STATS 790 Assignment 4

Yiran Zhang
400119421

04 April, 2023

Contents

Question 1

The raisin dataset is collected and published by Ilkay Cinar, Murat Koklu and Sakir Tasdemir on UCI repository in 2021. The dataset captures the morphological features of two types raisin from Turkey: Besni and Kecimen. The features include the area of raisin in pixels, perimeter length, etc. The raisin dataset has 900 observations and 8 variables, with 1 target variable `Class`, denoting the raisin type, and the remaining 7 variables being potential predictors.

```
# Import library
library(readxl) # read xlsx file
library(DataExplorer) # correlation plot
library(reshape2) # data reshape for ggplot
library(tidymodels) # model fitting
library(vip) # variable importance plot

# Import dataset
raisin <- read_xlsx("Raisin_Dataset/Raisin_Dataset.xlsx") %>% # read dataset
  as.data.frame() %>% # change to data.frame type
  mutate(across(Class, factor)) %>% # change the target variable to factor
  mutate(across(where(is.numeric), scale)) # scale the numeric variables
```

There are 900 observations and 8 variables. The target variable `Class` describes the type of raisin and therefore is categorical, it has two levels: Besni and Kecimen. The remaining 7 predicting variables are all continuous and describe the numeric measurement of each raisin. There are no missing values observed from this dataset. The dataset is evenly distributed with 50% Besni raisin and 50% Kecimen raisin.

We also observe from the correlation plot (Figure 1) and the boxplots (Figure 2) that the variable `Extent` has relative poor performance on distinguishing two types of raisin, it seems to have no correlation with the raisin types. Moreover, all other predictors seem to have strong correlation between each other, and they can also help identify raisin types.

```r
dim(raisin) # 900 observations and 8 variables (7 predictors and 1 target)
```

```
## [1] 900   8
```

```r
# str(raisin) # data type of each variable, hidden because its output is too long
colSums(is.na(raisin)) # there are no missing values
```

```
##                                        Class
##     0     0     0     0     0     0     0     0
```

```r
# Proportion of each type of raisin
raisin %>%
  group_by(Class) %>%
  summarise(n=n(), prop=n()/nrow(.))
```

```
## # A tibble: 2 x 3
##   Class       n  prop
##   <fct>   <int> <dbl>
## 1 Besni     450   0.5
## 2 Kecimen   450   0.5
```

```r
# Correlation plot
raisin %>%
  plot_correlation()
```

```r
# Boxplots on all numeric variables
raisin %>%
  melt(id='Class') %>% # change to long format for plotting
  ggplot(aes(x = variable, y = value, color = Class)) +
  geom_boxplot() + # boxplots
  ggtitle("Boxplot of each numeric predictor") + # add a title
  scale_x_discrete(guide = guide_axis(angle = 45))
```

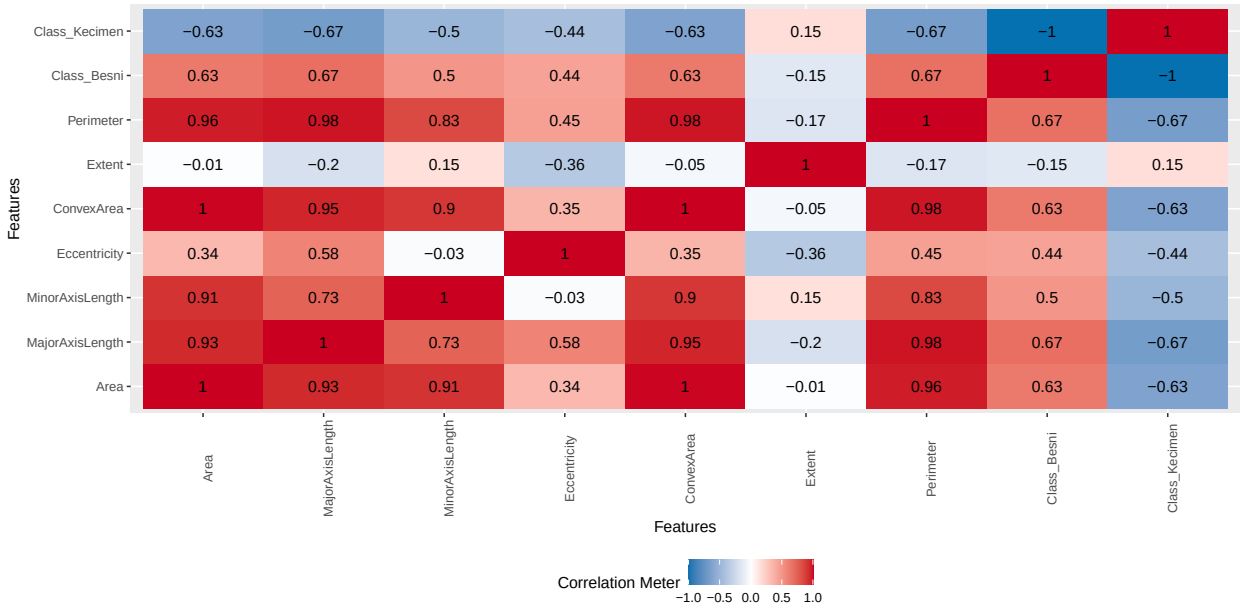

Figure 1: Correlation plot of raisin dataset

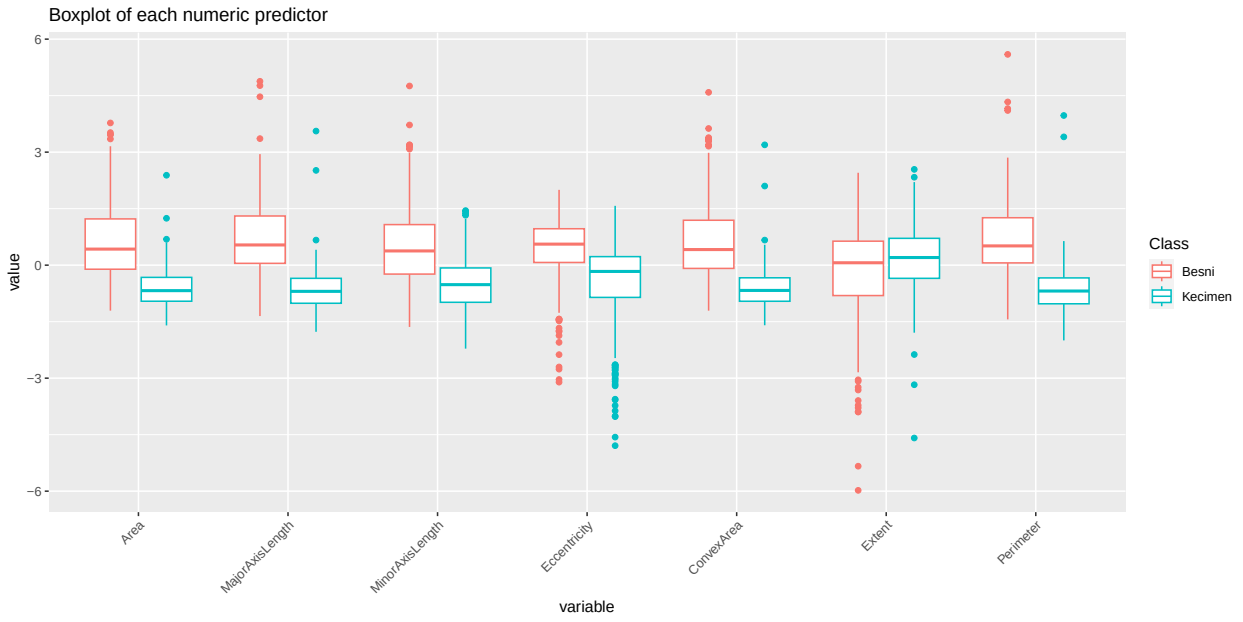

Figure 2: Boxplot of raisin dataset

We use `initial_split` function to split the dataset into 66% training set and 33% testing set. The numeric variables in the original raisin dataset all have different unit, therefore they are scaled in the previous step for a better comparison. From the conclusion above, we will remove `Extent` variable as it has no correlation with the target variable. The codes are learned from Lecture code (Bolker, 2023).

```
# Create train and test set
set.seed(120)
data_split <- initial_split(raisin, prop = 2/3, strata = Class)
train_raisin <- training(data_split) # 600 training data
test_raisin <- testing(data_split) # 300 testing data


# Recipe
raisin_recipe <- recipe(Class ~ ., train_raisin) %>%
  # remove variables with zero variances
  step_nzv(all_predictors(), freq_cut = 0, unique_cut = 0) %>%
  # remove Extent variable
  step_rm(Extent)
```

The target variable `Class` is categorical with two levels, therefore the classification tree is the most appropriate tree-based model to fit this dataset. We also include a 10-fold cross-validation to avoid overfitting.

```
# 10-fold cross validation
raisin_cv <- vfold_cv(train_raisin, v = 10, strata = Class)


# Specify a decision tree model
raisin_tree <- decision_tree() %>% # use decision tree model
  set_mode("classification") %>% # classification tree
  set_engine("rpart") # use rpart package
```

We will tune the following parameters:cost complexity, tree depth (number of splits) and minimum value of n (minimum number of observations in each node). We select a range from 3 to 10 as

possible `tree_depth` values because a depth less that 3 is too little to split the dataset, and more than 6 might be too complicated and may overfit the data. Similarly, a range from 5 to 10 is selected as the minimum number of observations for a node, as a number that is too large will oversimplify the tree model, and a too small number will overfit.

```
# Tune parameters
raisin_tree <- raisin_tree %>%
    set_args(cost_complexity = tune(),
             tree_depth = tune(),
             min_n = tune()) # add tuning

# Define workflow
raisin_flow <- workflow() %>%
  add_recipe(raisin_recipe) %>%
  add_model(raisin_tree)

# Tune the variables and set a grid of values to test out
raisin_grid <- grid_regular(cost_complexity(range = c(-5, -1)),
                            tree_depth(range = c(3, 6)),
                            min_n(range = c(5, 10)), levels = 4)
tree_rs <- tune_grid(
  raisin_flow,
  Class ~ .,
  resamples = raisin_cv,
  grid = raisin_grid,
  metrics = metric_set(accuracy))
```

We select the best model by the `select_best()` function, and the best model turns out to have 1e-10 cost complexity, 3 splits for the tree, and minimum 20 observations in each node. See below for the details of the split in this tree model.

```r
# Determine the best model
best_model <- select_best(tree_rs)
best_model
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1         0.00464          3     5 Preprocessor1_Model03
```

```r
# Update the workflow with the best model
raisin_tree_best <- finalize_workflow(raisin_flow, best_model)

# Fit the best model
raisin_fit <- fit(raisin_tree_best, data = train_raisin)
raisin_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_nzv()
## * step_rm()
##
## -- Model -----------------------------------------------------------------------
## n= 600
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 600 300 Besni (0.50000000 0.50000000)
##    2) MajorAxisLength>=-0.05671383 269  27 Besni (0.89962825 0.10037175)
##      4) Perimeter>=0.185745 207   8 Besni (0.96135266 0.03864734) *
##      5) Perimeter< 0.185745 62  19 Besni (0.69354839 0.30645161)
##       10) Area< -0.0007852851 45   8 Besni (0.82222222 0.17777778) *
##       11) Area>=-0.0007852851 17   6 Kecimen (0.35294118 0.64705882) *
##    3) MajorAxisLength< -0.05671383 331  58 Kecimen (0.17522659 0.82477341)
##      6) Perimeter>=-0.1610971 35  16 Besni (0.54285714 0.45714286)
##       12) MajorAxisLength< -0.1903866 15   4 Besni (0.73333333 0.26666667) *
##       13) MajorAxisLength>=-0.1903866 20   8 Kecimen (0.40000000 0.60000000) *
##      7) Perimeter< -0.1610971 296  39 Kecimen (0.13175676 0.86824324)
##       14) Eccentricity>=0.9909086 6   2 Besni (0.66666667 0.33333333) *
##       15) Eccentricity< 0.9909086 290  35 Kecimen (0.12068966 0.87931034) *
```

We apply this model to the test set and build a confusion matrix to check the number of observations that are mis-classified. We can see that there are 25 Besni raisin that are mis-classified as Kecimen, and 15 Kecimen raisin are mis-classified as Besni. The accuracy is about 86.7%, in other words, the mis-classification rate is about 13.3%, which is a fairly good rate.

A detailed plot for the decision tree model is shown in Figure 3, and the variable importance plot from Figure 4 indicates that the top three most important predictors are `MajorAxisLength`, `Perimeter` and `ConvexArea`.

```
# Confusion matrix
augment(raisin_fit, new_data = test_raisin) %>%
  conf_mat(truth = Class, estimate = .pred_class)
```

```
##           Truth
## Prediction Besni Kecimen
##    Besni     125      15
##    Kecimen    25     135
```

```
# Mis-classification rate = 1 - accuracy
augment(raisin_fit, new_data = test_raisin) %>%
  accuracy(truth = Class, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.867
```

```
# Tree details
raisin_fit %>%
  extract_fit_engine() %>%
  rattle::fancyRpartPlot()
```

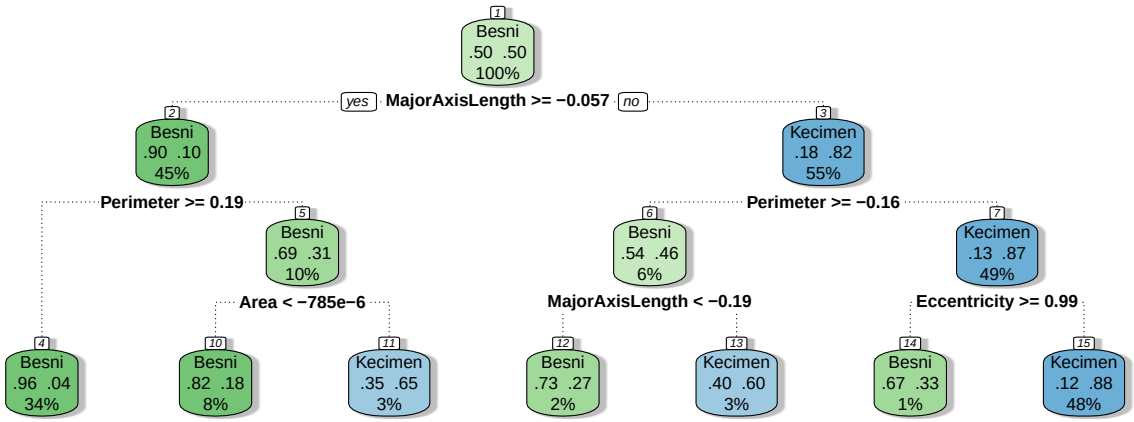

Figure 3: Details of the tree model

```
# Variable importance
raisin_fit %>%
  extract_fit_parsnip() %>%
  vip()
```

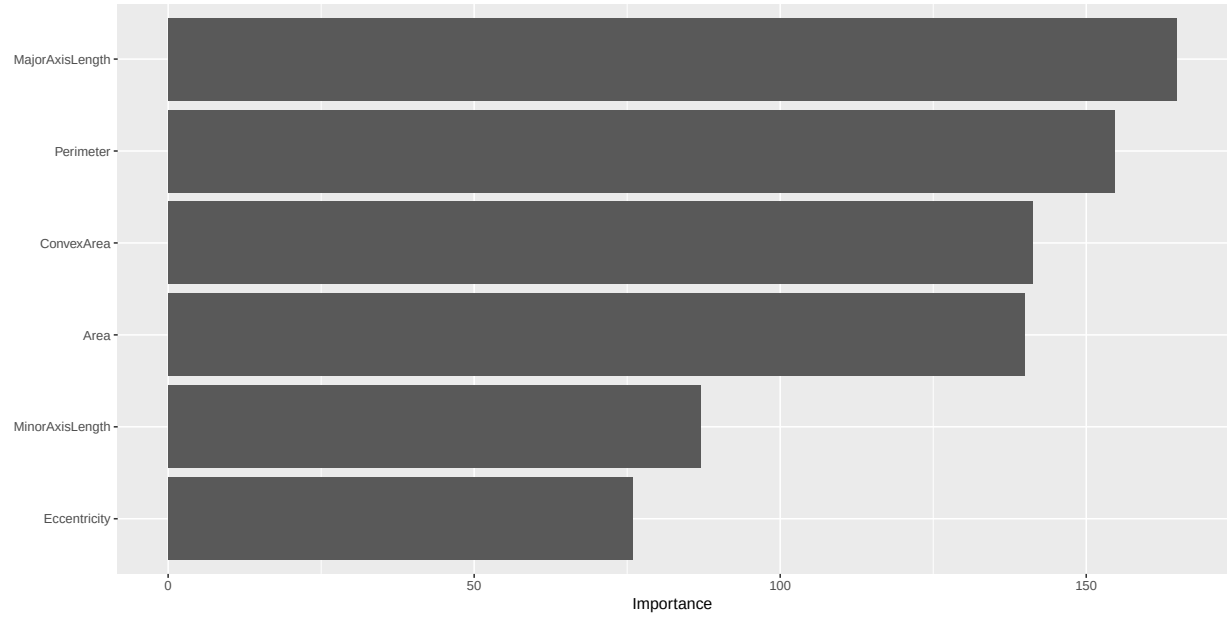

Figure 4: The importance of each predictors

Question 2

(a)

MSE (L2) Loss Function The formula for a L2 loss function is $L = \frac{1}{2}(y-\hat{y})^2$, substitute it into algorithm 10.3 step 2(c), we get the weight $\gamma_{jm} = \text{argmin}_\gamma \sum_{i=1}^n \frac{1}{2}(y_i - \hat{y}_i)^2 = \text{argmin}_\gamma \sum_{i=1}^n \frac{1}{2}(y_i - (f_{m-1}(x_i)+\gamma))^2$. In words, the weights γ_{jm} is the value γ that minimized the function $\sum_{i=1}^n \frac{1}{2}(y_i - (f_{m-1}(x_i)+\gamma))^2$.

To find the minimum value of the above formula, we take the first derivative and set it to 0 and solve for γ.

$$\begin{aligned}
\frac{\partial}{\partial\gamma}\sum_{i=1}^n \frac{1}{2}(y_i - (f_{m-1}(x_i)+\gamma))^2 &= \frac{\partial}{\partial\gamma}\sum_{i=1}^n \frac{1}{2}(y_i - f_{m-1}(x_i) - \gamma)^2 \\
&= -\sum_{i=1}^n (y_i - f_{m-1}(x_i) - \gamma) \\
&= 0
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sum_{i=1}^{n}(y_i - f_{m-1}(x_i) - \gamma) &= 0 \\
\sum_{i=1}^{n}(y_i - f_{m-1}(x_i)) - \sum_{i=1}^{n}\gamma &= 0 \\
\sum_{i=1}^{n}(y_i - f_{m-1}(x_i)) - n\gamma &= 0 \\
\sum_{i=1}^{n}(y_i - f_{m-1}(x_i)) &= n\gamma \\
\gamma &= \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{m-1}(x_i))
\end{aligned}$$

The above calculation shows that when the loss function is L2 loss function, the weights $\gamma = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{m-1}(x_i))$, which is the mean of residuals.

Binomial Deviance Loss Function According to ESL (Hastie et al, 2009) page 346, the equivalent binomial log-likelihood loss function is $l(y, p(x)) = ylog(p) + (1-y)log(1-p)$, substitute it into step 2(c) from algorithm 10.3 and get $\gamma_{jm} = \text{argmin}_{\gamma}\sum_{i=1}^{n} y_i log(f_{m-1}(x_i)+\gamma) + (1-y_i)log(1-(f_{m-1}(x_i)+\gamma))$, similarly, we take the first derivative with respect to γ and set it to 0.

$$\begin{aligned}
&\frac{\partial}{\partial\gamma}\sum_{i=1}^{n} y_i log(f_{m-1}(x_i)+\gamma) + (1-y_i)log(1-(f_{m-1}(x_i)+\gamma)) \\
&= \sum_{i=1}^{n}\frac{y_i}{f_{m-1}(x_i)+\gamma} + \frac{-1}{1-f_{m-1}(x_i)-\gamma} + \frac{y_i}{1-f_{m-1}(x_i)-\gamma} \\
&= \sum_{i=1}^{n}\frac{y_i}{f_{m-1}(x_i)+\gamma} + \frac{y_i-1}{1-f_{m-1}(x_i)-\gamma} \\
&= \sum_{i=1}^{n}\frac{y_i(1-f_{m-1}(x_i)-\gamma) + (y_i-1)(f_{m-1}(x_i)+\gamma)}{(f_{m-1}(x_i)+\gamma)(1-f_{m-1}(x_i)-\gamma)} \\
&= \sum_{i=1}^{n}\frac{y_i - y_i f_{m-1}(x_i) - y_i\gamma + y_i f_{m-1}(x_i) + y_i\gamma - f_{m-1}(x_i) - \gamma}{(f_{m-1}(x_i)+\gamma)(1-f_{m-1}(x_i)-\gamma)} \\
&= \sum_{i=1}^{n}\frac{y_i - f_{m-1}(x_i) - \gamma}{(f_{m-1}(x_i)+\gamma)(1-f_{m-1}(x_i)-\gamma)} \\
&= 0
\end{aligned}$$

Therefore, $\sum_{i=1}^{n} y_i - f_{m-1}(x_i) - \gamma = 0$, rearrange and get $\sum_{i=1}^{n} y_i - f_{m-1}(x_i) - \sum_{i=1}^{n}\gamma = 0$, $\sum_{i=1}^{n} y_i - f_{m-1}(x_i) = \sum_{i=1}^{n}\gamma = n\gamma$, and finally we get $\gamma = \frac{1}{n}\sum_{i=1}^{n} y_i - f_{m-1}(x_i)$.

(b)

MSE (L2) Loss Function According to the notes from Justin Domke (2010), the Newton Boosting is using the second order Taylor Expansion of the loss function to approximate its value. That is, $L(y_i, f_{m-1}(x_i) + \gamma) \approx L(y_i, f_{m-1}(x_i)) + \gamma g(y_i, f_{m-1}(x_i)) + \frac{1}{2}\gamma^2 h(y_i, f_{m-1}(x_i))$, where $g(y_i, f_{m-1}(x_i)) = \frac{\partial}{\partial f_{m-1}(x_i)} L(y_i, f_{m-1}(x_i))$ (first partial derivative), and $h(y_i, f_{m-1}(x_i)) = \frac{\partial^2}{\partial f^2_{m-1}(x_i)} L(y_i, f_{m-1}(x_i))$ (second partial derivative).

In the MSE setting,

$$\begin{aligned} g(y_i, f_{m-1}(x_i)) &= \frac{\partial}{\partial f_{m-1}(x_i)} \frac{1}{2}(y_i - f_{m-1}(x_i))^2 \\ &= -(y_i - f_{m-1}(x_i)) \end{aligned}$$

$$\begin{aligned} h(y_i, f_{m-1}(x_i)) &= \frac{\partial^2}{\partial f^2_{m-1}(x_i)} \frac{1}{2}(y_i - f_{m-1}(x_i))^2 \\ &= \frac{\partial}{\partial f_{m-1}(x_i)} [-(y_i - f_{m-1}(x_i))] \\ &= 1 \end{aligned}$$

Substituting them in to the formula above and we get $L(y_i, f_{m-1}(x_i) + \gamma) \approx \frac{1}{2}(y_i - f_{m-1}(x_i))^2 - (y_i - f_{m-1}(x_i))\gamma + \frac{1}{2}\gamma^2$, therefore $\gamma_{jm} = \text{argmin}_\gamma \sum_{i=1}^n \frac{1}{2}(y_i - f_{m-1}(x_i))^2 - (y_i - f_{m-1}(x_i))\gamma + \frac{1}{2}\gamma^2$. Take the first derivative and set to 0 to solve for γ.

$$\begin{aligned} &\frac{\partial}{\partial \gamma} \sum_{i=1}^n \frac{1}{2}(y_i - f_{m-1}(x_i))^2 - (y_i - f_{m-1}(x_i))\gamma + \frac{1}{2}\gamma^2 \\ &= \sum_{i=1}^n -(y_i - f_{m-1}(x_i)) + \gamma \\ &= \sum_{i=1}^n -(y_i - f_{m-1}(x_i)) + \sum_{i=1}^n \gamma \\ &= 0 \end{aligned}$$

Therefore, we get $\sum_{i=1}^n (y_i - f_{m-1}(x_i)) = \sum_{i=1}^n \gamma = n\gamma$, rearrange and get $\gamma = \frac{1}{n}\sum_{i=1}^n (y_i - f_{m-1}(x_i))$.

Binomial Deviance Loss Function γ is found by taking the first derivative of $\gamma_{jm} = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i - f_{m-1}(x_i)) + \gamma g(y_i, f_{m-1}(x_i)) + \frac{1}{2}\gamma^2 h(y_i, f_{m-1}(x_i))$ with respect to γ:

$$\begin{aligned}
\frac{\partial}{\partial \gamma} \sum_{i=1}^{n} L(y_i - f_{m-1}(x_i)) + \gamma g(y_i, f_{m-1}(x_i)) + \frac{1}{2}\gamma^2 h(y_i, f_{m-1}(x_i)) &= 0 \\
\sum_{i=1}^{n} g(y_i, f_{m-1}(x_i)) + \gamma h(y_i, f_{m-1}(x_i)) &= 0 \\
\sum_{i=1}^{n} g(y_i, f_{m-1}(x_i)) + \sum_{i=1}^{n} \gamma h(y_i, f_{m-1}(x_i)) &= 0 \\
\gamma \sum_{i=1}^{n} h(y_i, f_{m-1}(x_i)) &= -\sum_{i=1}^{n} g(y_i, f_{m-1}(x_i))
\end{aligned}$$

Therefore, $\gamma = -\frac{\sum_{i=1}^{n} g(y_i, f_{m-1}(x_i))}{\sum_{i=1}^{n} h(y_i, f_{m-1}(x_i))}$

In binomial log-likelihood loss function setting,

$$\begin{aligned}
l(y, p(x)) &= ylog(p) + (1-y)log(1-p) \\
&= ylog(\frac{p}{1-p}) + log(1-p)
\end{aligned}$$

Let $q = log(\frac{p}{1-p})$, then:

$$\begin{aligned}
\frac{p}{1-p} &= e^q \\
p &= e^q - pe^q \\
1 + e^q &= \frac{e^q}{p} \\
p &= \frac{e^q}{1+e^q}
\end{aligned}$$

Then $l(y, p) = yq - log(1 + e^q)$, so:

$g = y - \frac{e^q}{1+e^q} = y - p$, and $h = -\frac{e^q}{1+e^q} + \frac{e^{2q}}{(1+e^q)^2} = -p + p^2 = p(1-p)$, substituting them into the γ function and we get: $\gamma = -\frac{\sum_{i=1}^{n} y_i - p}{\sum_{i=1}^{n} p(1-p)}$

References

- Bolker, B. (2023). STAT790 Notes Pipelines. Github. https://github.com/bbolker/stat790/blob/main/notes/pipelines.qmd
- Bujokas, E. (2022). Gradient Boosting in Python from Scratch. Medium. https://towardsdatascience.com/gradient-boosting-in-python-from-scratch-788d1cf1ca7
- Domke, J. (2010). Statistical Machine Learning Notes 9 Boosting. https://people.cs.umass.edu/~domke/courses/sml2010/09boosting.pdf
- Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer.
- Kuhn, M., Wickham, H. (2020). Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles. https://www.tidymodels.org
- Saini, A. (2021). Gradient Boosting Algorithm: A Complete Guide for Beginners. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/