the training sample:

```r
train %>%
    select(-numerical_vars, -Legendary, -gen_1, -gen_2, -gen_3, -gen_4, -gen_5, -gen_6) %>%
    summarise(across(everything(), sum, na.rm = TRUE)) %>%
    pivot_longer(cols = everything(), names_to = "Feature", values_to = "Total_Sum") %>%
    mutate(Percentage = (Total_Sum / sum(Total_Sum)) * 100) %>%
    select(Feature, Percentage) %>%
    kable()
```

Feature	Percentage
Grass	7.420495
Fire	5.418139
Water	11.660777
Bug	6.007067
Normal	7.891637
Poison	4.711425
Electric	4.240283
Ground	6.007067
Fairy	3.415783
Fighting	4.358068
Psychic	7.302709
Rock	4.711425
Ghost	3.651355
Ice	3.062426
Dragon	3.769140
Dark	3.886926
Steel	4.122497
Flying	8.362780

Check if the probability of having a Legendary is equal accross generation to decide whether to keep the variable.

```r
train %>%
    select(gen_1, gen_2, gen_3, gen_4, gen_5, gen_6) %>%
    summarise(across(everything(), sum, na.rm = TRUE)) %>%
    pivot_longer(cols = everything(), names_to = "Feature", values_to = "Sum") %>%
    mutate(Percentage = (Sum / nrow(train)) * 100) %>% # Use nrow(train) here
    select(Feature, Percentage) %>%
    kable()
```

Feature	Percentage
gen_1	20.357143
gen_2	13.750000
gen_3	20.535714

Feature	Percentage
gen_4	14.464286
gen_5	21.250000
gen_6	9.642857

Now we can drop Generation 1, so that it is the baseline:

```
train <- train %>%
    select(-gen_1)
```

Since it's not equally likely to find a legendary Pokemon in each generation, with odd generations presenting more datapoints, it is important to keep it.

Numerical data

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

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

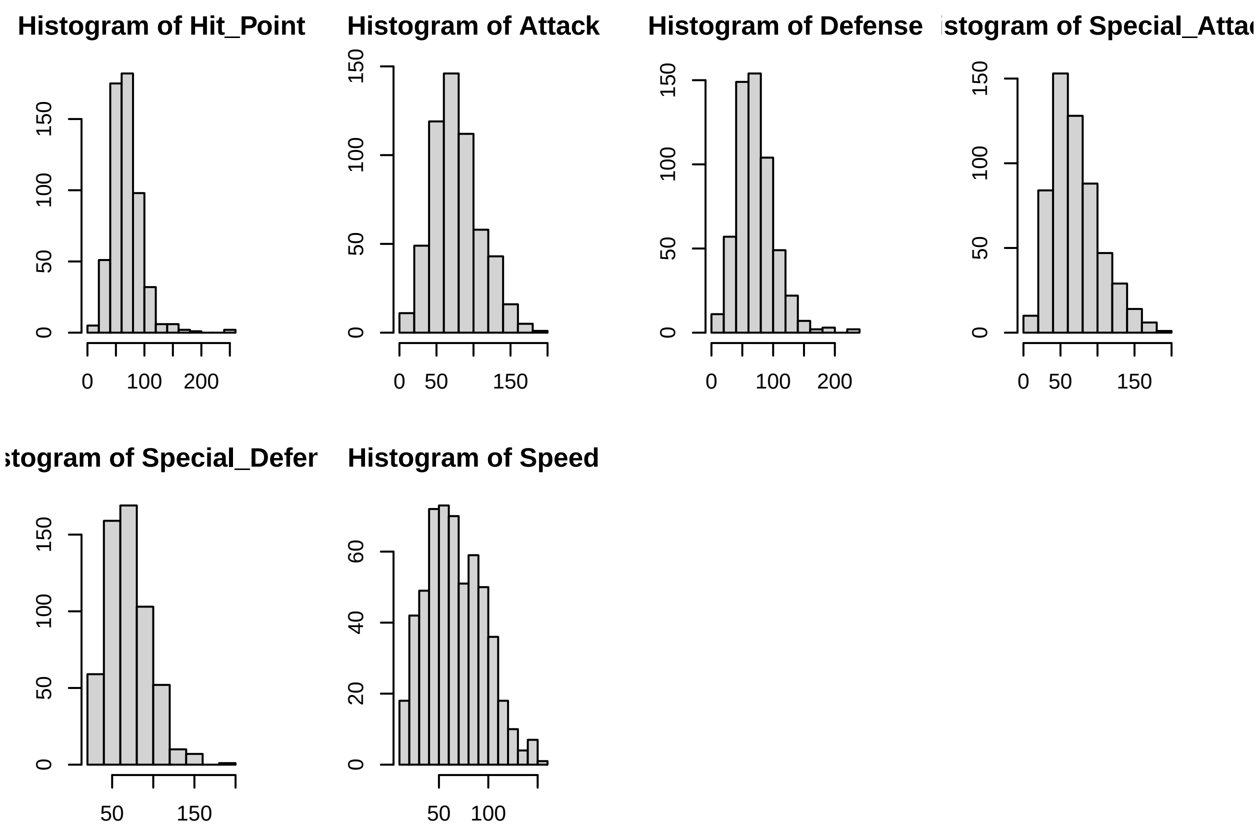
```
## [[1]]
## $breaks
##  [1]   0  20  40  60  80 100 120 140 160 180 200 220 240 260
##
## $counts
##  [1]   5  51 175 182  98  32   6   6   2   1   0   0   2
##
## $density
##  [1] 4.464286e-04 4.553571e-03 1.562500e-02 1.625000e-02 8.750000e-03
##  [6] 2.857143e-03 5.357143e-04 5.357143e-04 1.785714e-04 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))
```

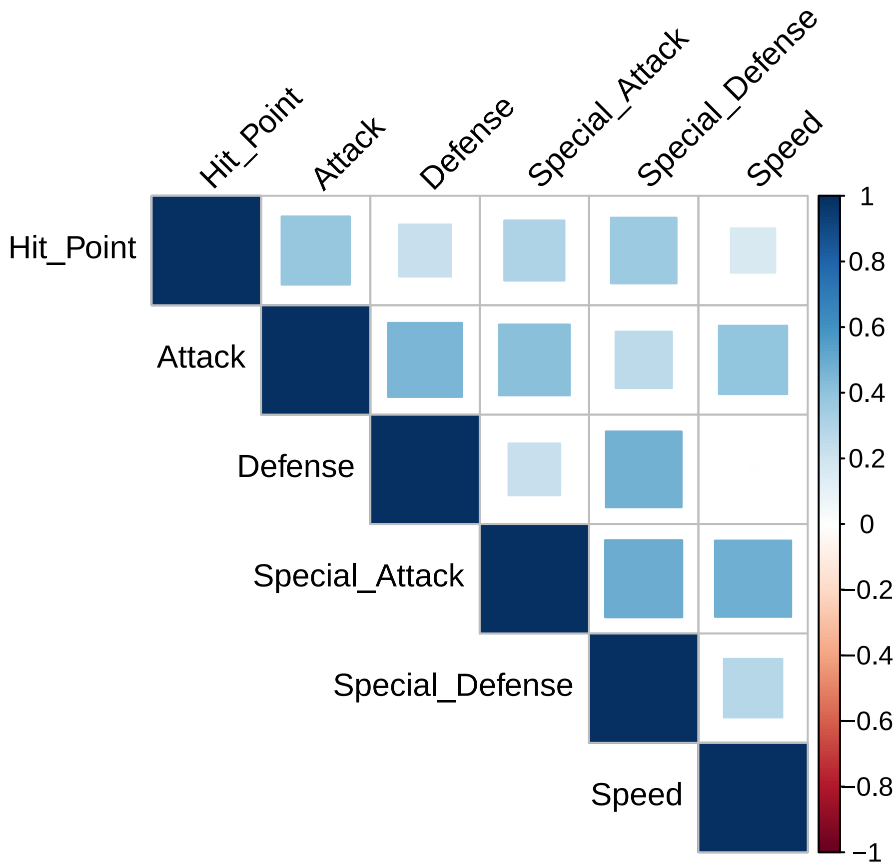



```r
cor_mat <- cor(train[numerical_vars])
print(cor_mat)
```

```
##                 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
scaling_params <- sapply(train[numerical_vars], mean)
scaling_params_sd <- sapply(train[numerical_vars], sd)

train_std <- train
train_std[numerical_vars] <- sweep(train_std[numerical_vars], 2, scaling_params, "-")
train_std[numerical_vars] <- sweep(train_std[numerical_vars], 2, scaling_params_sd, "/")

# Apply the same scaling to test data
test_std <- test
test_std[numerical_vars] <- sweep(test_std[numerical_vars], 2, scaling_params, "-")
test_std[numerical_vars] <- sweep(test_std[numerical_vars], 2, scaling_params_sd, "/")
```

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            4.330444e-17  1
```

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

```
##                         mean        sd
## Hit_Point       -0.142726299 0.8682415
## Attack          -0.029657333 0.9690458
## Defense          0.003810264 1.0290146
## Special_Attack   0.022203552 1.0254181
## Special_Defense  0.014580900 1.1861049
## Speed           -0.036205070 1.0046306
```

```r
cv_seed <- list(
    c(7, 18, 21, 34, 50, 67, 71, 85, 90, 44, 62, 37, 12, 55, 28, 99, 46, 19, 38, 68, 23),
    c(9, 16, 11, 40, 51, 53, 45, 64, 39, 57, 69, 27, 72, 61, 36, 56, 75, 80, 66, 26, 32),
    c(18, 25, 48, 31, 42, 35, 63, 22, 20, 77, 24, 74, 49, 10, 16, 82, 33, 13, 14, 58, 60),
    c(39, 41, 17, 55, 59, 26, 65, 30, 79, 19, 73, 12, 27, 70, 50, 84, 76, 28, 20, 35, 37),
    c(61, 43, 33, 44, 52, 72, 31, 78, 57, 49, 22, 76, 56, 47, 35, 69, 66, 21, 62, 9, 36),
    c(24, 83, 75, 59, 32, 64, 60, 52, 25, 58, 48, 71, 40, 50, 54, 39, 53, 23, 15, 12, 20),
    c(45, 14, 37, 19, 80, 53, 28, 55, 41, 23, 51, 29, 64, 47, 67, 60, 22, 32, 49, 66, 68),
    c(63, 15, 48, 40, 26, 34, 77, 39, 61, 29, 52, 46, 69, 73, 16, 59, 79, 41, 17, 10, 54),
    c(62, 55, 77, 56, 24, 38, 81, 22, 18, 71, 48, 63, 60, 35, 45, 73, 49, 68, 32, 50, 28),
    c(84, 36, 29, 68, 16, 59, 14, 79, 25, 57, 71, 34, 53, 67, 40, 51, 15, 46, 69, 76, 33),
    99 # Last element with a single integer
)
```

```r
extract_best_f <- function(fit) fit$results[which.max(fit$results$F), ]
```

```r
grid_search_threshold <- function(fit) {
    best_given_threshold <- data.frame(matrix(ncol = 4, nrow = 0))
    colnames(best_given_threshold) <- c("alpha", "lambda", "prob_threshold", "F1")
    all_threhsolds <- seq(0.3, 0.8, 0.1)
    for (tr in all_threhsolds) {
        res <- thresholder(fit, tr, F, "F1")
        best <- res[which.max(res$F1), ]
        best_given_threshold <- rbind(best_given_threshold, best)
    }
    return(best_given_threshold)
}
extract_best_threshold_f <- function(grid_df) {
    grid_df[which.max(grid_df$F1), ]
}
```

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

```r
grid_ <- expand.grid(
    .alpha = seq(0, 1, by = 0.1),
    .lambda = 10^(-c(0, 1, 10, 100, 1000))
)

train_control_1 <- trainControl(
    method = "cv",
```

```r
    number = 10,
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seeds = cv_seed
)

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

print(logistic_elnet_1$bestTune)
```

```
##    alpha lambda
## 13   0.2  1e-10
```

```r
print(extract_best_f(logistic_elnet_1))
```

```
##    alpha lambda       AUC Precision    Recall         F      AUCSD PrecisionSD
## 11   0.2      0 0.9738336 0.9581359 0.9651207 0.9613257 0.00643858  0.02575424
##      RecallSD        FSD
## 11 0.02177563 0.01621389
```

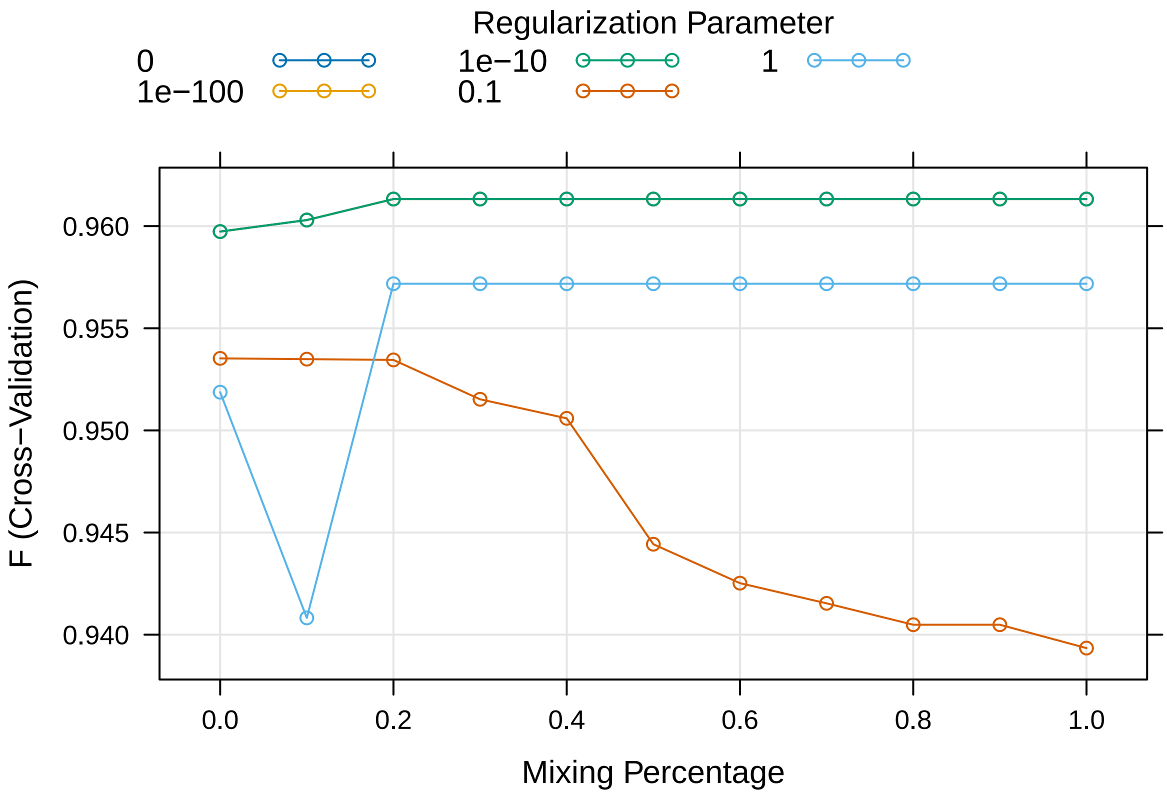
```r
best_f_1 <- grid_search_threshold(logistic_elnet_1)
print(best_f_1)
```

```
##     alpha lambda prob_threshold        F1
## 6     0.1      0            0.3 0.9617127
## 61    0.1      0            0.4 0.9604477
## 11    0.2      0            0.5 0.9613257
## 1     0.0      0            0.6 0.9639693
## 111   0.2      0            0.7 0.9536440
## 112   0.2      0            0.8 0.9480063
```

```r
print(extract_best_threshold_f(best_f_1))
```

```
##   alpha lambda prob_threshold        F1
## 1     0      0            0.6 0.9639693
```

```r
plot(logistic_elnet_1)
```



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

```
train_control_2 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "smote",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_2 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_2,
    intercept = FALSE
)
print(logistic_elnet_2$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_2))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9179107      1 0.9571795     0 0.008648344        0
##            FSD
## 25 0.004694285
```

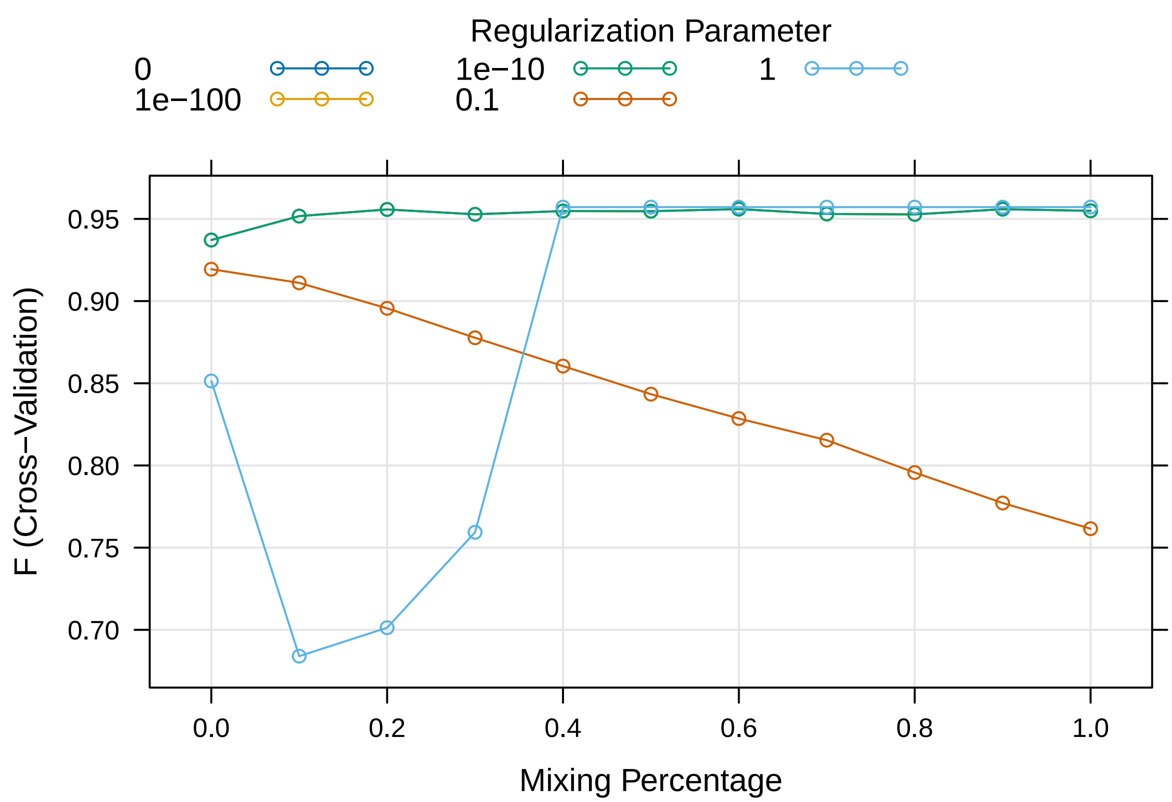
```r
best_f_2 <- grid_search_threshold(logistic_elnet_2)
print(best_f_2)
```

```
##    alpha lambda prob_threshold        F1
## 4    0.0    0.1            0.3 0.9662289
## 15   0.2    1.0            0.4 0.9653466
## 31   0.6    0.0            0.5 0.9559835
## 41   0.8    0.0            0.6 0.9514466
## 36   0.7    0.0            0.7 0.9472411
## 51   1.0    0.0            0.8 0.9350926
```

```r
print(extract_best_threshold_f(best_f_2))
```

```
##   alpha lambda prob_threshold        F1
## 4     0    0.1            0.3 0.9662289
```

```r
plot(logistic_elnet_2)
```



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

```r
train_control_3 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "up",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_3 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_3,
    intercept = FALSE
)
print(logistic_elnet_3$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_3))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9179375      1 0.9571954     0 0.008346938        0
##           FSD
## 25 0.00453091
```

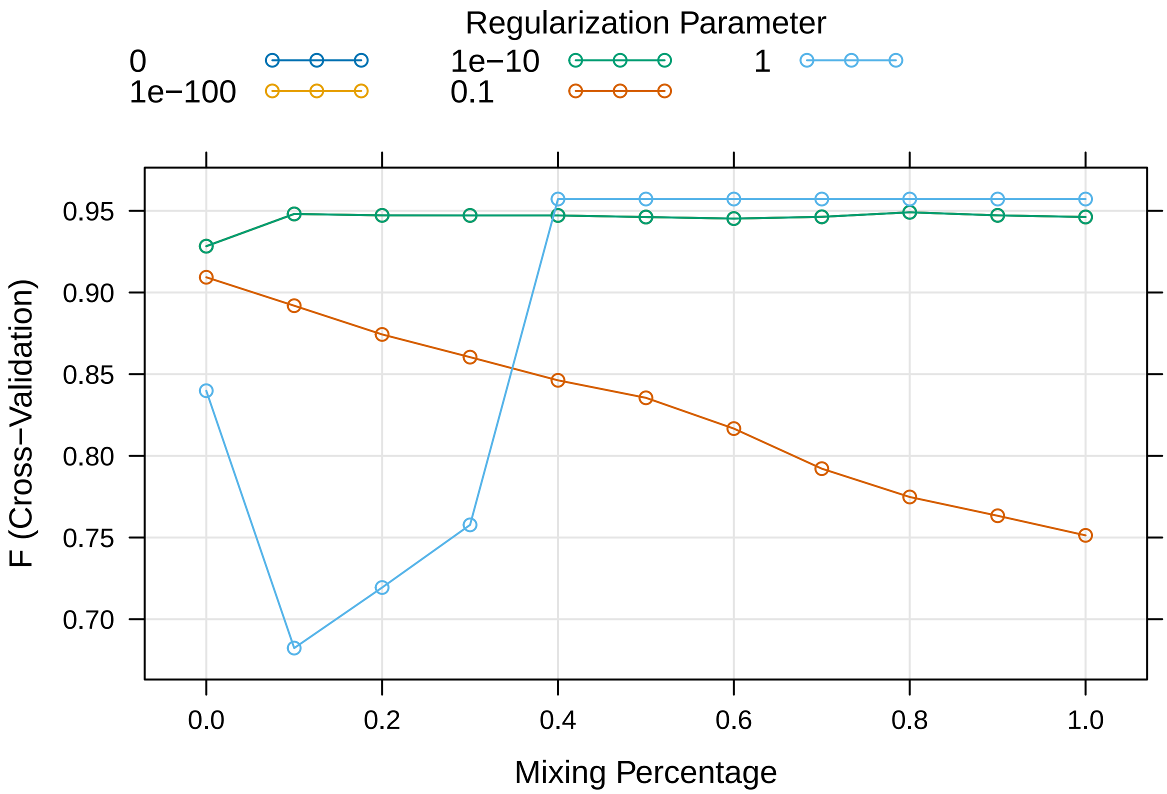
```r
best_f_3 <- grid_search_threshold(logistic_elnet_3)
print(best_f_3)
```

```
##    alpha lambda prob_threshold        F1
## 5    0.0      1            0.3 0.9687647
## 10   0.1      1            0.4 0.9623554
## 41   0.8      0            0.5 0.9490746
## 11   0.2      0            0.6 0.9480656
## 26   0.5      0            0.7 0.9460062
## 36   0.7      0            0.8 0.9370288
```

```r
print(extract_best_threshold_f(best_f_3))
```

```
##   alpha lambda prob_threshold        F1
## 5     0      1            0.3 0.9687647
```

```r
plot(logistic_elnet_3)
```



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

```
train_control_4 <- trainControl(
    method = "cv",
    number = 10,
    sampling = "down",
    classProbs = TRUE,
    summaryFunction = prSummary,
    savePredictions = "all",
    seed = cv_seed
)

logistic_elnet_4 <- train(
    Legendary ~ .,
    data = train_std,
    method = "glmnet",
    family = "binomial",
    metric = "F",
    tuneGrid = grid_,
    trControl = train_control_4,
    intercept = FALSE
)
print(logistic_elnet_4$bestTune)
```

```
##    alpha lambda
## 25   0.4      1
```

```r
print(extract_best_f(logistic_elnet_4))
```

```
##    alpha lambda AUC Precision Recall         F AUCSD PrecisionSD RecallSD
## 25   0.4      1   0 0.9178839      1 0.9571637     0 0.008939504        0
##            FSD
## 25 0.004852104
```

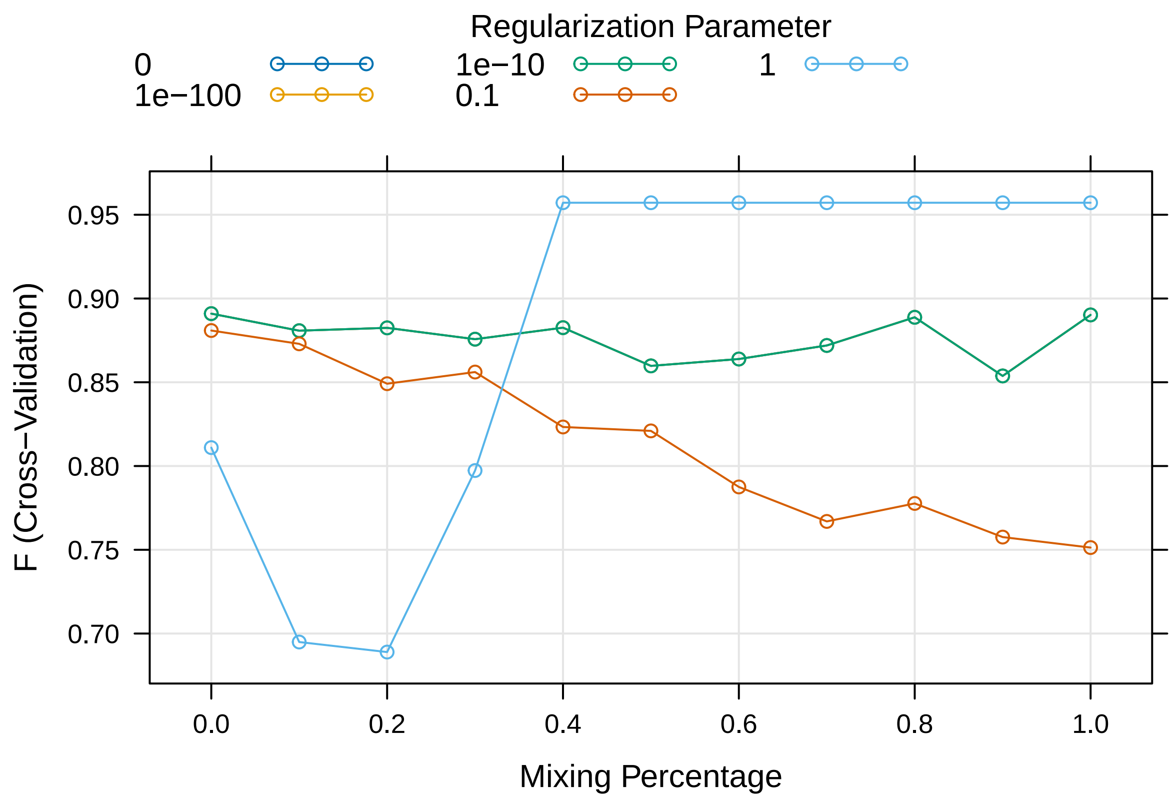
```r
best_f_4 <- grid_search_threshold(logistic_elnet_4)
print(best_f_4)
```

```
##     alpha lambda prob_threshold        F1
## 5     0.0      1            0.3 0.9665062
## 10    0.1      1            0.4 0.9650965
## 1     0.0      0            0.5 0.8909693
## 51    1.0      0            0.6 0.8899894
## 511   1.0      0            0.7 0.8851691
## 512   1.0      0            0.8 0.8803295
```

```r
print(extract_best_threshold_f(best_f_4))
```

```
##   alpha lambda prob_threshold        F1
## 5     0      1            0.3 0.9665062
```

```r
plot(logistic_elnet_4)
```



Prediction

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

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] 0.9375
##
## [[2]]
## [1] 0.9208333
##
## [[3]]
## [1] 0.9208333
##
## [[4]]
## [1] 0.9208333
```

```r
get_prediction_thresh <- function(best_f, trainControl) {
    best <- extract_best_threshold_f(best_f)
    alpha <- best$alpha
    lambda <- best$lambda
    tr <- best$prob_threshold
    grid_ <- data.frame(.alpha = alpha, .lambda = lambda)
    fit <- train(
        Legendary ~ .,
        data = train_std,
        method = "glmnet",
        family = "binomial",
        metric = "F",
        tuneGrid = grid_,
        trControl = trainControl,
        intercept = FALSE
    )
    probs <- get_prediction(fit, "prob")[, 2]
    y_hat <- as.factor(ifelse(probs > tr, "Yes", "No"))
    attr(y_hat, "threshold") <- tr
    attr(y_hat, "alpha") <- alpha
    attr(y_hat, "lambda") <- lambda
    return(y_hat)
}
get_accuracy_thresh <- function(y_hat) {
    return(mean(test_std$Legendary == y_hat))
}
```

```r
get_prediction_thresh(best_f_1, train_control_1) |> get_accuracy_thresh()
```

```
## [1] 0.925
```

```r
get_prediction_thresh(best_f_2, train_control_2) |> get_accuracy_thresh()
```

```
## [1] 0.6875
```

```r
get_prediction_thresh(best_f_3, train_control_3) |> get_accuracy_thresh()
```

```
## [1] 0.2458333
```

```r
get_prediction_thresh(best_f_4, train_control_4) |> get_accuracy_thresh()
```

```
## [1] 0.2
```

```r
get_confusion_matrix <- function(y_hat) {
    confusionMatrix(y_hat, as.factor(test_std$Legendary))
}
get_prediction_thresh(best_f_1, train_control_1) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  219  16
##        Yes   2   3
##
##                Accuracy : 0.925
##                  95% CI : (0.8841, 0.9549)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 0.465702
##
##                   Kappa : 0.2244
##
##  Mcnemar's Test P-Value : 0.002183
##
##             Sensitivity : 0.9910
##             Specificity : 0.1579
##          Pos Pred Value : 0.9319
##          Neg Pred Value : 0.6000
##              Prevalence : 0.9208
##          Detection Rate : 0.9125
##    Detection Prevalence : 0.9792
##       Balanced Accuracy : 0.5744
##
##        'Positive' Class : No
##
```

```r
get_prediction_thresh(best_f_2, train_control_2) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  146   0
##        Yes  75  19
```

```
##
##                Accuracy : 0.6875
##                  95% CI : (0.6247, 0.7456)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2356
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6606
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.2021
##              Prevalence : 0.9208
##          Detection Rate : 0.6083
##    Detection Prevalence : 0.6083
##       Balanced Accuracy : 0.8303
##
##        'Positive' Class : No
##
```

```
get_prediction_thresh(best_f_3, train_control_3) |> get_confusion_matrix()
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No   40   0
##        Yes 181  19
##
##                Accuracy : 0.2458
##                  95% CI : (0.1927, 0.3053)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0338
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.1810
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.0950
##              Prevalence : 0.9208
##          Detection Rate : 0.1667
##    Detection Prevalence : 0.1667
##       Balanced Accuracy : 0.5905
##
##        'Positive' Class : No
##
```

```
get_prediction_thresh(best_f_4, train_control_4) |> get_confusion_matrix()
```

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

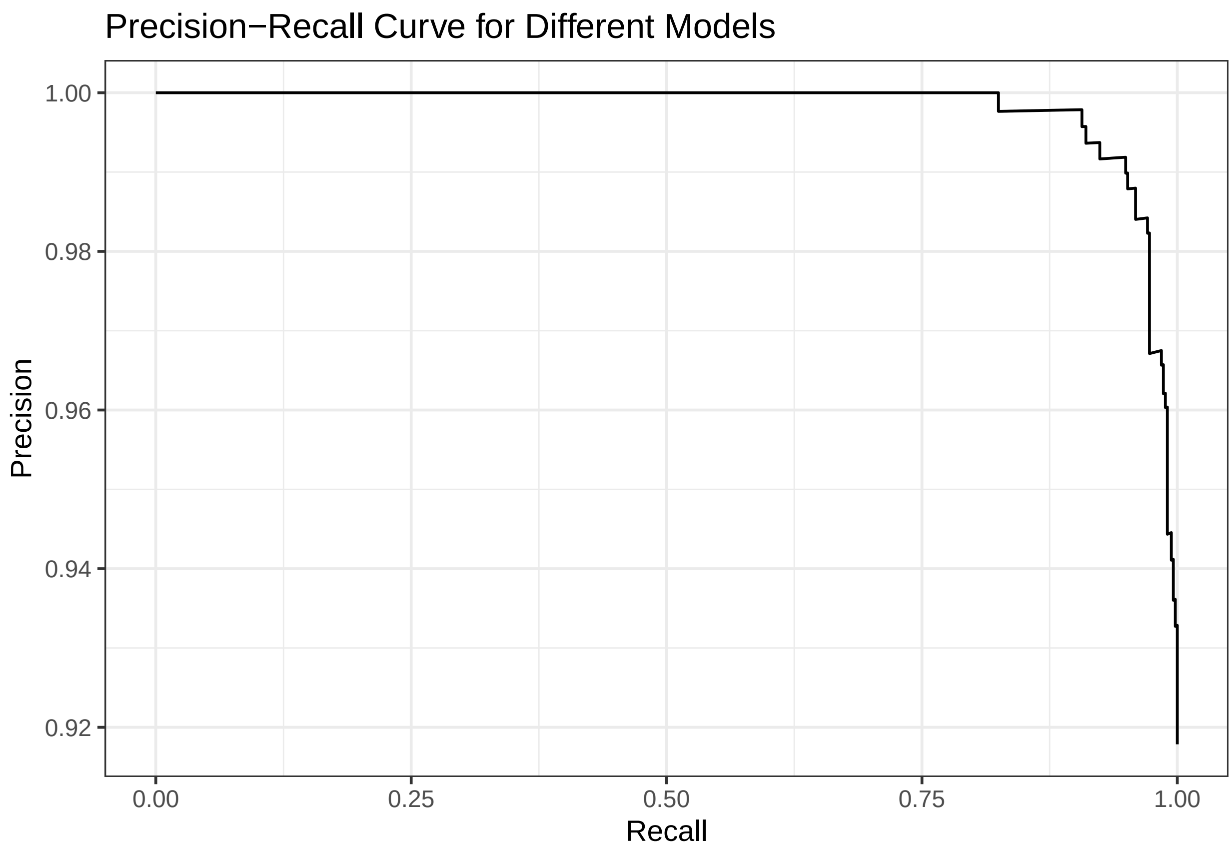
```
##
##           Reference
## Prediction  No Yes
##        No   29   0
##        Yes 192  19
##
##                Accuracy : 0.2
##                  95% CI : (0.1513, 0.2563)
##     No Information Rate : 0.9208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0234
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.13122
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.09005
##              Prevalence : 0.92083
##          Detection Rate : 0.12083
##    Detection Prevalence : 0.12083
##       Balanced Accuracy : 0.56561
##
##        'Positive' Class : No
##
```

```r
# install.packages("PRROC")
# library(PRROC)

# plot_precision_recall <- function(true_labels, pred_model1, pred_model2, #red_model3) {
#    pr_model1 <- pr.curve(scores.class0 = logistic_elnet_1, weights.class0 = test$Legendary, curve = T
#    pr_model2 <- pr.curve(scores.class0 = pred_model2, weights.class0 = rue_labels, curve = TRUE)
#    pr_model3 <- pr.curve(scores.class0 = pred_model3, weights.class0 = rue_labels, curve = TRUE)

#    plot(pr_model1, main = "Precision-Recall Curve Comparison", col = #1f77b4", lwd = 2)
#    lines(pr_model2, col = "darkableue", lwd = 2)
#    lines(pr_model3, col = "#66b3ff", lwd = 2)

#    legend("bottomright",
#       legend = c("Model 1", "Model 2", "Model 3"),
#        col = c("#1f77b4", "darkableue", "#66b3ff"), lwd = 2
#    )
# }

# plot_precision_recall(test$Legendary, logistic_elnet_1, xgb_model2, rf_model4)
```

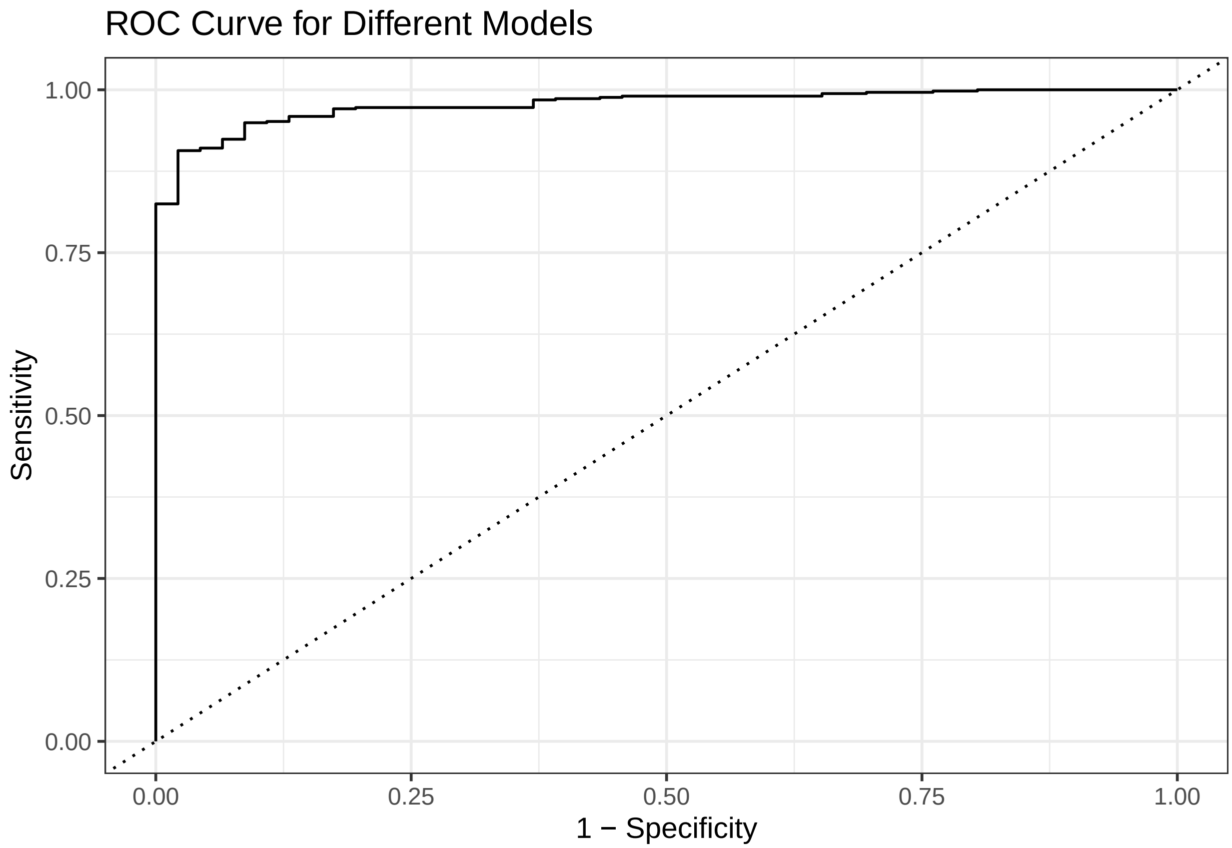
###test

```r
library(yardstick)
bind_cols(train_std, predict(logistic_elnet_1, type = "prob")) %>%
    pr_curve(truth = Legendary, No) |>
    ggplot(aes(x = recall, y = precision)) +
    geom_path() +
    labs(
```

```
        title = "Precision-Recall Curve for Different Models",
        x = "Recall", y = "Precision"
    ) +
    theme_bw()
```

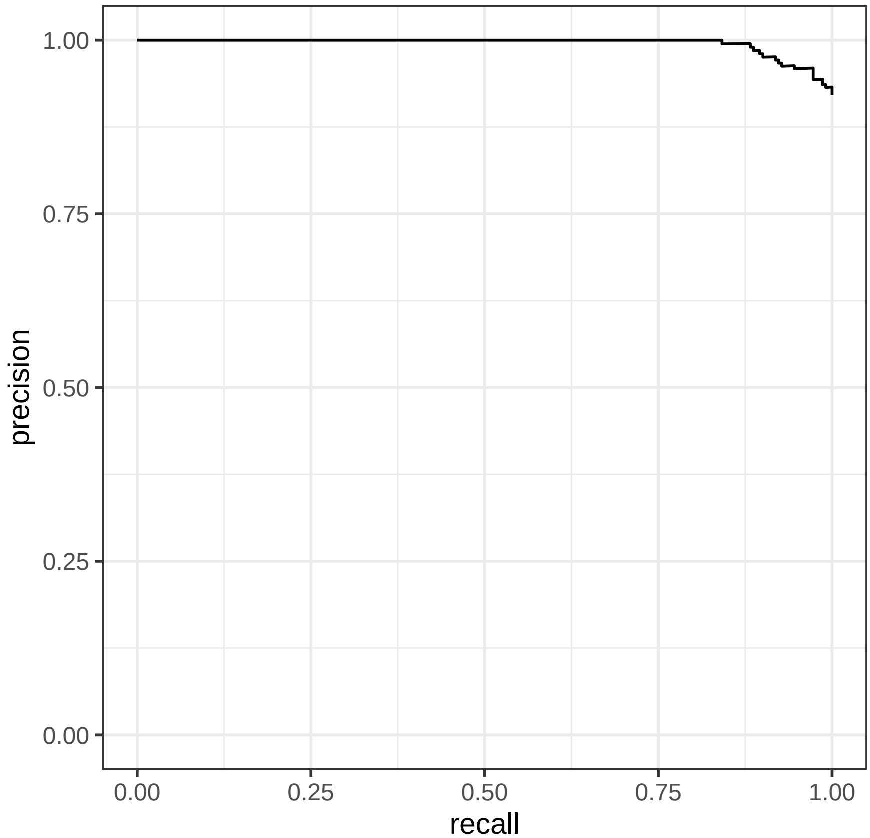



```
bind_cols(train_std, predict(logistic_elnet_1, type = "prob")) |>
    roc_curve(truth = Legendary, No) |>
    ggplot(aes(1 - specificity, y = sensitivity)) +
    labs(
        title = "ROC Curve for Different Models",
        x = "1 - Specificity", y = "Sensitivity"
    ) +
    geom_path() +
    geom_abline(lty = 3) +
    theme_bw()
```



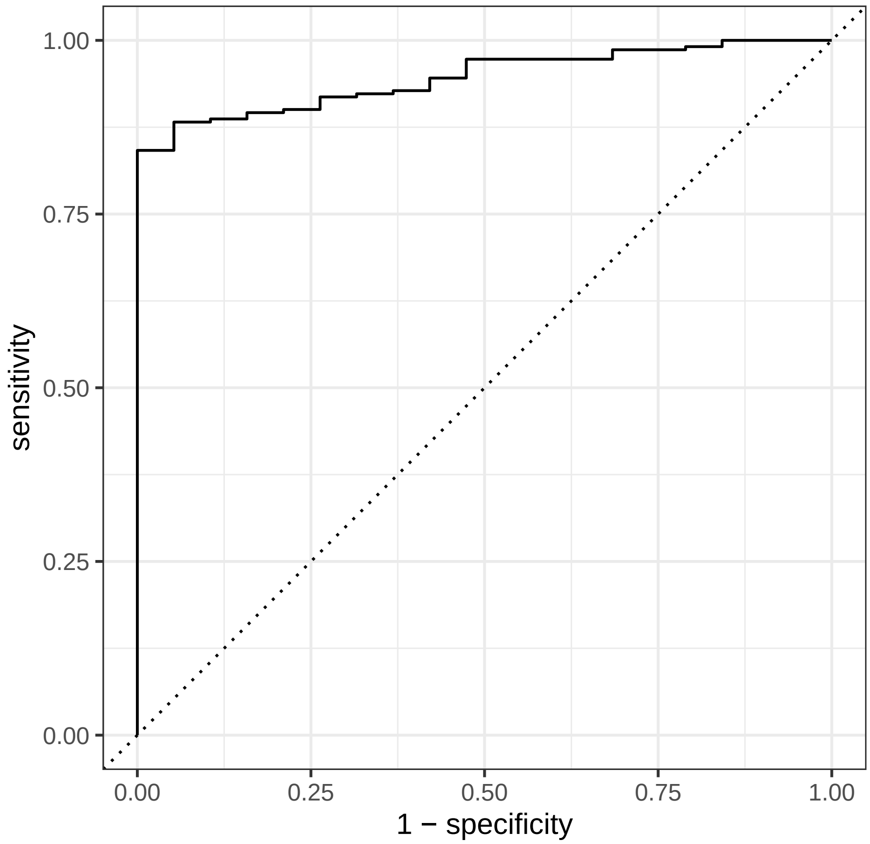
###train

```
library(yardstick)
bind_cols(test_std, predict(logistic_elnet_1, newdata = test_std |> select(-Legendary), type = "prob"))
    pr_curve(truth = Legendary, No) |>
    autoplot()
```




```r
# ggplot(aes(x = recall, y = precision)) +
# geom_path() +
# labs(title = "Precision-Recall Curve for Different Models",
#   x = "Recall", y = "Precision") +
# theme_bw()
```

```r
bind_cols(test_std, predict(logistic_elnet_1, newdata = test_std |> select(-Legendary), type = "prob"))
    roc_curve(truth = Legendary, No) |>
    autoplot()
```




```
# ggplot(aes(1 - specificity, y = sensitivity)) +
# labs(title = "ROC Curve for Different Models",
#   x = "1 - Specificity", y = "Sensitivity") +
# geom_path() +
# geom_abline(lty = 3) +
# theme_bw()
```