Classificacao de Imagens RGB com SVM

CEAO-802 - Metodos de Analise de Dados

AUTHORS
1T Generoso
1T João Marcos
1T Vitor Cesa

1 Introducao

Este trabalho aplica tecnicas de classificacao supervisionada ao dataset RGB disponibilizado na disciplina CEAO-802 - Metodos de Analise de Dados.

O documento de datasets apresenta um problema de classificacao de cenas urbanas com imagens RGB e TIR coletadas por drone sobre Guaratingueta (SP), com 13 classes de cobertura urbana. Nesta analise sera utilizada apenas a parte RGB do dataset, armazenada localmente em `D:/Vitor/Documents/DADOS CEAO 2026/RGB`.

A abordagem adotada segue a sugestao do material da disciplina: em vez de extrair atributos estatisticos simples (media, desvio, histogramas), serao utilizadas **redes neurais convolucionais pre-treinadas** como extratores de atributos (*transfer learning*). A rede MobileNetV2, pre-treinada no ImageNet, e carregada sem a camada de classificacao final. Cada imagem e redimensionada para `224x224` pixels e processada pela rede, que produz um vetor de 1280 atributos representando caracteristicas visuais de alto nivel aprendidas em milhoes de imagens. Esses vetores sao entao usados como entrada para os classificadores SVM e Random Forest.

A aula de Support Vector Machine (SVM) apresentou o metodo como um classificador baseado na busca de um hiperplano de separacao entre classes. A ideia central e encontrar uma fronteira que maximize a margem entre os grupos. Quando os dados nao sao linearmente separaveis, o SVM pode usar variaveis de folga, controladas pelo parametro `cost`, e tambem kernels, como o kernel radial ou RBF, para criar fronteiras de decisao nao lineares por meio do chamado *kernel trick*.

Como a pasta RGB possui grande volume de dados, sera usada uma amostra estratificada e balanceada de 50 imagens por classe. Essa escolha torna o processamento viavel em um computador comum, preservando a comparacao entre as classes.

Neste relatorio, serao avaliados:

1. SVM com kernel linear;
2. SVM com kernel radial;
3. SVM radial com ajuste simples de hiperparametros;
4. PCA + SVM radial;
5. Random Forest, como modelo de comparacao.

2 Preparacao do ambiente

```
# ============================================================
# 1. Instalacao e carregamento dos pacotes
```

```r
# ============================================================

pacotes <- c(
  "tidyverse",
  "keras3",
  "caret",
  "e1071",
  "randomForest",
  "knitr"
)

pacotes_faltando <- setdiff(pacotes, rownames(installed.packages()))

if (length(pacotes_faltando) > 0) {
  install.packages(pacotes_faltando)
}

library(tidyverse)
library(keras3)
library(caret)
library(e1071)
library(randomForest)
library(knitr)

backend_ativo <- tryCatch(config_backend(), error = function(e) NA_character_)
cat("Backend keras3:", backend_ativo, "\n")
```

```
Backend keras3: tensorflow
```

3 Caminho do dataset

Em R, e mais seguro usar barras `/` no caminho, mesmo no Windows.

```r
# ============================================================
# 2. Caminho do dataset RGB
# ============================================================

caminho_rgb <- "D:/Vitor/Documents/DADOS CEAO 2026/RGB"

if (!dir.exists(caminho_rgb)) {
  stop("A pasta do dataset RGB nao foi encontrada. Verifique o caminho informado em caminho_rgb
}

caminho_rgb
```

```
[1] "D:/Vitor/Documents/DADOS CEAO 2026/RGB"
```

4 Leitura dos arquivos de imagem

O codigo procura imagens dentro da pasta RGB. Se o dataset estiver organizado em subpastas, o nome da subpasta sera usado como classe. No dataset usado neste trabalho, os arquivos TIFF estao diretamente na pasta `RGB`, entao a classe e inferida pelo prefixo do nome do arquivo, como `C1`, `C2`, ..., `C13`.

```
# ============================================================
# 3. Listar imagens
# ============================================================

extensoes_imagem <- "\\.(jpg|jpeg|png|bmp|tif|tiff)$"

arquivos <- list.files(
  path = caminho_rgb,
  pattern = extensoes_imagem,
  recursive = TRUE,
  full.names = TRUE,
  ignore.case = TRUE
)

if (length(arquivos) == 0) {
  stop("Nenhuma imagem foi encontrada na pasta RGB.")
}

length(arquivos)
```

```
[1] 1248
```

```
# ============================================================
# 4. Criar metadados
# ============================================================

obter_classe <- function(arquivo, raiz) {
  arquivo_norm <- normalizePath(arquivo, winslash = "/", mustWork = FALSE)
  raiz_norm <- normalizePath(raiz, winslash = "/", mustWork = FALSE)

  caminho_relativo <- stringr::str_remove(
    arquivo_norm,
    paste0("^", stringr::fixed(raiz_norm), "/?")
  )

  partes <- strsplit(caminho_relativo, "/", fixed = TRUE)[[1]]

  if (length(partes) >= 2) {
    return(partes[1])
  }

  nome_sem_extensao <- tools::file_path_sans_ext(basename(arquivo))
  classe_inferida <- stringr::str_extract(nome_sem_extensao, "^C\\d+")

  if (is.na(classe_inferida)) {
    classe_inferida <- stringr::str_extract(nome_sem_extensao, "^[^_-]+")
  }

  classe_inferida
```

```
}

metadados_completo <- tibble(
  arquivo = arquivos,
  classe = map_chr(arquivos, obter_classe, raiz = caminho_rgb)
) %>%
  mutate(
    classe = factor(classe, levels = paste0("C", 1:13)),
    tamanho_mb = file.info(arquivo)$size / 1024^2
  ) %>%
  arrange(classe, arquivo)

resumo_dataset <- metadados_completo %>%
  group_by(classe) %>%
  summarise(
    n_imagens = n(),
    tamanho_total_mb = sum(tamanho_mb, na.rm = TRUE),
    tamanho_medio_mb = mean(tamanho_mb, na.rm = TRUE),
    .groups = "drop"
  )

resumo_dataset %>%
  mutate(across(where(is.numeric), ~ round(.x, 2))) %>%
  kable()
```

classe	n_imagens	tamanho_total_mb	tamanho_medio_mb
C1	96	3388.31	35.29
C2	96	3388.31	35.29
C3	96	3388.31	35.29
C4	96	3388.31	35.29
C5	96	3388.31	35.29
C6	96	3388.31	35.29
C7	96	3388.31	35.29
C8	96	3388.31	35.29
C9	96	3388.31	35.29
C10	96	3388.31	35.29
C11	96	3388.31	35.29
C12	96	3388.31	35.29
C13	96	3388.31	35.29

5 Amostra balanceada

A amostragem estratificada foi usada para manter a mesma quantidade de imagens em cada classe. Isso evita que classes maiores dominem o treinamento e reduz o tempo de processamento.

```r
# ============================================================
# 5. Criar amostra balanceada por classe
# ============================================================

set.seed(123)

max_imagens_por_classe <- 50

metadados <- metadados_completo %>%
  group_by(classe) %>%
  group_modify(~ {
    qtd <- min(max_imagens_por_classe, nrow(.x))
    slice_sample(.x, n = qtd)
  }) %>%
  ungroup() %>%
  mutate(classe = factor(classe, levels = paste0("C", 1:13)))

cat("Total de imagens no dataset completo:", nrow(metadados_completo), "\n")
```

```
Total de imagens no dataset completo: 1248
```

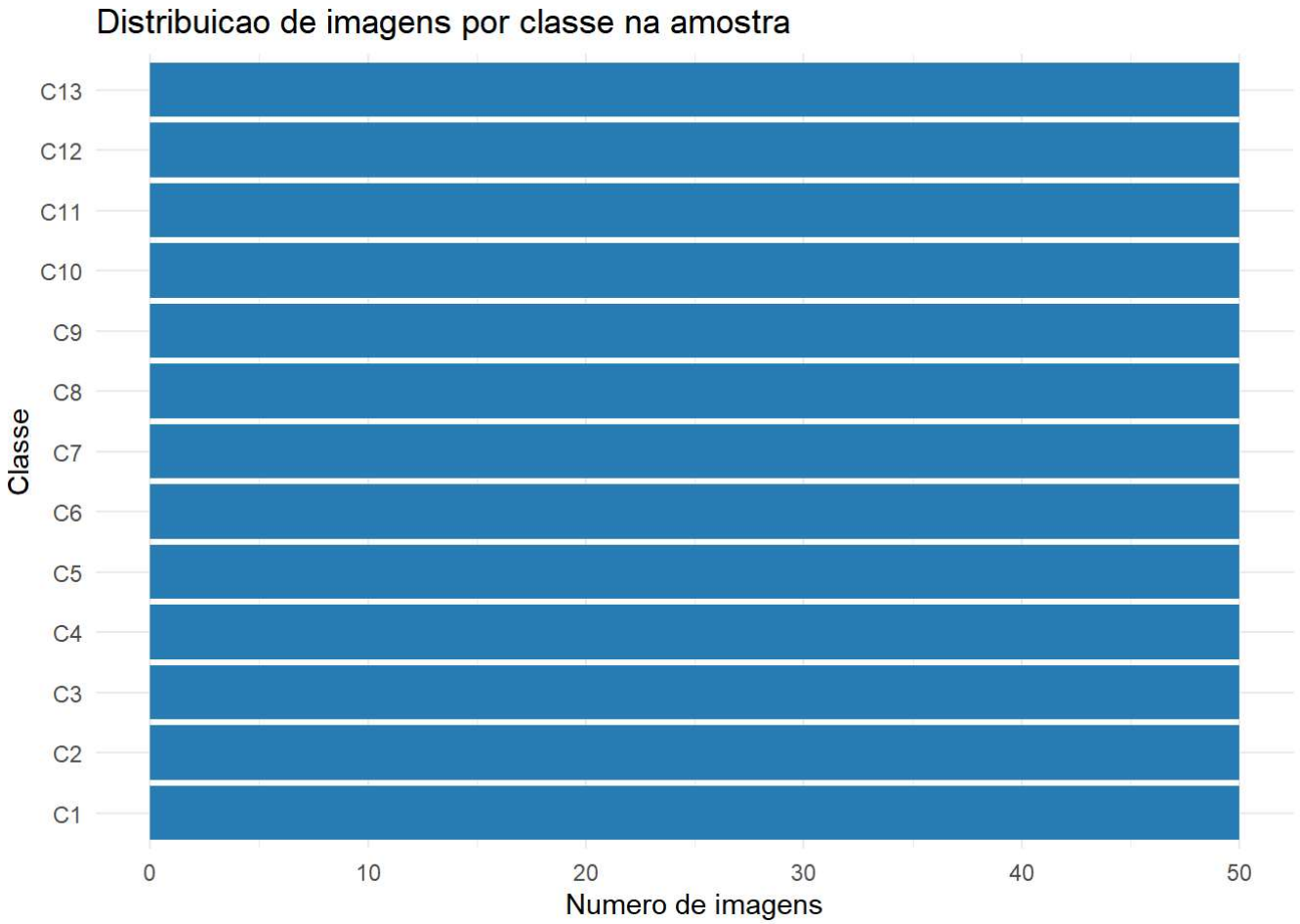
```r
cat("Total de imagens na amostra:", nrow(metadados), "\n")
```

```
Total de imagens na amostra: 650
```

```r
metadados %>%
  count(classe, name = "n_imagens_amostra") %>%
  arrange(classe) %>%
  kable()
```

classe	n_imagens_amostra
C1	50
C2	50
C3	50
C4	50
C5	50
C6	50
C7	50
C8	50
C9	50
C10	50
C11	50
C12	50
C13	50

```r
ggplot(metadados, aes(x = classe)) +
  geom_bar(fill = "#2C7FB8") +
  coord_flip() +
  labs(
    title = "Distribuicao de imagens por classe na amostra",
    x = "Classe",
    y = "Numero de imagens"
  ) +
  theme_minimal()
```



6 Modelo pre-treinado (Transfer Learning)

A estrategia de *transfer learning* consiste em usar uma rede neural convolucional (CNN) ja treinada em um grande dataset — neste caso, o ImageNet, com mais de 1 milhao de imagens e 1000 classes — e aproveitá-la como extrator de atributos.

A rede escolhida e a **MobileNetV2**, uma arquitetura compacta e eficiente desenvolvida pelo Google. Ela e carregada sem a camada de classificacao final (`include_top = FALSE`) e com pooling global medio (`pooling = "avg"`), o que produz um vetor de **1280 atributos** por imagem. Esses atributos representam caracteristicas visuais de alto nivel (bordas, texturas, formas, padroes complexos) que a rede aprendeu ao longo de seu treinamento no ImageNet.

```r
# ============================================================
# 6. Carregar modelo pre-treinado MobileNetV2
```

```
# ============================================================

dir.create("outputs", showWarnings = FALSE)

modelo_base <- application_mobilenet_v2(
  include_top = FALSE,
  weights     = "imagenet",
  pooling     = "avg",
  input_shape = c(224L, 224L, 3L)
)

cat("Modelo carregado:", modelo_base$name, "\n")
```

```
Modelo carregado: mobilenetv2_1.00_224
```

```
cat("Dimensao do vetor de features:", modelo_base$output_shape[[2]], "\n")
```

```
Dimensao do vetor de features: 1280
```

7 Extracao de atributos por Transfer Learning

Cada imagem e redimensionada para `224x224` pixels (formato padrao do MobileNetV2), normalizada com a funcao de preprocessamento especifica da rede e entao passada pela CNN. O vetor de saida e armazenado como atributos da imagem.

O processamento e feito em lotes (*batch*) para maior eficiencia, e o resultado e armazenado em cache para evitar reprocessamento em execucoes posteriores.

```
# ============================================================
# 7. Extracao de features via CNN pre-treinada (com cache)
# ============================================================

arquivo_cache_features <- "outputs/features_rgb_mobilenetv2_50.rds"
arquivo_cache_parcial  <- "outputs/features_rgb_mobilenetv2_50_parcial.rds"

tamanho_imagem <- 224L
tamanho_lote   <- 16L

extrair_features_batch <- function(arquivos, modelo, tamanho = 224L, lote = 16L) {
  n <- length(arquivos)
  resultados <- matrix(NA_real_, nrow = n, ncol = as.integer(modelo$output_shape[[2]]))

  for (inicio in seq(1, n, by = lote)) {
    fim <- min(inicio + lote - 1L, n)
    bloco <- arquivos[inicio:fim]
    tam_bloco <- length(bloco)

    batch_array <- array(0, dim = c(tam_bloco, tamanho, tamanho, 3L))

    for (j in seq_len(tam_bloco)) {
      img <- image_load(bloco[j], target_size = c(tamanho, tamanho))
      batch_array[j, , , ] <- image_to_array(img)
```

```
    }

    batch_prep <- (batch_array / 127.5) - 1.0
    features   <- predict(modelo, batch_prep, verbose = 0L)
    resultados[inicio:fim, ] <- features
  }

  resultados
}

if (file.exists(arquivo_cache_features)) {
  dados <- readRDS(arquivo_cache_features)
  cat("Features carregadas do cache:", arquivo_cache_features, "\n")
} else {
  cat("Extraindo features com MobileNetV2. Isso pode levar alguns minutos...\n")

  erros_idx <- integer(0)

  if (file.exists(arquivo_cache_parcial)) {
    features_matrix <- readRDS(arquivo_cache_parcial)
    linhas_prontas  <- which(complete.cases(features_matrix))
    cat("Cache parcial encontrado:", length(linhas_prontas), "imagens ja processadas.\n")
  } else {
    n_features      <- as.integer(modelo_base$output_shape[[2]])
    features_matrix <- matrix(NA_real_, nrow = nrow(metadados), ncol = n_features)
    linhas_prontas  <- integer(0)
  }

  linhas_pendentes <- setdiff(seq_len(nrow(metadados)), linhas_prontas)

  if (length(linhas_pendentes) > 0) {
    for (inicio in seq(1, length(linhas_pendentes), by = tamanho_lote)) {
      fim   <- min(inicio + tamanho_lote - 1L, length(linhas_pendentes))
      idxs  <- linhas_pendentes[inicio:fim]
      arquivos_bloco <- metadados$arquivo[idxs]

      resultado <- tryCatch(
        extrair_features_batch(arquivos_bloco, modelo_base, tamanho_imagem, length(idxs)),
        error = function(e) {
          message("Erro no lote ", inicio, "-", fim, ": ", conditionMessage(e))
          erros_idx <<- c(erros_idx, idxs)
          NULL
        }
      )

      if (!is.null(resultado)) {
        features_matrix[idxs, ] <- resultado
      }

      cat(sprintf("  Processadas: %d / %d\r", min(fim, length(linhas_pendentes)), length(linha

      if (fim %% (tamanho_lote * 5L) == 0L) {
        saveRDS(features_matrix, arquivo_cache_parcial)
      }
```

```
    }
    cat("\n")
    saveRDS(features_matrix, arquivo_cache_parcial)
  }

  imagens_ok <- complete.cases(features_matrix)
  n_features <- ncol(features_matrix)
  nomes_feat <- paste0("feat_", stringr::str_pad(seq_len(n_features), 4, pad = "0"))
  colnames(features_matrix) <- nomes_feat

  dados <- bind_cols(
    metadados[imagens_ok, ],
    as_tibble(features_matrix[imagens_ok, ])
  )

  if (length(erros_idx) > 0) {
    erros_df <- metadados[erros_idx, ] %>% mutate(erro = "falha na extracao CNN")
    write_csv(erros_df, "outputs/erros_processamento_rgb.csv")
    cat("Imagens com erro:", length(erros_idx), "(ver outputs/erros_processamento_rgb.csv)\n")
  }

  saveRDS(dados, arquivo_cache_features)
  if (file.exists(arquivo_cache_parcial)) file.remove(arquivo_cache_parcial)

  cat("Features salvas em:", arquivo_cache_features, "\n")
}
```

```
Features carregadas do cache: outputs/features_rgb_mobilenetv2_50.rds
```

```
cat("Imagens na base final:", nrow(dados), "\n")
```

```
Imagens na base final: 650
```

```
cat("Numero de atributos extraidos (MobileNetV2):", ncol(dados) - 3L, "\n")
```

```
Numero de atributos extraidos (MobileNetV2): 1280
```

```
dados %>%
  count(classe, name = "n_imagens_processadas") %>%
  arrange(classe) %>%
  kable()
```

classe	n_imagens_processadas
C1	50
C2	50
C3	50
C4	50
C5	50

classe	n_imagens_processadas
C6	50
C7	50
C8	50
C9	50
C10	50
C11	50
C12	50
C13	50

```
dados %>%
  select(arquivo, classe, feat_0001, feat_0002, feat_0003, feat_0004, feat_0005) %>%
  head(10) %>%
  mutate(across(starts_with("feat_"), ~ round(.x, 4))) %>%
  kable()
```

arquivo	classe	feat_0001	feat_0002	feat_0003	feat_0004	feat_0005
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_09_T_03_RGB.tiff	C1	0.4923	0.0000	0.0283	0.8341	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2024_03_T_03_RGB.tiff	C1	0.0000	0.0000	0.0000	0.1868	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_12_M_03_RGB.tiff	C1	0.1640	0.0000	0.0000	0.0000	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_07_T_02_RGB.tiff	C1	0.0364	0.0782	0.0000	0.2644	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2024_02_M_03_RGB.tiff	C1	0.0000	0.0045	0.0000	0.0260	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_11_M_02_RGB.tiff	C1	0.0000	0.0000	0.0472	0.0027	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_12_M_02_RGB.tiff	C1	0.2422	0.0000	0.0045	0.0000	0.1251
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_11_M_03_RGB.tiff	C1	0.0865	0.0000	0.0076	1.1901	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2024_05_T_01_RGB.tiff	C1	0.6739	0.0044	0.0148	0.0000	0.0000
D:/Vitor/Documents/DADOS CEAO 2026/RGB/C1_2023_09_M_01_RGB.tiff	C1	0.3725	0.1405	0.0000	0.0020	0.0000

8 Separacao entre treino e teste

A separacao treino/teste sera feita de forma estratificada, mantendo a proporcao das classes sempre que possivel.

```
# ============================================================
# 8. Separacao treino/teste
# ============================================================

set.seed(123)

idx_treino <- createDataPartition(
  y = dados$classe,
  p = 0.70,
  list = FALSE
)

treino <- dados[idx_treino, ]
teste <- dados[-idx_treino, ]

cat("Imagens no treino:", nrow(treino), "\n")
```

```
Imagens no treino: 455
```

```
cat("Imagens no teste:", nrow(teste), "\n")
```

```
Imagens no teste: 195
```

```
treino %>%
  count(classe, name = "treino") %>%
  full_join(
    teste %>% count(classe, name = "teste"),
    by = "classe"
  ) %>%
  arrange(classe) %>%
  kable()
```

classe	treino	teste
C1	35	15
C2	35	15
C3	35	15
C4	35	15
C5	35	15
C6	35	15
C7	35	15
C8	35	15
C9	35	15
C10	35	15
C11	35	15
C12	35	15
C13	35	15

9 Pre-processamento

Como o SVM e sensivel a escala das variaveis, os atributos serao centralizados e padronizados. Tambem serao removidas variaveis com variancia quase nula.

```
# ============================================================
# 9. Preparar matrizes X e vetor y
# ============================================================

colunas_nao_preditoras <- c("arquivo", "classe", "tamanho_mb")

x_treino <- treino %>%
  select(-all_of(colunas_nao_preditoras))

x_teste <- teste %>%
  select(-all_of(colunas_nao_preditoras))

y_treino <- droplevels(treino$classe)
y_teste <- factor(teste$classe, levels = levels(y_treino))

variaveis_nzv <- nearZeroVar(x_treino)

if (length(variaveis_nzv) > 0) {
  x_treino <- x_treino[, -variaveis_nzv, drop = FALSE]
  x_teste <- x_teste[, colnames(x_treino), drop = FALSE]
}

preproc <- preProcess(
  x_treino,
  method = c("center", "scale")
)

x_treino_norm <- predict(preproc, x_treino)
x_teste_norm <- predict(preproc, x_teste)

cat("Preditores usados nos modelos:", ncol(x_treino_norm), "\n")
```

```
Preditores usados nos modelos: 1265
```

10 Modelo 1: SVM linear

O SVM linear tenta separar as classes por hiperplanos lineares. E o modelo mais simples e serve como referencia inicial.

```
# ============================================================
# 10. SVM linear
# ============================================================

set.seed(123)

modelo_svm_linear <- svm(
```

```r
  x = x_treino_norm,
  y = y_treino,
  kernel = "linear",
  cost = 1,
  scale = FALSE
)

pred_svm_linear <- predict(modelo_svm_linear, x_teste_norm)

cm_svm_linear <- confusionMatrix(pred_svm_linear, y_teste)

cm_svm_linear
```

```
Confusion Matrix and Statistics

          Reference
Prediction C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13
       C1   4  3  0  1  0  0  0  0  1   1   0   0   0
       C2   4  6  0  0  0  2  0  0  0   0   0   0   1
       C3   0  0  7  6  1  0  0  2  1   0   0   0   0
       C4   3  2  0  6  0  0  2  1  0   0   0   1   4
       C5   0  0  2  0  9  0  0  0  0   0   0   0   1
       C6   2  3  0  0  1  8  1  4  1   3   2   3   0
       C7   0  1  0  0  1  0  6  0  2   0   0   6   2
       C8   0  0  1  0  0  1  1  8  0   0   1   1   0
       C9   1  0  1  0  0  0  0  0  9   1   0   0   1
       C10  0  0  1  0  0  0  0  0  1   5   8   0   0
       C11  1  0  0  1  0  1  0  0  0   5   4   1   0
       C12  0  0  0  0  1  3  5  0  0   0   0   2   2
       C13  0  0  3  1  2  0  0  0  0   0   0   1   4

Overall Statistics

               Accuracy : 0.4
                 95% CI : (0.3307, 0.4724)
    No Information Rate : 0.0769
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.35

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: C1 Class: C2 Class: C3 Class: C4 Class: C5
Sensitivity            0.26667   0.40000   0.46667   0.40000   0.60000
Specificity            0.96667   0.96111   0.94444   0.92778   0.98333
Pos Pred Value         0.40000   0.46154   0.41176   0.31579   0.75000
Neg Pred Value         0.94054   0.95055   0.95506   0.94886   0.96721
Prevalence             0.07692   0.07692   0.07692   0.07692   0.07692
Detection Rate         0.02051   0.03077   0.03590   0.03077   0.04615
Detection Prevalence   0.05128   0.06667   0.08718   0.09744   0.06154
Balanced Accuracy      0.61667   0.68056   0.70556   0.66389   0.79167
                     Class: C6 Class: C7 Class: C8 Class: C9 Class: C10
```

```
Sensitivity            0.53333   0.40000   0.53333   0.60000    0.33333
Specificity            0.88889   0.93333   0.97222   0.97778    0.94444
Pos Pred Value         0.28571   0.33333   0.61538   0.69231    0.33333
Neg Pred Value         0.95808   0.94915   0.96154   0.96703    0.94444
Prevalence             0.07692   0.07692   0.07692   0.07692    0.07692
Detection Rate         0.04103   0.03077   0.04103   0.04615    0.02564
Detection Prevalence   0.14359   0.09231   0.06667   0.06667    0.07692
Balanced Accuracy      0.71111   0.66667   0.75278   0.78889    0.63889
                     Class: C11 Class: C12 Class: C13
Sensitivity             0.26667    0.13333    0.26667
Specificity             0.95000    0.93889    0.96111
Pos Pred Value          0.30769    0.15385    0.36364
Neg Pred Value          0.93956    0.92857    0.94022
Prevalence              0.07692    0.07692    0.07692
Detection Rate          0.02051    0.01026    0.02051
Detection Prevalence    0.06667    0.06667    0.05641
Balanced Accuracy       0.60833    0.53611    0.61389
```

11 Modelo 2: SVM radial

O kernel radial, tambem chamado de RBF, permite construir fronteiras de decisao nao lineares. Nesta primeira versao, serao usados valores fixos para `cost` e `gamma`.

```
# ============================================================
# 11. SVM radial com parametros fixos
# ============================================================

set.seed(123)

modelo_svm_radial <- svm(
  x = x_treino_norm,
  y = y_treino,
  kernel = "radial",
  cost = 10,
  gamma = 0.01,
  scale = FALSE
)

pred_svm_radial <- predict(modelo_svm_radial, x_teste_norm)

cm_svm_radial <- confusionMatrix(pred_svm_radial, y_teste)

cm_svm_radial
```

```
Confusion Matrix and Statistics

          Reference
Prediction C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13
       C1   2  1  0  0  0  0  0  0  0   0   0   0   0
       C2   2  0  0  0  0  0  0  1  0   0   0   0   1
       C3   0  0  3  2  0  0  0  1  1   0   0   0   0
       C4   0  0  0  2  0  0  1  0  0   0   0   0   3
```

```
       C5   0  0  1  0  4  0  0  0  0   0   0   0   2
       C6   0  3  0  0  0  2  0  5  0   2   0   2   0
       C7   0  0  0  0  0  0  0  0  1   0   0   3   1
       C8   0  0  0  0  0  2  1  1  1   0   0   1   0
       C9   1  0  1  0  0  0  0  0  2   0   0   0   0
       C10  0  0  0  0  0  1  0  0  1   2   9   0   0
       C11  1  0  0  0  0  0  0  1  0   3   0   0   0
       C12  0  0  0  0  0  1  4  0  1   1   0   2   0
       C13  9 11 10 11 11  9  9  6  8   7   6   7   8

Overall Statistics

               Accuracy : 0.1436
                 95% CI : (0.0976, 0.2008)
    No Information Rate : 0.0769
    P-Value [Acc > NIR] : 0.001068

                  Kappa : 0.0722

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: C1 Class: C2 Class: C3 Class: C4 Class: C5
Sensitivity            0.13333   0.00000   0.20000   0.13333   0.26667
Specificity            0.99444   0.97778   0.97778   0.97778   0.98333
Pos Pred Value         0.66667   0.00000   0.42857   0.33333   0.57143
Neg Pred Value         0.93229   0.92147   0.93617   0.93122   0.94149
Prevalence             0.07692   0.07692   0.07692   0.07692   0.07692
Detection Rate         0.01026   0.00000   0.01538   0.01026   0.02051
Detection Prevalence   0.01538   0.02051   0.03590   0.03077   0.03590
Balanced Accuracy      0.56389   0.48889   0.58889   0.55556   0.62500
                     Class: C6 Class: C7 Class: C8 Class: C9 Class: C10
Sensitivity            0.13333   0.00000  0.066667   0.13333    0.13333
Specificity            0.93333   0.97222  0.972222   0.98889    0.93889
Pos Pred Value         0.14286   0.00000  0.166667   0.50000    0.15385
Neg Pred Value         0.92818   0.92105  0.925926   0.93194    0.92857
Prevalence             0.07692   0.07692  0.076923   0.07692    0.07692
Detection Rate         0.01026   0.00000  0.005128   0.01026    0.01026
Detection Prevalence   0.07179   0.02564  0.030769   0.02051    0.06667
Balanced Accuracy      0.53333   0.48611  0.519444   0.56111    0.53611
                     Class: C11 Class: C12 Class: C13
Sensitivity             0.00000    0.13333    0.53333
Specificity             0.97222    0.96111    0.42222
Pos Pred Value          0.00000    0.22222    0.07143
Neg Pred Value          0.92105    0.93011    0.91566
Prevalence              0.07692    0.07692    0.07692
Detection Rate          0.00000    0.01026    0.04103
Detection Prevalence    0.02564    0.04615    0.57436
Balanced Accuracy       0.48611    0.54722    0.47778
```

12 Modelo 3: Ajuste de hiperparametros do SVM radial

O parametro `cost` controla o quanto o modelo penaliza erros de classificacao. O parametro `gamma` influencia o alcance do kernel radial. Valores maiores podem gerar fronteiras mais complexas.

```
# ============================================================
# 12. Ajuste simples de hiperparametros do SVM radial
# ============================================================

set.seed(123)

k_cv <- min(5, as.integer(min(table(y_treino))))

if (k_cv >= 2) {
  ajuste_svm_radial <- tune.svm(
    x = x_treino_norm,
    y = y_treino,
    kernel = "radial",
    cost = c(0.1, 1, 10, 100),
    gamma = c(0.001, 0.01, 0.05, 0.1),
    tunecontrol = tune.control(cross = k_cv)
  )

  modelo_svm_radial_ajustado <- ajuste_svm_radial$best.model

  pred_svm_radial_ajustado <- predict(
    modelo_svm_radial_ajustado,
    x_teste_norm
  )

  cm_svm_radial_ajustado <- confusionMatrix(
    pred_svm_radial_ajustado,
    y_teste
  )

  ajuste_svm_radial$best.parameters %>%
    kable()
} else {
  warning("Poucas amostras por classe para validacao cruzada. O ajuste automatico sera ignorad

  ajuste_svm_radial <- NULL
  modelo_svm_radial_ajustado <- modelo_svm_radial
  pred_svm_radial_ajustado <- pred_svm_radial
  cm_svm_radial_ajustado <- cm_svm_radial
}
```

	gamma	cost
5	0.001	1

```
cm_svm_radial_ajustado
```

Confusion Matrix and Statistics

Prediction	Reference C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
C1	4	3	0	1	0	0	0	0	0	1	0	0	0
C2	3	3	0	0	0	0	0	0	0	0	0	0	1
C3	0	0	2	2	0	0	0	0	1	0	0	0	0
C4	4	1	0	6	0	0	2	1	0	0	0	0	4
C5	0	0	3	0	9	0	0	0	0	0	0	0	1
C6	0	6	0	0	1	8	1	7	1	2	1	4	0
C7	0	1	0	1	0	2	6	0	0	0	0	6	2
C8	0	0	2	0	0	0	0	6	2	0	0	0	0
C9	2	0	1	0	2	1	0	0	8	1	0	0	2
C10	0	0	1	0	0	1	0	0	1	3	7	0	0
C11	1	0	1	1	1	2	0	1	1	7	6	0	0
C12	0	0	0	0	0	1	5	0	0	0	0	4	1
C13	1	1	5	4	2	0	1	0	1	1	1	1	4

Overall Statistics

Accuracy	0.3538
95% CI	(0.2869, 0.4254)
No Information Rate	0.0769
P-Value [Acc > NIR]	< 2.2e-16
Kappa	0.3
Mcnemar's Test P-Value	NA

Statistics by Class:

	Class: C1	Class: C2	Class: C3	Class: C4	Class: C5
Sensitivity	0.26667	0.20000	0.13333	0.40000	0.60000
Specificity	0.97222	0.97778	0.98333	0.93333	0.97778
Pos Pred Value	0.44444	0.42857	0.40000	0.33333	0.69231
Neg Pred Value	0.94086	0.93617	0.93158	0.94915	0.96703
Prevalence	0.07692	0.07692	0.07692	0.07692	0.07692
Detection Rate	0.02051	0.01538	0.01026	0.03077	0.04615
Detection Prevalence	0.04615	0.03590	0.02564	0.09231	0.06667
Balanced Accuracy	0.61944	0.58889	0.55833	0.66667	0.78889

	Class: C6	Class: C7	Class: C8	Class: C9	Class: C10
Sensitivity	0.53333	0.40000	0.40000	0.53333	0.20000
Specificity	0.87222	0.93333	0.97778	0.95000	0.94444
Pos Pred Value	0.25806	0.33333	0.60000	0.47059	0.23077
Neg Pred Value	0.95732	0.94915	0.95135	0.96067	0.93407
Prevalence	0.07692	0.07692	0.07692	0.07692	0.07692
Detection Rate	0.04103	0.03077	0.03077	0.04103	0.01538
Detection Prevalence	0.15897	0.09231	0.05128	0.08718	0.06667
Balanced Accuracy	0.70278	0.66667	0.68889	0.74167	0.57222

	Class: C11	Class: C12	Class: C13
Sensitivity	0.40000	0.26667	0.26667
Specificity	0.91667	0.96111	0.90000
Pos Pred Value	0.28571	0.36364	0.18182
Neg Pred Value	0.94828	0.94022	0.93642

Prevalence	0.07692	0.07692	0.07692
Detection Rate	0.03077	0.02051	0.02051
Detection Prevalence	0.10769	0.05641	0.11282
Balanced Accuracy	0.65833	0.61389	0.58333

13 Modelo 4: PCA + SVM radial

O PCA transforma os atributos originais em componentes principais. Aqui, serao mantidos componentes suficientes para explicar pelo menos 95% da variancia, com limite maximo de 30 componentes.

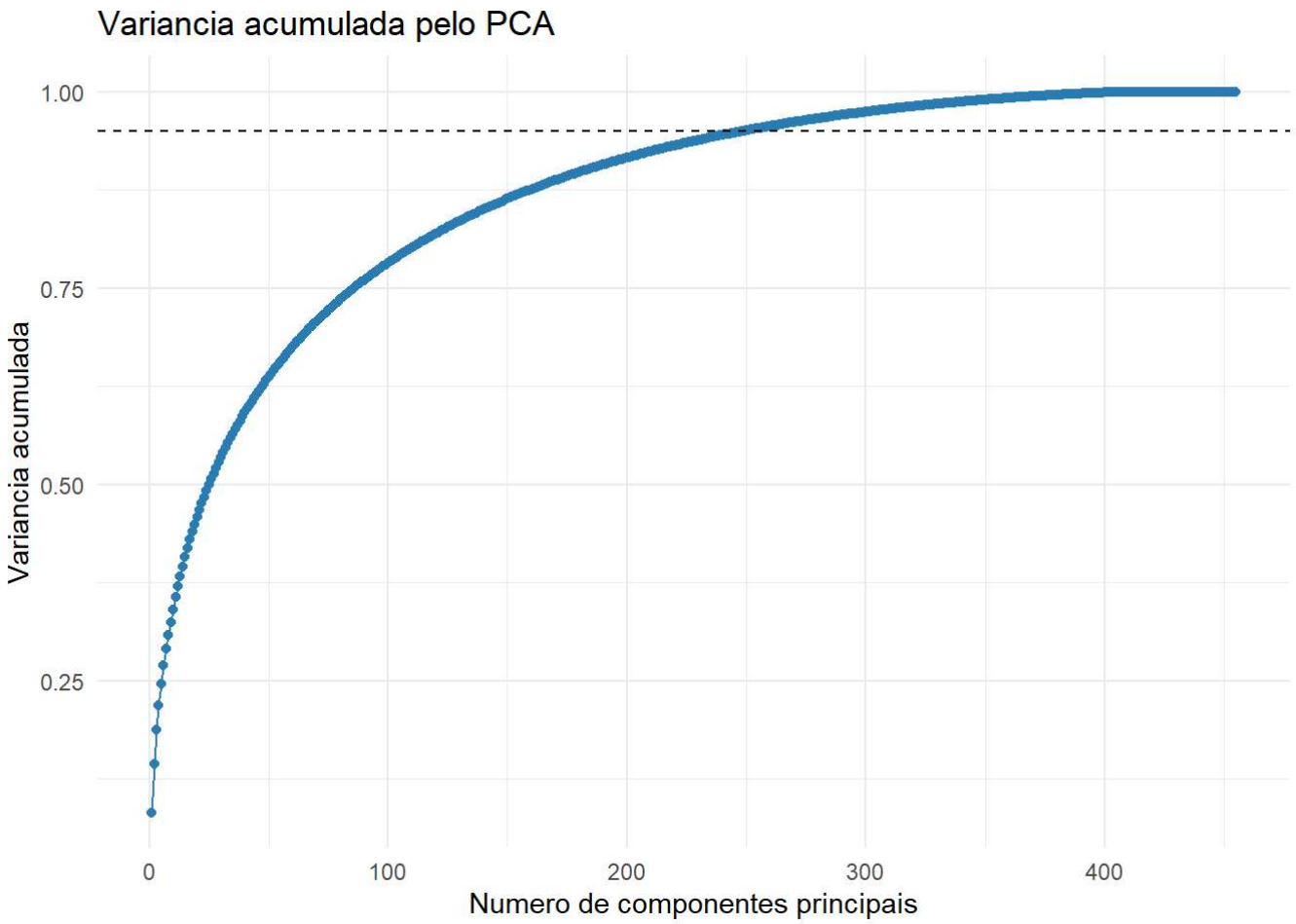
```
# ============================================================
# 13. PCA
# ============================================================

pca <- prcomp(
  x_treino_norm,
  center = FALSE,
  scale. = FALSE
)

variancia <- pca$sdev^2
variancia_exp <- variancia / sum(variancia)
variancia_acum <- cumsum(variancia_exp)

n_comp_95 <- which(variancia_acum >= 0.95)[1]
n_comp <- min(n_comp_95, 30, ncol(x_treino_norm))

cat("Componentes necessarios para 95% da variancia:", n_comp_95, "\n")
```

```
Componentes necessarios para 95% da variancia: 249
```

```
cat("Componentes usados no modelo:", n_comp, "\n")
```

```
Componentes usados no modelo: 30
```

```
tibble(
  componente = seq_along(variancia_acum),
  variancia_acumulada = variancia_acum
) %>%
  ggplot(aes(x = componente, y = variancia_acumulada)) +
  geom_line(color = "#2C7FB8") +
  geom_point(color = "#2C7FB8") +
  geom_hline(yintercept = 0.95, linetype = "dashed") +
  labs(
    title = "Variancia acumulada pelo PCA",
    x = "Numero de componentes principais",
    y = "Variancia acumulada"
  ) +
  theme_minimal()
```



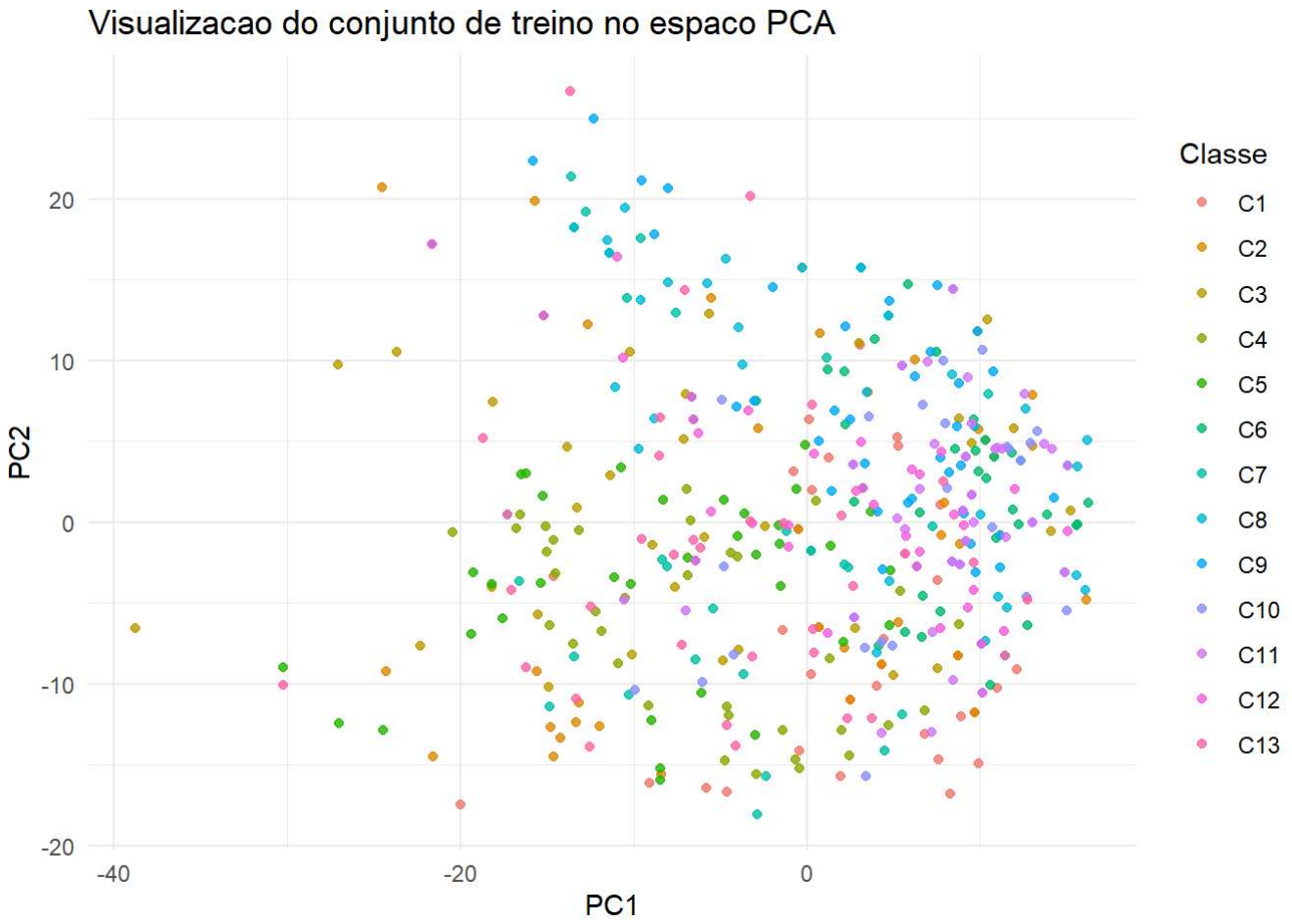

```r
x_treino_pca <- as.data.frame(pca$x[, 1:n_comp, drop = FALSE])

x_teste_pca <- as.data.frame(
  predict(pca, newdata = x_teste_norm)[, 1:n_comp, drop = FALSE]
)

set.seed(123)

modelo_svm_pca <- svm(
  x = x_treino_pca,
  y = y_treino,
  kernel = "radial",
  cost = 10,
  gamma = 0.01,
  scale = FALSE
)

pred_svm_pca <- predict(modelo_svm_pca, x_teste_pca)

cm_svm_pca <- confusionMatrix(pred_svm_pca, y_teste)

cm_svm_pca
```

```
Confusion Matrix and Statistics

          Reference
```

```
Prediction C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13
       C1   6  2  0  0  1  0  0  0  0   1   0   0   0
       C2   3  5  0  0  0  0  0  0  0   1   1   0   1
       C3   0  0  7  2  1  0  0  0  1   0   0   0   0
       C4   1  0  0  6  0  0  1  1  0   0   0   0   4
       C5   0  0  2  0  8  0  0  0  0   0   0   0   1
       C6   1  3  0  0  0  3  1  6  0   3   2   3   0
       C7   0  0  0  1  1  3  7  0  1   0   0   6   3
       C8   0  2  1  0  0  4  0  7  2   0   0   0   0
       C9   1  0  1  0  2  1  0  0  6   1   0   0   2
       C10  0  0  1  0  0  1  0  0  0   5   9   0   0
       C11  1  0  0  0  1  1  0  1  0   3   2   0   0
       C12  0  0  0  1  0  2  5  0  2   1   1   4   0
       C13  2  3  3  5  1  0  1  0  3   0   0   2   4

Overall Statistics

               Accuracy : 0.359
                 95% CI : (0.2917, 0.4306)
    No Information Rate : 0.0769
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3056

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: C1 Class: C2 Class: C3 Class: C4 Class: C5
Sensitivity            0.40000   0.33333   0.46667   0.40000   0.53333
Specificity            0.97778   0.96667   0.97778   0.96111   0.98333
Pos Pred Value         0.60000   0.45455   0.63636   0.46154   0.72727
Neg Pred Value         0.95135   0.94565   0.95652   0.95055   0.96196
Prevalence             0.07692   0.07692   0.07692   0.07692   0.07692
Detection Rate         0.03077   0.02564   0.03590   0.03077   0.04103
Detection Prevalence   0.05128   0.05641   0.05641   0.06667   0.05641
Balanced Accuracy      0.68889   0.65000   0.72222   0.68056   0.75833
                     Class: C6 Class: C7 Class: C8 Class: C9 Class: C10
Sensitivity            0.20000   0.46667   0.46667   0.40000    0.33333
Specificity            0.89444   0.91667   0.95000   0.95556    0.93889
Pos Pred Value         0.13636   0.31818   0.43750   0.42857    0.31250
Neg Pred Value         0.93064   0.95376   0.95531   0.95028    0.94413
Prevalence             0.07692   0.07692   0.07692   0.07692    0.07692
Detection Rate         0.01538   0.03590   0.03590   0.03077    0.02564
Detection Prevalence   0.11282   0.11282   0.08205   0.07179    0.08205
Balanced Accuracy      0.54722   0.69167   0.70833   0.67778    0.63611
                     Class: C11 Class: C12 Class: C13
Sensitivity             0.13333    0.26667    0.26667
Specificity             0.96111    0.93333    0.88889
Pos Pred Value          0.22222    0.25000    0.16667
Neg Pred Value          0.93011    0.93855    0.93567
Prevalence              0.07692    0.07692    0.07692
Detection Rate          0.01026    0.02051    0.02051
```

```
Detection Prevalence     0.04615     0.08205     0.12308
Balanced Accuracy        0.54722     0.60000     0.57778
```

```r
scores_pca <- as_tibble(pca$x[, 1:2, drop = FALSE]) %>%
  mutate(classe = y_treino)

ggplot(scores_pca, aes(x = PC1, y = PC2, color = classe)) +
  geom_point(alpha = 0.8) +
  labs(
    title = "Visualizacao do conjunto de treino no espaco PCA",
    x = "PC1",
    y = "PC2",
    color = "Classe"
  ) +
  theme_minimal()
```



14 Modelo 5: Random Forest

O Random Forest sera usado como classificador de comparacao. Ele costuma lidar bem com relacoes nao lineares e tambem permite estimar a importancia das variaveis.

```r
# ============================================================
# 14. Random Forest
# ============================================================
```

```r
set.seed(123)

modelo_rf <- randomForest(
  x = x_treino_norm,
  y = y_treino,
  ntree = 500,
  importance = TRUE
)

pred_rf <- predict(modelo_rf, x_teste_norm)

cm_rf <- confusionMatrix(pred_rf, y_teste)

cm_rf
```

```
Confusion Matrix and Statistics

          Reference
Prediction C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13
       C1   5  3  1  1  1  0  0  0  1   1   0   0   1
       C2   1  4  0  0  0  0  0  0  0   0   0   0   1
       C3   0  0  3  3  0  0  0  0  1   0   0   0   0
       C4   5  3  1  9  0  0  3  1  0   0   0   0   4
       C5   0  0  4  0  9  0  0  0  0   0   0   0   2
       C6   1  5  0  0  3  8  1  6  1   3   2   4   0
       C7   0  0  0  1  0  1  4  0  0   0   0   4   2
       C8   0  0  3  0  0  2  2  6  2   0   1   0   0
       C9   1  0  1  0  2  1  0  0  8   1   0   0   1
       C10  1  0  0  0  0  0  0  1  0   6  10   2   0
       C11  1  0  0  0  0  0  0  0  1   4   2   0   0
       C12  0  0  0  0  0  3  5  1  1   0   0   5   0
       C13  0  0  2  1  0  0  0  0  0   0   0   0   4

Overall Statistics

               Accuracy : 0.3744
                 95% CI : (0.3063, 0.4463)
    No Information Rate : 0.0769
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3222

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: C1 Class: C2 Class: C3 Class: C4 Class: C5
Sensitivity            0.33333   0.26667   0.20000   0.60000   0.60000
Specificity            0.95000   0.98889   0.97778   0.90556   0.96667
Pos Pred Value         0.35714   0.66667   0.42857   0.34615   0.60000
Neg Pred Value         0.94475   0.94180   0.93617   0.96450   0.96667
Prevalence             0.07692   0.07692   0.07692   0.07692   0.07692
Detection Rate         0.02564   0.02051   0.01538   0.04615   0.04615
Detection Prevalence   0.07179   0.03077   0.03590   0.13333   0.07692
```

```
Balanced Accuracy      0.64167   0.62778   0.58889   0.75278   0.78333
                     Class: C6 Class: C7 Class: C8 Class: C9 Class: C10
Sensitivity            0.53333   0.26667   0.40000   0.53333    0.40000
Specificity            0.85556   0.95556   0.94444   0.96111    0.92222
Pos Pred Value         0.23529   0.33333   0.37500   0.53333    0.30000
Neg Pred Value         0.95652   0.93989   0.94972   0.96111    0.94857
Prevalence             0.07692   0.07692   0.07692   0.07692    0.07692
Detection Rate         0.04103   0.02051   0.03077   0.04103    0.03077
Detection Prevalence   0.17436   0.06154   0.08205   0.07692    0.10256
Balanced Accuracy      0.69444   0.61111   0.67222   0.74722    0.66111
                     Class: C11 Class: C12 Class: C13
Sensitivity             0.13333    0.33333    0.26667
Specificity             0.96667    0.94444    0.98333
Pos Pred Value          0.25000    0.33333    0.57143
Neg Pred Value          0.93048    0.94444    0.94149
Prevalence              0.07692    0.07692    0.07692
Detection Rate          0.01026    0.02564    0.02051
Detection Prevalence    0.04103    0.07692    0.03590
Balanced Accuracy       0.55000    0.63889    0.62500
```

15 Comparacao dos modelos

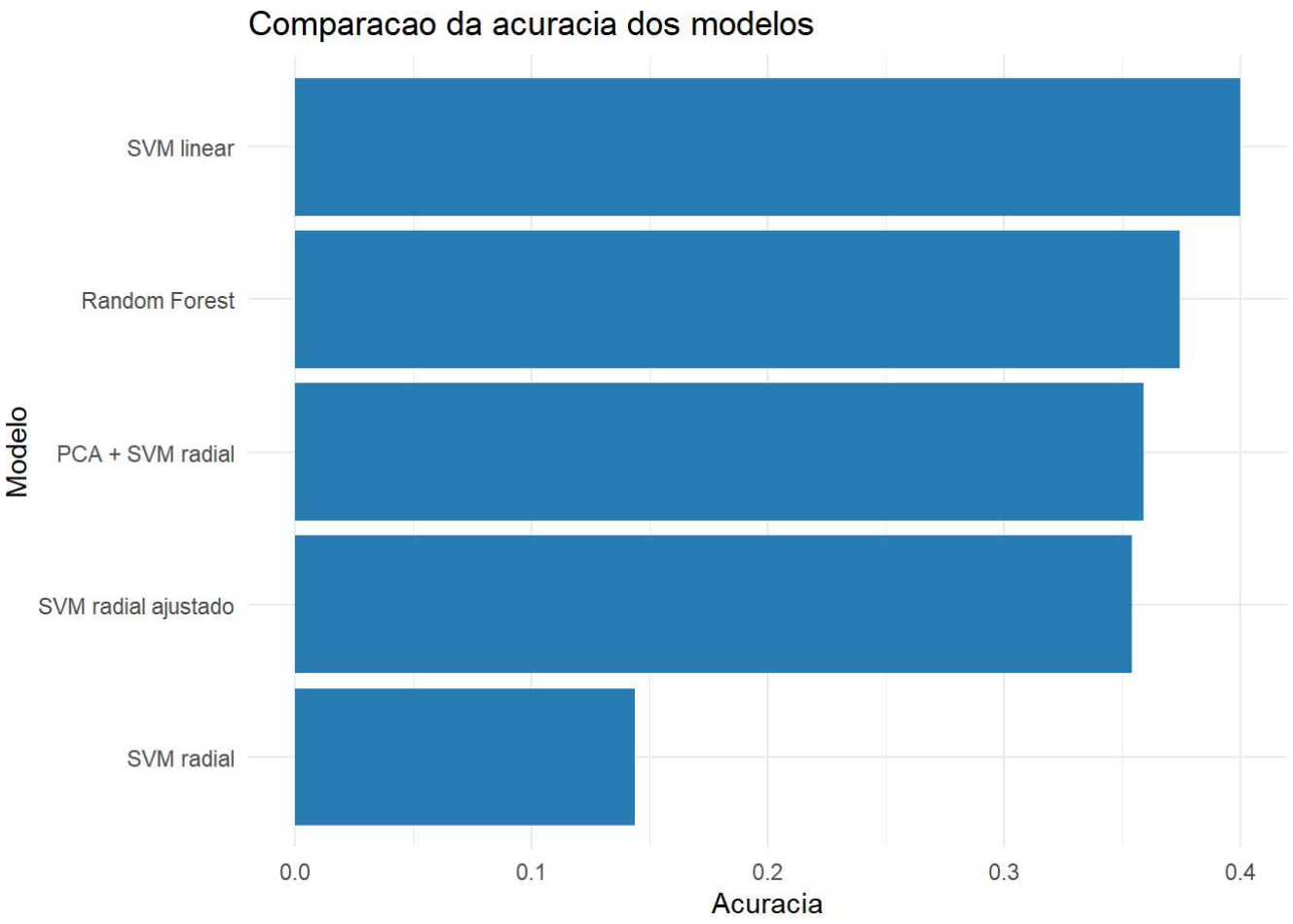
Os modelos serao comparados por acuracia, Kappa, sensibilidade macro, especificidade macro e F1 macro.

```
# ============================================================
# 15. Funcao de avaliacao dos modelos
# ============================================================

avaliar_modelo <- function(nome, matriz_confusao) {
  overall <- matriz_confusao$overall
  by_class <- matriz_confusao$byClass

  if (is.matrix(by_class)) {
    sensibilidade_macro <- mean(by_class[, "Sensitivity"], na.rm = TRUE)
    especificidade_macro <- mean(by_class[, "Specificity"], na.rm = TRUE)
    f1_macro <- mean(by_class[, "F1"], na.rm = TRUE)
  } else {
    sensibilidade_macro <- by_class["Sensitivity"]
    especificidade_macro <- by_class["Specificity"]
    f1_macro <- by_class["F1"]
  }

  tibble(
    modelo = nome,
    acuracia = as.numeric(overall["Accuracy"]),
    kappa = as.numeric(overall["Kappa"]),
    sensibilidade_macro = as.numeric(sensibilidade_macro),
    especificidade_macro = as.numeric(especificidade_macro),
    f1_macro = as.numeric(f1_macro)
  )
}

resultados <- bind_rows(
```

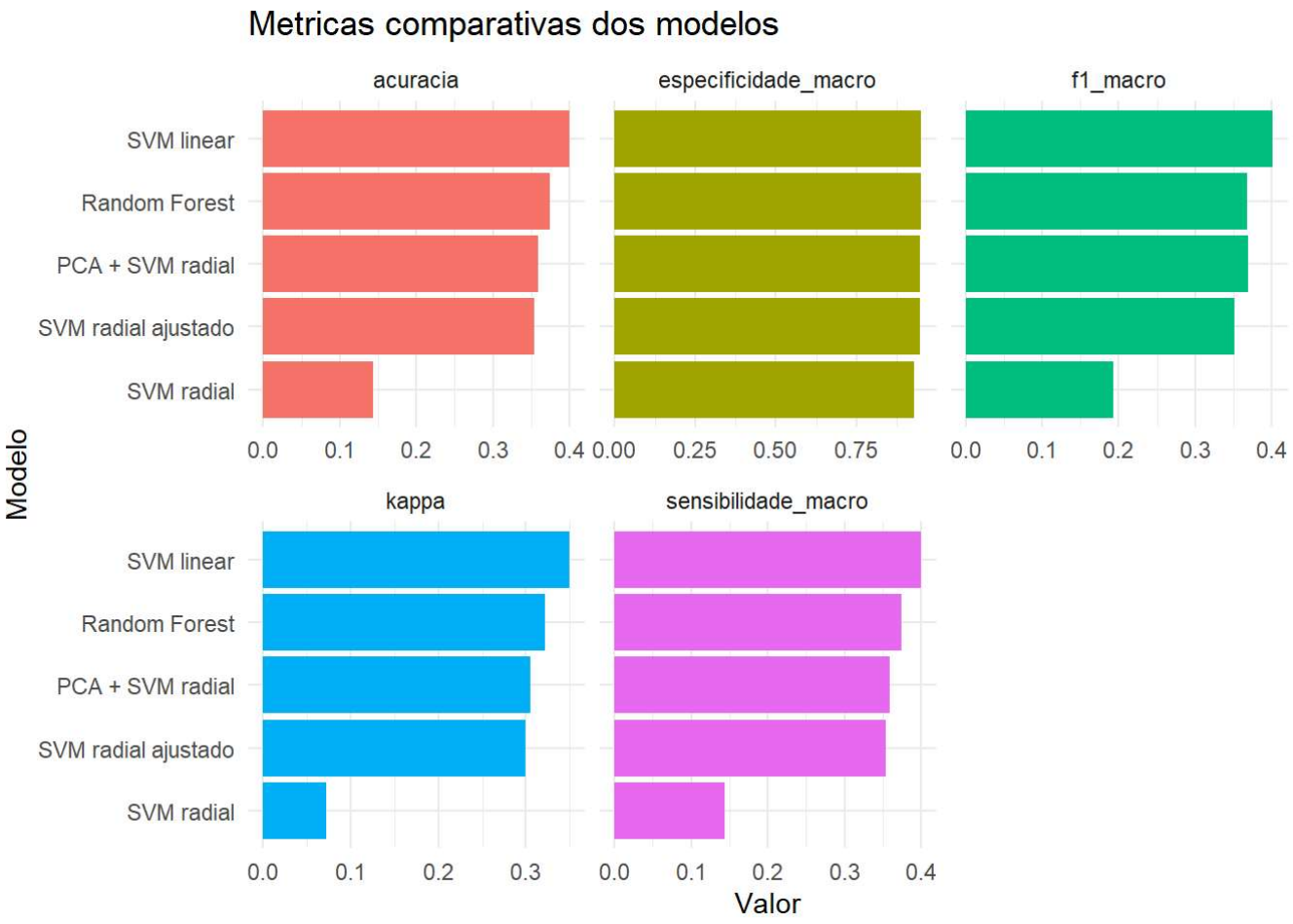
```
  avaliar_modelo("SVM linear", cm_svm_linear),
  avaliar_modelo("SVM radial", cm_svm_radial),
  avaliar_modelo("SVM radial ajustado", cm_svm_radial_ajustado),
  avaliar_modelo("PCA + SVM radial", cm_svm_pca),
  avaliar_modelo("Random Forest", cm_rf)
) %>%
  arrange(desc(acuracia))

resultados %>%
  mutate(across(where(is.numeric), ~ round(.x, 4))) %>%
  kable()
```

modelo	acuracia	kappa	sensibilidade_macro	especificidade_macro	f1_macro
SVM linear	0.4000	0.3500	0.4000	0.9500	0.4019
Random Forest	0.3744	0.3222	0.3744	0.9479	0.3688
PCA + SVM radial	0.3590	0.3056	0.3590	0.9466	0.3696
SVM radial ajustado	0.3538	0.3000	0.3538	0.9462	0.3520
SVM radial	0.1436	0.0722	0.1436	0.9286	0.1928

```
ggplot(resultados, aes(x = reorder(modelo, acuracia), y = acuracia)) +
  geom_col(fill = "#2C7FB8") +
  coord_flip() +
  labs(
    title = "Comparacao da acuracia dos modelos",
    x = "Modelo",
    y = "Acuracia"
  ) +
  theme_minimal()
```




```r
resultados_long <- resultados %>%
  pivot_longer(
    cols = c(acuracia, kappa, sensibilidade_macro, especificidade_macro, f1_macro),
    names_to = "metrica",
    values_to = "valor"
  )

ggplot(resultados_long, aes(x = reorder(modelo, valor), y = valor, fill = metrica)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  facet_wrap(~ metrica, scales = "free_x") +
  labs(
    title = "Metricas comparativas dos modelos",
    x = "Modelo",
    y = "Valor"
  ) +
  theme_minimal()
```



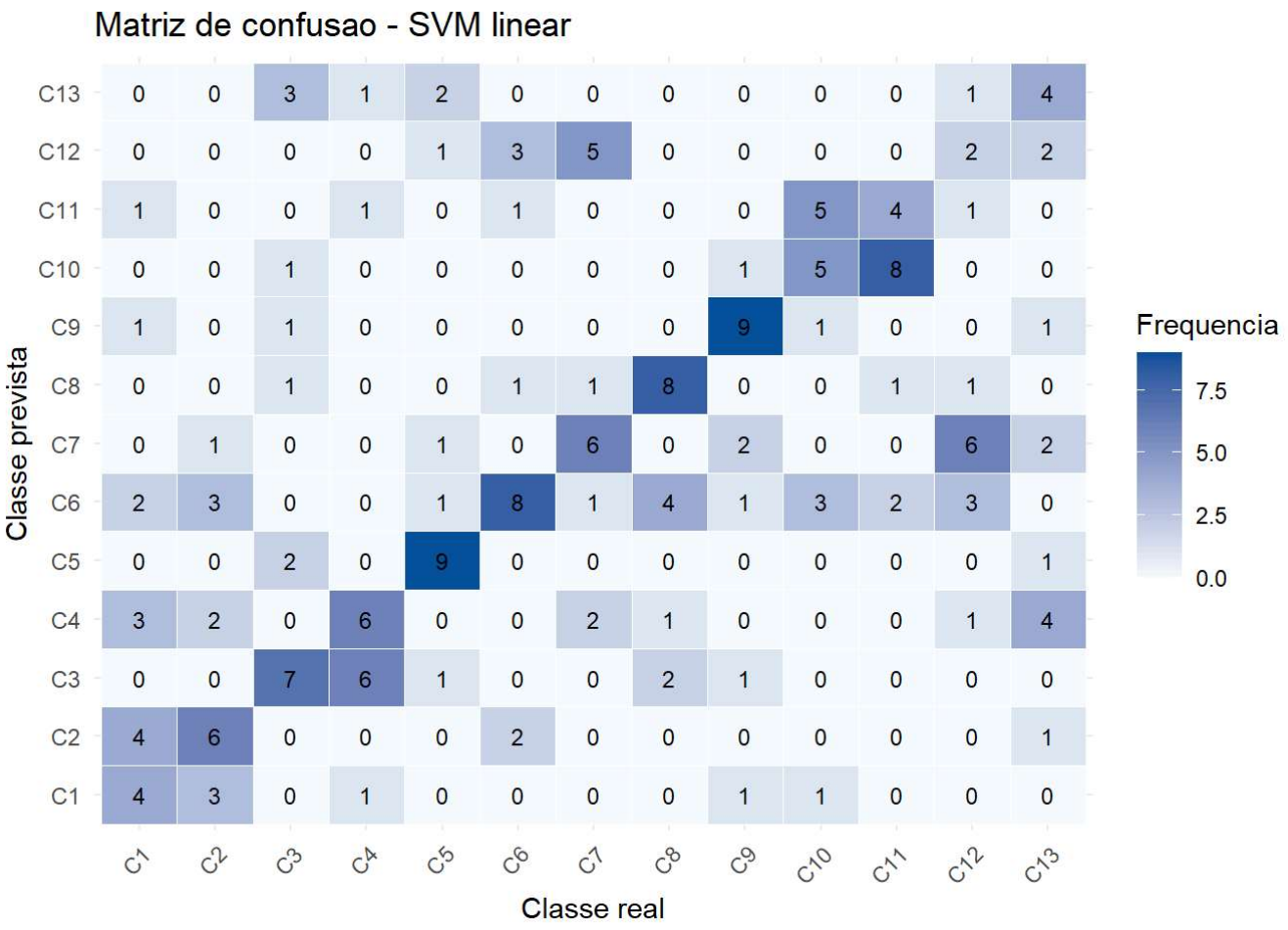
16 Melhor modelo

```
melhor_modelo <- resultados$modelo[1]

cat("Melhor modelo pela acuracia:", melhor_modelo, "\n")
```

```
Melhor modelo pela acuracia: SVM linear
```

```
cm_melhor <- switch(
  melhor_modelo,
  "SVM linear" = cm_svm_linear,
  "SVM radial" = cm_svm_radial,
  "SVM radial ajustado" = cm_svm_radial_ajustado,
  "PCA + SVM radial" = cm_svm_pca,
  "Random Forest" = cm_rf
)

as.data.frame(cm_melhor$table) %>%
  ggplot(aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), size = 3) +
  scale_fill_gradient(low = "#F7FBFF", high = "#08519C") +
  labs(
    title = paste("Matriz de confusao -", melhor_modelo),
    x = "Classe real",
```

```
    y = "Classe prevista",
    fill = "Frequencia"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



17 Importancia das variaveis

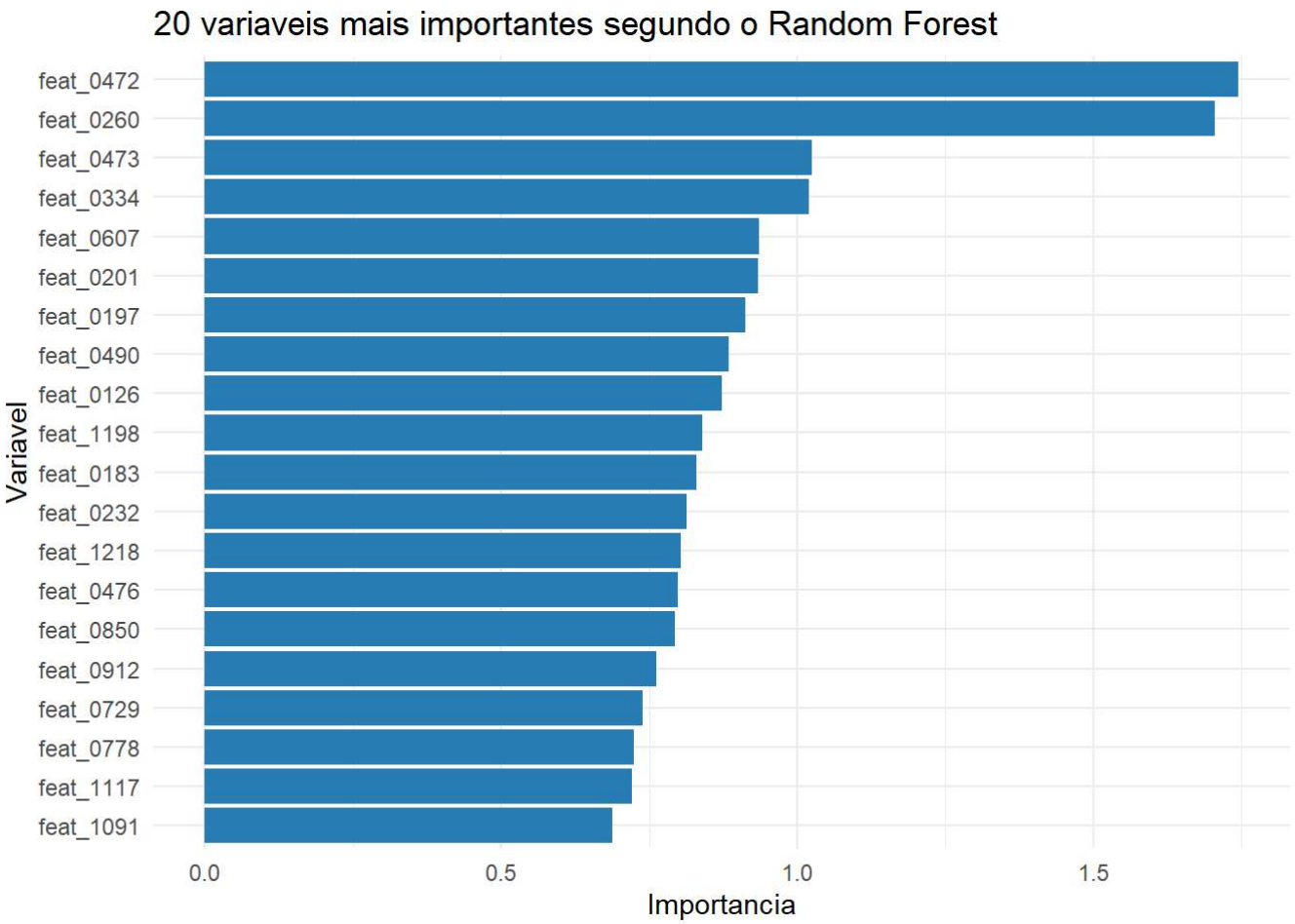
A importancia das variaveis sera estimada pelo Random Forest. Essa etapa ajuda a observar quais atributos contribuiram mais para a classificacao.

```
# ============================================================
# 16. Importancia das variaveis
# ============================================================

importancia <- importance(modelo_rf)

coluna_importancia <- if ("MeanDecreaseGini" %in% colnames(importancia)) {
  "MeanDecreaseGini"
} else {
  colnames(importancia)[1]
}

importancia_df <- tibble(
  variavel = rownames(importancia),
```

```
  importancia = importancia[, coluna_importancia]
) %>%
  arrange(desc(importancia))

importancia_df %>%
  slice_head(n = 20) %>%
  kable()
```

variavel	importancia
feat_0472	1.7446133
feat_0260	1.7042444
feat_0473	1.0239011
feat_0334	1.0189685
feat_0607	0.9355144
feat_0201	0.9340196
feat_0197	0.9121311
feat_0490	0.8838067
feat_0126	0.8725346
feat_1198	0.8392922
feat_0183	0.8291562
feat_0232	0.8134642
feat_1218	0.8038597
feat_0476	0.7979804
feat_0850	0.7941928
feat_0912	0.7613126
feat_0729	0.7384886
feat_0778	0.7236667
feat_1117	0.7209036
feat_1091	0.6878679

```
importancia_df %>%
  slice_head(n = 20) %>%
  ggplot(aes(x = reorder(variavel, importancia), y = importancia)) +
  geom_col(fill = "#2C7FB8") +
  coord_flip() +
  labs(
    title = "20 variaveis mais importantes segundo o Random Forest",
    x = "Variavel",
    y = "Importancia"
  ) +
  theme_minimal()
```



18 Salvamento dos resultados

```
# ============================================================
# 17. Salvar saidas
# ============================================================

dir.create("outputs", showWarnings = FALSE)

write_csv(dados, "outputs/features_rgb.csv")
write_csv(resultados, "outputs/resultados_modelos_rgb.csv")
write_csv(importancia_df, "outputs/importancia_variaveis_rgb.csv")

saveRDS(
  list(
    preproc                   = preproc,
    modelo_svm_linear         = modelo_svm_linear,
    modelo_svm_radial         = modelo_svm_radial,
    ajuste_svm_radial         = ajuste_svm_radial,
    modelo_svm_radial_ajustado = modelo_svm_radial_ajustado,
    modelo_svm_pca            = modelo_svm_pca,
    modelo_rf                 = modelo_rf,
    pca                       = pca,
    resultados                = resultados,
    extrator_cnn              = "MobileNetV2 (ImageNet, sem topo, pooling=avg, 1280 features)
  ),
  "outputs/modelos_rgb.rds"
```

```
)

cat("Arquivos salvos na pasta outputs.\n")
```

```
Arquivos salvos na pasta outputs.
```

19 Validacao Cruzada K-Fold

Na abordagem anterior (holdout), o conjunto de dados foi dividido uma unica vez em treino (70%) e teste (30%). Isso pode gerar uma estimativa otimista ou pessimista do desempenho real, dependendo de como a divisao ocorreu.

A validacao cruzada k-fold resolve esse problema dividindo o dataset em `k` partes iguais e estratificadas. Em cada rodada, uma parte diferente e usada como teste e as restantes como treino. O desempenho final e a media das `k` rodadas, o que aproveita todas as observacoes para avaliacao e produz uma estimativa mais estavel.

Nesta etapa, sera usado `k=5`. Para o SVM radial ajustado, serao reutilizados os melhores parametros encontrados no holdout (`cost` e `gamma`), evitando o custo computacional de re-otimizar em cada fold.

```
# ============================================================
# 18. Validacao cruzada k-fold (k=5, estratificada)
# ============================================================

set.seed(123)

k_folds <- 5L

folds <- createFolds(dados$classe, k = k_folds, list = TRUE, returnTrain = FALSE)

melhor_cost_kfold  <- if (!is.null(ajuste_svm_radial)) ajuste_svm_radial$best.parameters$cost
melhor_gamma_kfold <- if (!is.null(ajuste_svm_radial)) ajuste_svm_radial$best.parameters$gamma

cat("Parametros reutilizados no SVM ajustado: cost =", melhor_cost_kfold,
    "| gamma =", melhor_gamma_kfold, "\n")
```

```
Parametros reutilizados no SVM ajustado: cost = 1 | gamma = 0.001
```

```
metricas_folds <- vector("list", k_folds)

for (fold_i in seq_len(k_folds)) {
  idx_teste  <- folds[[fold_i]]
  idx_treino <- setdiff(seq_len(nrow(dados)), idx_teste)

  fold_treino <- dados[idx_treino, ]
  fold_teste  <- dados[idx_teste,  ]

  x_fold_tr <- fold_treino %>% select(-all_of(colunas_nao_preditoras))
  x_fold_te <- fold_teste  %>% select(-all_of(colunas_nao_preditoras))
  y_fold_tr <- droplevels(fold_treino$classe)
```

```r
  y_fold_te <- factor(fold_teste$classe, levels = levels(y_fold_tr))

  nzv_fold <- nearZeroVar(x_fold_tr)
  if (length(nzv_fold) > 0) {
    x_fold_tr <- x_fold_tr[, -nzv_fold, drop = FALSE]
    x_fold_te <- x_fold_te[, colnames(x_fold_tr), drop = FALSE]
  }

  pp_fold      <- preProcess(x_fold_tr, method = c("center", "scale"))
  x_fold_tr_n  <- predict(pp_fold, x_fold_tr)
  x_fold_te_n  <- predict(pp_fold, x_fold_te)

  m1 <- svm(x = x_fold_tr_n, y = y_fold_tr,
            kernel = "linear", cost = 1, scale = FALSE)

  m2 <- svm(x = x_fold_tr_n, y = y_fold_tr,
            kernel = "radial", cost = 10, gamma = 0.01, scale = FALSE)

  m3 <- svm(x = x_fold_tr_n, y = y_fold_tr,
            kernel = "radial", cost = melhor_cost_kfold,
            gamma = melhor_gamma_kfold, scale = FALSE)

  pca_fold    <- prcomp(x_fold_tr_n, center = FALSE, scale. = FALSE)
  var_acum_f  <- cumsum(pca_fold$sdev^2 / sum(pca_fold$sdev^2))
  n_comp_f    <- min(which(var_acum_f >= 0.95)[1], 30L, ncol(x_fold_tr_n))
  x_pca_tr    <- as.data.frame(pca_fold$x[, 1:n_comp_f, drop = FALSE])
  x_pca_te    <- as.data.frame(predict(pca_fold, x_fold_te_n)[, 1:n_comp_f, drop = FALSE])
  m4 <- svm(x = x_pca_tr, y = y_fold_tr,
            kernel = "radial", cost = 10, gamma = 0.01, scale = FALSE)

  m5 <- randomForest(x = x_fold_tr_n, y = y_fold_tr, ntree = 200L)

  metricas_folds[[fold_i]] <- bind_rows(
    avaliar_modelo("SVM linear",          confusionMatrix(predict(m1, x_fold_te_n), y_fold_te)
    avaliar_modelo("SVM radial",          confusionMatrix(predict(m2, x_fold_te_n), y_fold_te)
    avaliar_modelo("SVM radial ajustado", confusionMatrix(predict(m3, x_fold_te_n), y_fold_te)
    avaliar_modelo("PCA + SVM radial",    confusionMatrix(predict(m4, x_pca_te),    y_fold_te)
    avaliar_modelo("Random Forest",       confusionMatrix(predict(m5, x_fold_te_n), y_fold_te)
  ) %>% mutate(fold = fold_i)

  cat(sprintf("Fold %d/%d concluido.\n", fold_i, k_folds))
}
```

```
Fold 1/5 concluido.
Fold 2/5 concluido.
Fold 3/5 concluido.
Fold 4/5 concluido.
Fold 5/5 concluido.
```

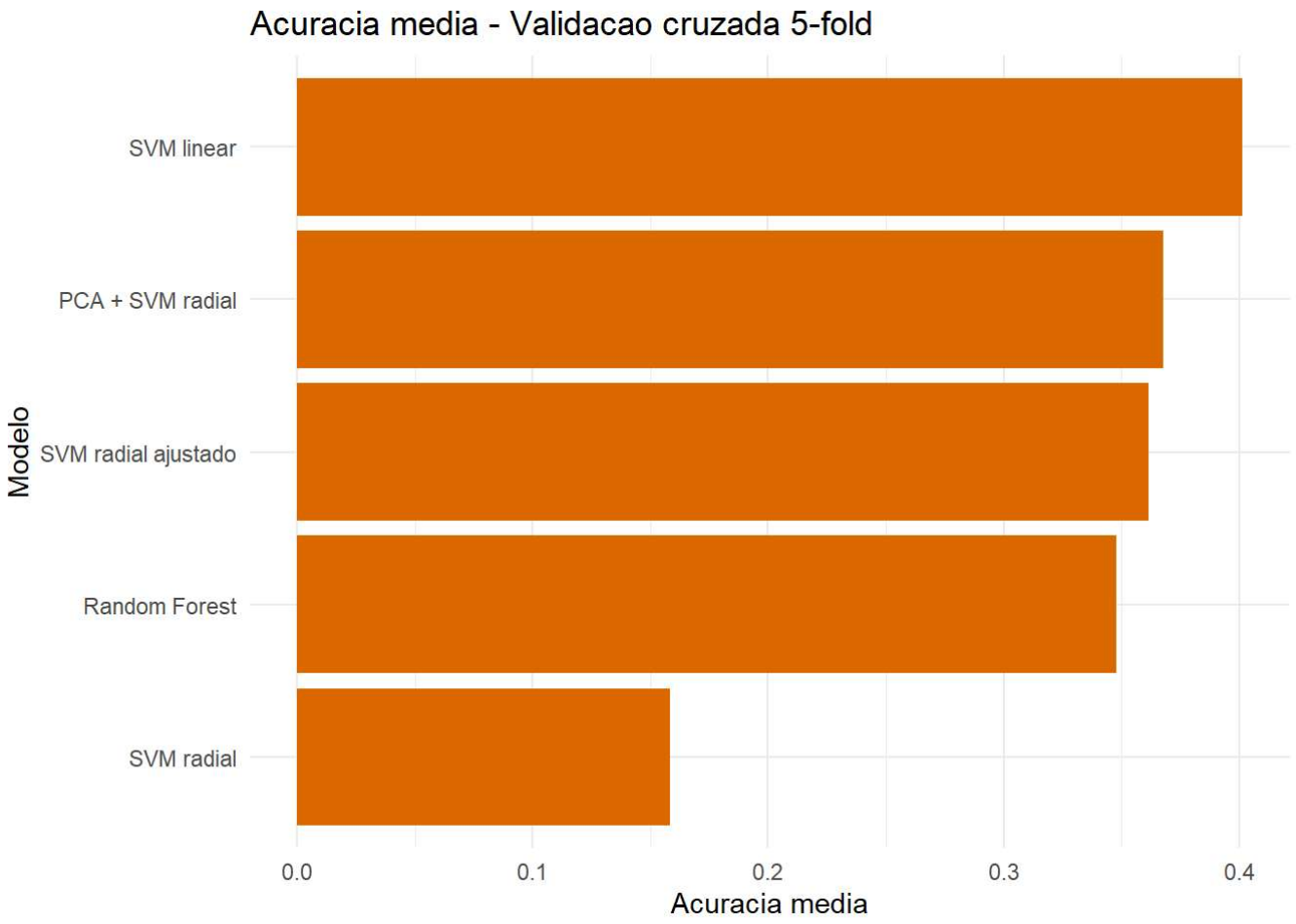
```r
resultados_kfold <- bind_rows(metricas_folds) %>%
  group_by(modelo) %>%
  summarise(across(c(acuracia, kappa, sensibilidade_macro, especificidade_macro, f1_macro), me
            .groups = "drop") %>%
```

```
  arrange(desc(acuracia))

write_csv(resultados_kfold, "outputs/resultados_kfold_rgb.csv")

resultados_kfold %>%
  mutate(across(where(is.numeric), ~ round(.x, 4))) %>%
  kable(caption = "Acuracia media - K-Fold (k=5)")
```

Acuracia media - K-Fold (k=5)

modelo	acuracia	kappa	sensibilidade_macro	especificidade_macro	f1_macro
SVM linear	0.4015	0.3517	0.4015	0.9501	0.4093
PCA + SVM radial	0.3677	0.3150	0.3677	0.9473	0.3784
SVM radial ajustado	0.3615	0.3083	0.3615	0.9468	0.3668
Random Forest	0.3477	0.2933	0.3477	0.9456	0.3502
SVM radial	0.1585	0.0883	0.1585	0.9299	0.2233

```
ggplot(resultados_kfold, aes(x = reorder(modelo, acuracia), y = acuracia)) +
  geom_col(fill = "#E06C00") +
  coord_flip() +
  labs(
    title = "Acuracia media - Validacao cruzada 5-fold",
    x = "Modelo",
    y = "Acuracia media"
  ) +
  theme_minimal()
```



20 Comparacao: Holdout vs K-Fold

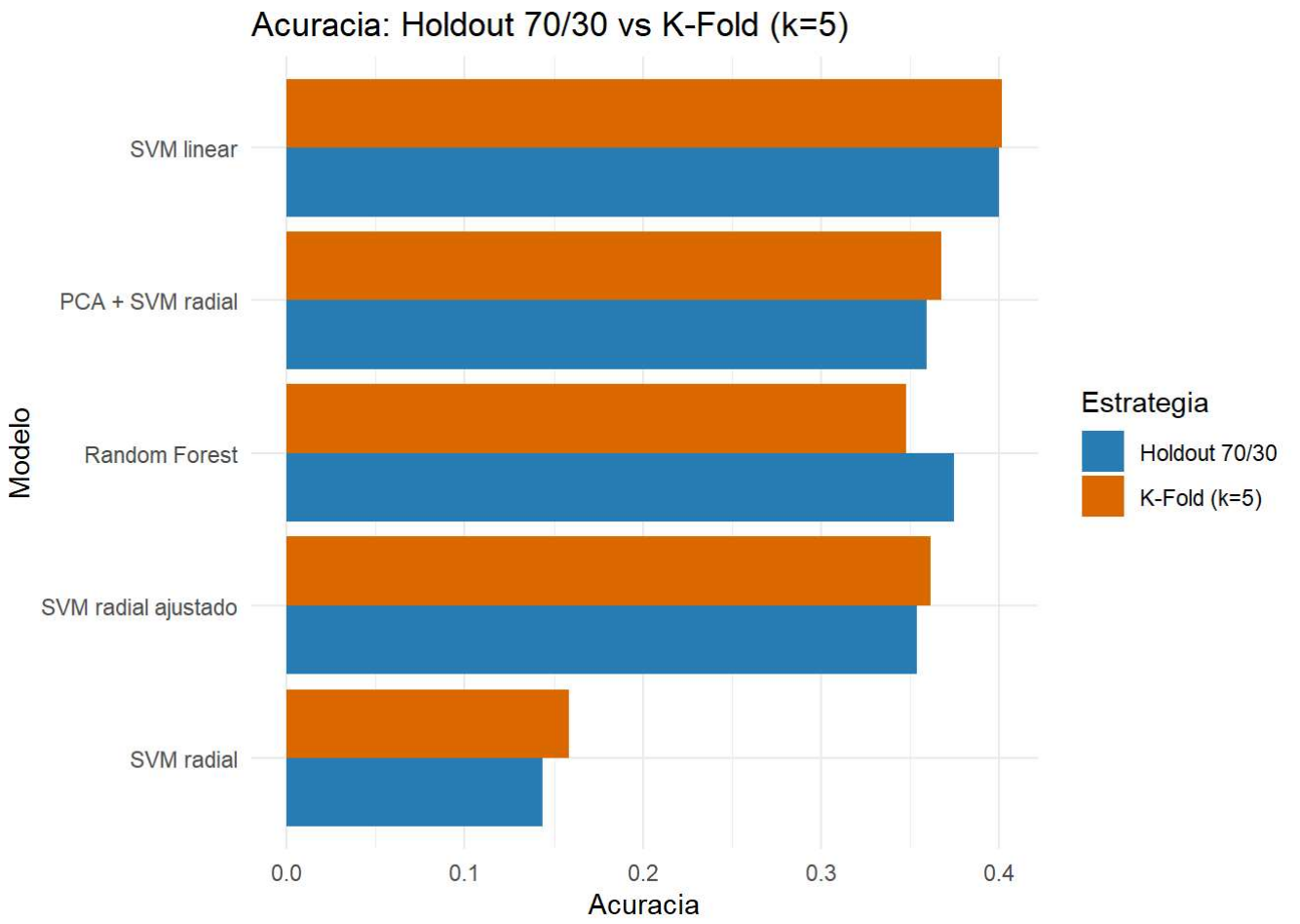
```
# ============================================================
# 19. Comparacao direta entre as duas estrategias
# ============================================================

comparacao <- bind_rows(
  resultados      %>% mutate(estrategia = "Holdout 70/30"),
  resultados_kfold %>% mutate(estrategia = "K-Fold (k=5)")
) %>%
  select(estrategia, modelo, acuracia, kappa, f1_macro) %>%
  arrange(estrategia, desc(acuracia))

comparacao %>%
  mutate(across(where(is.numeric), ~ round(.x, 4))) %>%
  kable(caption = "Comparacao entre Holdout 70/30 e Validacao Cruzada K-Fold")
```

Comparacao entre Holdout 70/30 e Validacao Cruzada K-Fold

estrategia	modelo	acuracia	kappa	f1_macro
Holdout 70/30	SVM linear	0.4000	0.3500	0.4019
Holdout 70/30	Random Forest	0.3744	0.3222	0.3688
Holdout 70/30	PCA + SVM radial	0.3590	0.3056	0.3696

estrategia	modelo	acuracia	kappa	f1_macro
Holdout 70/30	SVM radial ajustado	0.3538	0.3000	0.3520
Holdout 70/30	SVM radial	0.1436	0.0722	0.1928
K-Fold (k=5)	SVM linear	0.4015	0.3517	0.4093
K-Fold (k=5)	PCA + SVM radial	0.3677	0.3150	0.3784
K-Fold (k=5)	SVM radial ajustado	0.3615	0.3083	0.3668
K-Fold (k=5)	Random Forest	0.3477	0.2933	0.3502
K-Fold (k=5)	SVM radial	0.1585	0.0883	0.2233

```
ggplot(comparacao, aes(x = reorder(modelo, acuracia), y = acuracia, fill = estrategia)) +
  geom_col(position = "dodge") +
  coord_flip() +
  scale_fill_manual(values = c("Holdout 70/30" = "#2C7FB8", "K-Fold (k=5)" = "#E06C00")) +
  labs(
    title = "Acuracia: Holdout 70/30 vs K-Fold (k=5)",
    x = "Modelo",
    y = "Acuracia",
    fill = "Estrategia"
  ) +
  theme_minimal()
```



```r
melhor_holdout  <- max(resultados$acuracia)
melhor_kf       <- max(resultados_kfold$acuracia)
melhor_metodo   <- if (melhor_kf >= melhor_holdout) "K-Fold (k=5)" else "Holdout 70/30"
resultados_finais <- if (melhor_kf >= melhor_holdout) resultados_kfold else resultados

cat("Acuracia maxima - Holdout:", round(melhor_holdout, 4), "\n")
```

```
Acuracia maxima - Holdout: 0.4
```

```r
cat("Acuracia maxima - K-Fold: ", round(melhor_kf,      4), "\n")
```

```
Acuracia maxima - K-Fold:  0.4015
```

```r
cat("Metodo adotado como referencia:", melhor_metodo, "\n")
```

```
Metodo adotado como referencia: K-Fold (k=5)
```

21 Discussao

Os resultados permitem comparar diferentes classificadores aplicados sobre atributos extraidos por *transfer learning* com a rede MobileNetV2, e tambem avaliar o impacto da estrategia de avaliacao adotada.

A extracao de atributos por CNN pre-treinada substitui o calculo manual de medias, desvios e histogramas de cor por um vetor de 1280 atributos que codificam padroes visuais de alto nivel aprendidos em mais de um milhao de imagens do ImageNet. Essa representacao tende a ser muito mais discriminativa para problemas de classificacao de cenas.

O SVM linear busca fronteiras lineares nesse espaco de 1280 dimensoes. O SVM radial permite fronteiras nao lineares, sendo mais flexivel. O PCA reduz a dimensionalidade antes do SVM, eliminando redundancias entre as 1280 features da CNN. O Random Forest serve como referencia nao-linear e permite identificar quais atributos foram mais relevantes.

Sobre a estrategia de avaliacao: na primeira abordagem, o dataset foi dividido uma unica vez em 70% treino e 30% teste (holdout). Embora simples, esse metodo pode gerar estimativas instáveis com datasets de tamanho moderado, pois o resultado depende de como essa divisao ocorreu. Para verificar se os resultados eram representativos, o experimento foi repetido com validacao cruzada k-fold com k=5: o dataset e dividido em 5 partes estratificadas e, em cada rodada, uma parte diferente e usada como teste. O desempenho final e a media das 5 rodadas.

```r
if (melhor_kf >= melhor_holdout) {
  cat("O k-fold produziu acuracia igual ou superior ao holdout, sendo adotado como resultado p
} else {
  cat("O holdout produziu acuracia superior ao k-fold. Isso pode indicar que a divisao holdout
      "O k-fold e metodologicamente mais robusto, mas neste caso o holdout resultou em estimat
}
```

```
O k-fold produziu acuracia igual ou superior ao holdout, sendo adotado como resultado
principal.
```

22 Conclusao

Este trabalho implementou uma pipeline de classificacao de imagens RGB baseada em *transfer learning*, reducao de dimensionalidade e classificadores supervisionados, avaliada por duas estrategias distintas.

A estrategia de usar a MobileNetV2 pre-treinada como extrator de atributos, seguida de SVM ou Random Forest, combina o poder de representacao das redes neurais com a interpretabilidade dos classificadores classicos, sem necessidade de treinar a CNN do zero.

```
cat(sprintf(
  "O melhor resultado foi obtido com '%s' usando a estrategia '%s', com acuracia de %.1f%%.\n"
  resultados_finais$modelo[1],
  melhor_metodo,
  max(resultados_finais$acuracia) * 100
))
```

```
O melhor resultado foi obtido com 'SVM linear' usando a estrategia 'K-Fold (k=5)', com
acuracia de 40.2%.
```

Como expansao futura, o trabalho pode ser repetido com outros extratores CNN presentes no material da disciplina (EfficientNetB0, ResNet50, VGG19) e com a incorporacao das imagens TIR ao vetor de atributos, aproveitando a informacao termica disponivel no dataset.