Comparação de Classificadores

Ciência de Dados I

Hanny Araujo Borges - 12221BSI249
Igor Melo Mesquita - 12221BSI206
Maria Eduarda Dutra Silva - 12221BSI228

Índice

```
library(readxl)
library(data.table)
library(ggplot2)
library(dplyr)
```

```
Anexando pacote: 'dplyr'

Os seguintes objetos são mascarados por 'package:data.table':

    between, first, last

Os seguintes objetos são mascarados por 'package:stats':

    filter, lag

Os seguintes objetos são mascarados por 'package:base':

    intersect, setdiff, setequal, union
```

```
library(nnet)
library(rpart)
library(caret)
```

```
Carregando pacotes exigidos: lattice
```

```
library(class)
```

1 Introdução e Objetivos

Este trabalho de comparação de classificadores foi realizado com o objetivo de analisar as particularidades de três diferentes algoritmos de classificação, de modo a compreender melhor o funcionamento e resultados obtidos através de cada um, ranqueando os melhores e piores para diferentes bases de dados utilizando diferentes parâmetros. De tal forma, foram escolhidos os algoritmos de Árvore de Decisão, KNN e Regressão Logística; sendo o último escolhido daqueles que não foram estudados em sala de aula.

2 Descrição das Bases de Dados

Para a realização deste trabalho, foram selecionadas duas bases de dados públicas, que são descritas a seguir.

2.1 Dry Bean Dataset

- Link para a fonte: https://www.kaggle.com/datasets/joebeachcapital/dry-beans
- Número de Instâncias: 13.611.
- Número de Atributos: 17 (16 atributos previsores e 1 atributo classe).
- Atributo Classe: *Class* (contendo 7 tipos de feijão: *Barbunya, Bombay, Cali, Dermason, Horoz, Seker* e *Sira*).
- Valores Ausentes: A base de dados não possui valores ausentes.
- Tipos de Atributos: Os atributos são numéricos (inteiros e reais), representando medidas extraídas das imagens dos grãos, como área, perímetro, comprimento dos eixos principal e menor, e outros fatores de forma. Sendo eles:
  - Área (A): A área de um feijão e o número de pixels dentro de seus limites. à Contínuo
  - Perímetro (P): Tamanho da borda do feijão. à Contínuo
  - Comprimento do eixo principal (L): Definido pela distância das extremidades da linha mais longa a ser desenhada a partir de um feijão. à Contínuo
  - Comprimento do eixo menor (l): Definido pela linha mais longa que pode ser traçada

a partir do feijão, mantendo-se perpendicular ao eixo principal. à Contínuo

- Proporção (K): Define a relação entre os atributos L e l. à Contínuo
- Excentricidade (Ec): Excentricidade da elipse que possui os mesmos momentos da região. à Contínuo
- Área convexa (C): Número de pixels no menor polígono convexo capaz de conter a área da semente de feijão. à Discreto
- Diâmetro equivalente (Ed): Diâmetro de um círculo que possui a mesma área que a área da semente de feijão. à Contínuo
- Extensão (Ex): Razão entre o número de pixels do retângulo delimitador e a área da semente. à Contínuo
- Solidez (S): Também conhecido como convexidade. É a razão entre o número de pixels na casca convexa e os encontrados na semente. à Contínuo
- Circularidade (R): Quão circular é a forma, calculada pela fórmula que utiliza área (A) e perímetro (P). à Contínuo
  - R=(4 A)/ P2
- Compacidade (CO): Mede o grau de arredondamento de um objeto, definido pela fórmula que une diâmetro (Ed) e maior eixo (L). à Contínuo
  - CO = Ed/L
- Fator de Forma 1 (SF1), Fator de Forma 2 (SF2), Fator de Forma 3 (SF3), Fator de Forma 4 (SF4): Representam diferentes combinações matemáticas de área, perímetro e diâmetro para caracterizar a forma da semente. à Contínuo
- Classe: Categoria da semente de feijão (S*eker, Barbunya, Bombay, Cali, Dermosan, Horoz* e *Sira*). à Categórico

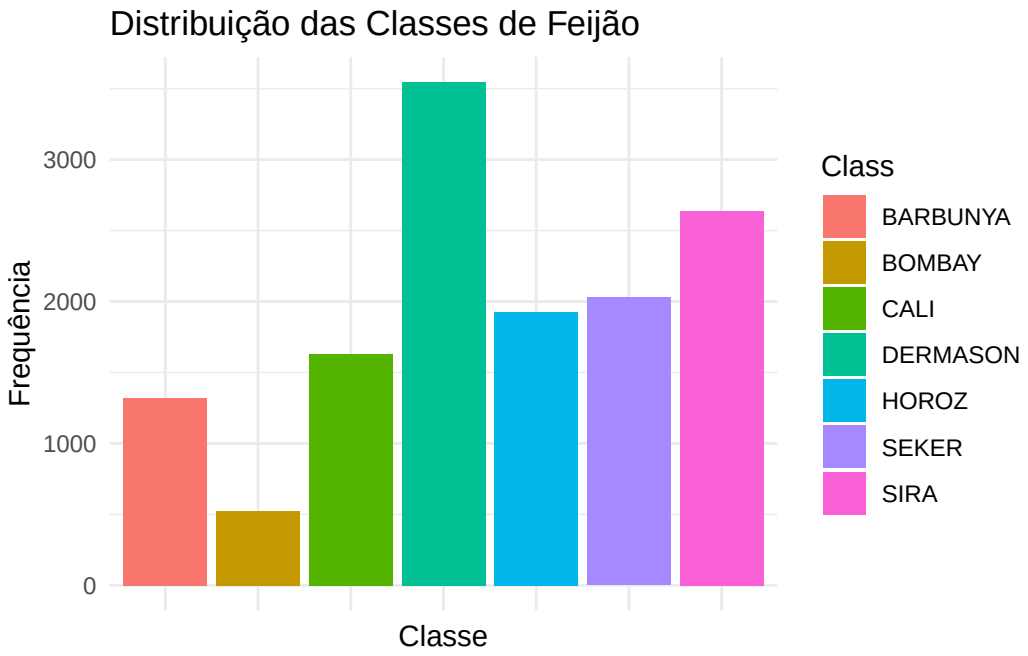
Observando os dados da classe, é perceptível um desbalanceamento entre as categorias de feijão, sendo a classe *Dermason* a de maior peso. Em contrapartida, a classe *Bombay* se encontra em desvantagem, por ter a menor das representações.

```
bean_df <- read_excel("Dry_Bean_Dataset.xlsx")
bean_df <- as.data.table(bean_df)   # se quiser table/dt

df <- bean_df %>%
  group_by(Class)%>%
```

```
  count(n())

ggplot(df, aes(x = Class, y = n, fill = Class)) +
  geom_col() +
  labs(title = "Distribuição das Classes de Feijão",
       x = "Classe",
       y = "Frequência") +
  theme_minimal() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



2.2 Glass Identification Dataset

- Link para a fonte: https://archive.ics.uci.edu/dataset/42/glass+identification
- Número de Instâncias: 214.
- Número de Atributos: 11 (10 atributos previsores e 1 atributo classe).
- Atributo Classe: *Type_of_glass* (com 7 tipos possíveis, embora um deles não tenha instâncias na base, resultando em 6 classes efetivas).

- Valores Ausentes: A base de dados não possui valores ausentes.
- Tipos de Atributos: Os atributos são numéricos contínuos, representando o índice de refração (RI) e a porcentagem em peso de óxidos de diferentes elementos químicos (Sódio, Magnésio, Alumínio, etc.). Sendo eles:
  - Id_number: número identificador. à Discreto
  - RI: Índice de refração. à Contínuo
  - Na: Percentual em peso de sódio. à Contínuo
  - Mg: Percentual em peso de magnésio. à Contínuo
  - Al: Percentual em peso de alumínio. à Contínuo
  - Si: Percentual em peso de silício. à Contínuo
  - K: Percentual em peso de potássio. à Contínuo
  - Ca: Percentual em peso de cálcio. à Contínuo
  - Ba: Percentual em peso de bário. à Contínuo
  - Fe: Percentual em peso de ferro. à Contínuo
  - Classe:

1. Janelas de edifícios - *float*
2. Janelas de edifícios - não *float*
3. Janelas de veículos - *float*
4. Janelas de veículos - não *float* (sem amostras na base)
5. Recipientes
6. Utensílios de mesa
7. Faróis de veículos

Nota: O termo "*float*" se refere à técnica de produção do vidro, onde uma camada fundida é nivelada sobre um banho de metal líquido que resulta num vidro uniforme e de superfície regular. Seu diferencial em relação ao método de "não *float*" está no índice de refração do material.

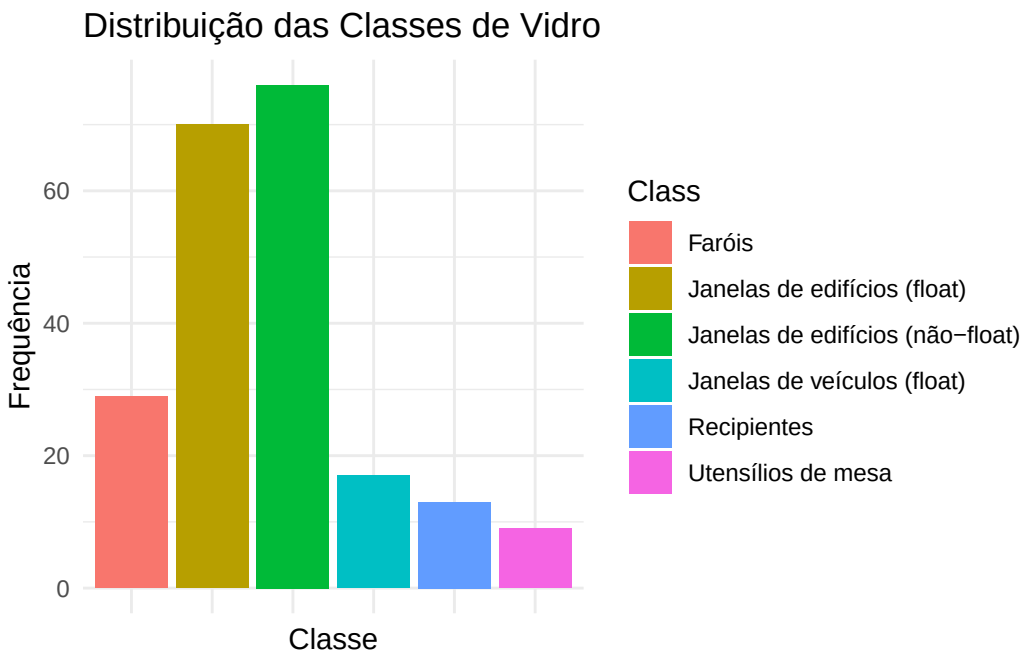
A base escolhida é desbalanceada, não contendo representação da classe 4 (janelas de veículos não *float*), e contendo discrepâncias de representação das demais classes, como ilustra o gráfico.

```
glass_df <- fread("glass.data", sep = ",", header = FALSE)
colnames(glass_df) <- c(
  "Id_number",
  "RI",  # V2
  "Na",               # V3
  "Mg",               # V4
  "Al",               # V5
  "Si",               # V6
  "K",                # V7
  "Ca",               # V8
  "Ba",               # V9
  "Fe",               # V10
  "Class"             # V11
)

df <- glass_df %>%
  group_by(Class)%>%
  count(n()) %>%
  mutate(Class = recode(Class,
                        `1` = "Janelas de edifícios (float)",
                        `2` = "Janelas de edifícios (não-float)",
                        `3` = "Janelas de veículos (float)",
                        `4` = "Janelas de veículos (não-float)",   # não há
                         ↪ amostras
                        `5` = "Recipientes",
                        `6` = "Utensílios de mesa",
                        `7` = "Faróis"
  ))



ggplot(df, aes(x = Class, y = n, fill = Class)) +
  geom_col() +
  labs(title = "Distribuição das Classes de Vidro",
       x = "Classe",
       y = "Frequência") +
  theme_minimal() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



3 Ferramenta

Para a execução dos experimentos realizados neste trabalho, foi utilizada a linguagem R no ambiente RStudio. Juntamente, foram usadas várias bibliotecas: readxl para ler os datasets em arquivos excel; data.table e dplyr para manipulação de dados; ggplot2 para criação de gráficos; nnet para implementações no modelo de Regressão Logística; rpart para implementações no modelo de Árvore de Decisão; caret para treinamento dos modelos; e class para implementações no modelo KNN.

4 Dry Bean Dataset

4.1 Pré-processamento

Na base Dry Beans, que contém 13.611 instâncias e 16 atributos numéricos derivados de medidas geométricas das sementes, foram aplicadas três etapas principais de pré-processamento: amostragem, redução de dimensionalidade e transformação de variáveis.

4.1.1 Amostragem

Devido a grande quantidade de instâncias do dataset Dry Beans, a amostragem se mostrou muito necessária, afinal, reduzindo o número de instâncias, reduz-se o esforço computacional preciso para processar os dados. Desta forma, escolheu-se a técnica de amostragem estratificada, de modo que todas as classes fossem representadas de maneira proporcional ao número de instâncias destas; essa técnica de proporção foi utilizada em decorrência do desbalanceamento das classes do dataset, ou seja, foi utilizada visando um balanceamento.

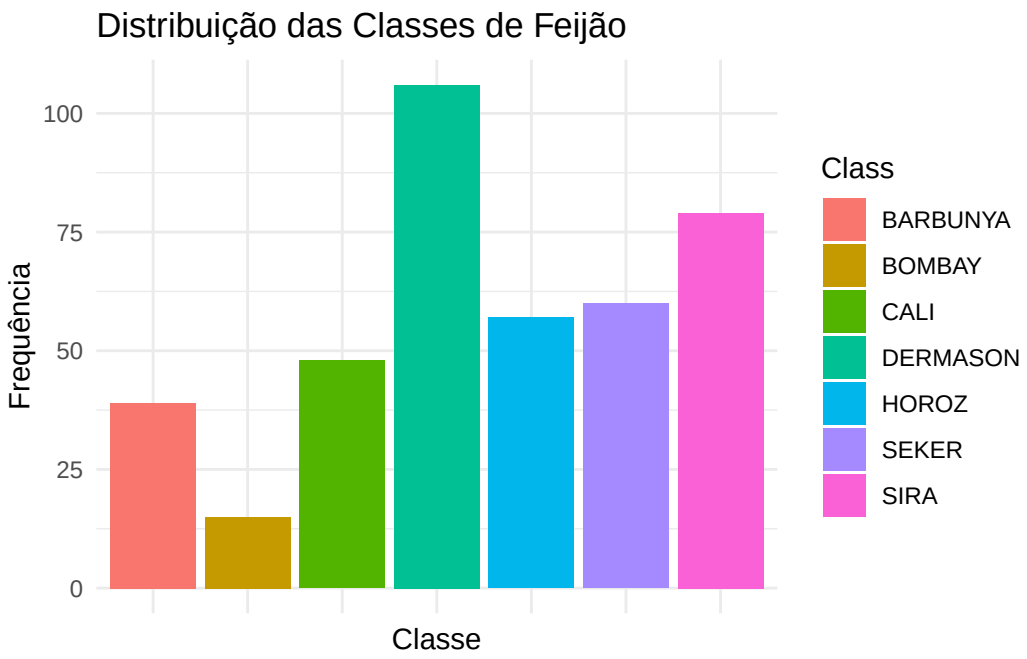
```
set.seed(42) # Reprodutibilidade

frac_global <- 0.03  # define o tamanho da partição para 3% do dataset

bean_df_bal <- bean_df %>%
  group_by(Class) %>%
  slice_sample(prop = frac_global) %>% # realiza a partiçao da amostragem
  ungroup()


# Garantia de manutenção das proporções
df <- bean_df_bal %>%
  group_by(Class)%>%
  count(n())

ggplot(df, aes(x = Class, y = n, fill = Class)) +
  geom_col() +
  labs(title = "Distribuição das Classes de Feijão",
       x = "Classe",
       y = "Frequência") +
  theme_minimal() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



4.1.2 Estratégia de Divisão da Base de Dados

Para a separação do dataset entre dados de treino e de teste, utilizou-se a técnica Holdout, que consiste em fazer uma única partição da amostra em 3 partes, sendo 2/3 para treino e 1/3 para teste.

```
set.seed(42) # Reprodutibilidade

idx_treino_bean <- bean_df_bal %>%
  mutate(row_id = dplyr::row_number()) %>%
  group_by(Class) %>%
  slice_sample(prop = 2/3) %>% # Realiza a partição da base para treino e
   ↪  teste
  pull(row_id)
```

4.1.3 Redução de Dimensionalidade

No dataset Dry Beans, observou-se que muitos atributos eram derivados de outros, e, portanto, havia redundância na base. Devido a isso, optou-se por utilizar a redução de dimensionalidade, a fim de retirar os atributos fundamentais, ou seja, os atributos que, a partir

deles, novos atributos surgiam. Por exemplo, o atributo *AspectRation* é calculado a partir da razão entre *MajorAxisLength* e *MinorAxisLength*, então *MajorAxisLenght* e *MinorAxisLength* seriam retirados.

- Os atributos retirados foram:
  - MinorAxisLength
  - MajorAxisLength
  - Area
  - Perimeter
  - ConvexArea

```
# Retirada das colunas fundamentais
bean_df_bal <- bean_df_bal %>%
  select(-c("MinorAxisLength", "MajorAxisLength", "Area", "Perimeter",
   ↪  "ConvexArea"))
```

4.1.4 Normalização

Analisando o dataset, é visível que ele possui instâncias com grande amplitude numérica, o que é ruim para algoritmos como KNN e Regressão Logística (que foram escolhidos para a realização deste trabalho). Sendo assim, foi necessária a normalização dos dados para que, conforme a necessidade de cada algoritmo, pudesse ter um melhor resultado. No caso do KNN, a normalização escolhida foi a re-escalar, que transforma os dados em um intervalo [0, 1], facilitando o cálculo das distâncias. Já para a Regressão Logística, escolheu-se a padronização, que converte os dados para uma média zero e desvio padrão 1, equilibrando as variáveis e facilitando a interpretação dos coeficientes.

4.1.4.1 Re-escalar

Normalização utilizada para o algoritmo KNN.

```
min_max_normalization <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Aplicando normalização (re-escala) nas colunas numéricas
```

```r
bean_df_res <- bean_df_bal %>%
  mutate(across(c("AspectRation", "Eccentricity", "EquivDiameter", "Extent",
   ↪  "Solidity", "roundness", "Compactness", "ShapeFactor1", "ShapeFactor2",
   ↪  "ShapeFactor3", "ShapeFactor4"), min_max_normalization))

bean_df_res %>%
  select(
    AspectRation,
    Eccentricity,
    EquivDiameter,
    Extent,
    Solidity,
    roundness,
    Compactness,
    ShapeFactor1,
    ShapeFactor2,
    ShapeFactor3,
    ShapeFactor4
  )
```

```
# A tibble: 404 x 11
   AspectRation Eccentricity EquivDiameter Extent Solidity roundness
          <dbl>        <dbl>         <dbl>  <dbl>    <dbl>     <dbl>
 1        0.280        0.640         0.371  0.548    0.832     0.644
 2        0.399        0.753         0.339  0.526    0.685     0.360
 3        0.301        0.663         0.454  0.445    0.766     0.396
 4        0.391        0.746         0.437  0.763    0.608     0.191
 5        0.174        0.493         0.479  0.668    0.840     0.716
 6        0.410        0.762         0.452  0.537    0.633     0.337
 7        0.376        0.735         0.377  0.537    0.791     0.524
 8        0.368        0.727         0.262  0.543    0.766     0.556
 9        0.346        0.708         0.446  0.788    0.790     0.592
10        0.286        0.648         0.243  0.864    0.932     0.868
# i 394 more rows
# i 5 more variables: Compactness <dbl>, ShapeFactor1 <dbl>,
#   ShapeFactor2 <dbl>, ShapeFactor3 <dbl>, ShapeFactor4 <dbl>
```

4.1.4.2 Padronização

Normalização utilizada para o algoritmo Regressão Logística.

```r
padronization_norm <- function(x){
  return((x-mean(x))/sd(x))
}

# Aplicando normalização (padronização) nas colunas numéricas
bean_df_pad <- bean_df_res %>%
  mutate(across(c("AspectRation", "Eccentricity", "EquivDiameter", "Extent",
   ↪  "Solidity", "roundness", "Compactness", "ShapeFactor1", "ShapeFactor2",
   ↪  "ShapeFactor3", "ShapeFactor4"), padronization_norm))

bean_df_pad %>%
  select(
    AspectRation,
    Eccentricity,
    EquivDiameter,
    Extent,
    Solidity,
    roundness,
    Compactness,
    ShapeFactor1,
    ShapeFactor2,
    ShapeFactor3,
    ShapeFactor4
  )
```

```
# A tibble: 404 x 11
   AspectRation Eccentricity EquivDiameter  Extent Solidity roundness
          <dbl>        <dbl>         <dbl>   <dbl>    <dbl>     <dbl>
 1      -0.649       -0.425         0.712  -0.730    -0.214   -0.415
 2      -0.0268       0.233         0.514  -0.851    -1.72    -2.11
 3      -0.540       -0.291         1.22   -1.30     -0.892   -1.89
 4      -0.0723       0.193         1.12    0.448    -2.52    -3.11
 5      -1.20        -1.28          1.38   -0.0765   -0.132    0.0115
 6       0.0297       0.282         1.21   -0.793    -2.26    -2.25
 7      -0.147        0.125         0.751  -0.790    -0.636   -1.13
 8      -0.193        0.0812        0.0384 -0.760    -0.888   -0.941
 9      -0.305       -0.0301        1.18    0.584    -0.642   -0.723
10      -0.616       -0.383        -0.0779  1.00      0.815    0.919
# i 394 more rows
# i 5 more variables: Compactness <dbl>, ShapeFactor1 <dbl>,
#   ShapeFactor2 <dbl>, ShapeFactor3 <dbl>, ShapeFactor4 <dbl>
```

4.2 Algoritmos

4.2.1 Árvore de Decisão

```
set.seed(42)  # Reprodutibilidade

bean_df_bal$Class <- as.factor(bean_df_bal$Class) # Garante que a classe é
 ↪  factor

# Armazena a parte de treino e base separada em idx_treino_bean
train_dt_bean <- bean_df_bal[idx_treino_bean, ]
test_dt_bean  <- bean_df_bal[-idx_treino_bean, ]

# Fórmula: alvo ~ todos os preditores
form_dt_bean <- reformulate(setdiff(names(train_dt_bean), 'Class'), response
 ↪  = 'Class')

# Executar Árvore de Decisão para cp (p1) = 0.05
cp_p1_dt_bean  <- rpart(form_dt_bean, data = train_dt_bean, method = "class",
 ↪  control = rpart.control(cp = 0.05))
pred_p1_dt_bean  <- predict(cp_p1_dt_bean,  newdata = test_dt_bean, type =
 ↪  "class")

# Executar Árvore de Decisão para cp (p2) = 0.001
cp_p2_dt_bean <- rpart(form_dt_bean, data = train_dt_bean, method = "class",
 ↪  control = rpart.control(cp = 0.001))
pred_p2_dt_bean <- predict(cp_p2_dt_bean, newdata = test_dt_bean, type =
 ↪  "class")
```

4.2.2 KNN

```
set.seed(42) # Reprodutibilidade

bean_df_res$Class <- as.factor(bean_df_res$Class) # Garante que a classe é
 ↪  factor

# Armazena a parte de treino e base separada em idx_treino_bean
train_knn_bean <- bean_df_res[idx_treino_bean, ]
```

```r
test_knn_bean  <- bean_df_res[-idx_treino_bean, ]

# Matriz de preditores (x) e vetores de classe (y)
X_train_knn_bean <- train_knn_bean %>%
  select(-all_of('Class')) %>%
  as.data.frame()

X_test_knn_bean  <- test_knn_bean %>%
  select(-all_of('Class')) %>%
  as.data.frame()

y_train_knn_bean <- train_knn_bean$Class
y_test_knn_bean <- test_knn_bean$Class

stopifnot(is.factor(y_train_knn_bean), is.factor(y_test_knn_bean)) # Verifica
 ↪  se todos os valores são factor (exigido pelo KNN)
stopifnot(all(sapply(X_train_knn_bean, is.numeric)),
 ↪  all(sapply(X_test_knn_bean, is.numeric))) # Verifica se todos os valores
 ↪  são numéricos (exigido pelo KNN)

# Executar KNN para k (p1) = 3
k_p1_knn_bean <- 3 # Define a quantidade de K-vizinhos mais próximos

pred_k_p1_knn_bean <- knn(
  train = X_train_knn_bean,   # base de treino (preditores)
  test  = X_test_knn_bean,    # base de teste  (preditores)
  cl    = y_train_knn_bean,   # classes de treino (factor)
  k     = k_p1_knn_bean       # número de vizinhos
)

# Executar KNN para k (p2) = 5
k_p2_knn_bean <- 5 # Define a quantidade de K-vizinhos mais próximos

pred_k_p2_knn_bean <- knn(
  train = X_train_knn_bean,   # base de treino (preditores)
  test  = X_test_knn_bean,    # base de teste  (preditores)
  cl    = y_train_knn_bean,   # classes de treino (factor)
  k     = k_p2_knn_bean        # número de vizinhos
)
```

4.2.3 Regressão Logística

4.2.3.1 Metodologia do Modelo de Regressão Logística Multinomial

A regressão logística multinomial é um modelo estatístico utilizado para prever bases que contenham variáveis dependentes e mais de duas categorias. O funcionamento da metodologia pode ser resumido como um processo de aprendizado contínuo: o modelo começa com estimativas iniciais e compara suas previsões com os resultados observados, após isso, reajusta seus parâmetros e repete o processo até estabilizar. Assim, captura padrões nos dados e os utiliza para realizar classificações em instâncias futuras. O passo a passo descrito seria algo como:

1. **Cálculo das probabilidades**

   A cada iteração, os dados são processados e o modelo gera pontuações para cada classe. Essas pontuações são transformadas em probabilidades através de uma função de normalização, garantindo que a soma das probabilidades de todas as classes seja igual a 1.

2. **Comparação com os valores reais**

   As probabilidades estimadas são comparadas com os valores observados na base de dados. O modelo verifica o grau de acerto e erro das previsões em relação às classes reais atribuídas a cada observação.

3. **Ajuste dos pesos**

   Com base na diferença entre valores previstos e reais, os pesos associados às variáveis são ajustados. Esse processo corrige a influência das variáveis, reforçando aquelas que ajudam na classificação correta e reduzindo o impacto daquelas que conduzem a erros. Abaixo segue exemplo de como essas correspondências são registradas, na parte de "*Coefficients*":

```r
set.seed(42)

# Armazena a parte de treino e base separada em idx_treino_bean
train_rl_bean <- bean_df_pad[idx_treino_bean, ]
test_rl_bean  <- bean_df_pad[-idx_treino_bean, ]

# Modelo gerado com atributos aleatórios, apenas para exemplificação da
↪ distribuição dos pesos
model_example <-  multinom(Class ~ Eccentricity + EquivDiameter + Extent
↪ + Solidity, data = train_rl_bean, trace = FALSE)

model_example
```

```
Call:
multinom(formula = Class ~ Eccentricity + EquivDiameter + Extent +
    Solidity, data = train_rl_bean, trace = FALSE)

Coefficients:
         (Intercept) Eccentricity EquivDiameter     Extent   Solidity
BOMBAY    -22.681027     1.582774     10.629813  0.5550999 1.20917285
CALI       -3.463678     7.928729      1.126689  0.6855321 0.74549467
DERMASON   -7.496789    10.425368    -39.956900 -1.4137513 2.70731595
HOROZ     -10.741360    25.417814    -19.422827 -1.4716727 0.09535921
SEKER      -3.089460    -4.824341    -10.511393 -2.1820014 2.75957230
SIRA        1.996261     9.699114    -21.796977 -0.7603323 0.85461759

Residual Deviance: 99.93688
AIC: 159.9369
```

4. **Iteração e convergência**

   O ciclo de cálculo de probabilidades, comparação com os valores reais e ajuste dos pesos é repetido sucessivamente. O processo segue até que os ajustes se tornem mínimos, caracterizando a convergência do modelo. Nesse ponto, considera-se que o modelo aprendeu a relação entre atributos e classes.

```r
# Continuação do algoritmo pós divisão de treino e teste
model_rl_bean <- multinom(Class ~ AspectRation + Eccentricity + EquivDiameter
 ↪  + Extent + Solidity + roundness + Compactness + ShapeFactor1 +
 ↪  ShapeFactor2 + ShapeFactor3 + ShapeFactor4, data = train_rl_bean, trace =
 ↪  FALSE)

coefs <- summary(model_rl_bean)$coefficients # pesos atribuidos a cada
 ↪  atributo de acordo com a classe

prediction_rl_bean <- predict(model_rl_bean, newdata = test_rl_bean)
```

Para testagem do desempenho do modelo com outros parâmetros, testaremos o treinamento apenas com os atributos mais relevantes para cada classe, baseado nos pesos a elas atribuídos.

```r
# Converter os coeficientes em um data frame para facilitar a manipulação
coefs_df <- as.data.frame(coefs)
# Calcular os coeficientes absolutos
coefs_abs <- abs(coefs_df)
```

```r
# Para cada variável (linha), pegar o maior valor absoluto e associar à
 ↪  variável
max_values <- apply(coefs_abs, 1, max)
# Agora associamos os valores máximos às variáveis (colunas)
max_vars <- apply(coefs_abs, 1, function(x) names(x)[which.max(x)])
# Criar um data frame para mostrar os atributos associados aos maiores
 ↪  valores
important_vars <- data.frame(Variable = max_vars, MaxValue = max_values)
# Ordenar pelas variáveis com maior coeficiente absoluto
important_vars_sorted <- important_vars[order(-important_vars$MaxValue), ]
# Mostrar as variáveis mais importantes
head(important_vars_sorted)
```

```
             Variable  MaxValue
BOMBAY   EquivDiameter 134.16990
HOROZ     AspectRation 100.76669
DERMASON  ShapeFactor1  97.92946
CALI      ShapeFactor1  86.57764
SEKER     AspectRation  73.02434
SIRA     EquivDiameter  72.88013
```

```r
model_rl_bean_less_att <- multinom(Class ~  EquivDiameter + roundness +
 ↪  ShapeFactor1 + ShapeFactor4, data = train_rl_bean, trace = FALSE)
coefs_bean_less_att <- summary(model_rl_bean_less_att)$coefficients # pesos
 ↪  atribuidos a cada atributo de acordo com a classe
prediction_rl_bean_less_att <- predict(model_rl_bean_less_att, newdata =
 ↪  test_rl_bean)
```

4.3 Medidas de Avaliação

Para realizar a avaliação dos classificadores, utilizou-se duas medidas principais: a matriz de confusão e a acurácia. A matriz de confusão é uma tabela usada para distinguir os tipos de erro de uma previsão com base no que foi previsto e no que seria a verdadeira classe. Já a acurácia calcula a proporção entre instâncias corretamentes classificadas em relação ao total de instâncias avaliadas, utilizando a fórmula abaixo.

$$\text{Acurácia} = (\text{TP} + \text{TN}) \,/\, (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

Em que: TP (true positive) e TN (true negative) representam as previsões corretas, enquanto FP (false positive) e FN (false negative) correspondem às classificações incorretas.

```r
# -------------------- Árvore de Decisão --------------------

# Avaliar | cp (p1) = 0.05
conf_p1_dt_bean <- confusionMatrix(pred_p1_dt_bean,test_dt_bean$Class)
acc_p1_dt_bean <- mean(pred_p1_dt_bean == test_dt_bean$Class)

# Avaliar | cp (p2) = 0.001
conf_p2_dt_bean <- confusionMatrix(pred_p2_dt_bean,test_dt_bean$Class)
acc_p2_dt_bean <- mean(pred_p2_dt_bean == test_dt_bean$Class)

print(paste("A acurácia do dataset DRY BEAN é superior com cp = ",
            ifelse(acc_p1_dt_bean > acc_p2_dt_bean, 0.05, 0.001)
            )
      )
```

```
[1] "A acurácia do dataset DRY BEAN é superior com cp =  0.001"
```

```r
# -------------------- KNN --------------------

# Avaliar | k (p1) = 3
conf_k_p1_knn_bean <- confusionMatrix(pred_k_p1_knn_bean, y_test_knn_bean)
acc_k_p1_knn_bean  <- mean(pred_k_p1_knn_bean == y_test_knn_bean)

# Avaliar | k (p2) = 5
conf_k_p2_knn_bean  <- confusionMatrix(pred_k_p2_knn_bean, y_test_knn_bean)
acc_k_p2_knn_bean   <- mean(pred_k_p2_knn_bean == y_test_knn_bean)

print(paste("A acurácia do dataset DRY BEAN é superior com knn = ",
            ifelse(acc_k_p2_knn_bean > acc_k_p1_knn_bean, 5, 3)
            )
      )
```

```
[1] "A acurácia do dataset DRY BEAN é superior com knn =  3"
```

```r
# -------------------- Regressão Logística --------------------


test_rl_bean$Class <- as.factor(test_rl_bean$Class)
# Avaliar | todas as classes
conf_rl_bean <- confusionMatrix(prediction_rl_bean, test_rl_bean$Class)
acc_rl_bean  <- mean(prediction_rl_bean == test_rl_bean$Class)
```

```
# Avaliar | apenas classes mais importantes (baseado no modelo que usa todas
 ↪  as classes)
conf_rl_bean_less_att <-
 ↪  confusionMatrix(prediction_rl_bean_less_att,test_rl_bean$Class)
acc_rl_bean_less_att  <- mean(prediction_rl_bean_less_att ==
 ↪  test_rl_bean$Class)

print(paste("A acurácia do dataset DRY BEAN é superior considerando ",
            ifelse(acc_rl_bean > acc_rl_bean_less_att, "todos os atributos",
             ↪  "apenas os atributos selecionados")
            )
      )
```

```
[1] "A acurácia do dataset DRY BEAN é superior considerando  apenas os
 ↪  atributos selecionados"
```

5 Glass Identification Dataset

5.1 Pré-processamento

Na base Glass Identification, composta por 214 instâncias e 9 atributos numéricos, foram aplicadas duas etapas principais de pré-processamento: seleção de atributos e transformação de variáveis.

5.1.1 Seleção de Atributos

Na seleção de atributos, foi realizada a remoção do campo *Id_number*, uma vez que esse atributo não possui relevância para a classificação e poderia introduzir ruído no modelo.

```
glass_df <- glass_df %>%
  select(-Id_number)
```

5.1.2 Estratégia da Divisão da Base de Dados

Para a separação do dataset entre dados de treino e de teste, utilizou-se a técnica Holdout, que consiste em fazer uma única partição da amostra em 3 partes, sendo 2/3 para treino e 1/3

para teste.

```
set.seed(42)  # Reprodutibilidade

idx_treino_glass <- glass_df %>%
  mutate(row_id = dplyr::row_number()) %>%
  group_by(Class) %>%
  slice_sample(prop = 2/3) %>% # Realiza a partição da base para treino e
   ↪  teste
  pull(row_id)
```

5.1.3 Normalização

Analisando o dataset, é visível que ele possui instâncias com grande amplitude numérica, o que é ruim para algoritmos como KNN e Regressão Logística (que foram escolhidos para a realização deste trabalho). Sendo assim, foi necessária a normalização dos dados para que, conforme a necessidade de cada algoritmo, pudesse ter um melhor resultado. No caso do KNN, a normalização escolhida foi a re-escalar, que transforma os dados em um intervalo [0, 1], facilitando o cálculo das distâncias. Já para a Regressão Logística, escolheu-se a padronização, que converte os dados para uma média zero e desvio padrão 1, equilibrando as variáveis e facilitando a interpretação dos coeficientes.

5.1.3.1 Re-escalar

Normalização utilizada para o algoritmo KNN.

```
glass_df_res <- glass_df %>%
  mutate(across(c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe"),
   ↪  min_max_normalization))

glass_df_res %>%
  select(-Class)
```

```
           RI        Na        Mg        Al        Si           K
        <num>     <num>     <num>     <num>     <num>       <num>
  1: 0.4328358 0.4375940 1.0000000 0.2523364 0.3517857 0.009661836
  2: 0.2835821 0.4751880 0.8017817 0.3333333 0.5214286 0.077294686
  3: 0.2208077 0.4210526 0.7906459 0.3894081 0.5678571 0.062801932
  4: 0.2857770 0.3729323 0.8218263 0.3115265 0.5000000 0.091787440
  5: 0.2752414 0.3819549 0.8062361 0.2959502 0.5839286 0.088566828
```

```
 ---
210: 0.2230026 0.5127820 0.0000000 0.8068536 0.5000000 0.012882448
211: 0.2502195 0.6300752 0.0000000 0.5295950 0.5803571 0.000000000
212: 0.4170325 0.5458647 0.0000000 0.5389408 0.6446429 0.000000000
213: 0.2352941 0.5488722 0.0000000 0.5140187 0.6785714 0.000000000
214: 0.2616330 0.5263158 0.0000000 0.5576324 0.6339286 0.000000000
           Ca        Ba    Fe
        <num>     <num> <num>
  1: 0.3085502 0.0000000     0
  2: 0.2230483 0.0000000     0
  3: 0.2184015 0.0000000     0
  4: 0.2592937 0.0000000     0
  5: 0.2453532 0.0000000     0
 ---
210: 0.3485130 0.3365079     0
211: 0.2760223 0.5047619     0
212: 0.2797398 0.5206349     0
213: 0.2834572 0.4984127     0
214: 0.2964684 0.5301587     0
```

5.1.3.2 Padronização

Normalização utilizada para o algoritmo Regressão Logística.

```
glass_df_pad <- glass_df %>%
  mutate(across(c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe"),
   ↪  padronization_norm))

glass_df_pad %>%
  select(-Class)
```

```
            RI         Na         Mg         Al          Si
         <num>      <num>      <num>      <num>       <num>
  1:  0.8708258  0.2842867  1.2517037 -0.6908222 -1.12444556
  2: -0.2487502  0.5904328  0.6346799 -0.1700615  0.10207972
  3: -0.7196308  0.1495824  0.6000157  0.1904651  0.43776033
  4: -0.2322859 -0.2422846  0.6970756 -0.3102663 -0.05284979
  5: -0.3113148 -0.1688095  0.6485456 -0.4104126  0.55395746
 ---
210: -0.7031664  0.8965789 -1.8611468  2.8743856 -0.05284979
211: -0.4990084  1.8517548 -1.8611468  1.0917817  0.52813587
212:  0.7522825  1.1659875 -1.8611468  1.1518695  0.99292440
```

```
213: -0.6109660  1.1904792 -1.8611468  0.9916354  1.23822946
214: -0.4133938  1.0067915 -1.8611468  1.2720450  0.91545965
              K          Ca          Ba          Fe
          <num>       <num>       <num>       <num>
  1: -0.67013422 -0.1454254 -0.3520514 -0.5850791
  2: -0.02615193 -0.7918771 -0.3520514 -0.5850791
  3: -0.16414813 -0.8270103 -0.3520514 -0.5850791
  4:  0.11184428 -0.5178378 -0.3520514 -0.5850791
  5:  0.08117845 -0.6232375 -0.3520514 -0.5850791
 ---
210: -0.63946840  0.1567205  1.7798049 -0.5850791
211: -0.76213169 -0.3913581  2.8457330 -0.5850791
212: -0.76213169 -0.3632515  2.9462923 -0.5850791
213: -0.76213169 -0.3351449  2.8055093 -0.5850791
214: -0.76213169 -0.2367718  3.0066278 -0.5850791
```

5.2 Algoritmos

5.2.1 Árvore de Decisão

```r
set.seed(42)  # Reprodutibilidade

glass_df$Class <- as.factor(glass_df$Class) # Garante que a classe é factor

# Armazena a parte de treino e base separada em idx_treino_bean
train_dt_glass <- glass_df[idx_treino_glass, ]
test_dt_glass  <- glass_df[-idx_treino_glass, ]

# Fórmula: alvo ~ todos os preditores
form_dt_glass <- reformulate(setdiff(names(train_dt_glass), 'Class'),
 ↪  response = 'Class')

# Treinar duas árvores com cp diferentes
cp_p1_dt_glass  <- rpart(form_dt_glass, data = train_dt_glass, method =
 ↪  "class", control = rpart.control(cp = 0.05))
cp_p2_dt_glass <- rpart(form_dt_glass, data = train_dt_glass, method =
 ↪  "class", control = rpart.control(cp = 0.001))

# Prever no conjunto de teste
pred_p1_dt_glass  <- predict(cp_p1_dt_glass,  newdata = test_dt_glass, type =
 ↪  "class")
```

```r
pred_p2_dt_glass <- predict(cp_p2_dt_glass, newdata = test_dt_glass, type =
 ↪  "class")
```

5.2.2 KNN

```r
set.seed(42) # Reprodutibilidade

glass_df_res$Class <- as.factor(glass_df_res$Class) # Garante que a classe é
 ↪  factor

# Armazena a parte de treino e base separada em idx_treino_bean
train_knn_glass <- glass_df_res[idx_treino_glass, ]
test_knn_glass  <- glass_df_res[-idx_treino_glass, ]

# Matriz de preditores (x) e vetores de classe (y)
X_train_knn_glass <- train_knn_glass %>% select(-all_of('Class')) %>%
 ↪  as.data.frame()
X_test_knn_glass  <- test_knn_glass  %>% select(-all_of('Class')) %>%
 ↪  as.data.frame()

y_train_knn_glass <- train_knn_glass$Class
y_test_knn_glass  <- test_knn_glass$Class

stopifnot(is.factor(y_train_knn_glass), is.factor(y_test_knn_glass)) #
 ↪  Verifica se todos os valores são factor (exigido pelo KNN)
stopifnot(all(sapply(X_train_knn_glass, is.numeric)),
 ↪  all(sapply(X_test_knn_glass, is.numeric))) # Verifica se todos os valores
 ↪  são numéricos (exigido pelo KNN)

# Executar KNN para k (p1) = 3
k_p1_knn_glass <- 3 # Define a quantidade de K-vizinhos mais próximos

pred_k_p1_knn_glass <- knn(
  train = X_train_knn_glass,
  test  = X_test_knn_glass,
  cl    = y_train_knn_glass,
  k     = k_p1_knn_glass
)

# Executar KNN para k (p2) = 5
```

```r
k_p2_knn_glass <- 5 # Define a quantidade de K-vizinhos mais próximos

pred_k_p2_knn_glass <- knn(
  train = X_train_knn_glass,
  test  = X_test_knn_glass,
  cl    = y_train_knn_glass,
  k     = k_p2_knn_glass
)
```

5.2.3 Regressão Logística

Ver seção 4.3.3.1 para explicação da **Metodologia do Modelo de Regressão Logística Multinomial.**

```r
set.seed(42)

# Armazena a parte de treino e base separada em idx_treino_bean
train_rl_glass <- glass_df_pad[idx_treino_glass, ]
test_rl_glass <- glass_df_pad[-idx_treino_glass, ]


model <- multinom(Class ~ RI + Na + Mg + Al + Si + K + Ca + Ba + Fe, data =
 ↪  train_rl_glass, trace = FALSE)
coefs <- summary(model)$coefficients # pesos atribuidos a cada atributo de
 ↪  acordo com a classe
prediction_rl_glass <- predict(model, newdata = test_rl_glass)
```

Para testagem do desempenho do modelo com outros parâmetros, testaremos o treinamento apenas com os atributos mais relevantes para cada classe, baseado nos pesos a elas atribuídos.

```r
# Converter os coeficientes em um data frame para facilitar a manipulação
coefs_df <- as.data.frame(coefs)
# Calcular os coeficientes absolutos
coefs_abs <- abs(coefs_df)
# Para cada variável (linha), pegar o maior valor absoluto e associar à
 ↪  variável
max_values <- apply(coefs_abs, 1, max)
# Agora associamos os valores máximos às variáveis (colunas)
max_vars <- apply(coefs_abs, 1, function(x) names(x)[which.max(x)])
```

```r
# Criar um data frame para mostrar os atributos associados aos maiores
 ↪  valores
important_vars <- data.frame(Variable = max_vars, MaxValue = max_values)
# Ordenar pelas variáveis com maior coeficiente absoluto
important_vars_sorted <- important_vars[order(-important_vars$MaxValue), ]
# Mostrar as variáveis mais importantes
head(important_vars_sorted)
```

```
  Variable  MaxValue
6        K 393.03459
7       Al 378.79727
5       Mg 281.84083
2       Mg  12.93198
3        K  12.61027
```

```r
model_less_att <- multinom(Class ~ Mg + Al + K, data = train_rl_glass, trace
 ↪  = FALSE)
coefs_less_att <- summary(model_less_att)$coefficients # pesos atribuidos a
 ↪  cada atributo de acordo com a classe
prediction_rl_glass_less_att <- predict(model_less_att, newdata =
 ↪  test_rl_glass)
```

5.3 Medidas de Avaliação

Para realizar a avaliação dos classificadores, utilizou-se duas medidas principais: a matriz de confusão e a acurácia. A matriz de confusão é uma tabela usada para distinguir os tipos de erro de uma previsão com base no que foi previsto e no que seria a verdadeira classe. Já a acurácia calcula a proporção entre instâncias corretamentes classificadas em relação ao total de instâncias avaliadas, utilizando a fórmula abaixo.

Acurácia = (TP + TN) / (TP + TN + FP + FN)

Em que: TP (true positive) e TN (true negative) representam as previsões corretas, enquanto FP (false positive) e FN (false negative) correspondem às classificações incorretas.

```r
# -------------------- Árvore de Decisão --------------------

# Avaliar | cp (p1) = 0.05
conf_p1_dt_glass  <- confusionMatrix(pred_p1_dt_glass,  test_dt_glass$Class)
acc_p1_dt_glass  <- mean(pred_p1_dt_glass == test_dt_glass$Class)
```

```r
# Avaliar | cp (p2) = 0.001
conf_p2_dt_glass <- confusionMatrix(pred_p2_dt_glass, test_dt_glass$Class)
acc_p2_dt_glass <- mean(pred_p2_dt_glass == test_dt_glass$Class)

print(paste("A acurácia do dataset GLASS IDENTIFICATION é superior com cp =
 ↪  ",
            ifelse(acc_p1_dt_glass > acc_p2_dt_glass, 0.05, 0.001)
            )
      )
```

```
[1] "A acurácia do dataset GLASS IDENTIFICATION é superior com cp =  0.001"
```

```r
# -------------------- KNN --------------------

# Avaliar | k (p1) = 3
conf_k_p1_knn_glass <- confusionMatrix(pred_k_p1_knn_glass, y_test_knn_glass)
acc_k_p1_knn_glass  <- mean(pred_k_p1_knn_glass == y_test_knn_glass)

# Avaliar | k (p2) = 5
conf_k_p2_knn_glass <- confusionMatrix(pred_k_p2_knn_glass, y_test_knn_glass)
acc_k_p2_knn_glass  <- mean(pred_k_p2_knn_glass == y_test_knn_glass)

print(paste("A acurácia do dataset GLASS IDENTIFICATION é superior com knn =
 ↪  ",
            ifelse(acc_k_p2_knn_glass > acc_k_p1_knn_glass, 5, 3)
            )
      )
```

```
[1] "A acurácia do dataset GLASS IDENTIFICATION é superior com knn =  3"
```

```r
# -------------------- Regressão Logística --------------------

test_rl_glass$Class <- as.factor(test_rl_glass$Class)

# Avaliar | todas as classes
conf_rl_glass <- confusionMatrix(prediction_rl_glass, test_rl_glass$Class)
acc_rl_glass  <- mean(prediction_rl_glass == test_rl_glass$Class)

# Avaliar | apenas classes mais importantes (baseado no modelo que usa todas
 ↪  as classes)
```

```
conf_rl_glass_less_att <-
 ↪  confusionMatrix(prediction_rl_glass_less_att,test_rl_glass$Class)
acc_rl_glass_less_att  <- mean(prediction_rl_glass_less_att ==
 ↪  test_rl_glass$Class)

print(paste("A acurácia do dataset GLASS IDENTIFICATION é superior
 ↪  considerando ",
            ifelse(acc_rl_glass > acc_rl_glass_less_att, "todos os
             ↪  atributos", "apenas os atributos selecionados")
            )
      )
```

```
[1] "A acurácia do dataset GLASS IDENTIFICATION é superior considerando
 ↪  todos os atributos"
```

6 Resultados

6.1 Dry Bean Dataset

```
# -------------------- Acurácia --------------------

comparative_table_bean <- data.frame(
  Dataset = "Dry Bean",
  Classificator = c("Árvore de Decisão", "KNN", "Regressão Logística"),
  Accuracy = c(acc_p2_dt_bean,acc_k_p2_knn_bean, acc_rl_bean_less_att)
)

comparative_table_bean
```

```
   Dataset       Classificator  Accuracy
1 Dry Bean   Árvore de Decisão 0.8676471
2 Dry Bean                 KNN 0.8897059
3 Dry Bean Regressão Logística 0.9338235
```

```
# -------------------- Matriz de Confusão --------------------

# Extrair os dados da matriz de confusão e definir modelos
cm_dt_bean <- as.data.frame(conf_p2_dt_bean$table)
```

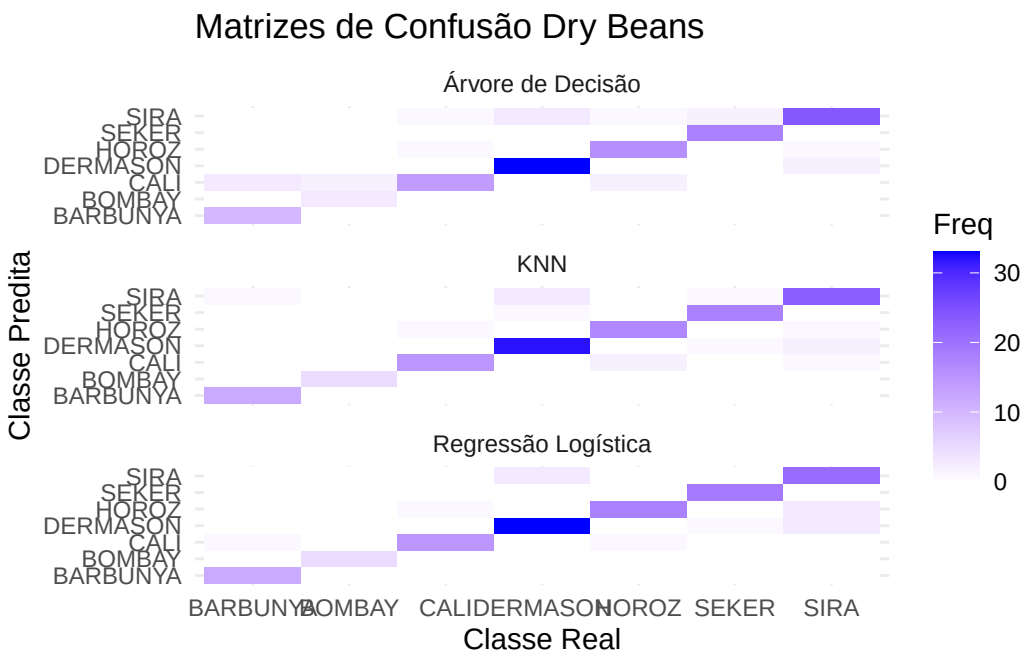
```r
cm_dt_bean$model <- "Árvore de Decisão"

cm_knn_bean <- as.data.frame(conf_k_p1_knn_bean$table)
cm_knn_bean$model <- "KNN"

cm_rl_bean <- as.data.frame(conf_rl_bean$table)
cm_rl_bean$model <- "Regressão Logística"

# Combinar as matrizes de confusão
cm_beans <- rbind(cm_dt_bean, cm_knn_bean, cm_rl_bean)

ggplot(cm_beans, aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile() +
  facet_wrap(~ model, ncol = 1) + # força 1 coluna → todas as matrizes
   ↪  embaixo da outra
  scale_fill_gradient(low = "white", high = "blue") +
  theme_minimal() +
  labs(
    title ="Matrizes de Confusão Dry Beans",
    x = "Classe Real",
    y = "Classe Predita"
  )
```



Através da análise dos avaliadores, observa-se o melhor desempenho da regressão logística em termos de acurácia, seguida pelo KNN e posteriormente pela árvore de decisão. Esse resultado se dá pela natureza dos atributos da base escolhida, dado que correspndem à medidas geométrica e fatores já derivados, com grande correlação entre os dados e próximos de uma relação linear com as classes. Por ser um modelo paramétrico, a regressão logística consegue explorar bem essas relações globais quando há uma boa quantidade de dados, além de se beneficiar das técnicas de pré processamento aplicadas para potencializar a convergência do modelo

O KNN obteve desempenho intermediário. Isso se deve ao fato de que o algoritmo depende fortemente da proximidade entre instâncias, e no caso de variedades de feijão com características muito semelhantes, os vizinhos podem pertencer a classes diferentes, o que aumenta o número de erros. Ainda assim, o KNN se mostrou competitivo, principalmente porque a normalização min–max aplicada no pré-processamento garantiu que todos os atributos contribuíssem de forma equilibrada no cálculo das distâncias.

A Árvore de Decisão foi o algoritmo com menor desempenho. Provavelmente isso se dá devido à tendência do classificador em criar divisões bem estabelecidas com base nos atributos, não sendo sensível às fronteiras sutis de decisão como as que distinguem as variedades de feijão. Mesmo com o aumento da complexidade da árvore e o procedente aumento da acurácia, o modelo ainda se manteve inferior.

6.2 Glass Identification Dataset

```
# -------------------- Acurácia --------------------
comparative_table_glass <- data.frame(
  Dataset = "Glass Identification",
  Classificator = c("Árvore de Decisão", "KNN", "Regressão Logística"),
  Accuracy = c(acc_p2_dt_glass,acc_k_p1_knn_glass, acc_rl_glass)
)

comparative_table_glass
```

```
               Dataset       Classificator  Accuracy
1 Glass Identification   Árvore de Decisão 0.7027027
2 Glass Identification                 KNN 0.7297297
3 Glass Identification Regressão Logística 0.6216216
```

```
# -------------------- Matriz de Confusão --------------------
```

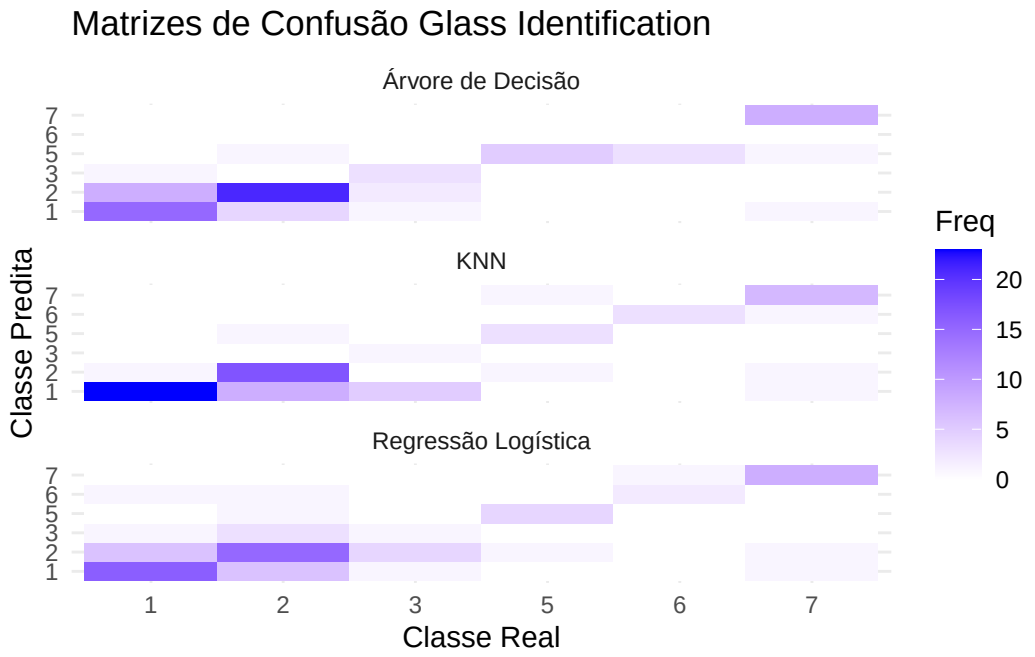
```r
# Extrair os dados da matriz de confusão e definir modelos
cm_dt_glass <- as.data.frame(conf_p2_dt_glass$table)
cm_dt_glass$model <- "Árvore de Decisão"

cm_knn_glass <- as.data.frame(conf_k_p1_knn_glass$table)
cm_knn_glass$model <- "KNN"

cm_rl_glass <- as.data.frame(conf_rl_glass$table)
cm_rl_glass$model <- "Regressão Logística"

# Combinar as matrizes de confusão
cm_glasss <- rbind(cm_dt_glass, cm_knn_glass, cm_rl_glass)

ggplot(cm_glasss, aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile() +
  facet_wrap(~ model, ncol = 1) + # força 1 coluna → todas as matrizes
   ↪  embaixo da outra
  scale_fill_gradient(low = "white", high = "blue") +
  theme_minimal() +
  labs(
    title ="Matrizes de Confusão Glass Identification",
    x = "Classe Real",
    y = "Classe Predita"
  )
```



Na base de Glass Identification, a ordem de desempenho se inverteu parcialmente: o KNN apresentou a maior acurácia, seguido pela Árvore de Decisão, e por último a Regressão Logística. Esse comportamento está relacionado às características da base, que é pequena (214 instâncias) e com classes muito próximas em termos de composição química. É observável que todos os modelos apresentaram uma acurácia intermediária, apontando uma possível necessidade de re-testagem da base em outros modelos de classificação ou com outras parametrizações.

O melhor desempenho do KNN em relação aos demais se deu pela facilidade do método em lidar com padrões locais, identificando semelhanças entre instâncias mesmo em um espaço não linear. Em bases menores como o Glass Identification, a ausência de informações abre espaço para suposições paramétricas e faz com que o KNN consiga se adaptar melhor a regiões complexas do espaço de atributos, estando esses normalizados.

A árvore de decisão obteve desempenho intermediário em relação aos outros modelos. Isso se deve ao fato de que o método é capaz de identificar interações não lineares entre os atributos químicos, formando divisões que traduzem combinações específica de valores. Apesar disso, a limitação de profundidade devido ao baixo número de instâncias e a sensibilidade a ruídos a deixa numa posição inferior em relação ao KNN.

A Regressão Logística, por sua vez, apresentou o pior desempenho no Glass Identification. A baixa acurácia pode ser atribuída ao tamanho reduzido do conjunto e à natureza das classes: as fronteiras entre tipos de vidro dificilmente são lineares, o que limita a capacidade do modelo

de separar class de forma ideal. Além disso, mesmo após a padronização das variáveis, o desbalanceamento entre as classes aumenta a instabilidade do modelo e consequentemente afeta as estimativas dos coeficientes.

6.3 Comparação Geral e Implicações

Os resultados mostram que não existe um único algoritmo que seja superior em todos os contextos. É perceptível que o desempenho das classificações depende das características individuais de cada conjunto de dados, assim como o significado de seus atributos que favorecem a interpretação de como esses se correlacionam. A partir disso, cada metodologia de classificação tratará melhor as bases que correspondam à lógica que aplicam em seu algoritmo.

Tal afirmação se comprova pela discrepância de desempenho dos classificadores nas bases. Observa-se, por exemplo, que as matrizes de confusão da base Dry Beans apresenta uma maior estabilidade e "nitidez" em comparação ao Glass Identification. Essa medida, unida às acurácias apresentadas ao final, indicam que os classificadores são ideais para bases que se comportam de forma semelhante ao Dry Beans, e possivelmente não seriam recomendáveis para bases como o Glass Identification. Essa dessemelhança também pode estar associada ao tamanho das bases. Em bases grandes, com atributos derivados e relações mais próximas da linearidade, como no Dry Beans, a Regressão Logística se mostrou a mais eficaz. Já em bases pequenas e com fronteiras de decisão complexas, como no Glass, métodos não paramétricos (KNN e Árvore) superaram a RL.

Ademais, o estudo evidencia a importância do pré-processamento: a normalização (re-escala para KNN e padronização para RL) foi essencial para que os algoritmos funcionassem adequadamente, e a redução de dimensionalidade no Dry Beans contribuiu para reduzir redundâncias, melhorando o desempenho dos modelos.

Com essas conclusões, é evidente a necessidade de conhecimento prévio dos dados presentes na base a ser analisada, assim como o que representam. Destaca-se ainda a importância de uma compreensão clara do objetivo a ser alcançado com os processamentos, para que assim seja possível traçar a relevância dos atributos, o algoritmo a ser utilizado e o resultado que se procura.