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

A área de Ciência de Dados tem se mostrado fundamental para a extração de conhecimento e suporte à tomada de decisão a partir de grandes volumes de dados. Dentro deste campo, a classificação supervisionada se destaca como uma das tarefas mais comuns e importantes, permitindo a categorização de novos dados em classes pré-definidas com base em um conjunto de dados previamente rotulado. A escolha do algoritmo de classificação e seu correto ajuste são etapas cruciais que impactam diretamente o desempenho e a precisão do modelo final.

Este trabalho, realizado no âmbito da disciplina de Ciência de Dados, tem como objetivo principal aplicar, avaliar e interpretar o desempenho de diferentes algoritmos de classificação supervisionada. Para isso, serão utilizadas duas bases de dados públicas com características distintas: o *Dry Bean Dataset* e o *Glass Identification Dataset*.

2 Descrição das Bases de Dados

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

2.1 Dry Bean Dataset

Esta base de dados foi obtida a partir de um estudo que utilizou um sistema de visão computacional para extrair características de imagens de grãos de feijão. O objetivo é classificar os grãos em sete variedades diferentes.

- Link para a fonte: https://www.kaggle.com/datasets/joebeachcapital/dry-beans
- Descrição Geral:
- 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

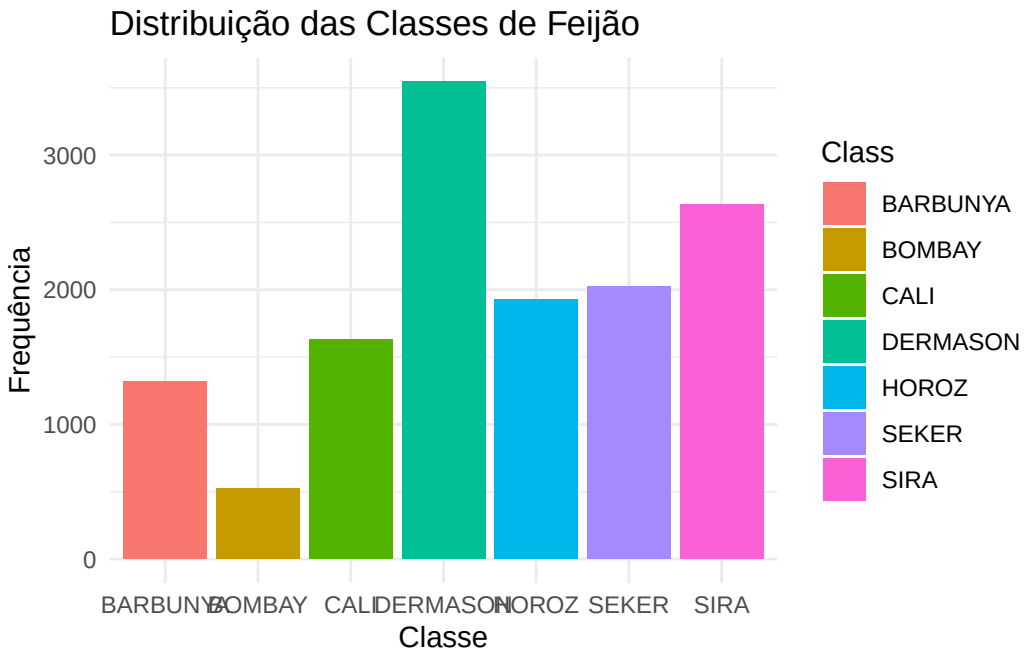
    * CO = Ed/L

  - 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()
```



2.2 Glass Identification Dataset

Esta base de dados foi criada para investigações de ciência forense, onde fragmentos de vidro encontrados em cenas de crime podem ser usados como evidência se corretamente identificados. A classificação se baseia na composição química do vidro.

- Link para a fonte: https://archive.ics.uci.edu/dataset/42/glass+identification
- Descrição Geral:
- 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:

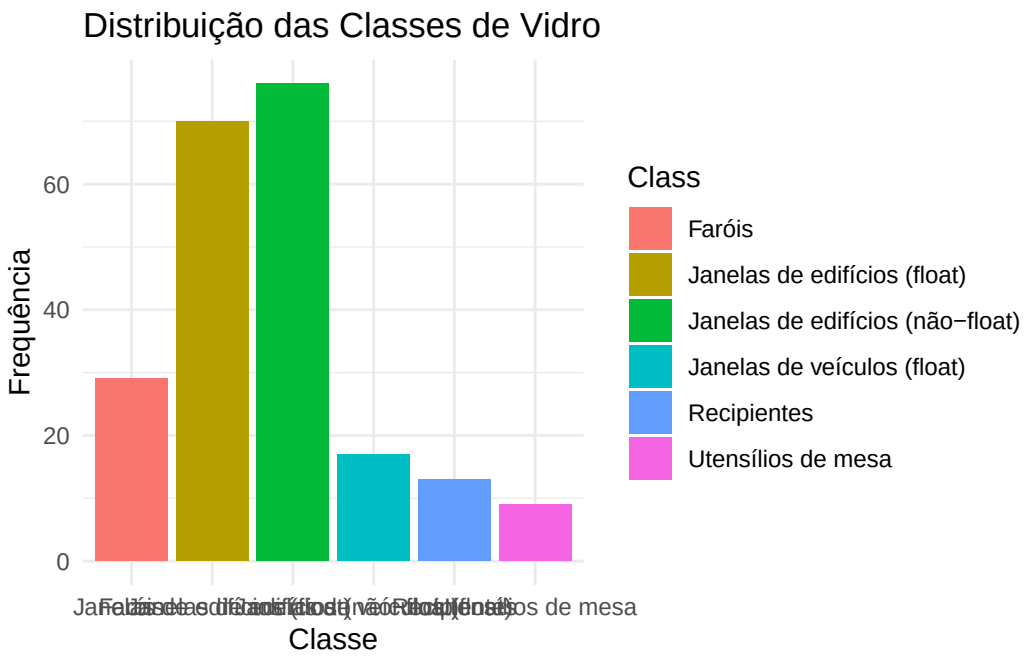
- Id_number: número identificador. à Discreto
- RI: Índice de refração. à Contínuo
- Na: Percentual em peso de sódio. à Contínuo
- Mg: Percentual em peso de magnésio. à Contínuo
- Al: Percentual em peso de alumínio. à Contínuo
- Si: Percentual em peso de silício. à Contínuo
- K: Percentual em peso de 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.

```r
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()
```



3 Ferramenta

Os experimentos foram realizados na linguagem **R**, em ambiente RStudio, utilizando pacotes amplamente empregados na área de ciência de dados. O pré-processamento e a manipulação das bases foram feitos principalmente com o pacote **dplyr**, enquanto a execução dos algoritmos contou com pacotes específicos: **class** (para o KNN), **nnet** (para a Regressão Logística Multinomial), **rpart** (para a Árvore de Decisão) e **caret** (para avaliação por meio da matriz de confusão e cálculo da acurácia). Essa combinação de ferramentas possibilitou a implementação consistente dos experimentos, assegurando reprodutibilidade e clareza na comparação dos diferentes algoritmos de classificação.

4 Dry Bean Dataset

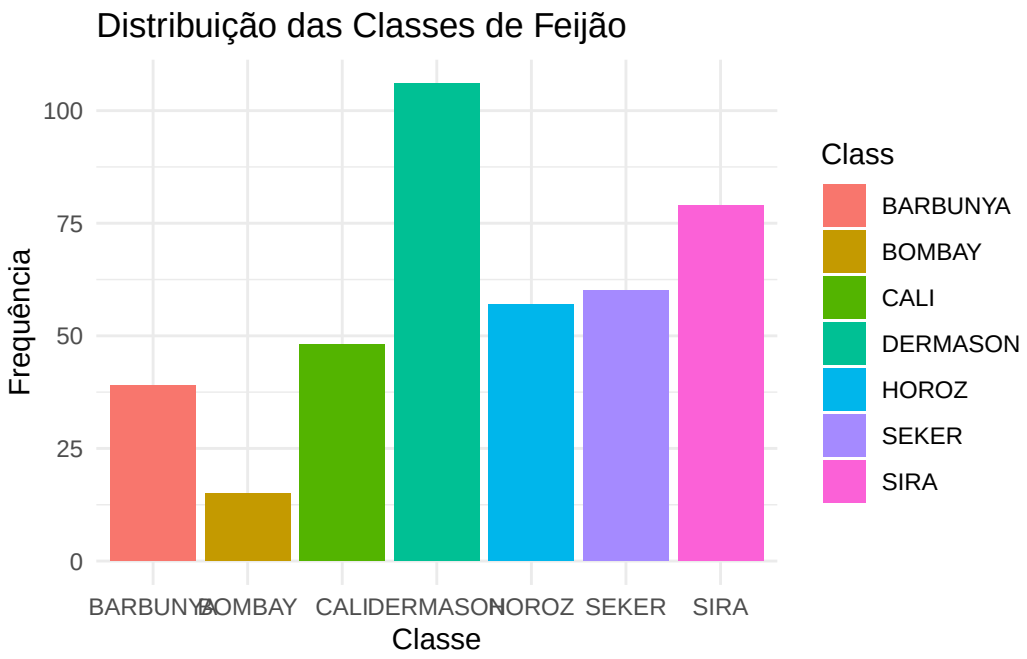
4.1 Pré-processamento

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

A amostragem foi considerada na base Dry Beans devido ao grande número de instâncias (13.611), permitindo reduzir o custo computacional durante os experimentos e possibilitando a execução de testes preliminares com subconjuntos menores de dados, garantindo maior eficiência na comparação entre classificadores. Optou-se pela amostragem estratificada, de modo a preservar as proporções originais das sete classes de feijões no subconjunto selecionado, o que se mostra essencial diante do desbalanceamento existente entre as classes, já que algumas variedades possuem muito mais instâncias que outras. Assim, essa estratégia assegura que o conjunto reduzido de dados mantenha a diversidade e representatividade necessárias para uma avaliação consistente do desempenho dos algoritmos de classificação.

```r
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()
```



4.1.2 Estratégia de Divisão da Base de Dados

Adotou-se a estratégia de divisão Holdout para a separação dos dados em treino e teste. Essa técnica consiste em particionar a base em dois subconjuntos mutuamente exclusivos: um destinado ao treinamento do modelo e outro reservado para a sua avaliação. Optou-se por utilizar aproximadamente 2/3 dos dados para o treino e 1/3 para o teste, de modo a garantir que o classificador tivesse acesso a uma quantidade suficiente de instâncias para o aprendizado, ao mesmo tempo em que fosse avaliado em exemplos não vistos durante o treinamento. Essa abordagem permite estimar o desempenho do modelo de forma direta e eficiente, sendo amplamente utilizada em cenários com bases de dados de tamanho moderado.

```
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

Na redução de dimensionalidade, observou-se que diversos atributos da base são derivados de outros já existentes, apresentando, portanto, redundância. Por exemplo, o atributo *AspectRation* é calculado a partir da razão entre *MajorAxisLength* e *MinorAxisLength*, enquanto *Roundness* e *Compactness* utilizam *Area* e *Perimeter* em suas fórmulas. Dessa forma, optou-se por manter os atributos derivados e remover os fundamentais. Essa abordagem preserva a informação essencial em métricas normalizadas e reduz a redundância entre variáveis, conforme ilustrado na tabela apresentada nesta seção.

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

4.1.4 Normalização

No processo de pré-processamento dos dados, adotou-se a normalização das variáveis preditoras, etapa fundamental para a execução de algoritmos de classificação baseados em medidas de distância e em modelos estatísticos multivariados. Foram utilizadas duas abordagens distintas conforme a necessidade de cada algoritmo: a normalização por re-escala e a padronização. A normalização por re-escala transforma os atributos para um intervalo comum, tipicamente [0, 1], evitando que variáveis com grande amplitude numérica dominem o cálculo das distâncias, como ocorre no KNN. Já a padronização converte cada atributo para média zero e desvio-padrão unitário, garantindo comparabilidade em algoritmos como a regressão logística, que assumem variáveis centradas e na mesma escala. Dessa forma, a aplicação dessas técnicas assegurou que todos os atributos contribuíssem de forma equilibrada no processo de classificação, aumentando a robustez e a interpretabilidade dos resultados obtidos.

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
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 variáveis dependentes categóricas com mais de duas classes. No presente estudo, este modelo foi aplicado para classificar amostras de acordo com suas características, conforme descrito a seguir.

1. **Preparação dos dados**

   Inicialmente, os dados foram organizados de forma a conter as variáveis independentes (atributos) e a variável resposta (classe). Uma das classes foi definida automaticamente como referência, servindo de base para a comparação com as demais.

2. **Definição inicial dos parâmetros**

   O modelo associa pesos (parâmetros) a cada variável em cada classe. Esses pesos indicam a intensidade e a direção da influência de uma característica sobre a probabilidade de pertencimento a determinada classe. No início do processo, os valores dos pesos são atribuídos de forma arbitrária, próximos de zero.

3. **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 100%.

4. **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.

5. **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.

6. **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.

7. **Modelo final e aplicação**

   Após a convergência, o modelo retém um conjunto de pesos ajustados para cada classe. Esses parâmetros permitem identificar quais variáveis exercem maior influência na diferenciação entre as classes. Em novos dados, o modelo aplica os pesos, calcula as probabilidades de cada classe e retorna como previsão a classe com maior probabilidade associada.

O funcionamento da regressão logística multinomial pode ser resumido como um processo de aprendizado iterativo: o modelo começa com estimativas iniciais, compara suas previsões com os resultados observados, ajusta seus parâmetros e repete o processo até estabilizar. Dessa forma, consegue capturar padrões nos dados e utilizá-los para realizar classificações em situações futuras.

```
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, ]


model_rl_bean <- multinom(Class ~ AspectRation + Eccentricity + EquivDiameter
 ↪  + Extent + Solidity + roundness + Compactness + ShapeFactor1 +
 ↪  ShapeFactor2 + ShapeFactor3 + ShapeFactor4, data = train_rl_bean)
```

```
# weights:  91 (72 variable)
initial  value 521.503920
iter  10 value 132.886753
iter  20 value 57.469306
iter  30 value 28.454307
iter  40 value 20.878223
iter  50 value 19.399664
iter  60 value 18.450299
iter  70 value 17.078242
iter  80 value 16.334450
iter  90 value 16.066841
iter 100 value 15.976636
final  value 15.976636
stopped after 100 iterations
```

```r
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)
# 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)
```

```
# weights:  42 (30 variable)
initial  value 521.503920
iter  10 value 130.809544
iter  20 value 58.402316
```

```
iter  30 value 40.529394
iter  40 value 39.089893
iter  50 value 38.178091
iter  60 value 37.939610
iter  70 value 37.868220
iter  80 value 37.781302
iter  90 value 37.742867
iter 100 value 37.710568
final  value 37.710568
stopped after 100 iterations
```

```r
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 a avaliação dos classificadores, foram utilizadas duas medidas principais: a matriz de confusão e a acurácia. A matriz de confusão consiste em uma tabela que organiza as previsões do modelo em relação aos valores reais, discriminando corretamente os acertos e os erros de classificação para cada classe. A acurácia, por sua vez, expressa a proporção de instâncias corretamente classificadas pelo modelo em relação ao total de instâncias avaliadas, sendo calculada pela fórmula:

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_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"
```

```
# -------------------- 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"
```

```
# -------------------- 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

Adotou-se a estratégia de divisão Holdout para a separação dos dados em treino e teste. Essa técnica consiste em particionar a base em dois subconjuntos mutuamente exclusivos: um destinado ao treinamento do modelo e outro reservado para a sua avaliação. Optou-se por utilizar aproximadamente 2/3 dos dados para o treino e 1/3 para o teste, de modo a garantir que o classificador tivesse acesso a uma quantidade suficiente de instâncias para o aprendizado, ao mesmo tempo em que fosse avaliado em exemplos não vistos durante o treinamento. Essa abordagem permite estimar o desempenho do modelo de forma direta e eficiente, sendo amplamente utilizada em cenários com bases de dados de tamanho moderado.

```
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

No processo de pré-processamento dos dados, adotou-se a normalização das variáveis preditoras, etapa fundamental para a execução de algoritmos de classificação baseados em medidas de distância e em modelos estatísticos multivariados. Foram utilizadas duas abordagens distintas conforme a necessidade de cada algoritmo: a normalização por re-escala e a padronização. A normalização por re-escala transforma os atributos para um intervalo comum, tipicamente [0, 1], evitando que variáveis com grande amplitude numérica dominem o cálculo das distâncias, como ocorre no KNN. Já a padronização converte cada atributo para média zero e desvio-padrão unitário, garantindo comparabilidade em algoritmos como a regressão logística, que assumem variáveis centradas e na mesma escala. Dessa forma, a aplicação dessas técnicas assegurou que todos os atributos contribuíssem de forma equilibrada no processo de classificação, aumentando a robustez e a interpretabilidade dos resultados obtidos.

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)
```

```
# weights:  66 (50 variable)
initial  value 250.846326
iter  10 value 103.284464
iter  20 value 79.698565
iter  30 value 68.244812
iter  40 value 64.165351
iter  50 value 62.166822
iter  60 value 61.223811
iter  70 value 60.744746
iter  80 value 60.452891
iter  90 value 60.433292
final  value 60.433227
converged
```

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

```r
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)])
# 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)
```

```
# weights:  30 (20 variable)
initial  value 250.846326
iter  10 value 139.993629
iter  20 value 124.077044
iter  30 value 122.338011
iter  40 value 122.221562
iter  50 value 122.219236
final  value 122.219223
converged
```

```r
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 a avaliação dos classificadores, foram utilizadas duas medidas principais: a matriz de confusão e a acurácia. A matriz de confusão consiste em uma tabela que organiza as previsões do modelo em relação aos valores reais, discriminando corretamente os acertos e os erros de classificação para cada classe. A acurácia, por sua vez, expressa a proporção de instâncias corretamente classificadas pelo modelo em relação ao total de instâncias avaliadas, sendo calculada pela fórmula:

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

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_glass  <- confusionMatrix(pred_p1_dt_glass,  test_dt_glass$Class)
acc_p1_dt_glass  <- mean(pred_p1_dt_glass == test_dt_glass$Class)

# 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
```

```r
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

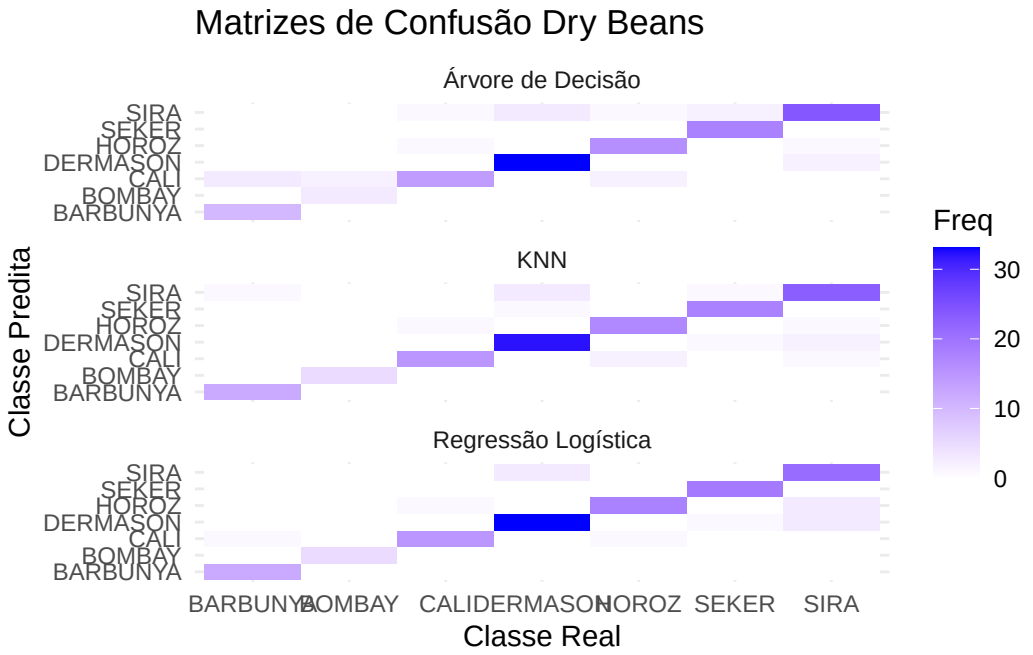
6.1 Dry Bean Dataset

```
# -------------------- 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$table)
cm_dt_bean$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"
)
```



No Dry Beans, observou-se que a Regressão Logística apresentou o melhor desempenho em termos de acurácia, seguida pelo KNN e, por último, pela Árvore de Decisão. Esse resultado pode ser explicado pela natureza dos atributos do conjunto: as variáveis correspondem a medidas geométricas e fatores de forma já derivados, muitos deles fortemente correlacionados e próximos de uma relação linear com as classes. A regressão logística, por ser um modelo paramétrico, consegue explorar bem essas relações globais quando há grande quantidade de dados, além de se beneficiar da padronização aplicada no pré-processamento, o que favorece 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. O motivo principal é que esse classificador tende a criar divisões eixos-paralelas no espaço dos atributos, o que não é suficiente para capturar fronteiras de decisão mais sutis e contínuas, como as que distinguem variedades de feijão. Embora valores menores de *cp* tenham aumentado a complexidade da árvore e melhorado levemente os resultados, o modelo ainda permaneceu inferior aos demais.

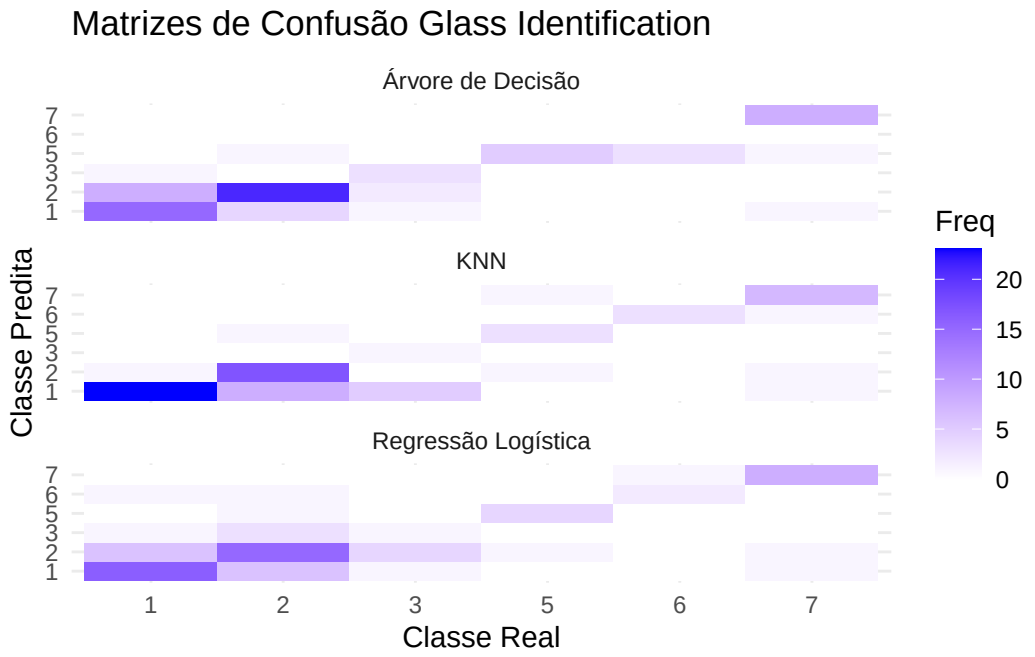
6.2 Glass Identification Dataset

```
# -------------------- 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 --------------------

# 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"
  )
```



No Glass, 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), desbalanceada e com classes muito próximas em termos de composição química.

O KNN apresentou a melhor performance porque o método lida bem com padrões locais, aproveitando a proximidade entre instâncias semelhantes mesmo em um espaço não linear. Em bases pequenas como o Glass, a ausência de suposições paramétricas faz com que o KNN consiga se adaptar melhor a regiões complexas do espaço de atributos, desde que os dados estejam normalizados.

A Árvore de Decisão obteve desempenho intermediário, superando a Regressão Logística. Isso se deve ao fato de que árvores conseguem capturar interações não lineares entre os atributos químicos, formando divisões que refletem combinações específicas de valores. Contudo, a

limitação de profundidade e a sensibilidade a ruídos ainda reduzem seu desempenho em relação ao KNN.

A Regressão Logística, por sua vez, apresentou o pior desempenho no Glass. 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 adequadamente as categorias. Além disso, o desbalanceamento das classes aumenta a instabilidade dos coeficientes estimados, mesmo após a padronização das variáveis.

6.3 Comparação Geral e Implicações

Os resultados mostram que não existe um único algoritmo superior em todos os contextos; o desempenho depende das características do conjunto de dados. 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.

Além disso, 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.

Esses achados estão em consonância com a teoria de aprendizado de máquina: modelos paramétricos tendem a ser favorecidos em conjuntos grandes e bem estruturados, enquanto modelos baseados em vizinhança ou árvores se beneficiam quando o espaço de decisão é mais irregular ou quando há poucos dados disponíveis.